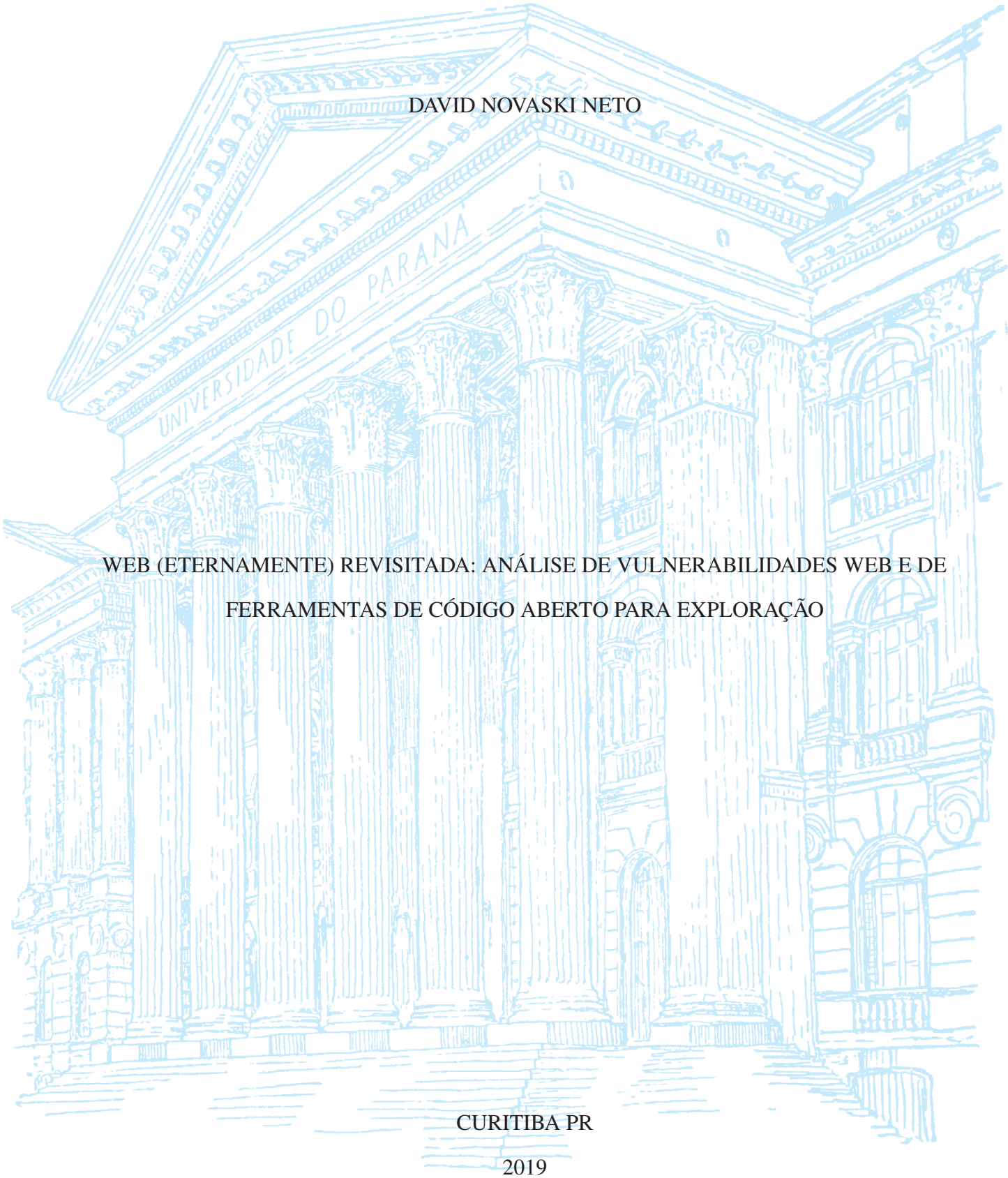
UNIVERSIDADE FEDERAL DO PARANÁ

DAVID NOVASKI NETO




# WEB (ETERNAMENTE) REVISITADA: ANÁLISE DE VULNERABILIDADES WEB E DE FERRAMENTAS DE CÓDIGO ABERTO PARA EXPLORAÇÃO

CURITIBA PR

2019

DAVID NOVASKI NETO

# WEB (ETERNAMENTE) REVISITADA: ANÁLISE DE VULNERABILIDADES WEB E DE FERRAMENTAS DE CÓDIGO ABERTO PARA EXPLORAÇÃO

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Profa. Dra. Letícia Mara Peres.

Coorientador: Prof. Dr. André Grégio.

CURITIBA PR

2019

Catalogação na Fonte: Sistema de Bibliotecas, UFPR
Biblioteca de Ciência e Tecnologia

N936w Novaski Neto, David
Web (eternamente) revisitada: análise de vulnerabilidades web e de ferramentas de código aberto para exploração [recurso eletrônico] / David Novaski Neto. – Curitiba, 2019.

Dissertação – Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós – Graduação em Informática, 2019.

Orientadora: Letícia Mara Peres. Coorientador: André Grégio.

1. Acesso à Internet. 2. Aplicações Web. 3. Teste de invasão (Medidas de segurança para computadores). I. Universidade Federal do Paraná. II. Peres, Letícia Mara. III. Grégio, André. IV. Título.

CDD: 658.478

Bibliotecária: Vanusa Maciel CRB- 9/1928

MINISTÉRIO DA EDUCAÇÃO
SETOR DE CIENCIAS EXATAS
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO INFORMÁTICA - 40001016034P5

# TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **DAVID NOVASKI NETO** intitulada: **WEB (ETERNAMENTE) REVISITADA: ANÁLISE DE VULNERABILIDADES WEB E DE FERRAMENTAS DE CÓDIGO ABERTO PARA EXPLORAÇÃO**, sob orientação da Profa. Dra. LETICIA MARA PERES, que após após terem inquirido o aluno e realizado a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 14 de Agosto de 2019.

LETICIA MARA PERES
Presidente da Banca Examinadora (UNIVERSIDADE FEDERAL DO PARANÁ)

MARIA CLAUDIA FIGUEIREDO PEREIRA EMER
Avaliador Externo (UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ)

CARLOS ALBERTO MAZIERO
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
Setor de Ciências Exatas
Coordenação de Pós-Graduação em Informática

*Esta dissertação é dedicada às minhas filhas, Nicole e Laís.*

## AGRADECIMENTOS

Agradeço à Profa. Dra. Letícia Mara Peres e ao Prof. Dr. André Grégio pela orientação e paciência na elaboração deste trabalho.

Agradeço a ITAIPU Binacional, ao Parque Tecnológico Itaipu – PTI e a Universidade Federal do Paraná – UFPR, por viabilizarem este programa de mestrado em Foz do Iguaçu.

## RESUMO

As aplicações Web evoluíram de somente apresentar páginas estáticas para aplicações transacionais que suportam conteúdo dinâmico, na qual diversas organizações oferecem seus serviços pela Internet, muito deles considerados críticos, como serviços financeiros. Em função da criticidade destes serviços, as aplicações Web são alvos de constantes ataques. Neste sentido, testes de segurança devem ser executados visando tornar as aplicações Web mais seguras. Diante deste contexto, o objetivo deste trabalho é levantar e explorar pela execução ferramentas de código aberto para testes de intrusão em aplicações Web, abordando o estado das ferramentas em relação ao panorama de tecnologias Web e aspectos relacionados à integração entre as ferramentas. A partir do estudo dos riscos presentes em todas as publicações do projeto OWASP Top 10, identifica-se que o principal problema de segurança em aplicações Web é ocasionado pela falta de validação nos dados de entrada, assim, foram selecionadas três vulnerabilidades para serem exploradas: Injeção de SQL (A1), *Cross Site Scripting* (A7) e Entidades Externas de XML (A4). O alvo selecionado para execução dos testes de intrusão foi a aplicação WebGoat, a qual trata-se de uma *Single-Page Application*. Foram encontradas 104 ferramentas de código aberto para execução de testes de intrusão em aplicações Web, com foco nas etapas de análise e exploração de vulnerabilidades da metodologia PTES. Dentre as 104 ferramentas, foram selecionadas 14 para serem aplicadas na execução dos testes de intrusão, a saber: Arachni, Beef, Htcap, IronWASP, Metasploit, Skipfish, SQLMap, Vega, W3af, Wapiti, Wfuzz, XSSer, Xenotix e ZAP. O resultados dos testes indicam que o componente *crawler* das ferramentas não está preparado para trabalhar com aplicações *Single-Page Application*, uma vez que somente duas ferramentas conseguem obter os pontos de entrada da aplicação WebGoat, a saber: Arachni e Htcap. O componente de detecção teve baixo percentual de identificação, somente as ferramentas IronWASP, Vega, ZAP e SQLmap detectaram a vulnerabilidade Injeção de SQL, enquanto que a vulnerabilidade XSS refletido, somente foi detectada pelas ferramentas ZAP e Xenotix. Nenhuma das ferramentas detectou as vulnerabilidades XSS armazenado, XSS DOM e XXE. As ferramentas para explorar *Cross Site Scripting* não foram avaliadas em função das validações que a aplicação WebGoat implementa. Com relação à vulnerabilidade XXE, não foi encontrada nenhuma ferramenta para explorá-la. Somente foi possível executar um teste de intrusão completo na vulnerabilidade Injeção de SQL, e para executá-lo foi necessário aplicar três ferramentas distintas, a saber: wapiti-getcookie, Htcap e SQLmap. A primeira ferramenta foi utilizada para obter o identificador de sessão, a segunda para obter os pontos de entrada e a terceira para detectar e explorar a vulnerabilidade. Como resultado da exploração, a ferramenta SQLmap forneceu uma cópia dos dados da aplicação.



Palavras-chave: segurança Web, vulnerabilidade, teste de intrusão

## ABSTRACT

Web applications have progressed from an application that only rendered static pages to a transactional application, in which several organizations offer their services over the Internet, much of them considered critical, such as financial services. Due to the criticality of these services, Web applications are the targets of constant attacks. In this sense, security testing should be performed to make Web applications more secure. In this context, the objective of this work is to raise and explore by running open source tools for pen test in Web applications, addressing the state of the tools in relation to the Web technologies scenario and aspects related to integration between the tools. From the study of risks present in all publications of the OWASP Top 10 project, it is identified that the main security problem in Web applications is caused by the lack of validation in the input data. Three vulnerabilities were selected to be exploited related to this problem: SQL Injection (A1), Cross Site Scripting (A7) and XML External Entities (A4). The target selected to perform the pen tests was the WebGoat application, which is a Single-Page Application. 104 open source tools have been found to perform pen tests on Web applications, focusing on the steps of analysis and exploitation of vulnerabilities of PTES methodology. Among the 104 tools, 14 were selected to be applied in the execution of the pen tests. The tools applied in the tests were Arachni, Beef, Htcap, IronWASP, Metasploit, Skipfish, SQLMap, Vega, W3af, Wapiti, Wfuzz, XSSer, Xenotix and ZAP. The results of the tests demonstrated that the tool's crawler component is not prepared to work with Single-Page Application, since only two tools are able to obtain WebGoat application entry points: Arachni and Htcap. The detection component had a low percentage of identification, only the IronWASP, Vega, ZAP, and SQLmap tools detected the SQL Injection vulnerability, while the XSS vulnerability was only detected by the ZAP and Xenotix tools. None of the tools detected the stored XSS, XSS DOM, and XXE vulnerabilities. The tools for exploring Cross Site Scripting have not been evaluated because of the validations that the WebGoat application implements. About the XXE vulnerability, no tool was found to exploit it. It was only possible to perform a full pen test on the SQL Injection vulnerability, and in order to execute it, it was necessary to apply three different tools: wapiti-getcookie, Htcap, and SQLmap. The first tool was used to obtain the session identifier, the second tool to obtain the entry points, and the third tool to detect and exploit the vulnerability. As a result of the exploit, the SQLmap tool provided a copy of the application data.



Keywords: Web security, vulnerability, pen test

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE CÓDIGOS

## LISTA DE ACRÔNIMOS

| | |
|---|---|
| AJAX | Asynchronous JavaScript and XML |
| API | Application Programming Interface |
| ASP | Active Server Pages |
| DOM | Document Object Model |
| EL | Expression Language |
| HTML | Hypertext Markup Language |
| HTTP | Hypertext Transfer Protocol |
| ISAAF | Information Systems Security Assessment Framework |
| JSON | JavaScript Object Notation |
| JSP | JavaServer Pages |
| LDAP | Lightweight Directory Access Protocol |
| NIST | National Institute of Standards and Technology |
| NoSQL | Non Structured Query Language |
| ORM | Object-relational mapping |
| OSSTMM | Open Source Security Testing Methodology Manual |
| OWASP | Open Web Application Security Project |
| PHP | Hypertext Preprocessor |
| PTES | Penetration Testing Execution Standard |
| SQL | Structured Query Language |
| SPA | Single Page Application |
| URL | Uniform Resource Locator |
| WAVSEP | Web Application Vulnerability Scanners Evaluation Project |
| XML | Extensible Markup Language |
| XSS | Cross Site Scripting |

**SUMÁRIO**

# 1 INTRODUÇÃO

## 1.1 CONTEXTO

As aplicações Web evoluíram de somente apresentar páginas estáticas para aplicações distribuídas baseadas em transações que suportam conteúdo dinâmico. Estas aplicações podem ser consideradas um ecossistema composto por um grande número de componentes e tecnologias, como protocolo HTTP (Hypertext Markup Language), servidores Web, tecnologias para desenvolvimento de componentes que são executadas no lado servidor (JSP, PHP e ASP), navegadores e tecnologias para desenvolvimento de componentes que são executadas no lado cliente (*JavaScript*, *Flash* e *Java Applets*) (Li e Xue, 2014). A Figura 1.1 mostra uma visão geral de uma aplicação Web.

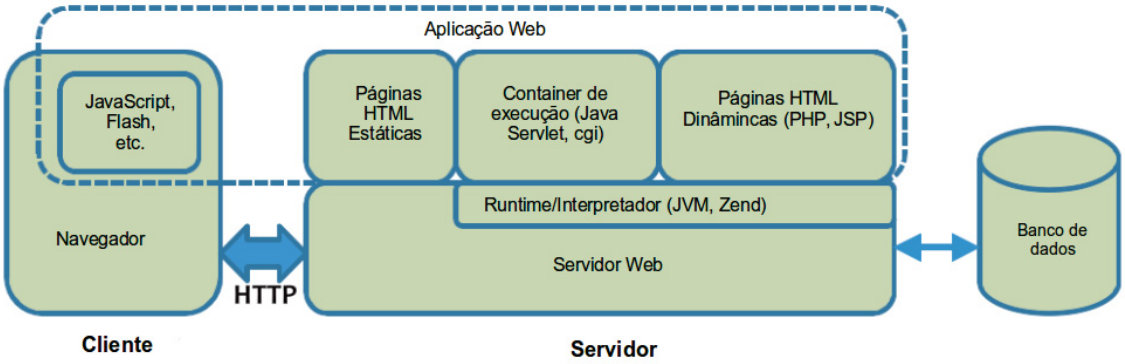



Figura 1.1: Visão geral de uma aplicação Web. Fonte: Li e Xue (2014)

A popularidade da plataforma Web pode ser atribuído aos seguintes fatores: facilidade para acessar a aplicação de forma remota, o cliente da aplicação (navegador) pode ser executado em diversos sistemas operacionais garantindo independência de plataforma e facilidade de distribuição de novas atualizações da aplicação (Li e Xue, 2014).

Diversos serviços críticos são disponibilizados através de aplicações Web. Geralmente, tais serviços armazenam e manipulam informações importantes, por exemplo, informações financeiras, saúde e pessoais (Li e Xue, 2014).

## 1.2 MOTIVAÇÃO

Em função da criticidade dos serviços disponibilizados e da facilidade de acesso às aplicações Web, tais aplicações são alvos de ataques (Li e Xue, 2014), como roubo de dados confidenciais. Segundo relatório de Stephens (2017), 30 % dos ataques são direcionados para este tipo de aplicação. Um ataque bem sucedido pode causar danos severos ao negócio de uma empresa, por exemplo, perdas financeiras e acarretar consequências legais (Li e Xue, 2014).

Metodologias de desenvolvimento de aplicações Web e *framework* de testes fornecem suporte limitado a segurança, ocasionando em grande esforço para implementar medidas de segurança, sendo que tal esforço é deixado de lado em função dos prazos assumidos. Como resultado, existem diversas aplicações Web publicadas na Internet com vulnerabilidades (Li e Xue, 2014). Testar uma aplicação Web não é tarefa simples uma vez que possui natureza heterogênea, composta por diferentes componentes que normalmente são implementados em diferentes linguagens de programação e *framework* de desenvolvimento. (Li et al., 2014b). Conforme

relatório da Acunetix (2017), 55% das aplicações Web avaliadas continham vulnerabilidades consideradas de alto risco, por exemplo, Injeção de SQL e *Cross Site Scripting*. Além de vulnerabilidades de alto risco, 84% também continham vulnerabilidades consideradas de médio risco.

Diversas pesquisas foram realizadas e como resultado várias técnicas, métodos e ferramentas de testes de intrusão foram desenvolvidos para detectar e explorar vulnerabilidades em aplicações Web. Geralmente, as ferramentas para testes de intrusão endereçam tipos diferentes de vulnerabilidades ou quando identificam uma mesma vulnerabilidade utilizam técnicas e implementações diferentes para detectá-la, gerando resultados distintos. Por exemplo, cada ferramenta pode utilizar de distintas heurísticas para geração dos dados de teste. Diante deste cenário, os profissionais de segurança têm dificuldade em selecionar uma ferramenta ou uma combinação delas para resolver suas necessidades (Li e Xue, 2014).

## 1.3 OBJETIVO

### 1.3.1 Objetivo Geral

O objetivo deste trabalho é levantar e explorar pela sua execução ferramentas de código aberto para testes de intrusão em aplicações Web, abordando o estado das ferramentas em relação ao panorama de tecnologias Web e aspectos relacionados à integração entre as ferramentas.

### 1.3.2 Objetivos específicos

Os objetivos específicos deste trabalho são:

- Apresentar a evolução das vulnerabilidades em aplicações Web, relatando os maiores problemas de segurança enfrentados por este tipo de aplicação.
- Levantar o estado da arte das ferramentas de código aberto para execução de testes de intrusão em aplicações Web que focam na análise e exploração de vulnerabilidades.
- Executar testes de intrusão em aplicações Web usando as ferramentas levantadas, avaliando a viabilidade de integrar as ferramentas aproveitando o resultado produzido por uma ferramenta na execução das demais.

## 1.4 METODOLOGIA DO TRABALHO

Esta seção apresenta a metodologia utilizada para atingir o objetivo proposto, contendo às seguintes etapas: apresentar a evolução das vulnerabilidade, identificar ferramentas, planejar a execução dos testes de intrusão, efetuar o reconhecimento da aplicação alvo, executar os testes de intrusão e analisar os resultados. A Figura 1.2 apresenta as etapas desta metodologia. Segue uma descrição das seis etapas da metodologia.

1. **Apresentar a evolução das vulnerabilidades**: esta etapa tem como objetivo apresentar os maiores problemas de segurança enfrentado pelas aplicações Web, tendo como base todas as publicações do projeto OWASP (*Open Web Application Security Project*) Top 10 (OWASP, 2017). Os resultados desta etapa são apresentados na Seção 3.1.
2. **Identificar ferramentas**: esta etapa tem como finalidade identificar ferramentas de código aberto para execução de testes de intrusão em aplicações Web com foco nas

etapas de análise e exploração de vulnerabilidades da metodologia PTES (*Penetration Testing Execution Standard*) (PTES, 2012). Será realizado um estudo para determinar quais vulnerabilidades cada ferramenta identifica e/ou explora, bem como, a qual etapa da metodologia PTES a ferramenta se aplica. O levantamento se dará pela realização de pesquisas em artigos, livros e sites da Internet. Os resultados desta etapa são apresentados na Seção 3.2.

3. **Planejar a execução dos testes de intrusão**: o objetivo desta etapa é definir o alvo dos testes de intrusão, as vulnerabilidades que serão exploradas, as ferramentas que serão aplicadas, os recursos computacionais necessários e as atividades que serão realizadas na execução dos testes de intrusão. As vulnerabilidades serão selecionadas de acordo com os resultados obtidos na etapa 1 desta metodologia. Com relação às ferramentas, serão selecionadas com base na etapa 2 da metodologia, nas avaliações e comparações encontradas em artigos, e na lista de vulnerabilidades que serão exploradas. O planejamento é apresentado na Seção 4.1.

4. **Efetuar o reconhecimento da aplicação alvo**: o propósito desta etapa é identificar os casos de uso, pontos de entradas e tecnologias empregadas referente aplicação alvo, bem como, entender os cabeçalhos HTTP, parâmetros e *cookies* encontrados nas requisições e respostas desta aplicação. Nesta etapa serão aplicadas ferramentas de *Proxy* e *debug* do navegador. Os resultados desta etapa são apresentados na Seção 4.2.1.

5. **Executar os testes de intrusão**: nesta etapa as ferramentas serão instaladas e os testes de intrusão serão efetuados. Cada teste realizado será relatado, com objetivo de registrar as dificuldades e as limitações encontradas durante execução de cada ferramenta. Os resultados desta etapa são apresentados na Seção 4.2.2.

6. **Analisar os resultados**: esta etapa tem como finalidade identificar e descrever o motivo das dificuldades e limitações encontradas na execução das ferramentas. Outros dois pontos serão analisados. O primeiro está relacionado com o panorama tecnológico Web, uma vez que as aplicações Web tiveram uma evolução significativa na forma de construção dos componentes de interface gráfica e na forma de comunicação com o servidor Web. Já o segundo diz respeito a forma de integrar as ferramentas aplicadas nos testes de intrusão, visto que durante a realização dos testes de intrusão é necessário a execução isolada de diversas ferramentas. Os resultados são analisados na Seção 4.2.3.

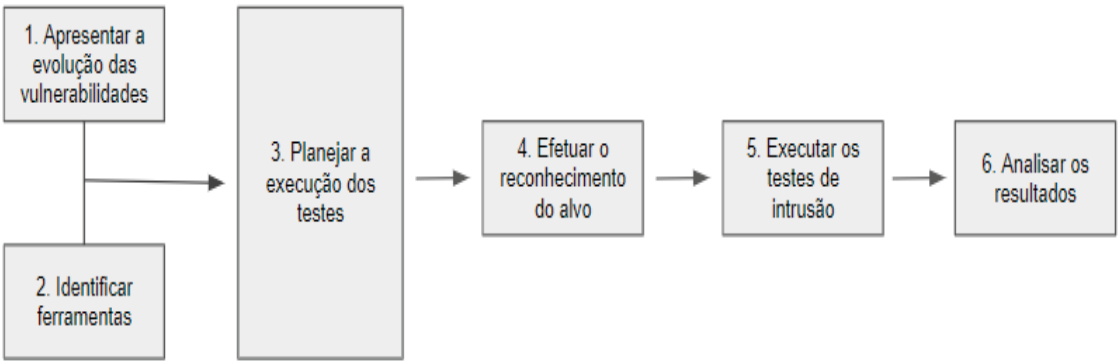



Figura 1.2: Etapas da metodologia do trabalho. Fonte: O autor (2019)

## 1.5 ESTRUTURA DO DOCUMENTO

Além deste capítulo introdutório, este documento contém outros quatro Capítulos:

- **Capítulo 2**: apresenta a revisão bibliográfica, a qual descreve conceitos relevantes para o trabalho proposto.
- **Capítulo 3**: descreve os resultados obtidos nas etapas de estudo da evolução das vulnerabilidades e identificação de ferramentas da metodologia de trabalho.
- **Capítulo 4**: apresenta os resultados obtidos nas etapas de planejamento da execução dos testes de intrusão, reconhecimento do alvo, execução dos testes de intrusão e análise de resultados da metodologia de trabalho.
- **Capítulo 5**: descreve as considerações finais do trabalho.

# 2 REVISÃO BIBLIOGRÁFICA

Este capítulo apresenta os principais conceitos que fundamentam o trabalho. A Seção 2.1 descreve conceitos relacionados à segurança de software, tecnologias Web e testes de software. A Seção 2.2 apresenta as vulnerabilidades mais críticas encontradas em aplicações Web. Por sua vez, a Seção 2.3 descreve conceitos vinculados aos testes de segurança e apresenta as técnicas existentes. Dentre as técnicas existentes, a Seção 2.4 detalhada a técnica testes de intrusão, a qual é o foco deste trabalho.

## 2.1 CONCEITOS

Esta seção apresenta conceitos relacionados à segurança de software, tecnologias Web, testes e demais conceitos utilizados ao longo do trabalho.

- **AJAX (*Asynchronous JavaScript and XML*)**: é uma técnica para criar aplicaçães Web mais interativas com o usuário, utilizando-se de requisições assíncronas. As requisições com servidor Web são implementadas utilizando o objeto JavaScript XMLHttpRequest. O uso deste objeto permite que seja aberta uma conexão HTTP com o servidor Web dentro de uma página HTML que reside no lado cliente. O servidor Web, neste caso, não processa toda a página, somente processa os dados enviados, retornando os dados necessários que são atualizados, de forma dinâmica, no lado cliente através de códigos JavaScript (Stuttard e Pinto, 2011).

- **Ameaça**: é uma circunstância que tem potencial para causar perda ou dano, como um atacante externo, através da exploração de uma vulnerabilidade (OWASP, 2014).

- **Ataque**: é a exploração de uma vulnerabilidade presente no software, violando algumas das propriedades de segurança especificados para o software (Sommerville, 2011).

- **Ativo**: é algo de valor que precisa ser protegido, podendo ser um componente do software ou dados usado por ele. Cada ativo pode estar relacionado a uma ou várias propriedades de segurança (Sommerville, 2011).

- **Autenticação**: é o procedimento para confirmar que algo ou alguém é autêntico, isto é, que as afirmações feitas sobre a coisa ou pessoa são verdadeiras. Autenticar um objeto significa confirmar sua procedência, ao passo que autenticar uma pessoa consiste em verificar sua identidade. Em sistemas de computador, a autenticação é o processo de tentar verificar a identidade digital do remetente de uma comunicação, por exemplo, através do *login* em uma aplicação (OWASP, 2014).

- **Autenticação baseada em formulário**: é uma forma de implementar o mecanismo de autenticação, onde a aplicação Web disponibiliza um formulário para informar usuário e senha (Stuttard e Pinto, 2011).

- **Autorização**: fornece privilégios de acesso concedidos a um usuário, programa ou processo. É permitir acesso a recursos somente àqueles autorizados a usá-los (OWASP, 2014).

- **Casos de uso**: é uma descrição de uma sequência de ações que o sistema executa para produzir um resultado de valor ao ator. O diagrama de casos de uso é utilizado para modelar aspectos dinâmicos do sistema e são aplicados para modelar a visão dos casos de uso do sistema (Booch et al., 1998).

- **Ciclo de vida para desenvolvimento de software seguro**: ciclo de vida de software é uma série de fases que fornecem um modelo para o desenvolvimento de aplicações. É uma estrutura imposta ao desenvolvimento dos artefatos de software. Um ciclo de vida genérico pode ser compostos das seguintes fases: análise de requisitos, projeto, implementação, testes, instalação e manutenção (Felderer et al., 2016). Uma forma de melhorar o ciclo de vida é considerar aspectos de segurança em todas suas fases. Ao integrar segurança em cada fase permite-se uma abordagem holística no tratamento de segurança nas aplicações. Um conceito importante vinculado ao ciclo de vida do software é o risco. A análise de risco é, principalmente, aplicada na fase de projeto e serve para identificar vulnerabilidades e guiar os testes de segurança (Felderer et al., 2016). A Figura 2.1 apresenta o ciclo de vida para desenvolvimento de software seguro.

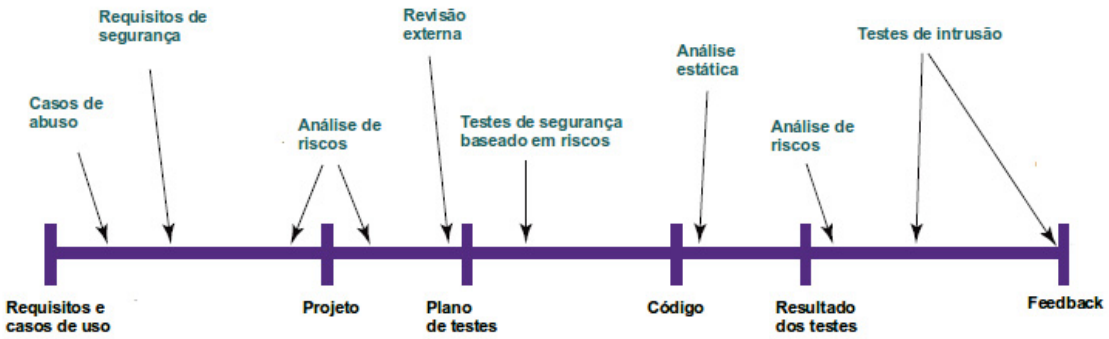



Figura 2.1: Ciclo de vida para desenvolvimento de software seguro. Fonte: Arkin et al. (2005)

- ***Crawler Web***: também conhecido como Web *spider* ou Web *robot*, é um software que navega em *sites* da Internet, de forma automatizada, com o objetivo de encontrar páginas HTML e indexar seu conteúdo, permitindo que sejam realizadas pesquisas nos conteúdos disponíveis nas páginas HTML encontradas (Olston e Najork, 2010).

- **Gerenciamento de identidade**: trata-se das funções para gerenciar usuários e papéis. Através destas funções, é possível conceder autorização para recursos na aplicação. Espera-se que pelo menos dois papéis existam, administradores e usuários regulares (OWASP, 2014).

- **Gerenciamento de sessão**: o protocolo HTTP não mantém estado, assim, é necessário um mecanismo pelo qual a aplicação Web controla e mantém o estado de um usuário que interage com ela. O gerenciamento de sessão é definido como o conjunto de controles que mantém o estado da interação entre um usuário e a aplicação. Neste sentido, é necessário a utilização de um identificador, geralmente, chamado de identificador de sessão (OWASP, 2014).

- **JSON (*JavaScript Object Notation*)**: é um formato compacto para troca de dados entre sistemas. É comumente usado em aplicativos AJAX como uma alternativa ao formato XML usado originalmente para transmissão de dados (Stuttard e Pinto, 2011).

- **Ponto de entrada**: refere-se a uma requisição HTTP feita por um usuário, via navegador, para a aplicação Web, recebendo uma resposta. Cada caso de uso da aplicação pode conter mais de um ponto de entrada (OWASP, 2014).

- ***Proxy* de aplicação Web**: são ferramentas que interceptam a comunicação entre o navegador e o servidor Web, permitindo que o testador visualize e manipule todo e qualquer conteúdo de dados que tramita entre os dois. Estas funcionalidades permitem que testador construa diferentes cenários de testes para encontrar as vulnerabilidades na aplicação Web (SPaCIoS, 2011).

- **Risco**: é a probabilidade da concretização de uma ameaça e suas consequências (Potter e McGraw, 2004). Riscos em aplicações Web são caminhos em potencial que um atacante pode utilizar para prejudicar uma organização, cada risco pode estar associado a uma ou mais vulnerabilidades (OWASP, 2017).

- **Segurança de software**: é um atributo do software que está relacionado com sua habilidade em se proteger de ataques externos, intencionais ou acidentais. Quando se fala em segurança a nível de software ou aplicação estamos diante de um problema de Engenharia de Software. Neste sentido, Engenheiros de Software devem garantir que o software é projetado para resistir a ataques. Por outro lado, segurança na infraestrutura (sistemas operacionais, banco de dados, servidor de aplicação) é um problema de administração/gerenciamento, ficando a cargo dos Administradores de Sistemas realizar a devida configuração na infraestrutura para resistir a ataques (Sommerville, 2011). Segurança tem como objetivo preservar às seguintes propriedades (ISO/IEC, 2018):

  - **Confidencialidade**: garantia que os dados somente serão acessados por pessoas autorizadas. É a garantia que informações não sejam divulgados para pessoas, processos ou dispositivos não autorizados.
  - **Integridade**: garantia que o software e seus dados não estejam danificados. É fornecida quando os dados não são alterados a partir de sua origem, os dados não foram acidentalmente ou maliciosamente modificados, alterados ou destruídos.
  - **Disponibilidade**: sistema deve ser robusto o suficiente para não ficar indisponível. Garante acesso aos dados no momento desejado de forma confiável por usuários autorizados.

- ***Single-Page Application* (SPA)** é uma aplicação Web onde existe apenas uma página HTML. Toda interface gráfica nas quais o usuário interage são implementadas em JavaScript e apresentadas de forma dinâmica na única página da aplicação, sobrescrevendo seu conteúdo atual, sem necessidade de carregar páginas do servidor Web. Portanto, toda a interface gráfica é processada no navegador do cliente, em oposição às aplicações Web tradicionais onde todo processamento da interface gráfica ocorre no lado servidor, neste caso, o lado cliente somente apresenta o HTML retornado pelo servidor. A comunicação com servidor Web ocorre através de AJAX (Fink e Flatow, 2014).

- **Teste**: é uma atividade realizada para avaliar a qualidade do produto, e para melhorá-lo, identificando defeitos e problemas (SWEBOK, 2004).

- **Teste de software**: consiste na verificação dinâmica do comportamento de um software através da execução de um conjunto finito de casos de teste contra o comportamento esperado. A noção dinâmica do teste avalia o software através da observação da sua

execução. O software que está sendo executado chama-se sistema em teste (SUT) (SWEBOK, 2004).

- **Testes estruturais ou caixa branca**: testes são derivados e executados com conhecimento do projeto ou código-fonte do software (Wazlawick, 2013).
- **Testes funcionais ou caixa preta**: testes executados sobre as entradas e saídas do software sem conhecimento do seu código-fonte (Wazlawick, 2013).
- **Vulnerabilidade**: é um tipo especial de falha, resultado de um defeito de software (Arkin et al., 2005). Este defeito pode estar presente no projeto, na implementação ou na operação do software, podendo ser explorado, comprometendo as propriedades de segurança (OWASP, 2014).

## 2.2 VULNERABILIDADES EM APLICAÇÕES WEB

Esta seção apresenta os riscos mais críticos encontradas em aplicações Web de acordo com o OWASP (Open Web Application Security Project), referentes ao ranking do ano de 2017. Cada risco está associado a uma ou várias vulnerabilidades.

OWASP é uma comunidade aberta cujo objetivo é possibilitar que as organizações desenvolvam e mantenham aplicações confiáveis. Dentro dos seus diversos projetos, destaca-se o OWASP Top 10 que tem por finalidade listar os dez riscos mais críticos relacionados com aplicações Web. Os riscos são selecionados e priorizados conforme sua prevalência nas pesquisas realizadas em combinação com estimativas consensuais de explorabilidade, detectabilidade e impacto (OWASP, 2017). A Figura 2.2 apresenta o ranking contendo os dez riscos mais críticos levantados em 2003 e 2017. Os valores A1, A2, A3, A4, A5, A6, A7, A8, A9 e A10 indicam a posição no ranking, onde A1 refere-se ao risco mais crítico e A10 ao menos crítico.

| OWASP Top 10 - 2013 | → | OWASP Top 10 - 2017 |
|---|---|---|
| A1 – Injeção | → | A1:2017-Injeção |
| A2 – Quebra de Autenticação e Gestão de Sessão | → | A2:2017-Quebra de Autenticação |
| A3 – Cross-Site Scripting (XSS) | ↘ | A3:2017-Exposição de Dados Sensíveis |
| A4 – Referência Insegura e Direta a Objetos (IDOR) [Agrupado+A7] | ∪ | A4:2017-Entidades Externas de XML (XXE) [NOVO] |
| A5 – Configurações de Segurança Incorrectas | ↘ | A5:2017-Quebra de Controlo de Acessos [AGRUPADO] |
| A6 – Exposição de Dados Sensíveis | ↗ | A6:2017-Configurações de Segurança Incorrectas |
| A7 – Falta de Função para Conrolo do Nível de Acesso [Agrupado+A4] | ∪ | A7:2017-Cross-Site Scripting (XSS) |
| A8 – Cross-Site Request Forgery (CSRF) | ☒ | A8:2017-Desserialização Insegura [NOVO, Comunidade] |
| A9 – Utilização de Componentes Vulneráveis | → | A9:2017-Utilização de Componentes Vulneráveis |
| A10 – Redirecionamentos e Encaminhamentos Inválidos | ☒ | A10:2017-Registo e Monitorização Insuficiente [NOVO, Comunidade] |

Figura 2.2: OWASP Top 2013 e 2017. Fonte: OWASP (2017)

Segue uma descrição dos 10 riscos de 2017, conforme (OWASP, 2017).

- **A1:2017-Injeção**: falhas de injeção, por exemplo, injeção de SQL, NoSQL, comandos do sistema operacional, LDAP, ORM e EL, ocorrem quando dados não confiáveis são enviados para um interpretador como parte de um comando ou consulta. Os dados enviados podem enganar o interpretador e executar comandos não desejados. Tais comandos podem resultar em perda de dados, corrupção ou divulgação de dados não autorizados e na indisponibilidade do sistema. O atacante envia um simples ataque baseado em texto que explora a sintaxe do interpretador alvo. Segue exemplo de cenário de ataque.

  O código 2.1 utiliza dados não confiáveis na construção de um comando SQL.

  Código 2.1: Exemplo de código Java que constrói comando SQL que permite Injeção de SQL. Fonte: OWASP (2017)

```
1 String query = "SELECT * FROM contas WHERE id\_cliente ='"
2 + request.getParameter("id") + "'";
```

  O atacante pode modificar o parâmetro id e enviar o valor `' or '1' = '1`, conforme exemplo: `http://exemplo.com.br/app/contaView?id=' or '1'='1`.

  Esta alteração faz com que todos o registros da tabela sejam retornados. Ataques mais perigosos podem alterar ou deletar dados e invocar *stored procedures*.

- **A2:2017-Quebra de Autenticação**: as funcionalidades da aplicação relacionadas à autenticação e ao gerenciamento de sessão geralmente são implementadas de forma incorreta, permitindo que os atacantes comprometam senhas, chaves ou tokens de sessão ou explorem outras falhas para assumir temporariamente ou permanentemente identidades de outros usuários. Exemplo de cenário de ataque: a utilização de lista de senhas conhecidas é um ataque comum e caso a aplicação não tenha proteção no preenchimento das credenciais poderá ser usada como um oráculo de senha.

- **A3:2017-Exposição de Dados Sensíveis**: aplicações Web e APIs não protegem de forma adequada dados confidenciais, por exemplo, dados financeiro e de saúde. Atacantes podem obter ou modificar estes dados, cometer fraudes em cartões de crédito ou furtar identidade. Exemplo de cenário de ataque: aplicação criptografa números de cartão de crédito utilizando criptografia a nível de banco de dados, contudo, caso esta informação seja recuperada, o banco de dados irá decriptar os dados, permitindo que uma ataque de Injeção de SQL recupere tais informações livre de criptografia.

- **A4:2017-Entidades Externas de XML (XXE)**: processadores XML configurados de forma incorreta avaliam referências de entidades externas em documentos XML. As entidades externas podem ser usadas para descobrir arquivos, compartilhar arquivos, fazer scanner de porta, executar código e realizar ataques de negação de serviço. Atacantes podem explorar processadores XML vulneráveis caso seja possível fazer *upload* de um documento XML. Exemplo de cenário de ataque: o atacante tentar extrair informações do servidor através do envio do XML apresentado no código 2.2.

Código 2.2: Exemplo de XML contendo definição de entidade externas. Fonte: OWASP (2017)

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

- **A5:2017-Quebra de Controle de Acessos**: às restrições sobre o que cada usuário autenticado pode efetuar na aplicação são geralmente implementadas de forma inadequada, permitindo que atacantes explorarem estas falha para acessar funcionalidades e dados não autorizados, como por exemplo, acessar contas de outros usuários, visualizar arquivos confidenciais, modificar dados de outros usuários e alterar os direitos de acesso. Exemplo de cenário de ataque: aplicação usa dados em um comando SQL sem nenhum tipo de validação para acessar os dados de um conta, conforme código 2.3.

Código 2.3: Exemplo de código Java que permite quebrar o controle de acesso. Fonte: OWASP (2017)

```
//atribui ao valor da variável de índice 1 da consulta SQL
//o valor do parâmetro id da requisição do usuário

pstmt.setString(1, request.getParameter("id"));

// executa a consulta SQL
ResultSet results = pstmt.executeQuery();
```

O atacante simplesmente modifica o parâmetro id colocando o número desejado, como não existe validação, o sistema retorna as informações sobre a conta.

- **A6:2017-Configuração de Segurança Incorretas**: configurações de segurança incorretas é um problema muito encontrado em função da utilização de configurações padrões, configurações feitas em código-fonte, cabeçalhos HTTP mal configurados e mensagens de erro contendo informações confidenciais. Todos os sistemas operacionais, *frameworks*, bibliotecas e aplicativos devem ser configurados para garantir segurança, bem como, devem ser atualizados frequentemente. Atacantes acessam contas default, páginas que não são mais utilizadas, falhas por falta de patches, arquivos e diretórios desprotegidos para obter acesso não autorizado ou conhecimento sobre o sistema. Exemplo de cenário de ataque: o servidor de aplicação contém aplicações exemplos que não são removidas no momento em que o servidor entrar em operação. Tais aplicações podem conter vulnerabilidades que são exploradas para comprometer o servidor.

- **A7:*Cross-Site Scripting* (XSS)**: As falhas XSS ocorrem sempre que uma aplicação Web adiciona dados não confiáveis em uma nova página da Web sem validação ou decodificação adequada, ou atualiza uma página da Web existente com dados fornecidos por um usuário usando uma API do navegador que pode criar HTML ou JavaScript. O XSS permite que os invasores executem scripts no navegador da vítima, que podem sequestrar os identificadores de sessões dos usuários e direcioná-los para sites maliciosos. Existem três forma de XSS, a saber: refletido, armazenado e DOM. No refletido a aplicação adiciona dados, informados pelo atacante, não validados e não codificados, como parte de saída HTML. No armazenado a aplicação persiste dados de um atacante sem validá-los. Em um outro momento um usuário recupera tal informação, a qual pode ser um JavaScript que será executado em seu navegador. Por fim, no DOM, os dados do atacante são executadas através da manipulação do DOM no navegador cliente.

Exemplo de cenário de ataque: a aplicação utiliza dados não confiáveis para construir HTML conforme código 2.4.

Código 2.4: Exemplo de código Java que permite XSS. Fonte: OWASP (2017)

```
//constroi um elemento input HTML de forma dinâmica
//setando o atributo value diretamente com
//o valor do parâmetro CC da requisição do usuário

(String) page += "<input name='creditcard' type='TEXT' value='" +
request.getParameter("CC") + "'>";
```

O atacante modifica o parâmetro CC no navegador, conforme código 2.5.

Código 2.5: Exemplo de código JavaScript que sequestra os cookies do navegador cliente em um ataque XSS. Fonte: OWASP (2017)

```
// fecha o elemento HTML onde foi atribuído o valor do parâmetro CC
// abre um novo script que será executado no momento em que a
// página for apresentada no navegador da vítima
// o script direciona a vítima para uma página hospedada no servidor
// do atacante enviando como parâmetros todos os cookies da vítima

'><script>document.location= 'http://www.attacker.com/cookie?foo='+
document.cookie</script>'
```

Esta alteração permite que o identificador de sessão seja enviada para o site do atacante, permitindo o sequestro de sessão.

- **A8:2017-Desserialização Insegura**: A desserialização insegura geralmente está relacionado à execução de código remoto, contudo, também pode ser utilizada para realizar ataques de injeção e escalar privilégios. Exemplo de cenário de ataque: aplicação usa serialização de objetos para salvar um cookie contendo o identificador do usuário, senha e função, conforme código 2.6.

Código 2.6: Exemplo de dados searializados original. Fonte: OWASP (2017)

```
a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";i:3;s:32:"b6a8b3bea87
fe0e05022f8f3c88bc960";}
```

O atacante altera o objeto serializado para lhe conceder função de administrador, conforme código 2.7.

Código 2.7: Exemplo de dados searializados alterado. Fonte: OWASP (2017)

```
a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";i:3;s:32:"b6a8b3bea87fe
0e05022f8f3c88bc960";}
```

- **A9:2017-Utilização de Componentes Vulneráveis**: componentes, como bibliotecas, *frameworks* e outros módulos de software, são executados com os mesmos privilégios da aplicação, assim, se um componente vulnerável for explorado, o atacante tem boa chance de obter êxito. Aplicativos e APIs que usam componentes com vulnerabilidades conhecidas podem prejudicar a implementação de mecanismos de segurança, permitindo ataques diversos.

- **A10:2017-Registro e Monitorização Insuficiente**: falta de log e monitoramento, juntamente com uma resposta a incidentes ineficaz, permitem que os atacantes, além

de realizarem o ataque propriamente dito, mantenham formas alternativas de acesso. Exemplo de cenário de ataque: atacante usa ferramenta de scanner para verificar todos os usuários que usam a mesma senha, assumindo o controle de todas as contas.

## 2.3 TESTES DE SEGURANÇA

Teste de segurança é um método para avaliar a segurança de um sistema de computador, validando e verificando metodicamente a eficácia dos controles de segurança das aplicações. Um dos objetivos dos testes de segurança é validar se os controles de segurança funcionam conforme o esperado. O funcionamento dos controles de segurança é especificado por meio de requisitos de segurança, os quais descrevem suas funcionalidades. Em alto nível, isso significa comprovar a confidencialidade, a integridade e a disponibilidade dos dados, bem como do serviço. Outro objetivo é validar se os controles de segurança foram implementados com poucas ou nenhuma vulnerabilidade. Testes de segurança em aplicações Web focam somente em avaliar a segurança deste tipo aplicação (OWASP, 2014). As técnicas de testes de segurança são: inspeção e revisão manual, modelagem de ameaças, revisão manual de código-fonte, análise automática de código-fonte e testes de intrusão.

- **Inspeção e revisão manual**: consiste em revisões realizadas por um humano, incluindo a revisão de: decisões de projeto, políticas de código seguro, requisitos de segurança, arquitetura do software, bem como, entrevistar os projetista do software. Embora o conceito de revisão manual seja simples, é uma técnica considerada eficaz, sendo recomendada em organizações com processo de seguranças maduros (OWASP, 2014).

- **Modelagem de ameaças**: consiste em uma análise de riscos da aplicação, auxiliando os projetistas a pensar nas ameaças que a aplicação pode enfrentar. Esta técnica permite ao projetista desenvolver estratégias de mitigação para potenciais vulnerabilidades e auxilia a focar em partes da aplicação que mais necessitam de atenção. Os modelos de ameaças devem ser criados o mais cedo possível durante o desenvolvimento da aplicação, e deve ser revisitado à medida que o desenvolvimento evolui. A base para desenvolver um modelo de ameaça é o padrão NIST 800-30 (OWASP, 2014).

- **Revisão manual de código-fonte**: consiste em manualmente verificar o código-fonte da aplicação em busca de vulnerabilidades de segurança. Muitas vulnerabilidades não podem ser detectadas de forma automática ou com o sistema em execução. Muitos peritos em segurança concordam que ainda não existe um substituto para esta técnica, uma vez que toda informação para identificar problemas de segurança está no código. São exemplos de vulnerabilidades com esta características: problemas de concorrência, falhas em lógica de negócio e problemas de controle de acesso (SPaCIoS, 2011).

- **Análise automática de código-fonte**: consiste em verificar o código-fonte da aplicação através de uma ferramenta que busca problemas de segurança baseado em conjunto de padrões ou regras que indicam possíveis vulnerabilidades (OWASP, 2014).

- **Testes de intrusão**: é um subconjunto do método de teste funcional que pode ser definido como uma tentativa legal e autorizada de explorar um sistema de computador com o propósito de torná-lo mais seguro e protegido. O testador age como um atacante, tentando detectar e explorar vulnerabilidades, podendo ser executado de forma remota, com auxílio de ferramentas ou manualmente (Potter e McGraw, 2004).

A Tabela 2.1 descreve as vantagens e desvantagens de cada técnica de teste de segurança.

Tabela 2.1: Vantagens e desvantagens das técnicas de testes de segurança. Fonte: OWASP (2014)

| **Técnica** | **Vantagem** | **Desvantagem** |
|---|---|---|
| Inspeção e revisão manual | não requer suporte tecnológico, pode ser aplicado a uma variedade de situações, flexível, promove o trabalho em equipe, no início do ciclo de desenvolvimento | alto consumo de tempo, material de suporte nem sempre disponível, requer pensamento humano significativo |
| Modelagem de ameaças | permite a visão da aplicação como um atacante, flexível, no início do ciclo de desenvolvimento | técnica relativamente nova, ter bons modelos de ameaça não significa ter boas aplicações |
| Revisão manual de código-fonte | completude, eficácia e precisão | requer especialistas de segurança altamente qualificados, não é possível detectar erros em tempo de execução facilmente, código-fonte realmente implementado pode diferir do código sendo analisado |
| Análise automática de código-fonte | uso de ferramentas, não requer profissional com conhecimento especializado | reportar vulnerabilidades onde não existe (falso positivo) |
| Testes de intrusão | pode ser rápido e, portanto, barato, requer um conjunto de habilidades relativamente menor que a revisão de código-fonte, testa o código que está sendo realmente exposto | no fim do ciclo de desenvolvimento |

## 2.4 TESTES DE INTRUSÃO

Dentre as técnicas de testes de segurança este trabalho tem foco nos testes de intrusão. Neste sentido, esta seção apresenta uma metodologia para realização de testes de intrusão. O objetivo da metodologia é fornecer uma descrição das etapas que devem ser seguidas, visando a completude e eficiência durante a execução dos testes. São metodologias para testes de intrusão: *Information Systems Security Assessment Framework (ISSAF)*, *The Open Source Security Testing Methodology Manual (OSSTMM)*, *National Institute of Standards and Technology* (NIST) SP 800-115 e *Penetration Testing Execution Standard (PTES)* (Haubris e Pauli, 2013). Neste trabalho será utilizada a metodologia PTES. Esta metodologia foi selecionada pois tem foco exclusivo em testes de intrusão, sendo a mais recente dentre as metodologias existentes. A sigla PTES significa padrão para execução de testes de intrusão.

PTES é um padrão projetado para fornecer as empresas e especialistas em segurança uma linguagem e escopo comuns para a realização de testes de intrusão. Foi desenvolvida por especialistas que atuam em áreas específicas do processo de testes (PTES, 2012). A Figura 2.3 apresenta as etapas desta metodologia.

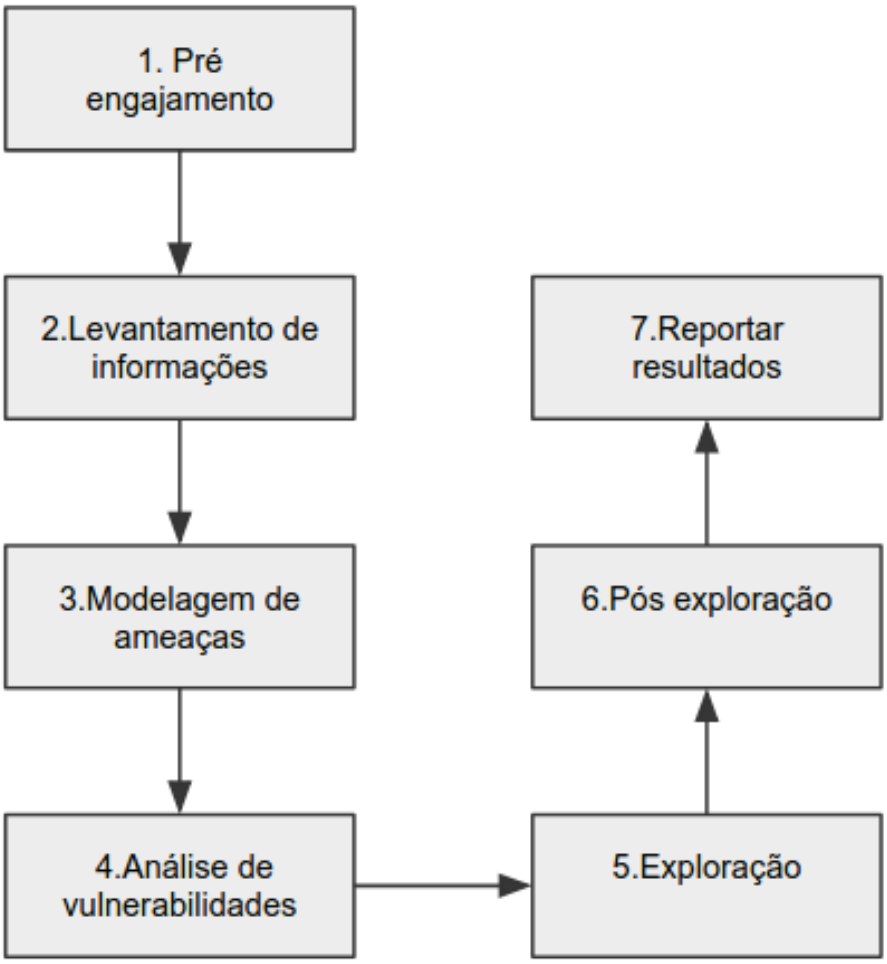



Figura 2.3: Etapas da metodologia PTES. Fonte: O autor baseado em PTES (2012)

Segue uma descrição das sete etapas da metodologia PTES, conforme (PTES, 2012).

1. **Pré-engajamento**: nesta etapa são obtidos os detalhes necessários para executar os testes, sendo definido o escopo e os objetivos dos testes. Quanto mais informações forem coletadas sobre a empresa neste estágio melhor serão os resultados. Os pontos chaves discutidos são quais as partes da empresa são mais valiosas e que geram mais receitas. A identificação destes pontos auxiliará os testadores na definição dos objetivos e dos locais onde será necessário ganhar acesso.

2. **Levantamento de informações**: o objetivo desta etapa é levantar o máximo de informação sobre o alvo, como informações de rede, tipos de sistemas utilizados, informações sobre os empregados e engenharia social. A entrega desta etapa deve incluir uma lista de todos ativos que são alvos, quais aplicações estão associadas aos ativos e os serviços utilizados. Ferramentas automatizadas podem ser utilizadas para auxiliar nesta etapa.

3. **Modelagem de ameaças**: com base nas informações encontradas nas etapas anteriores, é feita uma modelagem de ameaças para descobrir o que gera receita e como os negócios são executados. Deve-se determinar de onde as maiores ameaças podem vir e mapeá-las aos ativos encontrados.

4. **Análise de vulnerabilidades**: é o processo de identificar vulnerabilidades que podem ser exploradas por um atacante. Esta análise pode ser feita de duas formas, a saber: ativa, quando existe interação com o alvo sendo testado, por exemplo, enviando requisições com dados maliciosos e analisando a resposta, e passiva, a qual não interage com o alvo, somente monitora e analisa os dados durante alguma interação com o alvo. Esta etapa pode ser realizada de forma manual ou por ferramentas automatizadas.

5. **Exploração**: encontrar uma vulnerabilidade não significa que um atacante terá acesso ao sistema, assim, a fase de exploração concentra-se unicamente em estabelecer tal acesso, ignorando as restrições de segurança. As ferramentas utilizadas na etapa anterior implementam casos de testes genéricos para identificar vulnerabilidades, ou seja, são testes que podem ser aplicados em qualquer alvo, contudo, nesta etapa é necessário utilizar implementações específicas para o alvo em questão. Existem ferramentas que disponibilizam biblioteca com diversas implementações para explorar vulnerabilidades em produtos conhecidos, e permitem a customização e criação de novas implementações.

6. **Pós exploração**: os objetivos desta etapa são: manter o acesso ao alvo e esconder as evidências da invasão. Para manter o acesso ao alvo, são criados outros pontos de acesso, por exemplo, *backdoors* e novas contas de administrador, uma vez que medidas de segurança podem ser aplicadas ao alvo corrigindo suas vulnerabilidades, evitando assim que uma nova invasão aconteça.

7. **Reportar resultados**: esta etapa tem como objetivo confeccionar o relatório final, contendo um sumário executivo e um seção para reportar as questões técnicas.

### 2.4.1 Testes de intrusão em aplicações Web

Além das etapas apresentadas pela metodologia PTES, testes de intrusão em aplicações Web possuem atividades específicas que devem ser executadas, conforme apresentado no Guia de Testes do OWASP (OWASP, 2014).

O Guia de Testes do OWASP contém um *framework* e um guia de testes de intrusão em aplicações Web, descrevendo atividades e técnicas para testar os problemas de segurança mais comuns encontrados em aplicações Web. Os testes são baseados no método de teste funcional, sendo dividido em duas fases, a saber: modo passivo e modo ativo. No modo passivo, o testador deve entender a aplicação, observando as solicitações e respostas HTTP, enquanto que no modo ativo, o testador interage com a aplicação, enviando solicitações HTTP com dados maliciosos e analisando as respostas (OWASP, 2014). O Guia de Testes do OWASP é composto por 91 atividades, sendo dividido em 11 categorias, a saber: reconhecimento, gerenciamento de configuração, gerenciamento de identidade, autenticação, autorização, gerenciamento de sessão, validação de entrada de dados, erros de aplicação, lógica de negócio, lado cliente e criptografia. Segue uma descrição de cada categoria.

1. **Reconhecimento**: nesta categoria encontram-se as atividades que visam entender as funcionalidades, identificar os pontos de entradas, compreender as tecnologias aplicadas e a arquitetura da aplicação Web. No final destas atividades, o testador deve entender todos os pontos de acesso, cabeçalhos HTTP, parâmetros e *cookies*. Uma das atividades mais importantes na fase de reconhecimento é a varredura da aplicação Web que será explicada na Seção 2.4.1.1.

2. **Gerenciamento de configuração**: são atividades relacionadas a compreender e revisar as configurações dos servidores que compõem a arquitetura da aplicação. As plataformas

de aplicação são amplas e variadas, erros em sua configuração podem comprometer a aplicação da mesma forma que uma aplicação não segura pode comprometer o servidor.

3. **Gerenciamento de identidade**: as atividades desta categoria têm como objetivo testar se os papéis foram corretamente implementados na aplicação, bem como, os processos de criação de contas de usuários.

4. **Autenticação**: testar o esquema de autenticação significa entender como o processo de autenticação funciona e usar essas informações para tentar contornar o mecanismo de autenticação, obtendo acesso não autorizado.

5. **Autorização**: testar autorização significa entender como a autorização funciona e usar estas informações para contornar este mecanismo. A autorização é um processo que vem após uma autenticação bem-sucedida, portanto, o testador verificará esse ponto depois de ter credenciais válidas, associadas a um conjunto bem definido de funções e privilégios. Durante esse tipo de avaliação, deve-se verificar se é possível ignorar o esquema de autorização, encontrar maneiras de escalonar os privilégios atribuídos ao testador.

6. **Gerenciamento de sessão**: os objetivos dos testes desta categoria são garantir que o identificador de sessão não seja obtido por atacantes, validar se os cookies e tokens foram implementados de forma correta e se as informações de sessão são enviadas ao cliente por um canal seguro.

7. **Validação de entrada de dados**: a correta validação dos dados informados pelo usuário é o problema mais encontrado em aplicações Web. Os dados enviados pelo cliente nunca devem ser considerados confiáveis, uma vez que podem ser alterados por atacantes. Esta atividade tem como objetivo testar todas as formas possíveis de entrada para entender se a aplicação Web válida de forma correta os dados de entrada antes de usá-los.

8. **Erros de aplicação**: os códigos e mensagens retornados pela aplicação revelam informações sobre ela, como banco de dados e tecnologias utilizadas, esta categoria de testes visa validar se os códigos e mensagens retornados pela aplicação contém informações que podem ser utilizadas por atacantes.

9. **Lógica de negócio**: testes de lógica de negócios são semelhantes aos tipos de testes usados por testadores funcionais que se concentram em testes de estado finito. Esses tipos de testes exigem que os profissionais de segurança pensem um pouco diferente, desenvolva casos de abuso e uses técnicas de teste adotadas pelos testadores funcionais. As vulnerabilidades desta categoria não podem ser detectadas por um *Scanner* de aplicação Web, dependendo das habilidades e da criatividade do testador, uma vez que trata-se de questão específica da aplicação.

10. **Lado cliente**: testes do lado do cliente se preocupam com a execução de códigos na máquina cliente, normalmente de forma nativa ou via plugin no navegador. A execução do código no lado do cliente é diferente da execução no lado servidor. Em linhas gerais, os testes desta categoria verificam se é possível injetar código e checar se a aplicação permite manipular recursos do lado cliente.

11. **Criptografia**: dados confidenciais, como senha e número de cartão de crédito, devem ser transmitidos por um canal seguro. HTTP é um protocolo de texto não criptografado

e normalmente é protegido por Túnel SSL / TLS, resultando em tráfego HTTPS. Este protocolo garante não apenas confidencialidade, mas também autenticação. Servidores são autenticados usando certificados digitais. O objetivo dos testes é garantir que as configurações dos servidores estejam corretas e que dados confidenciais estejam trafegando por canal seguro.

A listagem completa das 91 atividades é descrita no Apêndice A. Para cada atividade é descrito seu objetivo e algumas ferramentas que podem ser utilizadas.

#### *2.4.1.1 Varredura de aplicações Web*

Varredura é usada nos testes de intrusão para levantar informações sobre a aplicação Web e para identificar vulnerabilidades em cada uma das páginas da aplicação. Para tanto, usa-se uma ferramenta chamada *scanner* de aplicação Web. Dentre os tipos de ferramentas existentes para execução de testes de intrusão, os *scanners* detém o maior número de ferramentas disponíveis, conforme apresentado em 3.2.

O *scanner* de aplicação Web é uma ferramenta composta por três módulos, a saber: *crawler*, teste e análise. Segue uma descrição da função de cada módulo, de acordo com Doup et al. (2010).

1. ***Crawler***: este módulo recebe um conjunto inicial de URL. A partir deste conjunto, recupera as respectivas páginas e segue os links encontrados para descobrir novas páginas. Adicionalmente, identifica todos o pontos de entrada da aplicação, como os parâmetros de requisições *GET* e *POST*.

2. **Teste**: este módulo recebe os pontos de entradas identificadas pelo *crawler*, e para cada ponto de entrada e vulnerabilidade que ferramenta suporta, gera valores que podem encontrar a vulnerabilidade. Por exemplo, injetar em cada campo de entrada código JavaScript para testar *Cross Site Scripting* ou partes de comandos SQL para testar por Injeção de SQL. Os dados de testes são, usualmente, gerados usando heurísticas ou valores pré-definidos.

3. **Análise**: este módulo recebe como entrada a resposta da requisição de ataque e detecta a vulnerabilidade. Por exemplo, caso o retorno de um ataque de Injeção de SQL contenha uma mensagem de erro do banco de dados, o módulo de análise pode inferir a existência da vulnerabilidade Injeção de SQL.

## CONSIDERAÇÕES DO CAPÍTULO

As propriedades de segurança são exploradas pelos atacantes por meio de vulnerabilidades existentes em aplicações Web. A Seção 2.2 apresentou as vulnerabilidades mais críticas que podem ser exploradas por atacantes neste tipo de aplicação. Uma forma de tornar as aplicações Web mais seguras é através da execução de testes de segurança. Existem diversas técnicas de testes de segurança que podem ser aplicadas. Dentre as técnicas, este trabalho foca nos testes de intrusão, os quais têm como finalidade detectar vulnerabilidades e realizar ataques visando comprovar que as vulnerabilidades identificadas são reais. A execução de testes dos intrusão deve ser guiada por uma metodologia. Neste sentido, foram apresentadas a metodologia PTES, a qual fornece um guia geral para execução de testes de intrusão, e a metodologia OWASP, a qual descreve atividades específicas para execução de testes de intrusão em aplicações Web. Os temas relacionados aos testes de segurança foram abordados na Seção 2.3 e 2.4.

Os temas apresentados neste capítulo têm o propósito de fundamentar o trabalho, o qual será apresentada nos próximos capítulos e trata do levantamento e exploração de ferramentas de código aberto para testes de intrusão em aplicações Web.

# 3 ANÁLISE DE VULNERABILIDADES E FERRAMENTAS

Este capítulo apresenta os resultados obtidos nas seguintes etapas da metodologia de trabalho: estudo da evolução das vulnerabilidades e identificação de ferramentas para execução de testes de intrusão em aplicações Web. A Secão 3.1 descreve os maiores problemas de segurança enfrentado pelas aplicações Web, enquanto que a Seção 3.2 apresenta uma lista de ferramentas de código aberto para execução de testes de intrusão neste tipo de aplicação, com foco nas etapas de análise e exploração de vulnerabilidades da metodologia PTES.

## 3.1 EVOLUÇÃO DAS VULNERABILIDADES EM APLICAÇÕES WEB

O projeto OWASP Top 10 apresenta um ranking contendo os 10 riscos mais críticos em aplicações Web desde do ano de 2003. A Tabela 3.1 apresenta os 10 riscos mais críticos em aplicações de todas as publicações, compreendendo os seguintes anos: 2003, 2004, 2007, 2010, 2013 e 2017. A tabela está em ordem alfabética pela coluna risco. O valor "-" significa que o risco não foi considerado naquele ano. Os valores A1, A2, A3, A4, A5, A6, A7, A8, A9 e A10 indicam a posição no ranking, onde A1 refere-se ao risco mais crítico e A10 ao menos crítico. Em função da unificação de riscos de anos anteriores, o risco *Broken Access Control* possui mais de uma posição em alguns anos.

Dentre os 19 riscos listados na Tabela 3.1, seis são riscos associados a problemas de validação de entrada de dados, a saber: *Injection*, *Cross Site Scripting (XSS)*, *Unvalidated Input*, *Buffer Overflows*, *XML External Entities (XEE)* e *Unvalidated Redirects and Forwards*. Dentre estes 6, os riscos *Injection*, *Cross Site Scripting (XSS)* e *XML External Entities (XEE)* estão presentes no Top 10 de 2017.

Tabela 3.1: OWASP Top 10 entre os anos de 2003 e 2017. Fonte: O autor (2019)

| **Risco** | **2003** | **2004** | **2007** | **2010** | **2013** | **2017** |
|---|---|---|---|---|---|---|
| *Broken Access Control* | A2 | A2 | A4 e A10 | A4 e A8 | A4 e A7 | A5 |
| *Broken Authentication* | A3 | A3 | A7 | A3 | A2 | A2 |
| *Buffer Overflows* | A5 | A5 | - | - | - | - |
| *Cross Site Request Forgery* (CSRF) | - | - | A5 | A5 | A8 | - |
| *Cross Site Scripting (XSS)* | A4 | A4 | A1 | A2 | A3 | A7 |
| *Denial of Service* | - | A9 | - | - | - | - |
| *Information Leakage\Error Handler* | A7 | A7 | A6 | - | - | - |
| *Injection* | A6 | A6 | A2 | A1 | A1 | A1 |
| *Insecure Communications* | - | - | A9 | A9 | - | - |
| *Insecure Deserialization* | - | - | - | - | - | A8 |
| *Insufficient Logging & Monitoring* | - | - | - | - | - | A10 |
| *Malicious File Execution* | - | - | A3 | - | - | - |
| *Remote Administration Flaws* | A9 | - | - | - | - | - |
| *Security Misconfiguration* | A10 | A10 | - | A6 | A5 | A6 |
| *Sensitive Data Exposure* | A8 | A8 | A8 | A7 | A6 | A3 |
| *Unvalidated Input* | A1 | A1 | - | - | - | - |
| *Unvalidated Redirects and Forwards* | - | - | - | A10 | A10 | - |
| *Using Known Vulnerable Components* | - | - | - | - | A9 | A9 |
| *XML External Entities (XEE)* | - | - | - | - | - | A4 |

Os 10 riscos mais críticos das publicações de 2003 e 2004 foram baseados unicamente no conhecimento dos especialistas que participaram da sua elaboração, uma vez que não existiam fontes estatísticas confiáveis sobre violações de segurança em aplicações Web. Entre estas duas publicações houve pouca mudança. *Denial of Service* foi adicionada enquanto *Remote Administration Flaws* foi unificada com *Broken Access Control*. A inclusão de *Denial of Service* se baseou na grande quantidade de organizações suscetíveis a este tipo de ataque e em seu impacto quando se obtém sucesso (OWASP, 2004).

A publicação de 2007 teve sua metodologia reformulada, sendo baseado no relatório de estatísticas das vulnerabilidades mais exploradas fornecido pelo MITRE (OWASP, 2007). De acordo com o relatório, o número total de vulnerabilidades em aplicações Web teve um aumento significativo, ao ponto de ultrapassar *Buffer Overflow*. Este fato se deve à facilidade de exploração de vulnerabilidades em aplicações Web, combinada com a proliferação de aplicações de baixa qualidade, construída por desenvolvedores inexperientes (Steve Christey, 2007). com relação à publicação de 2004, houveram diversas alterações. Foram removidos *Unvalidated Input*, *Buffer Overflow*, *Denial of Service* e *Insecure Configuration Management*. A primeira foi removida porque era uma superclasse de várias vulnerabilidades já presentes na lista, por exemplo, *Injection* e *Cross Site Scripting (XSS)*. A segunda foi eliminada uma vez que ocorre com mais frequências em programas que usam linguagem C/C++, sendo baixo o registro deste tipo de vulnerabilidade em aplicações Web. A terceira e quarta foram excluídas em função de sua baixa posição no ranking do relatório. Foram adicionadas *Insecure Remote File Include*, *Cross Site Request Forgery* e *Insecure Communications*. A primeira foi acrescentada em função do número elevado de aplicações Web escritas em PHP com este tipo de vulnerabilidade e a segunda devido sua posição no ranking do relatório. A terceira não foi inserida com base no ranking, e sim por representarem a causa raiz de muitos vazamentos de informações (OWASP, 2007).

Em 2010, a metodologia para estimar o risco teve alterações, afetando a ordem de alguns riscos no ranking. As vulnerabilidades *Security Misconfiguration* e *Unvalidated Redirects and Forwards* foram adicionadas enquanto que *Malicious File Execution Information Leakage and Improper Error Handling* foram removidas. *Security Misconfiguration* foi retirada de 2007 pois não era considerada um problema de software, contudo, do ponto de vista de risco organizacional e prevalência foi incorporada novamente, já *Unvalidated Redirects and Forwards* foi adicionada uma vez que a evidência mostra que esta questão, relativamente desconhecida, é generalizada e pode causar danos significativos. *Malicious File Execution* foi eliminada porque sua presença em 2007 foi motivada por um grande número de aplicações PHP com esta vulnerabilidade, entretanto, PHP foi melhorado no sentido de disponibilizar uma configuração mais segura por padrão, o que diminuiu a ocorrência desta questão. Tendo em vista o baixo impacto ocasionado pela divulgação de mensagens de erros, *Information Leakage and Improper Error Handling* foi removida (OWASP, 2010).

A publicação de 2013 uniu o risco *Insecure Communication* com *Sensitive Data Exposure*. O risco *Using Known Vulnerable Components*, antes mencionado como parte de *Security Misconfiguration*, ganhou categoria própria, pois o crescimento do desenvolvimento baseado em componentes aumentou significativamente o risco de se utilizar componentes vulneráveis. Em função das estatísticas, *Broken Authentication* subiu uma posição no ranking enquanto que *Cross-Site Request Forgery* perdeu três. Outra questão que contribuiu para queda deste último, presente no Top 10 por seis anos, foi que organizações e *frameworks* de desenvolvimento focaram em implementar mecanismos de proteção (OWASP, 2013).

Com relação à publicação de 2017, além de utilizar dados estáticos, consultou especialistas de diversas áreas. Baseada nas estatísticas, foi adicionado o risco *XML External Entities (XEE). Cross-Site Request Forgery (CSRF)* e *Unvalidated Redirects and Forwards* foram removidas. Pelos especialistas, *Insecure Deserialization* e *Insufficient Logging & Monitoring* ganharam presença no ranking (OWASP, 2017).

A Tabela 3.2 apresenta casos reais de ataques e os relacionam com os riscos do OWASP Top 10 2017.

Tabela 3.2: Casos reais de ataques em aplicações Web. Fonte: O autor (2019)

| **Ano** | **Ataque** | **Top 10 2017** |
|---|---|---|
| 2010 | A marinha Inglesa teve seu site violado pela exploração da vulnerabilidade *SQL Injection*. O atacante publicou as credenciais dos administradores do site (BBC, 2010). | A1 |
| 2011 | O site da Sony Picture foi atacado pela exploração da vulnerabilidade *SQL Injection*, expondo informações pessoais e senhas de um milhão (1.000.000) de usuários, três milhões e meio (3.500.000) de cupons digitais e setenta e cinco mil (75.000) códigos de músicas. As senhas estavam armazenadas sem criptografia (MACNN, 2011). | A1 |
| 2012 | Um grupo de atacantes postou em seu site quatrocentos e cinquenta mil (450.000) credenciais de usuário do Yahoo!. As credenciais estavam armazenadas sem criptografia e a técnica utilizada foi *union based SQL injection* (CBS, 2012). | A1 |
| 2013 | O site de administração do governo de Istambul foi violado pela exploração da vulnerabilidade *SQL Injection*, sendo eliminados diversos registros referentes a contas de gás, Internet, água, eletricidade e telefone (Softpedia, 2013). | A1 |

**Tabela 3.2 continuação da página anterior**

| Ano | Ataque | Top 10 2017 |
|---|---|---|
| 2014 | Atacantes russos roubaram um bilhão e duzentos milhões (1.200.000.000) credenciais de usuários e mais de quinhentos milhões (500.000.000) de endereços de e-mail. Estas informações foram obtidas de quatrocentos e vinte mil (420.000) aplicações Web (Times, 2014). | A1 |
| 2015 | A empresa de telecomunicações Britânica chamada TalkTalks's foi atacada através da exploração da vulnerabilidade *SQL Injection*. Os atacantes acessaram informações pessoais e bancárias de aproximadamente cento e cinquenta mil (150.000) clientes (ICO., 2016). | A1 |
| 2016 | Atacantes roubaram informações de salários e impostos de mais de quatrocentos e trinta mil (430.000) empregados da empresa de cartão de crédito Equifax. Para obter as informações bastava ao atacante acessar o portal do empregado e informar, para fins de autenticação, os últimos quatros dígitos do cartão social e o ano de nascimento (KrebsonSecurity, 2016). | A2 |
| 2016 | O site da empresa German Airline vem sendo atacado desde 2011. Mais de um milhão e meio (1.500.000) de informações de passageiros estavam disponíveis para consulta. Ao comprar uma passagem era enviado um email ao cliente contendo uma URL para página de informações da compra, bastava ao atacante modificar os parâmetros desta URL para acessar as informações de outros clientes (RT, 2016). | A5 |
| 2016 | A rede social para adolescentes i-Dressup teve exposto cinco milhões e meio (5.500.000) de contas de usuários pela exploração da vulnerabilidade *SQL Injection*. As credenciais estavam armazenadas sem criptografia (ARSTechnica, 2016). | A1 |
| 2016 | Mossack Fonseca, empresa do Panamá que vende offshores, teve roubado cerca de onze milhões (11.000.000) de documentos. O atacante explorou uma vulnerabilidade presente em uma versão antiga do Drupal (Obermaier, 2016). | A6 |
| 2016 | A falta de configuração de segurança em um banco de dados armazenado na Amazon expôs informações de noventa e três milhões (93.000.000) de eleitores mexicanos (MacKeeper, 2016). | A6 |
| 2017 | Seis milhões (6.000.000) de registros de clientes foram obtidos da empresa Verizon. O motivo foi falta de configuração de segurança que permitiu que qualquer pessoa que soubesse o endereço do servidor fizesse download de arquivos sem qualquer restrição de acesso (Theverge, 2017). | A6 |
| 2017 | A falta de configuração de segurança em um banco de dados armazenado na Amazon expôs exposto os dados de quatro milhões (4.000.000) de clientes da empresa Dow Jones & Co. (SCmagazine, 2017). | A6 |
| 2017 | Atacantes roubaram dados financeiros de uma grande empresa cartão de crédito dos EUA, a saber: Equifax. Os dados roubados afetam cento e quarenta e três milhões (143.000.000) de americanos. O ataque explorou uma vulnerabilidade no *framework* de desenvolvimento Struts (CNET, 2017). | A9 |
| 2018 | A empresa de comunicação espanhola Telefónica teve exposta milhões de registros de clientes. Para acessar os registros o atacante precisava estar logado e alterar informações na URL da aplicação (Español, 2018). | A5 |

## 3.2 FERRAMENTAS PARA TESTES DE INTRUSÃO EM APLICAÇÕES WEB

Neste trabalho, foi realizado um levantamento de ferramentas de código aberto para execução de testes de intrusão em aplicações Web com foco nas etapas de análise e exploração de vulnerabilidades da metodologia PTES. As ferramentas comerciais não entraram no escopo desta atividade, pois as versões gratuitas, quando disponíveis, possuem limitações em suas funcionalidades, como encontrado nas ferramentas Acutinex [1] e Burp Suit [2]. Outras ferramentas comerciais podem ser encontradas no site do OWASP [3]. Além do levantamento, foram realizados um estudo para determinar em qual etapa da metodologia PTES cada ferramenta se aplica e um mapeamento das vulnerabilidades que algumas das ferramentas são capazes de detectar.

As ferramentas foram identificadas através de pesquisas em artigos, livros e sites da Internet. Todas as referências utilizadas durante a pesquisa são encontradas no Apêndice B.

Como resultado do levantamento foram encontradas 104 ferramentas de código aberto para execução de testes de intrusão em aplicações Web. A listagem completa das ferramentas é descrita no Apêndice B.

Dentre o conjunto de ferramentas identificadas, 62.5% são aplicadas na etapa de análise de vulnerabilidade, 27.9% são aplicadas nas etapas de análise e exploração de vulnerabilidades, 5.8% são aplicadas na etapa de exploração e 3.8% são *framework* compostos por diversas ferramentas, conforme apresentado na Figura 3.1.

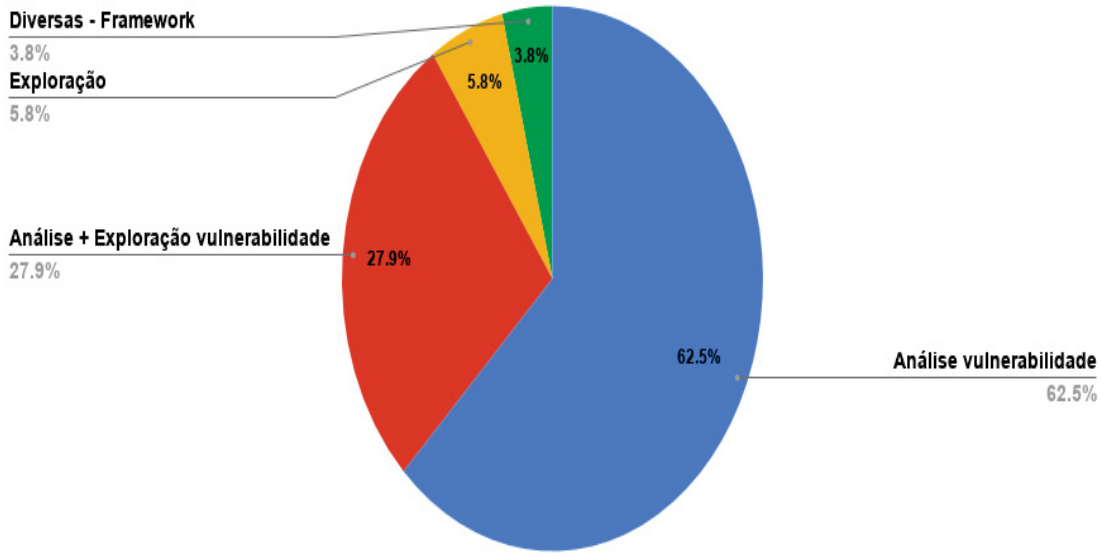



Figura 3.1: Percentual de ferramentas por etapa da metodologia geral para realização de testes de intrusão. Fonte: O autor (2019)

A Figura 3.2 apresenta a quantidade de ferramentas por vulnerabilidade. A vulnerabilidade que possui maior número de ferramentas é Injeção de SQL com 28. Em segundo lugar, com 25, encontram-se as ferramentas que identificam e/ou exploram diversas. A vulnerabilidade que detém o terceiro maior quantidade de ferramentas é *Cross Site Scripting (XSS)* com 14. Diversas significa que a ferramenta detecta mais de 3 vulnerabilidades.

[1] https://www.acunetix.com
[2] https://portswigger.net
[3] https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools

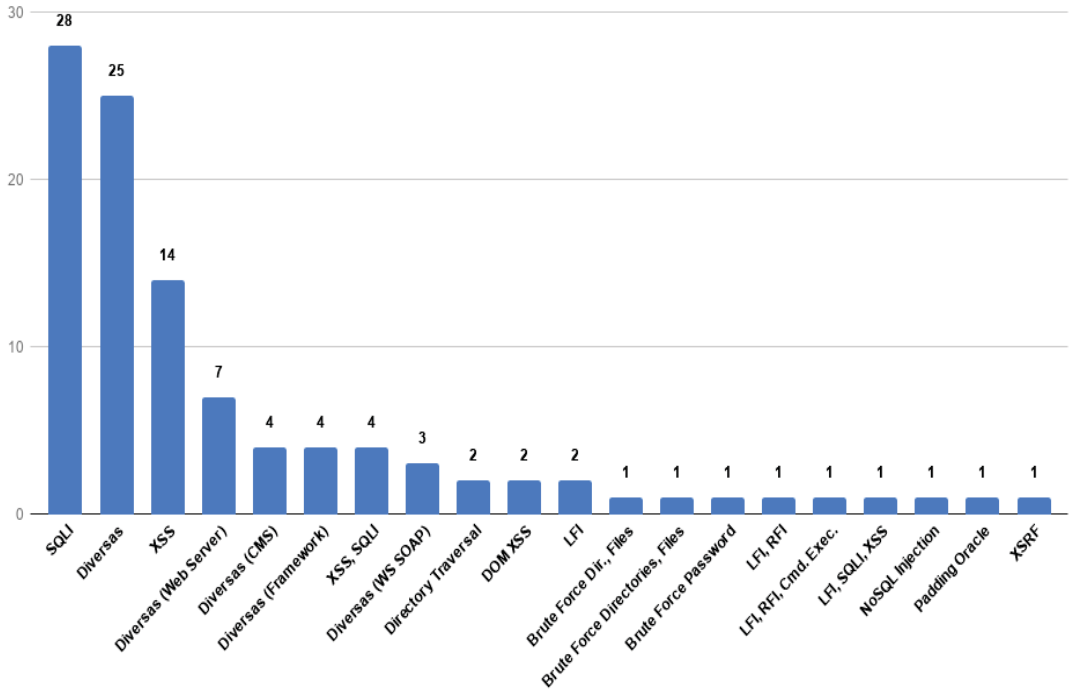



Figura 3.2: Quantidade de ferramentas por vulnerabilidade. Fonte: O autor (2019)

Com relação às ferramentas que identificam diversas vulnerabilidades, foram selecionadas sete com o objetivo de mapear quais são as principais vulnerabilidades suportadas por cada uma delas. As ferramentas selecionadas foram: Arachni [4] 1.5.1-0.5.12, IronWASP [5] 0.9.8.6, Skipfish [6] 2.10b, Vega [7] 1.0, W3af [8] 2018.8.22, Wapiti [9] 3.0.1 e ZAP [10] 2.7.0. A seleção teve como base a avaliação realizada por WAVSEP (2014). Esta avaliação apresenta um ranking com as ferramentas com maior taxa de detecção de um conjunto predefinido de vulnerabilidades, sendo que as ferramentas selecionadas estão entre as 10 maiores taxas de detecção. A Tabela 3.3 apresenta o mapeamento entre as vulnerabilidades, os riscos OWASP Top 10 2017 e as ferramentas selecionadas. A primeira coluna da tabela contém o nome da vulnerabilidade, a segunda o risco OWASP Top 10 2017 associado a vulnerabilidade e, a partir da terceira coluna em diante, o nome de cada ferramenta. A tabela está em ordem alfabética pela coluna vulnerabilidade. O valor "X" significa que a ferramenta pode identificar a vulnerabilidade.

Tabela 3.3: Mapeamento entre vulnerabilidades, OWASP TOP 2017 e ferramentas. Fonte: O autor (2019)

| **Vulnerabilidade** | **Top 10** | **Arach.** | **W3af** | **ZAP** | **Wap.** | **Skip.** | **Iron.** | **Vega** |
|---|---|---|---|---|---|---|---|---|
| *.htaccess misconfig.* | A6 | X | X | | | | | |
| *Allow. HTTP methods* | A6 | X | | | | | | |
| *Application Errors* | A6 | | | X | | | | X |
| *Auth. Bypass* | A2 | | | | | | | |

[4]http://www.arachni-scanner.com
[5]https://ironwasp.org
[6]https://code.google.com/archive/p/skipfish
[7]https://subgraph.com/vega
[8]http://w3af.org
[9]http://wapiti.sourceforge.net/
[10]https://www.owasp.org/index.php/ZAP

**Tabela 3.3 continuação da página anterior**

| Vulnerabilidade | Top 10 | Arach. | W3af | ZAP | Wap. | Skip. | Iron. | Vega |
|---|---|---|---|---|---|---|---|---|
| *Backup dir.* | A6 | X | | | | | | |
| *Backup File Disclose* | A6 | X | | X | X | | | |
| *Bruteforce Auth.* | A2 | | X | | | | X | |
| *Buffer Overflow* | A1 | | X | X | | | X | X |
| *Clickjacking* | A6 | | X | | | | X | |
| *Comand Injection* | A1 | X | X | X | | X | X | X |
| *Common adm. interf.* | A6 | X | | | | | | |
| *Common directories* | A6 | X | | | | | | |
| *Common files* | A6 | X | | | | | | |
| *CRLF Injection* | A1 | | | X | X | | X | |
| *CORS* | A6 | X | X | | | | | |
| *Cross Site Flashing* | A1 | | | | | | X | |
| *CSRF* | A2 | X | X | X | | X | | |
| *XSS Reflected* | A7 | X | X | X | X | X | X | X |
| *XSS Stored* | A7 | X | X | X | X | X | X | X |
| *Cross Site Tracing* | A6 | | X | | | | | |
| *DOM XSS* | A7 | X | | | | | X | |
| *DoS - Server Wide* | A3 | | | | | | X | |
| *Eval Code Injection* | A1 | | X | | X | | | X |
| *EL Injection* | A1 | | | X | | | X | |
| *Forceful dir. listing* | A1 | X | X | | X | | | |
| *Format String* | A1 | | X | X | | X | | X |
| *HTTP PUT* | A3 | X | | | | X | | |
| *HTTP TRACE detec.* | A6 | X | | | | | | X |
| *HttpOnly cookies* | A6 | X | | X | | | | |
| *HTTPS over an insec.* | A6 | | X | X | | X | | |
| *Insecure cookies* | A2 | X | X | | | | | |
| *Insuf. TLP for pwd* | A6 | X | | X | | | | |
| *Integer Overflow* | A1 | | | X | | X | | X |
| *LDAP Injection* | A1 | X | X | X | | | X | |
| *Local File Inclusion* | A1 | X | X | | X | | X | X |
| *Missing STS headers* | A6 | X | | X | | | | |
| *MX Injection* | A1 | | X | | | | X | |
| *NoSQL Injection* | A1 | X | | | | | | |
| *Padding Oracle* | A3 | | | X | | | | |
| *Parameter Pollution* | A1 | | | X | | | | |
| *Path/Dir. traversal* | A5 | X | | X | | X | | X |
| *ReDoS* | | | X | | | | | |
| *Remote File Inclusion* | A1 | X | X | X | X | | X | X |
| *Response splitting* | A1 | X | X | | X | | | |
| *Server Side Include* | A1 | | X | X | | | X | |
| *SSRF* | A2 | | | | X | | X | |
| *Session Fixation* | A2 | | | X | X | | X | |
| *Source code discl.* | A3 | X | | | X | | X | X |
| *SQL Injection* | A1 | X | X | X | X | X | X | X |

**Tabela 3.3 continuação da página anterior**

| Vulnerabilidade | Top 10 | Arach. | W3af | ZAP | Wap. | Skip. | Iron. | Vega |
|---|---|---|---|---|---|---|---|---|
| *Unval. DOM redirec.* | A1 | X | | | | | | |
| *Unval. Red. & Forw.* | A1 | X | X | X | | | | |
| *Up. Unex. File Types* | | | X | | | | X | |
| *URL Session ID Rewr.* | A2 | | X | X | | | | |
| *Verb Tampering* | A1 | | | X | | | X | |
| *WebDAV* | A6 | X | X | | | | | |
| *XML Injection* | A1 | X | | X | X | X | X | X |
| *XPath Injection* | A1 | X | X | X | X | X | X | X |
| **Total** | | **33** | **28** | **28** | **15** | **12** | **24** | **16** |

Conforme apresentado na Tabela 3.3, a ferramenta Arachni pode ser utilizada para identificar 33 vulnerabilidades, seguida das ferramentas ZAP e W3af com 28, IronWASP com 24, Vega com 16, Wapiti com 15 e Skipfish com 12. A Figura 3.3 apresenta a quantidade de vulnerabilidades que cada ferramenta identifica em relação ao OWASP Top 10 2017.

Desprende-se dos dados apresentados que as ferramentas podem complementar-se, aumentando a cobertura dos testes em relação ao número de vulnerabilidades identificadas.

Dentre as ferramentas analisadas nenhuma identifica vulnerabilidades associadas aos riscos A8, A9 e A10.

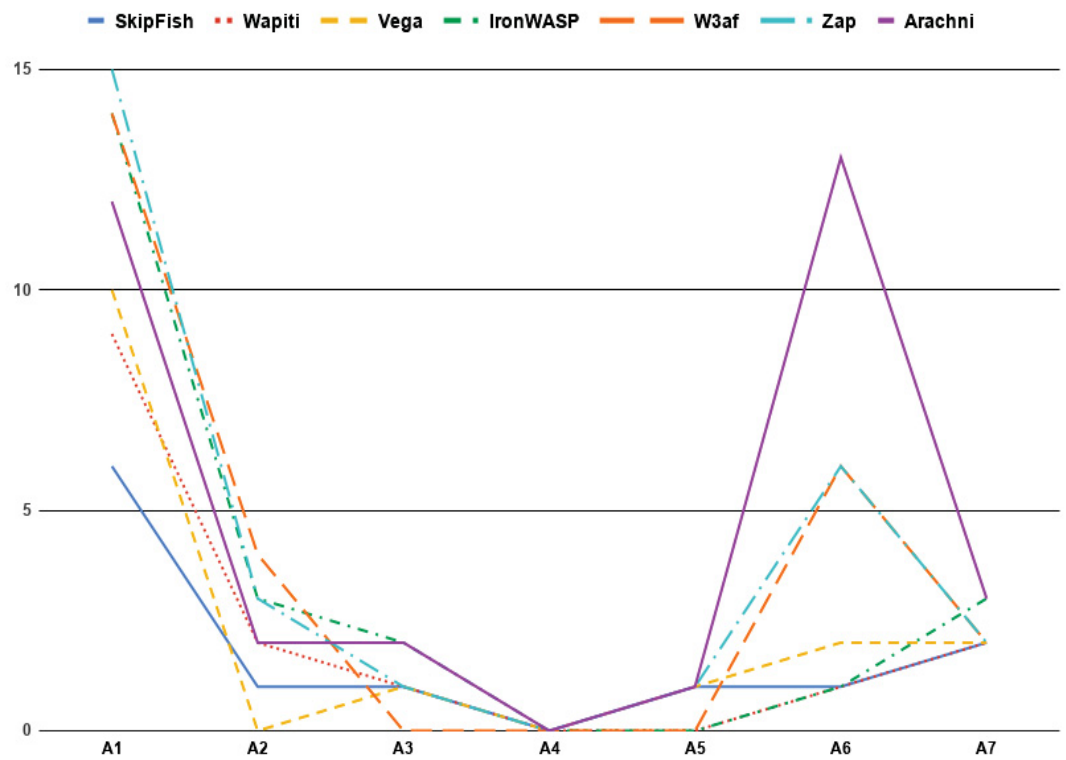



Figura 3.3: Quantidade de vulnerabilidades por ferramenta em relação ao OWASP Top 10 2017. Fonte: O autor (2019)

# 4 TESTES E RESULTADOS

Este capítulo apresenta os resultados obtidos nas seguintes etapas da metodologia de trabalho: planejamento da execução dos testes de intrusão, reconhecimento do alvo, execução dos testes de intrusão e análise dos resultados. A Seção 4.1 apresenta o planejamento da execução dos testes de intrusão definindo o alvo dos testes, as vulnerabilidades que serão exploradas, as ferramentas que serão usadas e as atividades que serão realizadas na execução dos testes. A Secão 4.2 apresenta os resultados obtidos no reconhecimento do alvo e na execução dos testes de intrusão, por fim, é apresentada uma discussão sobre os resultados.

## 4.1 PLANEJAMENTO DA EXECUÇÃO DOS TESTES

Esta seção tem como objetivo definir as vulnerabilidades que serão exploradas, o alvo dos testes de intrusão, as ferramentas que serão aplicadas, os recursos computacionais necessários e as atividades que serão realizadas na execução dos testes de intrusão.

As vulnerabilidades selecionadas para serem exploradas são associadas aos riscos do OWASP Top 2017, a saber: Injeção de SQL (A1), *Cross Site Scripting* (A7) e Entidades Externas de XML (A4). Estas vulnerabilidades foram selecionadas porque referem-se ao problema de validação de entrada de dados, o qual, conforme apresentado na Seção 3.1, é maior problema de segurança em aplicações Web.

O alvo escolhido para execução dos testes de intrusão foi a aplicação WebGoat (OWASP, 2018), pois contém as vulnerabilidades selecionadas. Trata-se de uma aplicação Web insegura, mantida pelo OWASP. Foi desenvolvida com o objetivo de fornecer um ambiente de ensino às pessoas interessadas em segurança de aplicações Web. Através de exercícios, dicas e códigos, as vulnerabilidades encontradas em aplicações Web são explicadas. A aplicação WebGoat foi desenvolvida nas seguintes plataformas de desenvolvimento: Java, Ruby on rails, PHP, NodeJs, Android e IOS. A aplicação utilizada neste trabalho foi a desenvolvida em Java, versão 8.0.0.M21. Esta versão da aplicação pode ser obtida através do endereço https://github.com/WebGoat/WebGoat/releases.

Dentre as ferramentas levantadas na Seção 3.2 foram selecionadas 14 para realizar os testes de intrusão, a saber: Arachni 1.5.1-0.5.12, Beef 0.4.7.0-alpha, Htcap 1.0.1, IronWASP 0.9.8.6, Metasploit v4.17.28, Skipfish 2.10b, SQLMap 1.2.8.8, Vega 1.0, W3af 2018.8.22, Wapiti 3.0.1, Wfuzz 2.2.11, XSSer 1.7-2, Xenotix 6.2 e ZAP 2.7.0. A Seção 3.2 mapeou as vulnerabilidades que podem ser detectadas pelas ferramentas Arachni, IronWASP, Skipfish, Vega, W3af, Wapiti e ZAP. Estas ferramentas têm foco na fase de análise de vulnerabilidades e foram selecionadas com base na avaliação realizada por WAVSEP (2014). Além das ferramentas para análise de vulnerabilidades, são necessárias ferramentas para exploração das vulnerabilidades selecionadas. Neste sentido, foram selecionadas as ferramentas SQLMap, Beef, XSSer e Xenotix. A ferramenta SQLMap explora a vulnerabilidade Injeção de SQL. Já as ferramentas Beef, XSSer e Xenotix exploram a vulnerabilidade *Cross Site Scripting*. As ferramentas Beef, Metasploit, SQLMap, Wfuzz e Xenotix foram selecionadas porque são ferramentas de referência do Guia de Testes do OWASP. Já a ferramenta XSSer foi selecionada pois pode ser aplicada nas etapas de análise e exploração da vulnerabilidade *Cross Site Scripting*. Por fim, a ferramenta Htcap foi selecionada em função de sua documentação indicar a possibilidade de obter pontos de entrada em *Single-Page Application*. A ferramenta comercial Acutinex [1] também indica em sua

[1] https://www.acunetix.com

documentação a possibilidade de obter pontos de entrada em *Single-Page Application*. Com relação à vulnerabilidade Entidades Externas de XML, não foi encontrada nenhuma ferramenta para explorá-la.

Segue uma descrição de cada ferramenta. O local da disponibilidade das ferramentas está no Apêndice B.

1. **Arachni**: é um *scanner* de aplicação Web desenvolvido em Ruby. O foco desta ferramenta é a etapa de reconhecimento do alvo. Suporta aplicações Web que são implementadas em tecnologias como JavaScript, HTML5 e AJAX (McPhee, 2017).

2. **Beef (Browser Exploitation Framework)**: é uma ferramenta de código aberto para exploração da vulnerabilidade *Cross Site Scripting*. O foco da ferramenta é explorar o navegador onde a aplicação vulnerável é apresentada (Prasad, 2016).

3. **Htcap**: é um scanner de aplicação Web de código aberto capaz de obter pontos de entradas em *Single-Page Application*, interceptando chamadas AJAX e alterações DOM. O foco desta ferramenta é seu componente de *crawler*. A ferramenta pode ser estendida através da implementação de módulos customizados para detecção e exploração de vulnerabilidades, o que possibilita a integração com ferramentas externas especializadas (Htcap, 2018).

4. **IronWASP (Iron Web Application Advanced Security Testing Platform)**: é uma ferramenta de código aberto para testes de vulnerabilidades em aplicações Web, sendo projetado para ser estendida pelos usuários através da criação de scanner personalizados (Suteva et al., 2013).

5. **Metasploit**: é um *framework* para testes de intrusão. O *framework* é composto por diferentes módulos que podem ser utilizados para atender diversos tipos de aplicações, por exemplo, módulos relacionados à detecção e exploração de vulnerabilidades em aplicações Web (Prasad, 2016).

6. **Skipfish**: é um *scanner* de aplicação Web de código aberto. Através de seu componente de *crawler* a ferramenta prepara um mapa contendo todas as páginas da aplicação Web que serão posteriormente checadas (Makino e Klyuev, 2015).

7. **SQLmap**: é uma ferramenta de código aberto que automatiza o processo de detecção e exploração da vulnerabilidade Injeção de SQL em aplicações Web. Com esta ferramenta é possível identificar qual banco de dados a aplicação utiliza, recuperar informações do banco de dados, acessar o sistema de arquivos e executar comandos no sistema operacional (Prasad, 2016).

8. **Vega**: é um *scanner* de aplicação Web de código aberto e uma plataforma para testes de segurança em aplicações Web. A ferramenta inclui um scanner para testes automatizados e um *Proxy* para inspecionar as requisições da aplicação (Subgraph, 2018).

9. **W3af (Web Application Attack and Audit Framework)**: é um *scanner* de aplicação Web de código aberto. O objetivo da ferramenta é criar um *framework* para detectar e explorar vulnerabilidades em aplicações Web (Suteva et al., 2013).

10. **Wapiti**: é um *scanner* de aplicação Web de código aberto. Após o componente *crawler* encontrar os pontos de entradas que devem ser testados, a ferramenta age como um

*fuzzer* injetando dados nos campos encontrados para verificar se são vulneráveis (Wapiti, 2018).

11. **Wfuzz**: é um *scanner* de aplicação Web de código aberto que utiliza um *fuzzer* para gerar os dados de testes (Wfuzz, 2018).

12. **XSSer (Cross Site Scripter)**: é uma ferramenta para detecção e exploração automática da vulnerabilidade *Cross Site Scripting* em aplicações Web (XSSer, 2018).

13. **Xenotix**: é uma ferramenta, criada pelo OWASP, para detecção e exploração da vulnerabilidade *Cross Site Scripting*, com baixo número de falsos positivos na detecção e um *framework* para exploração contendo diversos módulos (Xenotix, 2018).

14. **ZAP (Zed Attack Proxy)**: é um *scanner* de aplicação Web de código aberto mantido pelo OWASP. É uma das ferramentas de código aberto para testes de segurança mais populares sendo mantido por diversos voluntários. Fornece *scanners* automatizados, bem como, um conjunto de ferramentas que permitem encontrar vulnerabilidades manualmente (Makino e Klyuev, 2015).

Com relação aos recursos computacionais, serão utilizados dois computadores. O computador um será utilizado para executar os testes aplicando as ferramentas IronWASP e Xenotix, as quais são desenvolvidas na plataforma Windows. A especificação deste computador é apresentada na Tabela 4.1.

Tabela 4.1: Especificação Computador 1. Fonte: O autor (2019)

| | |
|---|---|
| **Sistema operacional** | Windows Home 64 bits |
| **Memória** | 8 GB |
| **Processador** | Processador Intel i7-4500U CPU @ 2.40GHz 2 núcleos |
| **Disco** | 1 TB |

O computador dois será utilizado para executar os testes aplicando as demais ferramentas. A especificação deste computador é apresentada na Tabela 4.2.

Tabela 4.2: Especificação Computador 2. Fonte: O autor (2019)

| | |
|---|---|
| **Sistema operacional** | Ubuntu 16.04 64 bits |
| **Memória** | 12 GB |
| **Processador** | Processador Intel Xeon CPU E5530 @ 2.40GHz 4 núcleos |
| **Disco** | 300 GB |

Por fim, seguem as atividades que serão realizadas na execução dos testes:

1. Reconhecimento do alvo
   (a) Instalar a aplicação WebGoat
   (b) Instalar ferramenta *Proxy*
   (c) Identificar os casos de uso
   (d) Identificar e analisar os pontos de entrada
   (e) Analisar o código-fonte do lado cliente

2. Testes de intrusão
   (a) Definir os fatores avaliados
   (b) Instalar as ferramentas
   (c) Realizar os testes
   (d) Relatar os testes
3. Discutir os resultados obtidos

## 4.2 EXECUÇÃO DOS TESTES E RESULTADOS OBTIDOS

Esta seção descreve os resultados alcançados no reconhecimento do alvo e nos testes de intrusão, por fim, é realizada uma discussão sobre os resultados obtidos, conforme atividades apresentadas no planejamento da execução dos testes.

### 4.2.1 Reconhecimento do alvo

O propósito desta seção é identificar os casos de uso, pontos de entradas e tecnologias empregadas referente aplicação WebGoat, bem como, entender os cabeçalhos HTTP, parâmetros e dados encontrados nas requisições e respostas desta aplicação.

**Casos de uso**. Os casos de uso representam as funcionalidades visíveis do ponto de vista do usuário e para encontrá-los, sem a documentação da aplicação, é necessário navegar por ela. Portanto, deve-se abrir o navegador e digitar o endereço da aplicação, http://localhost:8080/WebGoat/. O usuário será direcionado para página de *Login*, caso não possua um usuário, será necessário registrá-lo para acessar a aplicação. Para a realização deste trabalho, foi criado o usuário mestrado e senha mestrado. O próximo passo é se autenticar na aplicação. Feito isto, o usuário será direcionado para página *Home* que contém o *Menu* da aplicação com as opções disponíveis. Foram identificados os seguintes casos de uso: *Register new User*, *Login*, *WebGoat*, *WebWolf*, *Http Basics*, *HTTP Proxies*, *SQL Injection (advanced)*, *SQL Injection*, *SQL Injection (mitigation)*, *XXE*, *Authentication Bypasses*, *JWT tokens*, *Password reset*, *Cross Site Scripting*, *Insecure Direct Object References*, *Missing Function Level Access Control*, *Insecure Login*, *Insecure Deserialization*, *Vulnerable Components*, *Bypass front-end restrictions*, *Client side filtering*, *HTML tampering*, *WebGoat Challenge*, *Admin lost password*, *Without password*, *Creating a new account*, *Admin password reset* e *Without account*. Os casos de uso da aplicação WebGoat que serão utilizados nos testes de intrusão são os presentes no diagrama de casos de uso da Figura 4.1.

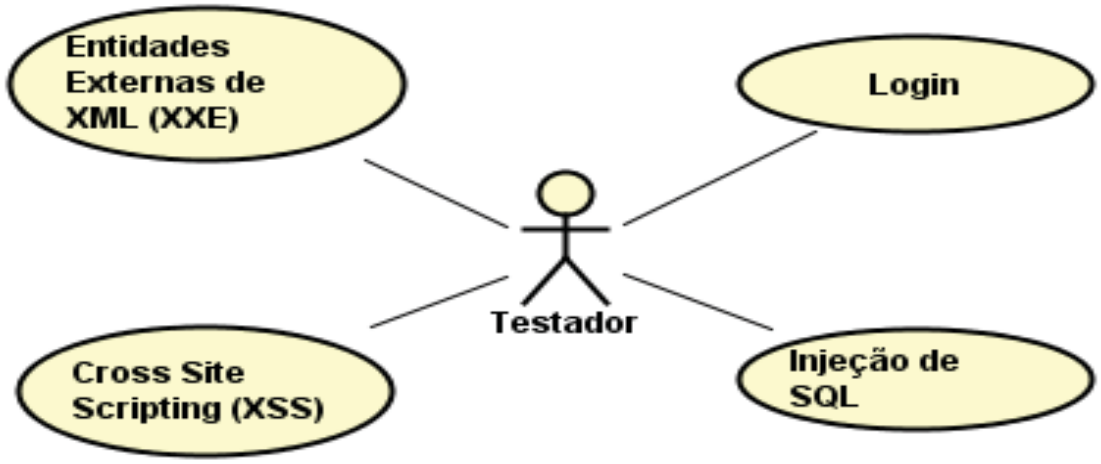



Figura 4.1: Diagrama de casos de uso da aplicação WebGoat utilizados nos testes de intrusão. Fonte: O autor (2019)

**Pontos de entrada dos casos de uso**. Um ponto de entrada refere-se a uma requisição feita pelo usuário, via navegador, para a aplicação. Cada caso de uso da aplicação pode conter mais de um ponto de entrada. Para identificá-los, é necessário navegar pelos casos de uso e encontrar todas as ações disponíveis aos usuário, por exemplo, botões e links. Deve-se clicar nas opções e, através de uma ferramenta de *Proxy*, interceptar as requisições e respostas. A ferramenta utilizada foi OWASP Webscarab [2] versão 20070504-1631. Foram identificados nove pontos de entrada descritos na Tabela 4.3. A primeira coluna da tabela contém o identificador da linha, a segunda o ponto de entrada, apresentando o método HTTP utilizado e a URL, a terceira os parâmetros enviados e a última indica a qual caso de uso pertence o ponto de entrada.

Tabela 4.3: Pontos de entrada dos casos de uso da aplicação WebGoat. Fonte: O autor (2019)

| **N°** | **Ponto de entrada** | **Parâmetros** | **Caso de uso** |
|---|---|---|---|
| 1 | POST http://localhost:8080/WebGoat/*Login* | username=mestrado & password=mestrado | *Login* |
| 2 | POST http://localhost:8080/WebGoat/SqlInjection/attack5a | account=teste | Injeção de SQL |
| 3 | POST http://localhost:8080/WebGoat/SqlInjection/attack5b | userid=1 | Injeção de SQL |
| 4 | GET http://localhost:8080/WebGoat/CrossSiteScripting/attack5a | QTY1=1&QTY2=1&QTY3=1&QTY4=1&field1=4128+3214+0002+1999&field2=111 | *Cross Site Scripting* |
| 5 | POST http://localhost:8080/WebGoat/CrossSiteScripting/dom-follow-up | successMessage=teste | *Cross Site Scripting* |

[2]https://www.owasp.org/index.php/Category:OWASP_WebScarab_Project

**Tabela 4.3 continuação da página anterior**

| N° | Ponto de entrada | Parâmetros | Caso de uso |
|---|---|---|---|
| 6 | POST http://localhost:8080/WebGoat/CrossSiteScripting/stored-xss | {"text":"teste"} | *Cross Site Scripting* |
| 7 | POST http://localhost:8080/WebGoat/xxe/simple | <?xml version="1.0"?> <comment> <text>teste</text> </comment> | Entidades Externas de XML |
| 8 | POST http://localhost:8080/WebGoat/xxe/content-type | {"text":"teste"} | Entidades Externas de XML |
| 9 | POST http://localhost:8080/WebGoat/xxe/blind | <?xml version="1.0"?> <comment><text>teste </text></comment> | Entidades Externas de XML |

Os pontos de entrada da Tabela 4.3 linhas 4, 5 e 6 referem-se, respectivamente, às vulnerabilidades XSS refletido, XSS armazenado e XSS DOM.

Uma vez identificados os pontos de entrada, o próximo passo é sua análise. A partir desta análise, é possível entender como ocorre a comunicação entre o navegador e a aplicação, o formato dos dados transferidos, os cabeçalhos HTTP, os parâmetros enviados e os dados de resposta.

Os pontos de entrada que serão detalhados para análise são os da linha 1 e 2 da Tabela 4.3, uma vez que os demais pontos de entrada são iguais ao da linha 2, mudando somente os dados de envio e de retorno, não acrescentando novas informações sobre a aplicação.

A Tabela 4.4 contém a requisição de *Login* da aplicação, ponto de entrada da Tabela 4.3 linha 1, detalhando os cabeçalhos e parâmetros enviados, enquanto que a Tabela 4.5 contém a resposta desta requisição, detalhando os parâmetros de retorno.

Tabela 4.4: Requisição de *Login* da aplicação WebGoat. Fonte: O autor (2019)

| **Caso de uso** | *Login* |
|---|---|
| **Descrição** | Requisição de *Login* enviada pela aplicação ao clicar no botão *Sign in*. Os dados informados na tela de *Login* foram usuário mestrado e senha mestrado |
| **Método URL Versão** | POST http://localhost:8080/WebGoat/*Login* HTTP/1.1 |

**Tabela 4.4 continuação da página anterior**

| | |
|---|---|
| **Cabeçalhos** | **Host**: localhost:8080<br>**User-Agent**: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:64.0) Gecko/20100101 Firefox/64.0<br>**Accept**: text/html,application/xhtml+xml,application/xml; q=0.9,*/*; q=0.8<br>**Accept-Language**: pt-BR,pt;q=0.8,en-US;q=0.5,en;q=0.3<br>**Accept-Encoding**: gzip, deflate<br>**Referer**: http://localhost:8080/WebGoat/*Login*<br>**Content-Type**: application/x-www-form-urlencoded<br>**Content-length**: 35<br>**Connection**: keep-alive<br>**Upgrade-Insecure-Requests**: 1 |
| **Parâmetros** | username=mestrado&password=mestrado |

Tabela 4.5: Resposta enviada pela aplicação para a requisição de *Login*. Fonte: O autor (2019)

| | |
|---|---|
| **Caso de uso** | *Login* |
| **Descrição** | Resposta enviada pela aplicação para a requisição de *Login* |
| **Versão Status Mensagem** | HTTP/1.1 200 |
| **Cabeçalhos** | **X-Content-Type-Options**: nosniff<br>**X-XSS-Protection**: 1; mode=block<br>**X-Frame-Options**: DENY<br>**Set-Cookie**: JSESSIONID=A5A9D2DE2414A2617179982AC45CD0D2; Path=/WebGoat<br>**Location**: http://localhost:8080/WebGoat/welcome.mvc<br>**Content-length**: 0 |

Com base nas informações da requisição e resposta de *Login* podemos concluir que a aplicação WebGoat implementa autenticação baseada em formulário, e que na resposta dessa requisição o *cookie* **JSESSIONID** (Set-Cookie) é atribuído com um novo identificador da sessão do usuário. Este *cookie* deverá ser enviado nas próximas requisições da aplicação.

A Tabela 4.6 contém as requisições do caso de uso Injeção de SQL, ponto de entrada referente a linha 2 da Tabela 4.3, detalhando os cabeçalhos e parâmetros enviados. Neste caso, foram realizadas duas requisições, uma enviando dados não maliciosos e outra enviando dados maliciosos que exploram a vulnerabilidade. Já a Tabela 4.7 contém a resposta para ambas requisições. A resposta para requisição que explora a vulnerabilidade retorna todos os registros da tabela de usuários da aplicação.

Tabela 4.6: Requisições do caso de uso Injeção de SQL. Fonte: O autor (2019)

| | |
|---|---|
| **Caso de uso** | Injeção de SQL |
| **Descrição** | Requisições do caso de uso Injeção de SQL, ponto de entrada referente a linha 2 da Tabela 4.3 |
| **Método URL Versão** | POST http://localhost:8080/WebGoat/SqlInjection/attack5a HTTP/1.1 |

**Tabela 4.6 continuação da página anterior**

| | |
|---|---|
| **Cabeçalhos** | **Host**: localhost:8080<br>**User-Agent**: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:64.0) Gecko/20100101 Firefox/64.0<br>**Acceptt**: */*<br>**Accept-Language**: pt-BR,pt;q=0.8,en-US;q=0.5,en;q=0.3<br>**Accept-Encoding**: gzip, deflate<br>**Connection**: keep-alive<br>**Content-length**: 13<br>**Referer**: http://localhost:8080/WebGoat/start.mvc<br>**Cookie**: JSESSIONID=A5A9D2DE2414A261717 9982AC45CD0D2<br>**Content-Type**: application/x-www-form-urlencoded; charset=UTF-8<br>**X-Requested-With**: XMLHttpRequest |
| **Parâmetros com dados não maliciosos** | account=teste |
| **Parâmetros com dados maliciosos** | account=’ OR ’1’=’1 |

Tabela 4.7: Respostas das requisições do caso de uso Injeção de SQL. Fonte: O autor (2019)

| | |
|---|---|
| **Caso de uso** | Injeção de SQL |
| **Descrição** | Repostas das requisições do caso de uso Injeção de SQL, ponto de entrada referente a linha 2 da Tabela 4.3 |
| **Versão Status Mensagem** | HTTP/1.1 200 |
| **Cabeçalhos** | **X-Application-Context**: application:8080<br>**X-Content-Type-Options**: nosniff<br>**X-XSS-Protection**: 1; mode=block<br>**X-Frame-Options**: DENY<br>**Content-Type**: application/json;charset=UTF-8<br>**X-Transfer-Encoding**: chunked<br>**Content-length**: 103 |
| **Resposta dados não maliciosos** | { "lessonCompleted": false, "feedback": "No results matched. Try Again.", "output": null } |

**Tabela 4.7 continuação da página anterior**

| | |
|---|---|
| **Resposta dados maliciosos** | {"lessonCompleted": true, "feedback": "You have succeed: USERID, FIRST_NAME,LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, *Login*_COUNT, <br/> 101, Joe, Snow, 987654321, VISA, , 0, <br/>101, Joe, Snow, 102, John, Smith, 2435600002222, MC, , 0, <br/>102, John, Smith, 4352209902222, AMEX, , 0, <br/>103, Jane, Plane, 123456789, MC, , 0, <br />103, Jane, Plane, 333498703333, AMEX, , 0, <br/>10312, Jolly, Hershey, 176896789, MC, , 0, <br/>10312, Jolly, Hershey, 333300003333, AMEX, , 0, <br/>10323, Grumpy, youaretheweakestlink, 673834489, MC, , 0, <br/>10323, Grumpy, youarethewe-akestlink, 33413003333, AMEX, , 0, <br/>15603, Pe-ter, Sand, 123609789, MC, , 0, <br/>15603, Peter, Sand, 338893453333, AMEX, , 0, <br/>15613, Joesph, Something, 33843453533, AMEX, , 0, </>15837, Chaos, Monkey, 32849386533, CM, , 0, </>19204, Mr, Goat, 33812953533, VISA, , 0, <br/> ", "output": null } |

Analisando as requisições da Tabela 4.6 e respostas da Tabela 4.7 da aplicação WebGoat, pode-se concluir duas questões sobre esta aplicação. A primeira, refere-se a comunicação com o servidor. A comunicação com o servidor é implementada utilizando o objeto Javascript XMLHttpRequest (X-Requested-With: XMLHttpRequest), ou seja, a aplicação faz uso de AJAX. Já a segunda, diz respeito ao formato dos dados retornados pela aplicação. A aplicação WebGoat retorna os dados no formato JSON (Content-Type: application/json).

A Figura 4.2 apresenta a tela da aplicação após clique no botão *Get Account info* enviando dados maliciosos.

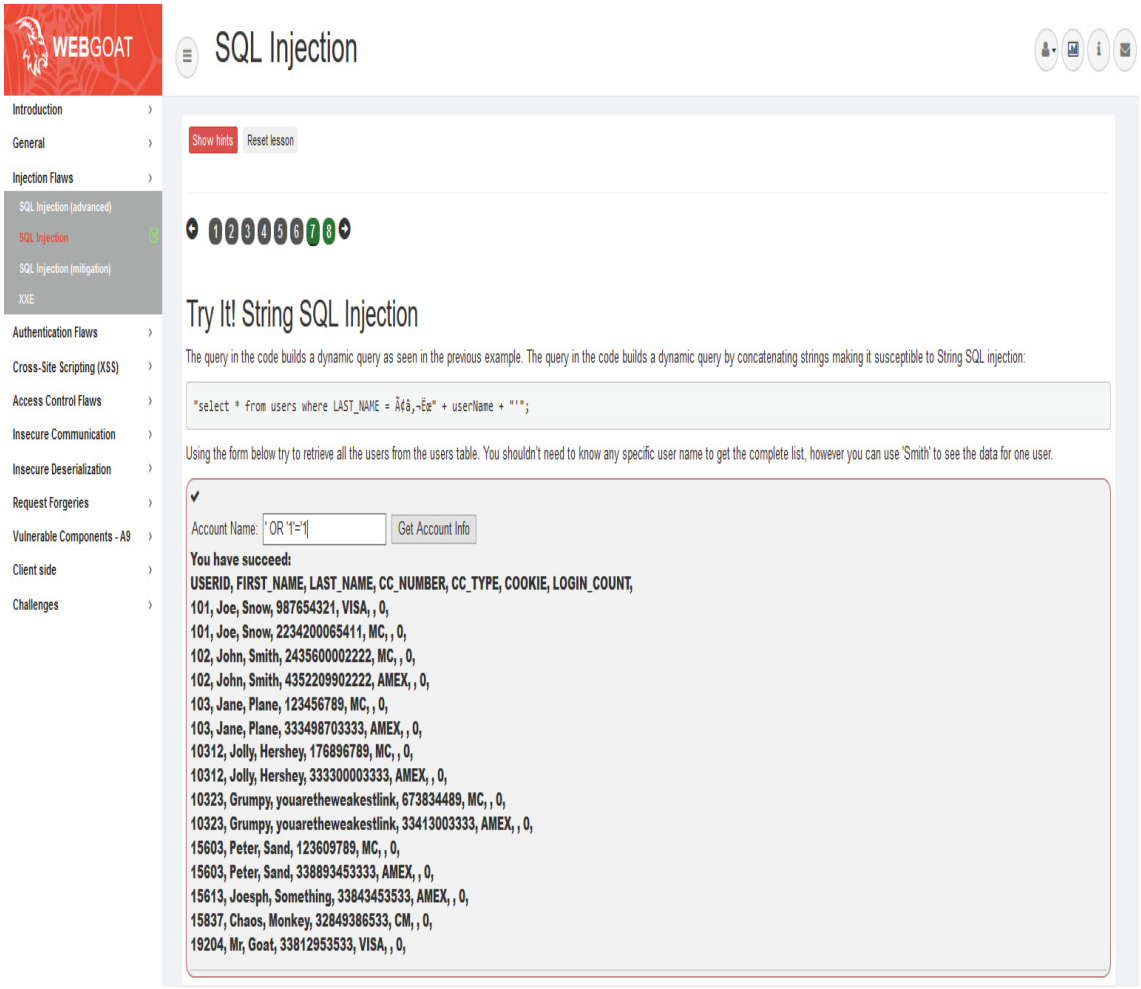



Figura 4.2: Tela do caso de uso Injeção de SQL da aplicação WebGoat. Fonte: OWASP (2018)

**Análise do código-fonte lado cliente**. Nos últimos anos, as aplicações Web tiveram uma evolução significativa na forma de construção dos componentes de interface gráfica. Ao contrário das aplicações Web clássicas, onde todo processamento tanto dos componentes de interface gráfica quanto dos componentes de regras de negócios era realizado no servidor, as novas aplicações Web possuem grande parte do processamento dos componentes da interface gráfica realizados no lado cliente, sendo JavaScript a linguagem predominante. Neste sentido, o objetivo desta seção é analisar como a camada de apresentação da aplicação WebGoat foi implementada. As ferramentas de desenvolvedor do navegador Google Chrome foram empregadas para obter as informações sobre a aplicação. A primeira atividade realizada foi analisar o código-fonte da página retornada após efetuar o *Login* na aplicação. A partir da análise do código-fonte desta página, extrai-se que as informações dinâmicas, por exemplo o *Menu*, não são retornadas pelo servidor. O código 4.1 refere-se a um trecho de código HTML da página retornado pelo servidor que apresenta o *Menu* da aplicação. No decorrer do texto será explicado como as informações dinâmicas são apresentadas.

Código 4.1: Código HTML retornado pelo servidor referente ao Menu da aplicação. Fonte: O autor (2019)

```
<aside class="sidebar">
   <div id="menu-container"></div>
</aside>
```

Usando as ferramentas de desenvolvedor do navegador é possível entender e visualizar a estrutura e o código-fonte da aplicação. A aplicação WebGoat é uma *Single-Page Application*. Existe uma única página HTML para toda a aplicação, **start.mvc**, as demais visões ou páginas são implementadas em Javascript e apresentadas de forma dinâmica dentro dela.

Através da Figura 4.3 pode-se visualizar a estrutura geral da aplicação do lado cliente. A aplicação é dividida utilizando o padrão *Model View Controller* (MVC). O *Model* contém os funções JavaScript para acesso a camada de negócio que reside no servidor, e como já identificado na seção anterior, é implementada utilizando o objeto JavaScript XMLHttpRequest. As *Views* referem-se às páginas HTML na qual o usuário interage, e são implementadas em JavaScript, processadas no navegador do cliente e apresentadas na única página HTML da aplicação. O *Controller* é responsável em receber os eventos das *Views*, repassá-los ao *Model* e determinar qual a próxima *View* que será apresentada.

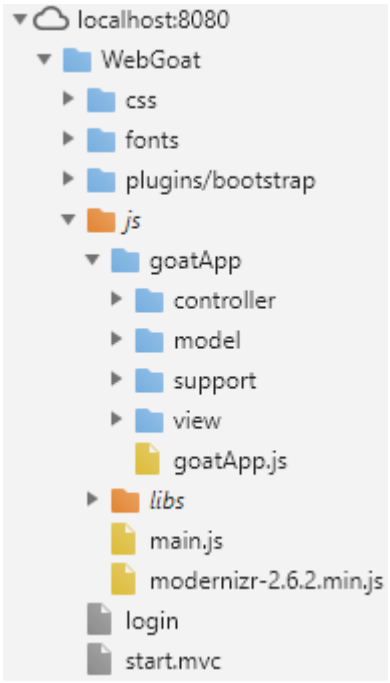



Figura 4.3: Estrutura geral da aplicação WebGoat lado cliente. Fonte: O autor (2019)

Os arquivos que contém o código JavaScript referentes às *Views* podem ser visualizados na Figura 4.4.

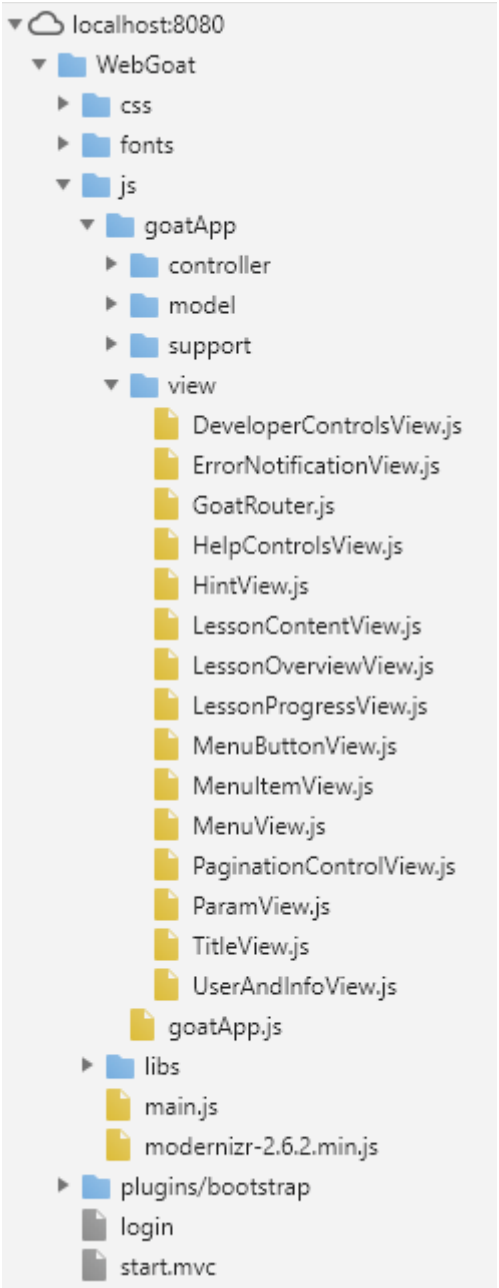



Figura 4.4: Arquivos que contém o código JavaScript referentes as *Views* da aplicação WebGoat. Fonte: O autor (2019)

A próxima questão a ser analisada refere-se a implementação do *Menu* da aplicação, uma vez que esta implementação pode afetar os resultados obtidos pelo componente de *crawler* das ferramentas. No caso da aplicação WebGoat, é a partir dos links do *Menu* da aplicação que as ferramentas deverão obter os pontos de entradas que serão testados.

O código 4.2 executa o serviço da camada de negócio que retorna os itens do *Menu* da aplicação, referente a URL **service/lessonmenu.mvc**. O serviço é executado a cada 5 segundos, linha 13, e ao receber a resposta com os dados do servidor, dispara o evento **menuData:load**, linha 19.

Código 4.2: MenuCollection.js - Código JavaScript que acessa a camada de negócio via XMLHttpRequest para obter os itens do *Menu* da aplicação. Fonte: O autor (2019)

```
1  define(['jquery',
2        'underscore',
3        'backbone',
4        'goatApp/model/MenuModel'],
5     function ($, _, Backbone, MenuModel) {
6        return Backbone.Collection.extend({
7           model: MenuModel,
8           url: 'service/lessonmenu.mvc',
9
10          initialize: function () {
11             var self = this;
12             this.fetch();
13             setInterval(function () {
14                this.fetch()
15             }.bind(this), 5000);
16          },
17
18          onDataLoaded: function () {
19             this.trigger('menuData:loaded');
20          },
21
22          fetch: function () {
23             var self = this;
24             Backbone.Collection.prototype.fetch.apply(this, arguments).then(
25                function (data) {
26                   this.models = data;
27                   self.onDataLoaded();
28                }
29             );
30          }
31       });
32    });
```

O código 4.3 implementa a parte *View* que renderiza o *Menu* da aplicação. Na linha 21, registra-se a função como ouvinte do evento **menuData:loaded**, assim, quando os dados, referentes ao itens de *Menu*, são retornados pelo servidor, a função **render** é executada, a partir da linha 25. A implementação de cada opção do *Menu* é encontrada nas linhas 59, 60, 62 e 63. Para cada opção do *Menu*, é construída, de forma dinâmica, um link utilizando a tag HTML <a ref=">.

Código 4.3: MenuView.js - Código JavaScript que implementa a geração do menu da aplicação. Fonte: O autor (2019)

```
1
2  define(['jquery',
3        'underscore',
4        'backbone',
5        'goatApp/model/MenuCollection',
6        'goatApp/view/MenuItemView',
7        'goatApp/support/GoatUtils'],
8     function($
```

```
9       _,
10      ,
11      Backbone,
12      MenuCollection,
13      MenuItemView,
14      GoatUtils) {
15
16   return Backbone.View.extend({el:'#menu-container',
17
18      initialize: function() {
19            this.collection = new MenuCollection();
20            this.addSpinner();
21            this.listenTo(this.collection,'menuData:loaded',this.render);
22            this.curLessonLinkId = '';
23      },
24
25      render: function (){
26            this.removeSpinner();
27            var items, catItems, stages;
28            items = this.collection.models; // top level items
29            var menuMarkup = '';
30            var menuUl = $('<ul>',{class:'nano-content'});
31            for(var i=0;i<items.length;i++) { //CATEGORY LEVEL
32                  var catId, category, catLink, catArrow,
33                  catLinkText, lessonName, stageName;
34                  catId = GoatUtils.makeId(items[i].get('name'));
35                  category = $('<li>',{class:'sub-menu ng-scope'});
36                  catLink = $('<a>',{'category':catId});
37                  catArrow = $('<i>',{class:'fa fa-angle-right pull-right'});
38                  catLinkText = $('<span>',{text:items[i].get('name')});
39                  catLink.append(catArrow);
40                  catLink.append(catLinkText);
41                  var self = this;
42                  catLink.click(_.bind(this.expandCategory,this,catId));
43                  category.append(catLink);
44                   MenuItemView({items:items[i].get('children')})
45                   .render();
46                  var lessons=items[i].get('children');
47                  if (lessons) {
48                        var categoryLessonList = $('<ul>',
49                        {class:'slideDown lessonsAndStages',
50                        id:catId}); //keepOpen
51                        for (var j=0; j < lessons.length;j++) {
52                              var lessonItem = $('<li>',{class:'lesson'});
53                              var lessonName = polyglot.t(lessons[j].name);
54                              var lessonId = catId + '-' +
55                              GoatUtils.makeId(lessonName);
56                              if (this.curLessonLinkId === lessonId) {
57                                    lessonItem.addClass('selected');
58                              }
59                              var lessonLink = $('<a>',
60                              {href:lessons[j].link,text:lessonName,
61                              id:lessonId});
62                              lessonLink.click(_.bind(this.onLessonClick,
63                              this,lessonId));
64                              lessonItem.append(lessonLink);
65                              //check for lab/stages
66                              categoryLessonList.append(lessonItem);
```

```
67                                  if (lessons[j].complete) {
68                                      lessonItem.append($('<span>',
69
70                                      {class:'glyphicon
71                                      glyphicon-check lesson-complete'}));
72                                  }
73                                  var stages = lessons[j].children;
74                                  for (k=0; k < stages.length; k++) {
75                                        var stageItem = $('<li>',
76                                        {class:'stage'});
77                                        var stageName = stages[k].name;
78                                        var stageId = lessonId + '-stage' + k;
79                                        if (this.curLessonLinkId === stageId) {
80                                              stageItem.addClass('selected');
81                                        }
82                                        var stageLink = $('<a>',
83                                        {href:stages[k].link,
84                                        text:stageName,id:stageId});
85                                        stageLink.click(_.bind(
86                                        this.onLessonClick, this,stageId));
87                                        stageItem.append(stageLink);
88                                        categoryLessonList.append(stageItem);
89                                        if (stages[k].complete) {
90                                              stageItem.append($('<span
91                                              >',{class:'glyphicon
92                                              glyphicon-check lesson-complete'})
93                                              );
94                                        }
95                                  }
96                            }
97                            category.append(categoryLessonList);
98                      }
99                      menuUl.append(category);
100               }
101               this.$el.html(menuUl);
102               //if we need to keep a menu open
103               if (this.openMenu) {
104                     $('#'+this.openMenu).show();
105               }
106         }});});
```

### 4.2.2 Testes de intrusão

O objetivo desta seção é apresentar os resultados da realização dos testes de intrusão na aplicação WebGoat. Conforme definido na Seção 4.1, as vulnerabilidades exploradas foram Injeção de SQL (A1), *Cross Site Scripting* (A7) e Entidades Externas de XML (A4). Com relação às ferramentas, foram aplicadas às seguintes: Arachni, Beef, Htcap, IronWASP, Metasploit, Skipfish, SQLMap, Vega, W3af, Wapiti, Wfuzz, XSSer, Xenotix e ZAP. Os pontos de entradas da aplicação WebGoat que foram utilizados nos testes são os apresentados na Tabela 4.3.

**Fatores Avaliados**. Os resultados serão apresentados levando em consideração os seguintes fatores:

- **Autenticação**: capacidade de detectar e explorar vulnerabilidades em aplicações Web que seja necessário realizar autenticação. Conforme apresentado na Seção 4.2.1, a aplicação WebGoat implementa autenticação baseada em formulário e para explorá-la é

necessário estar autenticado. Portando, é necessário que a ferramenta tenha a capacidade de em cada requisição enviar o identificador de sessão do usuário autenticado. Outra função que a ferramenta pode implementar, é a realização da autenticação de forma automática, a partir da URL de *Login*, usuário e senha, sem necessidade de intervenção manual do testador.

- ***Crawler***: capacidade de encontrar os pontos de entrada da aplicação Web de forma automática. A quantidade de pontos de entrada que são extraídos da aplicação Web afeta diretamente os testes de intrusão, uma vez que determina o percentual de cobertura dos testes da ferramenta. Geralmente, cada ferramenta possui sua implementação para extração. Portanto, podemos obter resultados distintos entre as ferramentas usadas para o mesmo propósito.

- **Detecção**: capacidade de identificar as vulnerabilidades selecionadas, as quais podem ser exploradas pelos atacantes.

- **Exploração**: capacidade de explorar as vulnerabilidades selecionadas, ou seja, causar dano a aplicação, uma vez que detectar uma vulnerabilidade não significa que o atacante cause algum dano.

- **Importar e exportar requisições**: capacidade de importar e exportar os pontos de entrada de/para outras ferramentas. O propósito desta análise é verificar a viabilidade de integrar as ferramentas aplicadas nos testes de intrusão, aumentar a capacidade de extração de pontos de entrada e o número de técnicas utilizadas para detecção das vulnerabilidades, uma vez que cada ferramenta pode ter foco em uma área específica do teste.

**Realização dos testes de intrusão**. Os resultados apresentados não levam em consideração a implementação de programas para detecção e exploração via API das ferramentas, quando existentes, somente via interface gráfica e linha de comando. A Tabela 4.8 contém os resultados por fator avaliado. A primeira coluna desta tabela contém o nome do fator avaliado, enquanto que a segunda apresenta o resultado detalhando a quantidade de ferramentas que atendem ao fator.

Tabela 4.8: Resultados da execução dos testes de intrusão na aplicação WebGoat detalhados por fator avaliado. Fonte: O autor (2019)

| **Fator** | **Resultado** |
|---|---|
| Autenticação | Dentre as ferramentas aplicadas, seis suportam autenticação baseada em formulário, a saber: Arachni, IronWASP, Skipfish, Vega, W3af e ZAP. Nas demais ferramentas, o identificador de sessão foi obtido através da ferramenta wapiti-getcookie. Dentre as seis ferramentas que suportam este tipo de autenticação, somente em três delas, Arachni, Skipfish e W3af, não existe intervenção manual, ou seja, a ferramenta consegue efetuar a autenticação de forma automática, sem intervenção do testador. |
| *Crawler* | Dentre as ferramentas aplicadas, 12 possuem o componente de *crawler*, a saber: Arachni, IronWASP, Skipfish, Vega, W3af, Wapiti, ZAP, SQLmap, Metasploit, Wfuzz, XSSer e Htcap. Dentre estas, somente duas encontram os pontos de entrada da aplicação, a saber: Arachni e Htcap. Nas demais ferramentas foi necessário encontrar os pontos de entrada través do *Proxy* embutido na ferramenta ou através da ferramenta Htcap. As ferramentas que contém *Proxy* embutido são IronWASP, Vega, W3af e ZAP. |

**Tabela 4.8 continuação da página anterior**

| Fator | Resultado |
|---|---|
| Detecção | Dentre as ferramentas aplicadas, 12 possuem o componente para detecção de vulnerabilidades, a saber: Arachni, IronWASP, Skipfish, Vega, W3af, Wapiti, ZAP, SQLmap, Metasploit, Wfuzz, Xenotix e XSSer. Dentre estas, 10 possuem testes para detectar Injeção de SQL, a saber: Arachni, IronWASP, Skipfish, Vega, W3af, Wapiti, ZAP, SQLmap, Metasploit e Wfuzz, contudo, somente quatro identificaram a vulnerabilidade, a saber: IronWASP, Vega, ZAP e SQLmap; oito possuem testes para detectar *Cross Site Scripting*, a saber: Arachni, IronWASP, Vega, Wapiti, ZAP, Wfuzz, Xenotix e XSSer, entretanto, somente duas detectaram a vulnerabilidade *XSS* refletido, a saber: ZAP e Xenotix. Os outros dois tipos de *XSS*, armazenado e DOM, não foram identificados por nenhuma das ferramentas; seis possuem testes para detectar XXE, a saber: Arachni, IronWASP, Vega, Wapiti, ZAP e Wfuzz, contudo, nenhuma ferramenta foi capaz de detectar esta vulnerabilidade. |
| Exploração | Dentre as ferramentas aplicadas, quatro são aplicadas na exploração de vulnerabilidades, a saber: SQLmap, Beef, Xenotix e XSSer. Dentre estas, uma é aplicada na exploração de Injeção de SQL, a saber: SQLmap. Através da ferramenta SQLmap foi possível obter uma cópia dos dados da aplicação WebGoat; três são aplicadas para explorar *Cross Site Scripting*, a saber: Beef, Xenotix e XSSer, entretanto, não foi possível explorar a vulnerabilidade com nenhuma delas. As ferramentas Beef e Xenotix, para explorar a vulnerabilidade, precisam enviar para aplicação, através do campo vulnerável, um código JavaScript que instala suas ferramentas de exploração no navegador da vítima, sendo que tal código JavaScript não pode ser enviado devido a validações realizadas pela aplicação WebGoat. Com relação à ferramenta XSSer, não foi possível explorar uma vez que a ferramenta não identificou a vulnerabilidade. |
| Imp/Exp | Dentre as ferramentas aplicadas: três importam e exportam pontos de entrada, a saber: IronWASP, W3af e SQLmap; três não importam e não exportam pontos de entrada, a saber: Arachni, Vega, Wapiti, Beef e Xenotix; quatro somente importam pontos de entrada, a saber: Wfuzz, XSSer, Beef e Xenotix; três somente exportam pontos de entrada, a saber: Skipfish, ZAP e Htcap. |

A Tabela 4.9 apresenta os resultados detalhados por ferramenta. A primeira coluna desta tabela contém o nome da ferramenta. A segunda coluna apresenta o resultado dos fatores avaliados.

Tabela 4.9: Resultados da execução dos testes de intrusão na aplicação WebGoat detalhados por ferramenta. Fonte: O autor (2019)

| Ferramenta | Resultado |
|---|---|
| Arachni | Esta ferramenta suporta autenticação baseada em formulário e possui *plugin* para realização da autenticação de forma automática, a partir da URL de *Login*, usuário e senha. O componente *crawler* consegue obter os pontos de entrada da aplicação, entretanto, não detectou nenhuma das vulnerabilidades testadas. É possível importar e exportar somente URL encontradas pelo *crawler* e não requisições completas. Esta ferramenta não é aplicada para exploração de vulnerabilidades. |

**Tabela 4.9 continuação da página anterior**

| Ferramenta | Resultado |
|---|---|
| Beef | Esta ferramenta não suporta autenticação baseada em formulário e não suporta autenticação automática. Neste caso, o identificador de sessão foi obtido através da ferramenta wapiti-getcookie. A ferramenta não possui componente de *crawler* e detectão, sendo somente aplicada para exploração da vulnerabilidade *Cross Site Scripting*. Os pontos de entradas da aplicação foram obtidos usando a ferramenta Htcap. No que tange a exploração, não foi possível utilizar a ferramenta. Para instalar as ferramentas de exploração no navegador do cliente, é preciso passar no campo vulnerável o seguinte *payload* `<script src='http://172.29.12.205:3000/hook.js'></script>`, todavia, a aplicação WebGoat aceita somente o *payload* `<script> alert ('my javascript here')</script>`, conforme mensagem retornada pela aplicação: `<script> alert ('my javascript here') </script> Try again. We do want to see this specific javascript (in case you are trying to do something more fancy)` |
| Htcap | Esta ferramenta não suporta autenticação baseada em formulário e não suporta autenticação automática. Neste caso, o identificador de sessão foi obtido através da ferramenta wapiti-getcookie. O foco desta ferramenta é o componente de *crawler*. A versão utilizada não possui por padrão componente para detecção e exploração de vulnerabilidades, contudo, permite que seja estendida através da implementação de módulos customizados para detecção e exploração. A versão aplicada possui módulos para as ferramentas SQLmap, Arachni e Wapiti. O componente *crawler* consegue obter os pontos de entrada da aplicação, sendo possível exportá-las. |
| IronWASP | Esta ferramenta suporta autenticação baseada em formulário e autenticação automática, entretanto, para configurar a autenticação automática é preciso intervenção manual para gravar o procedimento de autenticação. Com relação à autenticação automática, foi apresentado uma limitação, não sendo possível utilizá-lo junto com o componente de *crawler*, assim, não foi possível obter os pontos de entrada da aplicação de forma automática, e para obtê-los foi utilizado o *Proxy* embutido na ferramenta, sendo necessário navegar pelos casos de uso e executar os pontos de entrada. No que diz respeito a detecção, a ferramenta detectou somente Injeção de SQL e nos testes para Entidades Externas de XML reportou a vulnerabilidade *Local File Include*. Esta ferramenta não é aplicada para exploração das vulnerabilidades testadas, contudo, existe funcionalidades para explorar *Cross-Site Request Forgery*. É possível importar e exportar os pontos de entrada. |

**Tabela 4.9 continuação da página anterior**

| Ferramenta | Resultado |
|---|---|
| Metasploit | Esta ferramenta não suporta autenticação baseada em formulário e não suporta autenticação automática. Neste caso, o identificador de sessão foi obtido através da ferramenta wapiti-getcookie. O componente *crawler* não consegue obter os pontos de entrada da aplicação, e como a ferramenta não tem *Proxy* embutido, os pontos de entrada foram obtidas usando a ferramenta Htcap. No que diz respeito a detecção, a ferramenta somente possui módulo para detecção de Injeção de SQL, não sendo aplicada para as demais vulnerabilidades selecionadas. A vulnerabilidade Injeção de SQL não foi detectada. Esta ferramenta pode ser aplicada na exploração de vulnerabilidades através de *plugin* para ferramentas externas, como SQLmap e Beef. É possível importar e exportar requisições. |
| Skipfish | Esta ferramenta suporta autenticação baseada em formulário e autenticação automática, entretanto, ao tentar utilizar esta função a ferramenta apresenta um erro que impede prosseguir com os testes. Neste caso, o identificador de sessão foi obtido através da ferramenta wapiti-getcookie. O componente *crawler* não consegue obter os pontos de entrada da aplicação, e como a ferramenta não tem *Proxy* embutido e não importa pontos de entrada, não foi possível testar nenhuma das vulnerabilidades. Esta ferramenta não é aplicada para exploração de vulnerabilidades e exporta pontos de entrada. |
| SQLmap | Esta ferramenta não suporta autenticação baseada em formulário e não suporta autenticação automática. Neste caso, o identificador de sessão foi obtido através da ferramenta wapiti-getcookie. O componente *crawler* não consegue obter os pontos de entrada da aplicação, e como a ferramenta não tem *Proxy* embutido, os pontos de entrada foram obtidas usando a ferramenta Htcap. No que diz respeito a detecção, a ferramenta detectou Injeção de SQL, não sendo aplicada para as demais vulnerabilidades selecionadas. Durante a detecção da vulnerabilidade a ferramenta identificou que o banco de dados utilizado pela aplicação WebGoat é HSQLDB [3]. com relação à exploração da vulnerabilidade Injeção de SQL, foi possível realizar uma cópia dos dados da aplicação. Não foi possível realizar *upload* de arquivos e executar comandos no sistema operacional, uma vez que o banco de dados HSQLDB não suporta tais operações. É possível importar e exportar pontos de entrada. |
| Vega | Esta ferramenta suporta autenticação baseada em formulário e autenticação automática, entretanto, para configurar a autenticação automática é preciso intervenção manual para gravar o procedimento de autenticação. O componente *crawler* não consegue obter os pontos de entrada da aplicação e para obtê-los foi utilizado o *Proxy* embutido da ferramenta, sendo necessário navegar pelos casos de uso e executar os pontos de entrada. No que diz respeito a detecção, a ferramenta detectou Injeção de SQL e nos testes para Entidades Externas de XML reportou a vulnerabilidade *Local File System Paths*. Esta ferramenta não é aplicada para exploração de vulnerabilidades e não é possível importar ou exportar pontos de entrada. |

[3]http://hsqldb.org/

**Tabela 4.9 continuação da página anterior**

| Ferramenta | Resultado |
|---|---|
| W3af | Esta ferramenta suporta autenticação baseada em formulário e possui *plugin* para realização da autenticação de forma automática, a partir da URL de *Login*, usuário e senha. O componente *crawler* não consegue obter os pontos de entrada e para obtê-las foi utilizado o *Proxy* embutido da ferramenta, sendo necessário navegar pelos casos de uso e executar os pontos de entrada. No que diz respeito a detecção, a ferramenta não detectou nenhuma das vulnerabilidades. Esta ferramenta não é aplicada para exploração de vulnerabilidades, entretanto, é possível configurar e executar ferramentas para exploração a partir dela, como SQLmap. É possível importar e exportar os pontos de entrada. |
| Wapiti | Esta ferramenta não suporta autenticação baseada em formulário e não suporta autenticação automática, contudo, disponibiliza a ferramenta wapiti-getcookie para obter o identificador de sessão a partir da URL de *Login*, usuário e senha. O identificador de sessão é retornado em um arquivo cookies.json. Uma vez, encontrado o identificador deve-se passá-lo como parâmetro na execução da ferramenta. O componente *crawler* não consegue obter os pontos de entrada da aplicação, e como a ferramenta não tem *Proxy* embutido e não importa pontos de entrada, não foi possível testar nenhuma das vulnerabilidades. Esta ferramenta não é aplicada para exploração de vulnerabilidades e não exporta pontos de entrada. |
| Wfuzz | Esta ferramenta não suporta autenticação baseada em formulário e não suporta autenticação automática. Neste caso, o identificador de sessão foi obtido através da ferramenta wapiti-getcookie. Não possui componente de *crawler*, assim, os pontos de entrada foram obtidos usando a ferramenta Htcap. No que diz respeito a detecção, a ferramenta não detectou nenhuma das vulnerabilidades. Esta ferramenta não se aplica para exploração de vulnerabilidades. É possível importar os pontos de entrada. |
| XSSer | Esta ferramenta não suporta autenticação baseada em formulário e não suporta autenticação automática. Neste caso, o identificador de sessão foi obtido através da ferramenta wapiti-getcookie. O componente *crawler* não consegue obter os pontos de entrada, e como a ferramenta não tem *Proxy* embutido e não importa os pontos de entrada, não foi possível detectar e explorar a vulnerabilidade *Cross Site Scripting*. |

**Tabela 4.9 continuação da página anterior**

| Ferramenta | Resultado |
|---|---|
| Xenotix | Esta ferramenta não suporta autenticação baseada em formulário e não suporta autenticação automática. Neste caso, o identificador de sessão foi obtido através da ferramenta wapiti-getcookie. A ferramenta não possui componente de *crawler* sendo aplicada para detecção e exploração da vulnerabilidade *Cross Site Scripting*. Os pontos de entradas da aplicação foram obtidos usando a ferramenta Htcap. No que tange a detecção, a ferramenta identificou a vulnerabilidade *Cross Site Scripting* refletido. Com relação à exploração, não foi possível utilizar a ferramenta. Para instalar as ferramentas de exploração no navegador do cliente, é preciso passar no campo vulnerável o seguinte *payload* `<script src='http://172.29.12.205:3000/xook.js'></script>`, todavia, a aplicação WebGoat aceita somente o *payload* `<script> alert ('my javascript here')</script>`, conforme mensagem retornada pela aplicação: `<script> alert ('my javascript here') </script> Try again. We do want to see this specific javascript (in case you are trying to do something more fancy)` |
| ZAP | Esta ferramenta suporta autenticação baseada em formulário e autenticação automática, entretanto, para configurar a autenticação automática é preciso intervenção manual para gravar o procedimento de autenticação. O componente *crawler* não consegue obter os pontos de entrada da aplicação e para obtê-las foi utilizado o *Proxy* embutido da ferramenta, sendo necessário navegar pelos casos de uso e executar os pontos de entrada. No que diz respeito a detecção, a ferramenta detectou Injeção de SQL e *Cross Site Scripting* refletido. Nos testes para Entidades Externas de XML detectou a vulnerabilidade *Parameter Tampering*. Esta ferramenta não é aplicada para exploração das vulnerabilidades, entretanto, é possível configurar e executar ferramentas para exploração a partir dela, como SQLmap. É possível exportar pontos de entrada, contudo, não é possível importá-los. |

Por fim, a Tabela 4.10 fornece um resumo dos resultados. A primeira coluna da tabela contém o nome da ferramenta. A segunda coluna refere-se ao fator autenticação. Este fator foi dividido em outras três colunas. A coluna "Form" indica se a ferramenta suporta autenticação baseada em formulário, a coluna "Auto" indica se a ferramenta realiza a autenticação de forma automática e a coluna "S/ Int" indica se existe intervenção manual do testador. A terceira coluna refere-se ao fator *crawler*, indicando se a ferramenta obteve os pontos de entrada da aplicação. A quarta coluna refere-se ao fator detecção. Este fator foi dividido em outras cinco colunas. A coluna "SQLI" indica se a ferramenta detectou a vulnerabilidade Injeção de SQL, a coluna "XSSR" indica se a ferramenta detectou a vulnerabilidade *Cross Site Scripting* refletido, a coluna "XSSS" indica se a ferramenta detectou a vulnerabilidade *Cross Site Scripting* armazenado, a coluna "XSSD" indica se a ferramenta detectou a vulnerabilidade *Cross Site Scripting* DOM e a coluna "XEE" indica se a ferramenta detectou a vulnerabilidade Entidade Externas de XML. A quinta coluna refere-se ao fator exploração. Este fator foi dividido em outras cinco colunas. A coluna "SQLI" indica se a ferramenta explorou a vulnerabilidade Injeção de SQL, a coluna "XSSR" indica se a ferramenta explorou a vulnerabilidade *Cross Site Scripting* refletido, a coluna "XSSS" indica se a ferramenta explorou a vulnerabilidade *Cross Site Scripting* armazenado, a

coluna "XSSD" indica se a ferramenta explorou a vulnerabilidade *Cross Site Scripting* DOM e a coluna "XEE" indica se a ferramenta explorou a vulnerabilidade Entidade Externas de XML. A sexta coluna refere-se ao fator importação e exportação de pontos de entrada. Este fator foi divido em duas outras colunas. A coluna "IMP" indica se a ferramenta importa pontos de entrada e a coluna "EXP" indica se a ferramenta exporta pontos de entrada. A sétima coluna “ARD” indica se ferramenta detectou por completo, parcial ou nenhuma das vulnerabilidades descritas em sua documentação. Cada célula da tabela pode conter os seguintes valores "✓", "✕", "NA", "N*", "●", "◐"e "○". O valor "✓" indica que a ferramenta atende ao fator avaliado; o valor "✕" indica que a ferramenta não atende ao fator; o valor "NA" indica que a ferramenta não se aplica ao fator; o valor "N*" indica que não foi possível testar a ferramenta para o fator; o valor "●" indica que a ferramenta detecta todas as vulnerabilidades descritas em sua documentação; o valor "◐" indica que a ferramenta detecta parcialmente as vulnerabilidades descritas em sua documentação; e, por fim, o valor "○" indica que a ferramenta não detecta nenhuma das vulnerabilidades descritas em sua documentação.

Tabela 4.10: Resumo dos resultados da execução dos testes de intrusão na aplicação WebGoat. Fonte: O autor (2019)

| **Ferramenta** | **Autenticação** | | | **Crawler** | **Detecção** | | | | | **Exploração** | | | | | **Importação/ Exportação** | | **ARD** [a] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Form** [b] | **Auto** | **S/ Int** [c] | | **SQLI** | **XSSR** | **XSSS** | **XSSD** | **XEE** | **SQLI** | **XSSR** | **XSSS** | **XSSD** | **XEE** | **IMP** | **EXP** | |
| Arachni | ✓ | ✓ | ✓ | ✓ | × | × | × | × | × | NA | NA | NA | NA | NA | × | × | ○ |
| Beef | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | N* | N* | N* | NA | ✓ | × | NA |
| Htcap | × | × | × | ✓ | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | × | ✓ | NA |
| IronWASP | ✓ | ✓ | × | × | ✓ | × | × | × | × | NA | NA | NA | NA | NA | ✓ | ✓ | ◐ |
| Metasploit | × | × | × | × | × | NA | NA | NA | NA | NA | NA | NA | NA | NA | ✓ | ✓ | ○ |
| Skipfish | ✓ | ✓ | ✓ | × | × | × | × | × | × | NA | NA | NA | NA | NA | × | ✓ | ○ |
| SQLmap | × | × | × | × | ✓ | NA | NA | NA | NA | ✓ | NA | NA | NA | NA | ✓ | ✓ | ● |
| Vega | ✓ | ✓ | × | × | ✓ | × | × | NA | × | NA | NA | NA | NA | NA | × | × | ◐ |
| W3af | ✓ | ✓ | ✓ | × | × | × | × | NA | NA | NA | NA | NA | NA | NA | ✓ | ✓ | ○ |
| Wapiti | × | × | × | × | × | × | × | NA | × | NA | NA | NA | NA | NA | × | × | ○ |
| WFuzz | × | × | × | × | × | × | × | × | × | NA | NA | NA | NA | NA | ✓ | × | ○ |
| XSSer | × | × | × | × | NA | × | × | × | NA | NA | × | × | × | NA | ✓ | × | ○ |
| Xenotix | × | × | × | NA | NA | ✓ | × | × | NA | NA | N* | N* | N* | NA | ✓ | × | NA |
| ZAP | ✓ | ✓ | × | × | ✓ | ✓ | × | NA | × | NA | NA | NA | NA | NA | × | ✓ | ◐ |
| **Total ✓** | **6** | **6** | **3** | **2** | **4** | **2** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **8** | **7** | NA |

[a] ARD = Atende os requisitos da documentação referentes à detecção de vulnerabilidades
[b] Form = Autenticação baseada em formulário
[c] S/ Int = Sem intervenção manual

### 4.2.3 Discussão dos resultados obtidos

O objetivo desta seção é discutir os resultados dos testes de intrusão que foram apresentados na seção anterior. A discussão abordará questões relacionadas à autenticação, ao *crawler*, à detecção de vulnerabilidades, à exploração de vulnerabilidades, à integração das ferramentas e à escolha da aplicação WebGoat como alvo dos testes.

Conforme apresentado na Seção 4.2.1, a aplicação WebGoat é uma *Single-Page Application*. Todo conteúdo dinâmico da camada da apresentação, incluindo o *Menu*, é implementado em JavaScript. A aplicação faz uso de AJAX, uma vez que em diversos pontos de entrada se comunica com o servidor utilizando o objeto JavaScript XMLHttpRequest. Os dados de retorno das requisições são formatados em JSON, e para acessar os pontos de entrada da aplicação é preciso estar autenticado, sendo autenticação baseada em formulário a forma empregada.

Sobre a **autenticação**, este é um mecanismo muito encontrado em aplicações Web. Neste sentido, as ferramentas devem lidar de forma adequada com esta questão. Contudo, pelos resultados obtidos 46% das ferramentas avaliadas suportam autenticação baseada em formulário e são capazes de efetuar a autenticação de forma automática, repassando o identificador de sessão para os demais componentes da ferramenta. Esta falta de suporte não impede que as ferramentas sejam utilizadas nos testes, desde que forneçam um meio para informar o identificador de sessão de um usuário autenticado. Um dos problemas desta abordagem é a necessidade de utilização de uma outra ferramenta para obter o identificador de sessão do usuário autenticado, tornando a automatização dos testes complexa, uma vez que é necessário integrar as ferramentas. Nas ferramentas em que não foi utilizado *Proxy* para obter os pontos de entrada, a ferramenta wapiti-getcookie foi aplicada para obter o identificador de sessão.

Sobre o ***crawler***, este desempenha um papel crítico na execução automatizada dos testes de intrusão, uma vez que não encontrar pontos de entrada da aplicação Web significa realizar testes incompletos deixando de detectar vulnerabilidades. Verificou-se que duas ferramentas conseguem obter os pontos de entrada da aplicação WebGoat de forma automática. Portanto, o componente *crawler* da maioria das ferramentas não estão preparados para trabalhar com *Single-Page Application*. Esta questão já foi identificada e analisada por Giancarlo Pellegrino et al. (2015). Tradicionalmente, o *crawler* encontra novas páginas da aplicação Web através de padrões encontrados no conteúdo HTML retornado do servidor Web. Esta abordagem consegue extrair pontos de entrada em aplicações clássicas, entretanto, falha em aplicações modernas, onde grande parte da interface gráfica é construída em JavaScript. Neste sentido, links e formulários HTML são construídos de forma dinâmica, ou seja, em tempo de execução da página no navegador, assim ao analisar o conteúdo retornado pelo servidor Web, o *crawler* não encontrará os elementos esperados. O fato do *crawler* das ferramentas IronWASP, Vega, W3af, ZAP, SQLmap, Metasploit e Wfuzz não encontrar os pontos de entrada da aplicação alvo não inviabilizou a execução dos testes. Nas ferramentas IronWASP, Vega, W3af e ZAP foi utilizado o *Proxy* da ferramenta para obter os pontos de entrada. A utilização do *Proxy* impõe a necessidade do testador navegar pela aplicação e executar todos os pontos de entrada de forma manual, gerando uma dependência humana para obter os pontos de entrada. Nas ferramentas SQLmap, Metasploit e Wfuzz os pontos de entrada foram importados da ferramenta Htcap. O problema desta abordagem está na complexidade em integrar as duas ferramentas. A ferramenta Htcap foi utilizada porque suporta a exportação dos pontos de entrada, esta capacidade não está presente na Arachni que somente suporta a exportação de URL. Já nas ferramentas Skipfish,Wapiti, e XSSer não foi possível contornar o problema do *crawler* inviabilizando a execução dos testes, uma vez que não possuem *Proxy* e não é possível importar pontos de entrada de outra ferramenta.

Sobre a **detecção de vulnerabilidades**, somente 40% das ferramentas detectaram a vulnerabilidade Injeção de SQL e 20% das ferramentas detectaram a vulnerabilidade *Cross Site Scripting* refletido. Nenhuma das ferramentas detectou as vulnerabilidades *Cross Site Scripting* armazenado, *Cross Site Scripting* DOM e Entidades Externas de XML. Está fora do escopo deste trabalho entender os motivos pelas quais as ferramentas falharam na detecção. A Tabela 3.3 apresentou um mapeamento das vulnerabilidades que algumas ferramentas podem detectar. No mapeamento foram utilizadas sete ferramentas com foco na fase de análise de vulnerabilidades. As ferramentas mapeadas foram: Arachni, IronWASP, Skipfish, Vega, W3af, Wapiti e ZAP. Todas as ferramentas continham testes para as vulnerabilidades Injeção de SQL, *Cross Site Scripting* refletido e *Cross Site Scripting* armazenado. Contudo, nos testes realizados na aplicação Webgoat somente as ferramentas IronWASP, Vega e ZAP detectaram a vulnerabilidade Injeção de SQL. Com relação à vulnerabilidade *Cross Site Scripting* refletido, somente com a ferramenta ZAP foi possível detectá-la. Nenhuma ferramenta foi capaz de detectar a vulnerabilidade *Cross Site Scripting* armazenado. Com relação à vulnerabilidade *Cross Site Scripting* DOM, somente as ferramentas Arachni e IronWASP continham testes para esta vulnerabilidade, entretanto, nenhuma ferramenta detectou a vulnerabilidade. Com relação à vulnerabilidade Entidades Externas de XML, nenhuma das ferramentas analisadas possuem testes específicos para esta vulnerabilidade. As ferramentas possuem testes para vulnerabilidade Injeção de XML, a qual pode ser considerada uma generalização das vulnerabilidades relacionadas à XML, como Entidades Externas de XML. Dentre as ferramentas, somente W3af não continha testes para a vulnerabilidade Injeção de XML. As demais ferramentas, que continham testes para Injeção de XML, não foram capazes de detectar a vulnerabilidade Entidades Externas de XML.

Sobre a **exploração de vulnerabilidades**, somente foi possível explorar a vulnerabilidade Injeção de SQL. As ferramentas para explorar *Cross Site Scripting* não foram aplicadas em função da necessidade de instalação de ferramentas JavaScript, que não foram instaladas devido a validações realizadas pela aplicação WebGoat. Com relação à vulnerabilidade Entidades Externas de XML, não foi encontrada, durante a pesquisa, nenhuma ferramenta para explorá-la.

Sobre a **integração entre as ferramentas**, foi possível ratificar que uma ferramenta isolada não implementa todos os requisitos necessários para realizar um teste de intrusão completo na aplicação WebGoat. No contexto deste trabalho, realizar um teste de intrusão completo significa efetuar a autenticação, obter os pontos de entrada, detectar e explorar a vulnerabilidade. A única vulnerabilidade na qual foi possível executar um teste de intrusão completo foi Injeção de SQL, e para executá-lo foi necessário aplicar três ferramentas distintas que se integraram via seus resultados. A primeira ferramenta aplicada foi wapiti-getcookie, que gerou como resultado um arquivo no formato JSON contendo o identificador de sessão. Esta ferramenta recebe como entrada o usuário, senha e a URL para realizar a autenticação. A segunda ferramenta aplicada foi Htcap, que gerou como resultado um arquivo no formato do banco de dados SQLite [4] contendo os pontos de entrada identificados. Esta ferramenta recebe como entrada o identificador de sessão. Por fim, a terceira ferramenta aplicada foi SQLmap, sendo utilizada para detectar e explorar a vulnerabilidade. Esta ferramenta recebe como entrada o identificador de sessão e os pontos de entrada. Durante a etapa de detecção a ferramenta identificou que o banco de dados utilizado pela aplicação WebGoat é HSQLDB [5]. Como resultado da exploração, a ferramenta forneceu um arquivo texto contendo a cópia dos dados da aplicação.

Sobre a **escolha da aplicação WebGoat**, esta não se mostrou adequada para realizar testes de intrusão automatizados através do uso de ferramentas. Esta afirmação tem como base as limitações apresentadas nos parágrafos anteriores. São limitações impostas devido às validações

[4]https://www.sqlite.org

[5]http://hsqldb.org/

que a aplicação implementa e às tecnologias utilizadas por ela. Este fato pode ser considerado um indício de que ao realizar testes em outras aplicações, com implementações e tecnologias diferentes da aplicação WebGoat, serão encontrados resultados distintos e talvez melhores.

# 5 CONSIDERAÇÕES FINAIS

Através do estudo da evolução das vulnerabilidades em aplicações Web, tendo como base o projeto OWASP Top 10, foi possível identificar que o maior problema de segurança, enfrentado por este tipo de aplicação, é ocasionado pela falta de validação nos dados de entradas. Por exemplo, dentre os 19 riscos listados em todas as edições do Top 10, seis são riscos associados a este problema. Neste sentido, foram selecionadas três vulnerabilidades para serem exploradas, referentes ao problema de validação de entrada de dados. As vulnerabilidades selecionadas foram Injeção de SQL (A1), *Cross Site Scripting* (A7) e Entidades Externas de XML (A4).

Com relação às ferramentas, foram encontradas 104 ferramentas de código aberto para execução de testes de intrusão em aplicações Web, com foco nas etapas de análise e exploração de vulnerabilidades da metodologia PTES. A vulnerabilidade que possui maior percentual de ferramentas para sua detecção é Injeção de SQL, em segundo lugar encontram-se as ferramentas que identificam diversas vulnerabilidades e em terceiro lugar as ferramentas que identificam *Cross Site Scripting*. Dentre as 104 ferramentas, foram selecionadas 14 para serem aplicadas na execução dos testes de intrusão, a saber: Arachni, Beef, Htcap, IronWASP, Metasploit, Skipfish, SQLMap, Vega, W3af, Wapiti, Wfuzz, XSSer, Xenotix e ZAP.

O alvo selecionado para execução dos testes de intrusão foi a aplicação WebGoat. Através da etapa de reconhecimento do alvo, foi possível identificar que esta aplicação é uma *Single-Page Application*, onde toda camada da apresentação, incluindo o *Menu*, é implementado em JavaScript. A aplicação WebGoat faz uso de AJAX, uma vez que em diversos pontos de entrada se comunica com o servidor utilizando o objeto JavaScript XMLHttpRequest. Os dados retornados pelas requisições efetuadas são formatados em JSON, e para acessar os pontos de entrada da aplicação é preciso se autenticar, sendo autenticação baseada em formulário a forma empregada.

Na execução dos testes de intrusão foram avaliados os seguintes fatores: capacidade de autenticação, capacidade de obter os pontos de entrada da aplicação de forma automática através do componente *crawler*, capacidade de identificar e explorar as vulnerabilidades e capacidade de importar e exportar pontos de entrada de/para outras ferramentas. Os resultados dos testes de intrusão indicaram que um baixo percentual (46%) de ferramentas suportam autenticação baseada em formulário e são capazes de efetuar a autenticação de forma automática, a saber: Arachni, IronWASP, Skipfish, Vega, W3af e ZAP. Com relação ao *crawler*, verificou-se que somente duas ferramentas conseguem obter os pontos de entrada da aplicação WebGoat de forma automática, a saber: Arachni e Htcap. Assim, pode-se concluir que o componente *crawler* das ferramentas não estão preparados para trabalhar com aplicações *Single-Page Application*. O componente de detecção também teve baixo percentual detecção, somente 40% das ferramentas detectaram a vulnerabilidade Injeção de SQL, a saber: IronWASP, Vega, ZAP e SQLmap. com relação à vulnerabilidade *Cross Site Scripting* refletido, somente foi detectada pelas ferramentas ZAP e Xenotix. Nenhuma das ferramentas detectou as vulnerabilidades *Cross Site Scripting* armazenado, *Cross Site Scripting* DOM e Entidades Externas de XML. As ferramentas para explorar *Cross Site Scripting* não foram aplicadas em função da necessidade de instalação de ferramentas JavaScript, as quais não foram instaladas devido validações realizadas pela aplicação WebGoat. com relação à vulnerabilidade Entidade Externas de XML, não foi encontrada, durante a pesquisa, nenhuma ferramenta para explorá-la.

Dentre as vulnerabilidades selecionadas, somente foi possível executar um teste de intrusão completo na vulnerabilidade Injeção de SQL, e para executá-lo foi necessário aplicar

três ferramentas distintas, uma vez que a ferramenta SQLmap, usada para detecção e exploração, não suporta autenticação baseada em formulário e seu componente de *crawler* não obteve os pontos de entrada da aplicação. As outras duas ferramentas aplicadas foram wapiti-getcookie e Htcap. A primeira foi utilizada para obter identificador de sessão, enquanto que a segunda foi utilizada para obter os pontos de entrada da aplicação WebGoat.

Tendo em vista os resultados e as limitações encontradas durante os experimentos, a escolha da aplicação WebGoat pode ser considerada inadequada. É provável que a execução de testes de intrusão em outras aplicações, com implementações e tecnologias diferentes da aplicação WebGoat, encontrem resultados distintos e até melhores.

Os resultados apresentados podem ser utilizados como base para os seguintes trabalhos futuros:

- Execução de testes de intrusão em outras aplicações inseguras que sejam implementadas com tecnologias diferentes da aplicação WebGoat, visando comparar os resultados com os obtidos neste trabalho.
- Aplicação das ferramentas Beef e Xenotix em uma aplicação Web que seja possível realizar a instalação das ferramentas JavaScript para exploração da vulnerabilidade *Cross Site Scripting*.
- Análise das ferramentas com o objetivo de entender o motivo da baixa performance na detecção das vulnerabilidades na aplicação WebGoat.
- Levantamento e aplicação de ferramentas para detecção e exploração da vulnerabilidade Entidades Externas de XML.
- Aplicação das ferramentas utilizadas nos testes de intrusão da vulnerabilidade Injeção de SQL no *framework* OWTF visando possuir uma única interface de execução das ferramentas.
- Estudo da integração das ferramentas via API com o objetivo de criar novas ferramentas utilizando as melhores característica de cada ferramenta.

# REFERÊNCIAS


Abbas Saeed, F. e Abed Elgabar, E. E. (2014). Assessment of open source web application security scanners. *Journal of Theoretical and Applied Information Technology*, 61(2):281–287.

Acunetix (2017). Acunetix vulnerability testing report 2017. `https://www.acunetix.com/blog/articles/acunetix-vulnerability-testing-report-2017`. Acessado em 29/11/2017.

Ali, A. B. M., Shakhatreh, A. Y. I., Abdullah, M. S. e Alostad, J. (2011). SQL-injection vulnerability scanning tool for automatic creation of SQL-injection attacks. *Procedia Computer Science*, 3:453–458.

Antunes, N. e Vieira, M. (2014). Penetration testing for web services. *Computer*, 47(2):30–36.

Antunes, N. e Vieira, M. (2017). Designing vulnerability testing tools for web services: approach, components, and tools. *International Journal of Information Security*, 16(4):435–457.

Arkin, B., Stender, S. e McGraw, G. (2005). Software penetration testing. *IEEE Security and Privacy*, 3(1):84–87.

ARSTechnica (2016). As we speak, teen social site is leaking millions of plaintext passwords. `https://arstechnica.com/information-technology/2016/09/social-hangout-site-for-teens-leaks-millions-of-plaintext-passwords/`. Acessado em 19/07/2018.

Awang, N. F. e Manaf, A. A. (2013). Detecting Vulnerabilities in Web Applications Using Automated Black Box and Manual Penetration Testing. *Communications in Computer and Information Science*, 381 CCIS:230–239.

Bau, J., Bursztein, E., Gupta, D. e Mitchell, J. (2010). State of the art: Automated black-box web application vulnerability testing. *Proceedings - IEEE Symposium on Security and Privacy*, páginas 332–345.

Bau, J., Wang, F., Bursztein, E., Mutchler, P. e Mitchell, J. (2012). Vulnerability Factors in New Web Applications: Audit Tools, Developer Selection & Languages.

BBC (2010). Royal navy website attacked by romanian hacker. `https://www.bbc.co.uk/news/technology-11711478`. Acessado em 18/07/2018.

Blome, A., Ochoa, M., Li, K., Peroli, M. e Dashti, M. T. (2013). VERA: A flexible model-based vulnerability testing tool. *Proceedings - IEEE 6th International Conference on Software Testing, Verification and Validation, ICST 2013*, páginas 471–478.

Booch, G., Rumbaugh, J. e Jacobson, I. (1998). *Unified Modeling Language User Guide*.

Büchler, M., Oudinet, J. e Pretschner, A. (2012a). Semi-automatic security testing of web applications from a secure model. *Proceedings of the 2012 IEEE 6th International Conference on Software Security and Reliability, SERE 2012*, páginas 253–262.

Büchler, M., Oudinet, J. e Pretschner, A. (2012b). SPaCiTE - Web application testing engine. *Proceedings - IEEE 5th International Conference on Software Testing, Verification and Validation, ICST 2012*, páginas 858–859.

Calvi, A. e Viganò, L. (2016). An automated approach for testing the security of web applications against chained attacks. *Proceedings of the 31st Annual ACM Symposium on Applied Computing - SAC '16*, páginas 2095–2102.

CBS (2012). Yahoo reportedly hacked: Is your account safe? `https://www.cbsnews.com/news/yahoo-reportedly-hacked-is-your-account-safe/`. Acessado em 19/07/2018.

Chen, J. M. e Wu, C. L. (2010). An automated vulnerability scanner for injection attack based on injection point. *ICS 2010 - International Computer Symposium*, páginas 113–118.

Chen, S. (2014). The Web Application Vulnerability Scanners Benchmark. (February).

Ciampa, A. (2010). A heuristic-based approach for detecting SQL-injection vulnerabilities in web applications. *Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems - SESS '10*, páginas 43–49.

CNET (2017). Equifax data breach may affect nearly half the us population. `https://www.cnet.com/news/equifax-data-leak-hits-nearly-half-of-the-us-population/`. Acessado em 20/07/2018.

Dashevskyi, S., Dos Santos, D. R., Massacci, F. e Sabetta, A. (2017). TestREx: a framework for repeatable exploits. *International Journal on Software Tools for Technology Transfer*, páginas 1–15.

De Jimenez, R. E. L. (2017). Pentesting on web applications using ethical - Hacking. *2016 IEEE 36th Central American and Panama Convention, CONCAPAN 2016*, (503).

de Meo, F. e Viganò, L. (2017). A formal approach to exploiting multi-stage attacks based on file-system vulnerabilities of web applications. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10379 LNCS:196–212.

Deepa, G., Thilagam, P. S., Khan, F. A., Praseed, A., Pais, A. R. e Palsetia, N. (2018). Black-box detection of XQuery injection and parameter tampering vulnerabilities in web applications. *International Journal of Information Security*, 17(1):105–120.

Dessiatnikoff, A., Akrout, R., Alata, E., Kaaniche, M. e Nicomette, V. (2011). A clustering approach for web vulnerabilities detection. *Proceedings of IEEE Pacific Rim International Symposium on Dependable Computing, PRDC*, páginas 194–203.

Djuric, Z. (2013). A black-box testing tool for detecting SQL injection vulnerabilities. *2013 2nd International Conference on Informatics and Applications, ICIA 2013*, páginas 216–221.

Doup, A., Cova, M. e Vigna, G. (2010). Why Johnny Can't Pentest: An Analysis of Black-Box Web Vulnerability Scanners.

Doupé, A. (2014). Advanced Automated Web Application Vulnerability Analysis. *Ph.D. Dissertation*, (June).

Doupé, A., Cavedon, L., Kruegel, C. e Vigna, G. (2012). Enemy of the State: A State-Aware Black-Box Web Vulnerability Scanner. *USENIX Security Symposium*, páginas 523–538.

Duchene, F. (2015). Detection of Web Vulnerabilities via Model Inference assisted Evolutionary Fuzzing Detection of Web Vulnerabilities via Model Inference assisted Evolutionary Fuzzing.

Duchene, F., Rawat, S., Richier, J. L. e Groz, R. (2013). LigRE: Reverse-engineering of control and data flow models for black-box XSS detection. *Proceedings - Working Conference on Reverse Engineering, WCRE*, páginas 252–261.

Duchene, F., Rawat, S., Richier, J.-L. e Groz, R. (2014). KameleonFuzz. *Proceedings of the 4th ACM conference on Data and application security and privacy - CODASPY '14*, (February):37–48.

El Idrissi, S., Berbiche, N., Guerouate, F. e Sbihi, M. (2017). Performance evaluation of web application security scanners for prevention and protection against vulnerabilities. *International Journal of Applied Engineering Research*, 12(21):11068–11076.

Español, E. (2018). Telefónica tapa un agujero de seguridad que dejó al descubierto millones de datos de sus clientes. `https://www.elespanol.com/economia/empresas/20180716/telefonica-agujero-seguridad-descubierto-millones-datos-clientes/322967853_0.html`. Acessado em 20/07/2018.

Fallis, A. (2013). Black Hat Python, Python Programming for Hackers. *Journal of Chemical Information and Modeling*.

Felderer, M., Büchler, M., Johns, M., Brucker, A. D., Breu, R. e Pretschner, A. (2016). *Security Testing: A Survey*, volume 101.

Felmetsger, V. e Cavedon, L. (2010). Toward automated detection of logic vulnerabilities in web applications. *USENIX Security* . . . .

Ferreira, A. M. e Kleppe, H. (2011). Effectiveness of Automated Application Penetration Testing Tools.

Fink, G. e Flatow, I. (2014). *Pro Single Page Application Development Using Backbone.js and ASP.NET*.

Fung, A. P., Wang, T., Cheung, K. W. e Wong, T. Y. (2014). Scanning of real-world web applications for parameter tampering vulnerabilities. *ACM symposium on Information, computer and communications security*, páginas 341–352.

Giancarlo Pellegrino, Constantin Tschurtz, E. B., Rossow e Rossow, C. (2015). jak: Using Dynamic Analysis to Crawl and Test Modern Web Applications.

Gupta, S. e Sharma, L. (2011). Analysis and Assessment of Web Application Security Testing Tools. páginas 0–1.

Hanqing Wu, L. Z. (2015). *Web Security A WhiteHat Perspective*.

Haubris, K. P. e Pauli, J. J. (2013). Improving the Efficiency and Effectiveness of Penetration Test Automation. *2013 10th International Conference on Information Technology: New Generations*, páginas 387–391.

Hoglund, G. e Mcgraw, G. (2004). *Exploiting Software How to Break Code*.

Htcap (2018). Htcap. https://htcap.org/. Acessado em 03/11/2018.

ICO. (2016). Russian hackers amass over a billion internet passwords. https://ico.org.uk/about-the-ico/news-and-events/news-and-blogs/2016/10/talktalk-gets-record-400-000-fine-for-failing-to-prevent-october-2015-attack/. Acessado em 19/07/2018.

ISO/IEC (2018). INTERNATIONAL STANDARD ISO / IEC Information technology — Security techniques — Information security management systems — Overview and vocabulary. 2018.

Jaswal, N. (2014). *Mastering Metasploit*.

Jingling, Z. e Rulin, G. (2015). A New Framework of Security Vulnerabilities Detection in PHP Web Application. *Proceedings - 2015 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2015*, páginas 271–276.

Jon Erikson (2007). *The Art of Exploitation*.

Kals, S., Kirda, E., Kruegel, C. e Jovanovic, N. (2006). SecuBat: A Web Vulnerability Scanner. *Proceedings of the 15th international conference on World Wide Web - WWW '06*, página 247.

KrebsonSecurity (2016). Crooks grab w-2s from credit bureau equifax. https://krebsonsecurity.com/2016/05/crooks-grab-w-2s-from-credit-bureau-equifax/. Acessado em 18/07/2018.

Li, X., Si, X. e Xue, Y. (2014a). Automated black-box detection of access control vulnerabilities in web applications. *Codaspy*, páginas 49–60.

Li, X. e Xue, Y. (2011). BLOCK: a black-box approach for detection of state violation attacks towards web applications. *Proceedings of the 27th Annual Computer Security Applications Conference*, páginas 247–256.

Li, X. e Xue, Y. (2013). LogicScope: automatic discovery of logic vulnerabilities within web applications. *Proceedings of the 8th ACM SIGSAC symposium on . . .* , páginas 481–486.

Li, X. e Xue, Y. (2014). A survey on server-side approaches to securing web applications. *ACM Computing Surveys*, 46(4):1–29.

Li, Y. F., Das, P. K. e Dowe, D. L. (2014b). Two decades of Web application testing - A survey of recent advances. *Information Systems*, 43:20–54.

MacKeeper (2016). Breaking: Massive breach of mexican voter data. https://mackeeper.com/blog/post/217-breaking-massive-data-breach-of-mexican-voter-data. Acessado em 20/07/2018.

MACNN (2011). Lulzsec hacks sony pictures, reveals 1m passwords unguarded. http://www.macnn.com/articles/11/06/02/lulz.security.hits.sony.again.in.security.message/. Acessado em 18/07/2018.

Mainka, C., Somorovsky, J. e Schwenk, J. (2012). Penetration testing tool for web services security. *Proceedings - 2012 IEEE 8th World Congress on Services, SERVICES 2012*, páginas 163–170.

Makino, Y. e Klyuev, V. (2015). Evaluation of web vulnerability scanners. *Proceedings of the 2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2015*, 1(September):399–402.

Martirosyan, Y. (2012). Security Evaluation of Web Application Vulnerability Scanners Strengths and Limitations Using Custom Web Application.

McPhee, M. (2017). *Mastering Kali Linux for Web Penetration Testing*.

Mcquade, K. (2014). Open Source Web Vulnerability Scanners : The Cost Effective Choice? *2014 Proceedings of the Conference for Information Systems Applied Research*, (November):1–13.

Mirjalili, M., Nowroozi, A. e Alidoosti, M. (2014). A survey on web penetration test. *ACSIJ Advances in Computer Science*, 3(6):107–121.

Mukhopadhyay, I., Goswami, S. e Mandal, E. (2014). Web Penetration Testing using Nessus and Metasploit Tool. *IOSR Journal of Computer Engineering*, 16(3):126–129.

Muniz, J. e Lakhani, A. (2013). *Web Penetration Testing with Kali Linux*.

Nájera-Gutiérrez, G. (2016). *Kali Linux Web Penetration Testing Cookbook*.

Obermaier, F. (2016). About the panama papers. `https://panamapapers.sueddeutsche.de/articles/56febff0a1bb8d3c3495adf4/`. Acessado em 20/07/2018.

Olston, C. e Najork, M. (2010). Web Crawling. 4.

OWASP (2004). OWASP Top 10 - 2004 The Ten Most Critical Web Application Security Risks.

OWASP (2007). OWASP Top 10 - 2007 The Ten Most Critical Web Application Security Risks.

OWASP (2010). OWASP Top 10 - 2010 The Ten Most Critical Web Application Security Risks.

OWASP (2013). OWASP Top 10 - 2013 The Ten Most Critical Web Application Security Risks.

OWASP (2014). *OWASP Testing Guide 4.0*.

OWASP (2017). OWASP Top 10 - 2017 The Ten Most Critical Web Application Security Risks.

OWASP (2018). Webgoat. `https://github.com/WebGoat/WebGoat`. Acessado em 01/10/2018.

Pellegrino, G. e Balzarotti, D. (2014). Toward Black-Box Detection of Logic Flaws in Web Applications. *Network and Distributed System Security Symposium (NDSS)*, (February):23–26.

Pellegrino, G., Johns, M., Koch, S., Backes, M. e Rossow, C. (2017). Deemon: Detecting CSRF with Dynamic Analysis and Property Graphs.

Potter, B. e McGraw, G. (2004). Software security testing. *Security & Privacy, IEEE*, 2(5):81–85.

Prasad, P. (2016). *Mastering Modern Web Penetration Testing*.

PTES (2012). Penetration testing execution standard. `http://www.pentest-standard.org`. Acessado em 28/05/2018.

Román Muñoz, F., Sabido Cortes, I. I. e García Villalba, L. J. (2017). Enlargement of vulnerable web applications for testing. *Journal of Supercomputing*, páginas 1–20.

RT (2016). Millions of german airline passengers' data exposed to security gaps for years. https://www.rt.com/news/354191-german-airline-passengers-data/. Acessado em 18/07/2018.

Saeed, F. A. (2014). Using WASSEC to Evaluate Commercial Web Application Security Scanners. (1):177–181.

Scambray, J., Shema, M. e Sima, C. (2006). *Hacking Exposed Web Applications, 2nd Ed. (Hacking Exposed)*.

SCmagazine (2017). Millions of dow jones customer records exposed due an internal error. https://www.scmagazine.com/millions-of-dow-jones-customer-records-exposed-due-an-internal-error/article/675843/. Acessado em 20/07/2018.

Shar, L. K., Briand, L. C. e Tan, H. B. K. (2015). Web Application Vulnerability Prediction Using Hybrid Program Analysis and Machine Learning. *IEEE Transactions on Dependable and Secure Computing*, 12(6):688–707.

Shelly, D. A. (2010). Using a Web Server Test Bed to Analyze the Limitations of Web Application Vulnerability Scanners. (March).

Softpedia (2013). Redhack breaches istanbul administration site, hackers claim to have erased debts. https://news.softpedia.com/news/RedHack-Breaches-Istanbul-Administration-Site-Hackers-Claim-to-Have-Erased-Debts-364000.shtml. Acessado em 19/07/2018.

Sommerville, I. (2011). *Software Engineering*.

SPaCIoS (2011). SPaCIoS Tool mock up , Technology survey , Validation methodology patterns v . 1 Abstract Deliverable details. 4:1–38.

Stephens, J. C. (2017). Application Security Statistics Report. The case for DevSecOps. páginas 69–77.

Steve Christey, R. A. M. (2007). Vulnerability type distributions in cve. http://cwe.mitre.org/documents/vuln-trends/index.html. Acessado em 16/07/2018.

Stuttard, D. e Pinto, M. (2011). *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws, Second Edition*.

Subgraph (2018). Vega vulnerability scanner. https://subgraph.com/vega/. Acessado em 01/10/2018.

Suteva, N., Zlatkovski, D. e Mileva, A. (2013). Evaluation and Testing of Several Free / Open Source Web. (Ciit):221–224.

Suto, L. (2010). Analyzing the Accuracy and Time Costs of Web Application Security Scanners. *San Francisco, February*, (October 2007):20.

SWEBOK (2004). *Guide to the Software Engineering Body of Knowledge - SWEBOK*.

Tatli, E. I. e Urgun, B. (2017). WIVET - Benchmarking Coverage Qualities of Web Crawlers. *Computer Journal*, 60(4):555–572.

Theverge (2017). Verizon partner data breach exposes millions of customer records. https://www.theverge.com/2017/7/12/15962520/verizon-nice-systems-data-breach-exposes-millions-customer-records. Acessado em 20/07/2018.

Times, T. N. Y. (2014). Russian hackers amass over a billion internet passwords. https://www.nytimes.com/2014/08/06/technology/russian-gang-said-to-amass-more-than-a-billion-stolen-internet-credentials.html?_r=0. Acessado em 19/07/2018.

Uygur, S. (2014). *Penetration Testing with BackBox*.

van der Loo, F. (2011). Comparison of penetration testing tools for web applications. página 50.

Vega, Esteban Alejandro Armas, O. A. L. S. e Villalba (2017). Benchmarking of Pentesting Tools. 11.

Vieira, M., Antunes, N. e Madeira, H. (2009). Using Web Security Scanners to Detect Vulnerabilities in Web Services. *Word Journal Of The International Linguistic Association*, páginas 566–571.

Wapiti (2018). Wapiti. wapiti.sourceforge.net. Acessado em 17/10/2018.

WAVSEP (2014). The wivet score of web application scanners. http://sectoolmarket.com/wivet-score-of-open-source-web-application-scanners.html. Acessado em 10/04/2018.

Wazlawick, R. S. (2013). *Engenharia de Software : conceitos e práticas*.

Weissbacher, M., Robertson, W., Kirda, E., Kruegel, C. e Vigna, G. (2015). ZigZag: Automatically Hardening Web Applications Against Client-side Validation Vulnerabilities. *USENIX Security*, páginas 737–752.

Wfuzz (2018). Wfuzz: The web fuzzer. https://wfuzz.readthedocs.io/en/latest/. Acessado em 15/10/2018.

Xenotix (2018). Xenotix. https://xenotix.in/. Acessado em 05/11/2018.

Xiong, P. e Peyton, L. (2010). A model-driven penetration test framework for Web applications. *Privacy Security and Trust (PST), 2010 . . .* , páginas 173–180.

XSSer (2018). Xsser. https://xsser.03c8.net/. Acessado em 05/09/2018.

Zech, P., Felderer, M. e Breu, R. (2017). Knowledge-based security testing of web applications by logic programming. *International Journal on Software Tools for Technology Transfer*, páginas 1–26.

Zhou, Y. e Evans, D. (2014). SSOScan: Automated Testing of Web Applications for Single Sign-On Vulnerabilities. *23rd USENIX Security Symposium (USENIX Security 14)*, páginas 495–510.

# APÊNDICE A – LISTA DE ATIVIDADES PARA TESTES DE INTRUSÃO EM APLICAÇÃO WEB PROPOSTAS PELO GUIA DE TESTES DO OWASP

A Tabela A.1 contém o objetivo de cada uma das noventa e uma (91) atividades que compõe o Guia de Testes do OWASP (OWASP, 2014), bem como, as ferramentas que podem ser utilizadas.

Tabela A.1: Atividades para testes de intrusão em aplicação Web do Guia de Testes do OWASP. Fonte: O autor baseado em OWASP (2014)

| **Categoria** | **Atividade** | **Objetivo** | **Ferramenta** |
|---|---|---|---|
| Reconhecimento | Conduzir reconhecimento de mecanismos de pesquisa para evitar vazamento de informações | Entender quais informações confidenciais de projeto e configuração da aplicação estão expostas diretamente (no *site* da organização) ou indiretamente (em um *site* de terceiros). | Google Hacker, SiteDigger, PunkSpider |
| Reconhecimento | Identificar o servidor Web | Identificar a versão e o tipo do servidor Web em execução, para determinar as vulnerabilidades conhecidas e as explorações apropriadas a serem usadas durante os testes. | HTTPrint, HTTPrecon, Netcraft, Desenmascarame |
| Reconhecimento | Revisar os arquivos do servidor Web para evitar vazamento de informações | Encontrar vazamento de informações no diretório ou pasta da aplicação da Web. Criar lista de diretórios que devem ser evitados por *Spiders*, *Robots* ou *Crawlers*. | Browser, curl, wget, rockspider |
| Reconhecimento | Enumerar as aplicações do servidor Web | Identificar as aplicações que são executadas no mesmo servidor Web onde a aplicação é executada | DNS lookup, Máquinas de pesquisas, Nmap, Nessus, Nikto |
| Reconhecimento | Revisar os comentários das páginas Web da aplicação em busca de vazamento de informações | Revisar os comentários e metadados de páginas da Web para entender melhor a aplicação e encontrar vazamento de informações | Wget, Browser, Eyeballs,Curl |
| Reconhecimento | Identificar as funcionalidades e pontos de entrada da aplicação Web | Entender as requisições e as respostas da aplicação | ZAP, WebScarab, Burp Suite, CAT |
| Reconhecimento | Mapear os caminhos de execução da aplicação Web | Mapear a aplicação Web e entender os fluxos de trabalho | ZAP |

**Tabela A.1 continuação da página anterior**

| Categoria | Atividade | Objetivo | Ferramenta |
|---|---|---|---|
| Reconhecimento | Identificar tecnologias e *frameworks* da aplicação Web | Identificar o *framework* Web utilizado pela aplicação | WhatWeb, BlindElephant, Wappalyzer |
| Reconhecimento | Identificar a aplicação Web | Identificar a aplicação Web e sua versão para determinar as vulnerabilidades conhecidas e as explorações apropriadas a serem usadas durante o testes | WhatWeb, BlindElephant, Wappalyzer |
| Reconhecimento | Mapear a arquitetura da aplicação Web | Identificar quais são os componentes da aplicação Web | |
| Gerenciamento de configuração | Testar a configuração de rede/infraestrutura | Revisar toda infraestrutura garantindo que não contenha vulnerabilidades conhecidas | Nessus, OpenVAS |
| Gerenciamento de configuração | Testar a configuração da plataforma da aplicação Web | Garantir que as configurações estejam corretas evitando problemas de segurança na arquitetura | |
| Gerenciamento de configuração | Testar extensões de arquivos para evitar vazamento de informações | Evitar que o atacante tenha acesso aos arquivos da aplicação e obtenha conhecimento sobre ela | Scanner de aplicação |
| Gerenciamento de configuração | Revisar arquivos de backup antigos para evitar vazamento de informações | Evitar que o atacante tenha acesso a arquivos antigos e obtenha conhecimento sobre a aplicação | Scanner de aplicação , wget |
| Gerenciamento de configuração | Enumerar interfaces de administração de infraestrutura da aplicação | Garantir que as interfaces de administração da aplicação não sejam acessadas por usuários não autorizados | Dirbuster, THC-HYDRA |
| Gerenciamento de configuração | Testar métodos HTTP | Evitar que atacantes utilizem os métodos HTTP de forma maliciosa | NetCat, cURL |
| Gerenciamento de configuração | Testar segurança de transporte HTTP | Evitar que as informações da aplicação trafeguem por canais inseguros | *Proxy* de aplicação, cURL |
| Gerenciamento de configuração | Testar políticas de domínio | Garantir que os arquivos contendo as políticas estejam mal configurados | Nikto, ZAP, W3af |
| Gerenciamento de identidade | Testar as *roles* de segurança | Validar as *roles* de segurança garantindo que as funções e informação da aplicação sejam acessadas em conformidade com elas | |
| Gerenciamento de identidade | Testar o registro de usuários | Verificar se os requisitos de segurança definidos no registro de usuário foram implementados corretamente | *Proxy* de aplicação |
| Gerenciamento de identidade | Testar contas privilegiadas | Verificar quais os tipos de contas que podem criar novas contas | *Proxy* de aplicação |

**Tabela A.1 continuação da página anterior**

| Categoria | Atividade | Objetivo | Ferramenta |
|---|---|---|---|
| Gerenciamento de identidade | Testes para enumeração de contas e contas anônimas | Verificar se é possível descobrir nomes de usuários interagindo com o mecanismo de autenticação | WebScarab, cURL |
| Gerenciamento de identidade | Testar política de nome de usuário | Verificar se as políticas de nome de usuário e mensagem retornado pela aplicação permitem descobrir usuários | |
| Autenticação | Testar credenciais transportadas por canal criptografado | Verificar se as credenciais dos usuários são trafegadas por um canal seguro | WebScarab, ZAP |
| Autenticação | Testar por credenciais padrões | Garantir que a aplicação não contenha credenciais padrões | Burp Intruder, THC Hydra, Brutus, Nikto 2 |
| Autenticação | Testar mecanismos de bloqueio | Validar se o mecanismo de autenticação tem habilidade de não permitir força bruta de senhas | |
| Autenticação | Testar *bypass* de autenticação | Garantir que a autenticação não seja contornada e o usuário obtenha acesso ilegal | WebScarab, ZAP |
| Autenticação | Testar funcionalidade de relembrar usuário | Validar se está implementada de forma correta garantindo que o atacante não obtenha a senha através do navegador | |
| Autenticação | Testar *cache* do navegador | Garantir que o navegador não armazene informações confidenciais | ZAP, Firefox add-on CacheViewer2 |
| Autenticação | Testar políticas de senha | Determinar a resistência da aplicação contra força bruta de senha usando dicionários de senha disponíveis, avaliando os requisitos de tamanho, complexidade, reutilização e envelhecimento | |
| Autenticação | Testar perguntas de segurança | Garantir que as perguntas de segurança não permitam aos atacantes obterem acesso às senhas dos demais usuários | |
| Autenticação | Testar troca de senha | Garantir que um usuário não altere a senha de uma outra conta | |
| Autenticação | Testar autenticação em outros canais | Identificar e validar todas as funcionalidades de autenticação da aplicação | |

**Tabela A.1 continuação da página anterior**

| Categoria | Atividade | Objetivo | Ferramenta |
|---|---|---|---|
| Autorização | Testar *directory traversal/file include* | Garantir que um agressor explore o sistema para ler ou gravar arquivos que não devem ser acessados | DotDotPwn, WFuzz, *Proxy* de aplicação |
| Autorização | Testar *bypass* de autorização | Verificar como o esquema de autorização foi implementado para cada função ou privilégio, evitando obter acesso a funções e recursos não autorizados | WebScarab, ZAP |
| Autorização | Testar escalação de privilégio | Garantir que um usuário não modifique seus privilégios na aplicação | WebScarab, ZAP |
| Autorização | Testar *insecure direct object references* | Garantir que os atacantes não tenham acesso a arquivos e recursos da aplicação, contornando o mecanismo de autorização | |
| Gerenciamento de sessão | Testar o gerenciamento de sessão | Verificar se os *cookies* e outros *tokens* de sessão são criados de maneira segura | ZAP, Burp Sequencer, Foundstone |
| Gerenciamento de sessão | Testar *cookies* | Garantir que os *cookies* são implementados de maneira segura | ZAP |
| Gerenciamento de sessão | Testar *session fixation* | Validar se a aplicação renova o *cookies* de sessão após autenticação | Hijack, WebScarab |
| Gerenciamento de sessão | Testar variáveis de sessão | Garantir que as informações de sessão sejam enviadas ao cliente por canal seguro | |
| Gerenciamento de sessão | Testar *Cross Site Request Forgery* | Validar se a aplicação não é vulnerável a CSRF | CSRF Tester, Cross Site Requester |
| Gerenciamento de sessão | Testar *logout* | Garantir que a aplicação encerre a sessão de usuário corretamente | Burp Suite - Repeater |
| Gerenciamento de sessão | Testar *timeout* de sessão | Garantir que a aplicação finalize automaticamente uma sessão de usuário inativo | |
| Gerenciamento de sessão | Testar sobrecarga de variáveis de sessão | Validar se a aplicação permite alterar valores das variáveis de sessão através do cliente | |
| Validação de entrada de dados | Testar *cross site script* refletido | Verificar se a aplicação é vulnerável a XSS refletido | ZAP, Xenotix |
| Validação de entrada de dados | Testar *cross site script* armazenado | Verificar se a aplicação é vulnerável a XSS armazenado | ZAP, BeEF |
| Validação de entrada de dados | Testar HTTP *Verb Tampering* | Validar se a aplicação permite alterar os métodos HTTP da requisição original | |

**Tabela A.1 continuação da página anterior**

| Categoria | Atividade | Objetivo | Ferramenta |
|---|---|---|---|
| Validação de entrada de dados | Testar HTTP *Parameter pollution* | Garantir que a aplicação não permita múltiplos valores para o mesmo parâmetro uma mesma requisição | ZAP |
| Validação de entrada de dados | Testar *SQL Injection* | Verificar se a aplicação permite executar comandos SQL via requisição HTTP | ZAP, SQL Power Injector, SQLMap |
| Validação de entrada de dados | Testar *LDAP Injection* | Verificar se a aplicação permite executar comandos LDAP via requisição HTTP | ZAP, Softerra LDAP Browser |
| Validação de entrada de dados | Testar *ORM Injection* | Verificar se a aplicação permite executar comandos SQL via requisição HTTP em aplicação que utilize mapeamento objeto-relacional | ZAP |
| Validação de entrada de dados | Testar *XML Injection* | Validar se a aplicação permite executar comandos do sistema operacional via *upload* de arquivo XML | ZAP, WFuzz |
| Validação de entrada de dados | Testar *SSI Injection* | Validar se é possível executar comandos no sistema operacional através de páginas HTML | ZAP |
| Validação de entrada de dados | Testar *XPath Injection* | Verificar se a aplicação permite executar comandos XPath via requisição HTTP | ZAP |
| Validação de entrada de dados | Testar *IMAP/SMTP Injection* | Validar se é possível executar comandos no serviço de e-mail atráves de requisição HTTP | W3af |
| Validação de entrada de dados | Testar *local file inclusion* | Garantir que não seja possível executar arquivos presentes do servidor Web através da requisição HTTP | ZAP |
| Validação de entrada de dados | Testar *remote file inclusion* | Garantir que não seja possível executar arquivos remotos no servidor Web através de requisição HTTP | ZAP |
| Validação de entrada de dados | Testar *command injection* | Garantir que não seja possível executar comandos do sistema operacional através de requisição HTTP | ZAP |
| Validação de entrada de dados | Testar *buffer overflow* | Verificar se a aplicação é vulnerável a *buffer overflow* | ZAP |
| Validação de entrada de dados | Testar *heap overflow* | Verificar se a aplicação é vulnerável a *heap overflow* | Spike, OllyDbg |
| Validação de entrada de dados | Testar *stack overflow* | Verificar se a aplicação é vulnerável a *stack overflow* | OllyDbg |
| Validação de entrada de dados | Testar *format string* | Validar se é possível indisponibilizar a aplicação através de textos enviados via requisição HTTP | ZAP |

**Tabela A.1 continuação da página anterior**

| Categoria | Atividade | Objetivo | Ferramenta |
|---|---|---|---|
| Validação de entrada de dados | Testar vulnerabilidade incubadas | Verificar se a aplicação possui ataques armazenados | Metasploit |
| Validação de entrada de dados | Testar *HTTP Splitting/Smuggling* | Validar se é possível atacar a aplicação através do envio de caracteres especiais no cabeçalho da requisição HTTP | |
| Erros da aplicação | Analisar os códigos de erro | Verificar se os códigos de erro retornados pela aplicação podem facilitar seu conhecimento | ZAP |
| Erros da aplicação | Analisar os *stack traces* | Verificar se as mensagens de erros retornados pela aplicação podem facilitar seu conhecimento | ZAP |
| Criptografia | Testar por proteção insuficiente de canal de transporte | Verificar se dados confidenciais estão sendo transmitidos por canal inseguro | |
| Criptografia | Testar *padding oracle* | Verificar se dados confidenciais, criptografados pela aplicação e enviados ao cliente, podem ser acessados por atacantes | ZAP, PadBuster, Poracle |
| Criptografia | Testar por dados sensíveis sobre canais sem criptografia | Verificar se dados confidenciais são transmitidos por canais seguros | curl |
| Lógica de negócio | Testar lógica de validação de dados | Verificar se a aplicação permite inserção de dados sem validação | ZAP |
| Lógica de negócio | Testar requisições forjadas | Verificar se a aplicação permite ao usuário submter dados para algum componente da aplicação que ele não poderia ter acesso | ZAP |
| Lógica de negócio | Testar integridade de dados | Verificar se a aplicação permite que os usuários destruam a integridade de qualquer parte da aplicação ou dados | ZAP |
| Lógica de negócio | Testar tempo de transações | Verificar se a aplicação possui transações longas que podem ser manipuladas | |
| Lógica de negócio | Testar número de vezes que uma função pode ser chamada | Verificar se a aplicação permite que os usuários utilizem suas funções mais vezes do que o exigido pela lógica de negócios | *Proxy* de aplicação |
| Lógica de negócio | Testar *workflows* | Verificar se a aplicação permite que os usuários executem ações não permitidas pelo *workflow* | *Proxy* de aplicação |
| Lógica de negócio | Testar a aplicação contra uso indevido | Verificar se a aplicação permite que os usuários manipulem suas funções de maneira indevida | *Proxy* de aplicação |

**Tabela A.1 continuação da página anterior**

| Categoria | Atividade | Objetivo | Ferramenta |
|---|---|---|---|
| Lógica de negócio | Testar funcionalidade de *upload* com arquivos inesperados | Verificar se aplicação permite *upload* de arquivos não suportados | Metasploit |
| Lógica de negócio | Testar funcionalidade de *upload* com arquivos maliciosos | Verificar se aplicação permite *upload* de arquivos mal intencionados | *Proxy* de aplicação |
| Lado cliente | Testar DOM *cross site script* | Verificar se a aplicação é vulnerável a DOM XSS | Ironwasp, DOMinator |
| Lado cliente | Testar execução de *JavaScript* | Validar se a aplicação permite injetar código *JavaScript* no lado cliente | |
| Lado cliente | Testar *HTML injection* | Validar se a aplicação permite injetar código HTML no lado cliente | |
| Lado cliente | Testar redirecionamento de URL | Validar se a aplicação permite redirecionamento de URL no lado cliente | DOMinator |
| Lado cliente | Testar injeção de CSS | Validar se a aplicação permite injetar código CSS no lado cliente | |
| Lado cliente | Testar manipulação de recursos | Verificar se a aplicação permite manipular URL de recursos no lado cliente | DOMinator |
| Lado cliente | Testar *cross origin resource sharing* | Verificar se a aplicação permite alterar informação de CORS do cabeçalho da requisição no lado cliente | ZAP |
| Lado cliente | Testar *cross site flashing* | Verificar vulnerabilidades em implementações em *ActionScript - Flash* | SWFScan |
| Lado cliente | Testar *clickjacking* | Verificar se a aplicação permite direcionar o usuário para páginas maliciosas, capturando suas informações | Clickjacking Tool |
| Lado cliente | Testar *websockets* | Verificar vulnerabilidades em aplicações que utilizem *WebSocket* | ZAP |
| Lado cliente | Testar Web *messaging* | Verificar vulnerabilidades em aplicações que utilizem Web *messaging* | ZAP |
| Lado cliente | Testar *local storage* | Verificar se a aplicação armazena dados confidenciais no lado cliente | Firebug, ZAP |

# APÊNDICE B – LISTA DE FERRAMENTAS

A Tabela B.1 contém a listagem completa das 104 ferramentas identificadas. A primeira coluna da tabela contém o nome da ferramenta, a segunda a linguagem na qual foi construída, a terceira a etapa da metodologia de testes de intrusão em que é aplicada, a quarta coluna as vulnerabilidades que a ferramenta identifica e/ou explora e a última a URL onde pode ser encontrada a ferramenta. A tabela está em ordem alfabética pela coluna "Ferramenta". A coluna "Etapa metodologia" pode conter os seguintes valores: "Análise vulnerabilidade", "Exploração", "Análise e exploração vulnerabilidade"e "Diversas (*Framework*)". "Análise vulnerabilidade" significa que a ferramenta é aplicada somente na etapa de análise de vulnerabilidades. "Exploração" significa que a ferramenta é aplicada somente na etapa exploração. "Análise e exploração vulnerabilidade" significa que a ferramenta é aplicada nas etapas de análise de vulnerabilidade e exploração. "Diversas (*Framework*)" significa que a ferramenta é um *framework*, composta por outras ferramentas e que dependendo da configuração realizada pode atuar em uma ou mais etapas da metodologia. A coluna "Vulnerabilidade" pode conter os seguintes valores: "Nome da vulnerabilidade", "Diversas", "Diversas (*Web Server*)", "Diversas (*Framework*)", "Diversas (WS SOAP)" e "Diversas (CMS)". "Diversas" significa que a ferramenta identifica e/ou explora mais de três vulnerabilidades. Caso identifique e/ou explore menos de três, o nome de cada vulnerabilidade será apresentado. "Diversas (*Web Server*)" significa que a ferramenta identifica e/ou explora mais de três vulnerabilidades relacionadas ao servidor Web. "Diversas (*Framework*)" significa que a ferramenta é um *framework*, composta por outras ferramentas e que dependendo da configuração realizada pode identificar e/ou explorar mais de três vulnerabilidades. "Diversas (WS SOAP)" significa que a ferramenta identifica e/ou explora mais de três vulnerabilidades relacionadas à *Web Services SOAP*. "Diversas (CMS)" significa que a ferramenta identifica e/ou explora mais de três vulnerabilidades em sistema de gerenciamento de conteúdo, por exemplo, WordPress e phpBB.

A Tabela B.2 contém a listagem de todas as referências utilizadas no levantamento das ferramentas. A primeira coluna da tabela contém o título da referência. A tabela está em ordem alfabética pela coluna "Título".

Tabela B.1: Ferramentas. Fonte: O autor (2019)

| **Ferramenta** | **Linguagem** | **Etapa metodologia** | **Vulnerabilidade** | **URL** |
|---|---|---|---|---|
| 2gwvs | C | Análise vulnerabilidades | XSS, SQLI | https://code.google.com/archive/p/2gwvs |
| Absinthe | C | Análise vulnerabilidades | SQLI | https://sourceforge.net/projects/absinthe/ |
| Arachni | Ruby | Análise vulnerabilidade | Diversas | http://www.arachni-scanner.com/ |
| Automagic | Perl | Análise e exploração vulnerabilidade | SQLI | http://www.securiteam.com/tools/6P00L0AEKQ.html |
| BBQSQL | Python | Análise e exploração vulnerabilidade | SQLI | https://github.com/Neohapsis/bbqsql/ |
| BeEF | Ruby | Exploração | XSS | http://beefproject.com/ |
| Blind SQLI | Python | Análise e exploração vulnerabilidade | SQLI | https://github.com/awnumar/blind-sql-bitshifting |
| Blisqy | Python | Análise e exploração vulnerabilidade | SQLI | https://github.com/JohnTroony/Blisqy |
| BobCat | .NET | Análise e exploração vulnerabilidade | SQLI | https://www.darknet.org.uk/2006/10/bobcat-sql-injection-tool-based-on-data-thief/ |
| Bsqlbf | Perl | Análise e exploração vulnerabilidade | SQLI | https://github.com/Neohapsis/bbqsql/ |
| CrappScan | .NET | Análise vulnerabilidade | Diversas | https://sourceforge.net/projects/crappscan/?source=directory |
| DAVTest | Perl | Análise e exploração vulnerabilidade | Diversas (*Web Server*) | https://code.google.com/archive/p/davtest/ |
| DIRB | C | Análise vulnerabilidade | Diversas (*Web Server*) | https://sourceforge.net/projects/dirb/?source=navbar |
| DirBuster | Java | Análise vulnerabilidade | Brute Force Dir., Files | https://www.owasp.org/index.php/Category:OWASP_DirBuster_Project |
| DOMinator | Firefox Extension | Análise vulnerabilidade | DOM XSS | https://code.google.com/ archive/p/dominator/ |

**Tabela B.1 continuação da página anterior**

| Ferramenta | Linguagem | Etapa metodologia | Vulnerabilidade | URL |
|---|---|---|---|---|
| DotDotPwn | Perl | Análise vulnerabilidade | Directory Traversal | https://github.com/wireghoul/dotdotpwn |
| DSSS | Python | Análise vulnerabilidade | SQLI | https://github.com/stamparm/DSSS |
| ExploitMyUnion | Python | Análise e exploração vulnerabilidade | SQLI | https://sourceforge.net/projects/ exploitmyunion/ |
| FG-Injector | C | Análise e exploração vulnerabilidade | SQLI | https://sourceforge.net/projects/injection-fwk/?source=navbar |
| fimap | Python | Análise e exploração vulnerabilidade | LFI, RFI | https://tha-imax.de/git/root/fimap |
| Finddler XSRF | .NET | Análise vulnerabilidade | XSRF | https://www.autosectools.com/Fiddler-XSRF-Inspector-Quick-Start |
| Finddler XSS | .NET | Análise vulnerabilidade | XSS | https://www.autosectools.com/Fiddler-XSRF-Inspector-Quick-Start |
| Gamja | Perl | Análise vulnerabilidade | XSS, SQLI | https://sourceforge.net/projects/gamja/ |
| Gobuster | Go | Análise vulnerabilidade | Brute Force Dir., Files | https://github.com/OJ/gobuster |
| Grabber | Python | Análise vulnerabilidade | Diversas | http://rgaucher.info/beta/grabber/ |
| Grendel-Scan | Java | Análise vulnerabilidade | Diversas | https://sourceforge.net/projects/grendel/ |
| HexDorker | PHP | Análise vulnerabilidade | SQLI | https://sourceforge.net/projects/hexdorker/ |
| Hexjector | PHP | Análise e exploração vulnerabilidade | SQLI | https://sourceforge.net/projects/hexjector/ |
| Hscan | .NET | Análise vulnerabilidade | Diversas | https://sourceforge.net/projects/hscan/ ?source=directory |
| Htcap | Python | Análise vulnerabilidade | Diversas | https://htcap.org/ |
| HTTP DT Scanner | .NET | Análise vulnerabilidade | Directory Traversal | https://www.autosectools.com/HTTP-Directory-Traversal-Scanner |
| Immunio XSS Fuzzer | Python | Análise vulnerabilidade | XSS | https://github.com/immunio/immunio-xss-fuzzer |

**Tabela B.1 continuação da página anterior**

| Ferramenta | Linguagem | Etapa metodologia | Vulnerabilidade | URL |
|---|---|---|---|---|
| IronWASP | .NET | Análise vulnerabilidade | Diversas | https://ironwasp.org/ |
| JAEK | Python | Análise vulnerabilidade | XSS | https://github.com/ConstantinT/jAEk |
| Joomscan | Perl | Análise vulnerabilidade | Diversas (CMS) | http://www.vulnerabilityassessment.co.uk/joomsq.htm |
| joomsq | Python | Análise vulnerabilidade | Diversas (CMS) | http://www.vulnerabilityassessment.co.uk/joomsq.htm |
| JSQLInjecton | Java | Análise e exploração vulnerabilidade | SQLI | https://github.com/ron190/jsql-injection/ |
| KayRa | Java | Análise vulnerabilidade | Diversas | https://sourceforge.net/projects/kayra/ |
| LFI Scanner | Perl | Análise vulnerabilidade | LFI | https://packetstormsecurity.com/files/view/102848/lfi-scanner-ver4.0.pl.txt |
| lfimap | Python | Análise e exploração vulnerabilidade | LFI | https://code.google.com/archive/p/lfimap/ |
| LoverBoy | .NET | Análise e exploração vulnerabilidade | SQLI | https://sourceforge.net/projects/loverboy/?source=directory |
| Metasploit | Ruby, Perl | Diversas (*Framework*) | Diversas (*Framework*) | https://metasploit.com/ |
| MultiInjector | Python | Exploração | SQLI | https://github.com/nosperantos/ MultInjector |
| MySqlat0r | Java | Análise e exploração vulnerabilidade | SQLI | https://www.scrt.ch/en/attack/downloads/mini-mysqlat0r |
| Nikto | Perl | Análise vulnerabilidade | Diversas (*Web Server*) | https://www.cirt.net/Nikto2 |
| NoSQLMap | Python | Análise e exploração vulnerabilidade | NoSQL Injection | https://www.kitploit.com/2016/02/nosqlmap-v06-automated-nosql-database.html |
| OpenVAS | C e Python | Diversas (*Framework*) | Diversas (*Framework*) | http://www.openvas.org/ |

**Tabela B.1 continuação da página anterior**

| Ferramenta | Linguagem | Etapa metodologia | Vulnerabilidade | URL |
|---|---|---|---|---|
| OWTF | Python | Diversas (*Framework*) | Diversas (*Framework*) | https://owtf.github.io/ |
| PadBuster | Perl | Análise vulnerabilidade | Padding Oracle | https://github.com/GDSSecurity/PadBuster |
| Pangolin | .NET | Análise e exploração vulnerabilidade | SQLI | https://www.darknet.org.uk/2009/05/pangolin-automatic-sql-injection-tool/ |
| PHP-Injector | PHP | Análise vulnerabilidade | SQLI | https://sourceforge.net/projects/php-injector/ |
| phpBB | Perl | Análise vulnerabilidade | Diversas (CMS) | https://packetstormsecurity.com/files/view/95146/phpbbrfi-scanner.txt |
| Powerfuzzer | Python | Análise vulnerabilidade | Diversas | https://www.powerfuzzer.com/ |
| RABBIT | Python | Análise vulnerabilidade | Diversas | https://sourceforge.net/projects/rabbit-vs/ |
| Safe3 | .NET | Análise vulnerabilidade | Diversas | https://sourceforge.net/projects/safe3wvs/ |
| Safe3 SQL Injector | .NET | Análise e exploração vulnerabilidade | SQLI | https://sourceforge.net/projects/safe3si/ |
| SecuBat | .NET | Análise vulnerabilidade | XSS, SQLI | https://sourceforge.net/projects/secubat/?source=directory |
| Shedu | .NET | Análise vulnerabilidade | Diversas | https://sourceforge.net/projects/shedu/ |
| SixFu | Python | Análise vulnerabilidade | XSS | https://www.evilfingers.com/tools/SiXFu.php |
| SkipFish | C | Análise vulnerabilidade | Diversas | https://code.google.com/archive/p/skipfish/ |
| SnappingTurtle | Python | Exploração | LFI, SQLI, XSS | https://www.autosectools.com/Web-Exploitation-Tool |
| Space Monkey | C | Análise vulnerabilidade | Diversas | https://sourceforge.net/projects/ spacemonkey/ |
| Springenwerk | Python | Análise vulnerabilidade | XSS | https://sourceforge.net/projects/ springenwerk/?source=directory |

**Tabela B.1 continuação da página anterior**

| Ferramenta | Linguagem | Etapa metodologia | Vulnerabilidade | URL |
|---|---|---|---|---|
| sqid | Ruby | Análise e exploração vulnerabilidade | SQLI | http://carnal0wnage.attackresearch.com/ 2007/07/ using-sqid-sql-injection-digger-to-look.html |
| SQL Power Injector | .NET | Análise e exploração vulnerabilidade | SQLI | http://www.sqlpowerinjector.com/ |
| sqlbftools | .NET | Análise e exploração vulnerabilidade | SQLI | https://packetstormsecurity.com/ files/43795/sqlbftools-1.2.tar.gz.html |
| SQLiX | Perl | Análise vulnerabilidade | SQLI | https://www.owasp.org/index.php/ Category:OWASP_SQLiX_Project#tab=Main |
| SQLmap | Python | Análise e exploração vulnerabilidade | SQLI | http://sqlmap.org/ |
| SQLninja | Perl | Análise e exploração vulnerabilidade | SQLI | http://sqlninja.sourceforge.net/ |
| SQLSentinel | Java | Análise vulnerabilidade | SQLI | https://sourceforge.net/projects/ sqlsentinel/?source=navbar |
| SQLsus | Perl | Análise e exploração vulnerabilidade | SQLI | http://sqlsus.sourceforge.net/ |
| st4lk3r | Perl | Análise vulnerabilidade | Diversas (*Web Server*) | http://www.forum-invaders.com.br/vb/archive/index.php/t-29244.html |
| THC-Hydra | C | Exploração | Brute Force Password | https://github.com/vanhauser-thc/thc-hydra |
| Umbrella | .NET | Análise vulnerabilidade | Diversas | https://sourceforge.net/projects/ umbrella-project2012/?source=directory |
| Uniscan | Perl | Análise vulnerabilidade | LFI, RFI, Cmd. Exec. | https://sourceforge.net/projects/uniscan/ |
| Vega | Java | Análise vulnerabilidade | Diversas | https://subgraph.com/vega/ |

**Tabela B.1 continuação da página anterior**

| Ferramenta | Linguagem | Etapa metodologia | Vulnerabilidade | URL |
|---|---|---|---|---|
| VERA | Python | Análise vulnerabilidade | Diversas | http://www.spacios.eu/index.php/spacios-tool/ |
| W3af | Python | Análise e exploração vulnerabilidade | Diversas | http://w3af.org/ |
| Wapiti | Python | Análise vulnerabilidade | Diversas | http://wapiti.sourceforge.net/ |
| Webarmy | Python | Análise vulnerabilidade | XSS, SQLI | https://sourceforge.net/projects/webarmy/ |
| Webshag | Python | Análise e exploração vulnerabilidade | Diversas (*Web Server*) | https://www.scrt.ch/en/attack/ downloads/webshag |
| WebSlayer | Python | Análise vulnerabilidade | Diversas | http://www.edge-security.com/webslayer.php |
| WebSploit | Python | Diversas (*Framework*) | Diversas (*Framework*) | https://sourceforge.net/projects/websploit/ |
| Webvulscan | PHP | Análise vulnerabilidade | Diversas | https://github.com/dermotblair/webvulscan |
| Wfuzz | Python | Análise vulnerabilidade | Diversas | http://www.edge-security.com/wfuzz.php |
| WHCC | VB | Análise vulnerabilidade | Diversas (*Web Server*) | https://www.darknet.org.uk/2007/01/introducing-whcc-web-hack-control-center/ |
| Whitewindow | Ruby | Análise vulnerabilidade | SQLI | https://www.kitploit.com/2017/05/whitewidow-sql-vulnerability-scanner.html |
| Wikto | .NET | Análise vulnerabilidade | Diversas (*Web Server*) | https://github.com/sensepost/wikto |
| witchxtool | Perl | Análise vulnerabilidade | Diversas | https://sourceforge.net/projects/witchxtool-v10/?source=typ_redirect |
| WPSCAN | Ruby | Análise vulnerabilidade | Diversas (CMS) | https://wpscan.org/ |
| WS-Attacker | Java | Análise vulnerabilidade | Diversas (WS SOAP) | https://github.com/RUB-NDS/WS-Attacker |

**Tabela B.1 continuação da página anterior**

| Ferramenta | Linguagem | Etapa metodologia | Vulnerabilidade | URL |
|---|---|---|---|---|
| WSDigger | .NET | Análise vulnerabilidade | Diversas (WS SOAP) | https://sourceforge.net/projects/ foundstone/files/?source=navbar |
| WSFuzzer | Python | Análise vulnerabilidade | Diversas (WS SOAP) | https://www.owasp.org/index.php/ Category:OWASP_WSFuzzer_Project |
| WSTool | PHP | Análise vulnerabilidade | Diversas | https://sourceforge.net/projects/ wstool/?source=directory |
| Xenotix XSS Exploit | .NET | Análise e exploração vulnerabilidade | XSS | https://www.owasp.org/index.php/ OWASP_Xenotix_XSS_Exploit_ Framework |
| XSS Shell | .NET | Exploração | XSS | https://www.darknet.org.uk/2006/12/ xss-shell-v039-cross-site-scripting-backdoor-tool/ |
| XSS-Proxy | Perl | Exploração | XSS | https://sourceforge.net/projects/xss-proxy/?source=navbar |
| XSS-Scanner | Java | Análise vulnerabilidade | XSS | https://sourceforge.net/projects/xss-scanner/?source=directory |
| XSSer | Python | Análise e exploração vulnerabilidade | XSS | https://xsser.03c8.net/ |
| XSSMap | Go | Análise vulnerabilidade | DOM XSS | https://github.com/rverton/xssmap |
| XSSploit | Python | Análise e exploração vulnerabilidade | XSS | https://www.scrt.ch/en/attack/ downloads/xssploit |
| XSSSniper | Python | Análise vulnerabilidade | XSS | https://bitbucket.org/gbrindisi/xsssniper |
| XSStrike | Python | Análise vulnerabilidade | XSS | https://www.darknet.org.uk/2018/03/ xsstrike-advanced-xss-fuzzer-exploitation-suite/ |
| ZAP | Java | Análise vulnerabilidade | Diversas | https://www.owasp.org/index.php/ZAP |

Tabela B.2: Referências utilizadas no levantamento das ferramentas. Fonte: O autor (2019)

| **Título** |
|---|
| *A black-box testing tool for detecting SQL injection vulnerabilities* (Djuric, 2013) |
| *A clustering approach for web vulnerabilities detection* (Dessiatnikoff et al., 2011) |
| *A formal approach to exploiting multi-stage attacks based on file-system vulnerabilities of web applications* (de Meo e Viganò, 2017) |
| *A heuristic-based approach for detecting SQL-injection vulnerabilities in web applications* (Ciampa, 2010) |
| *A model-driven penetration test framework for Web applications* (Xiong e Peyton, 2010) |
| *A New Framework of Security Vulnerabilities Detection in PHP Web Application* (Jingling e Rulin, 2015) |
| *A survey on web penetration test* (Mirjalili et al., 2014) |
| *Advanced Automated Web Application Vulnerability Analysis* (Doupé, 2014) |
| *An automated approach for testing the security of web applications against chained attacks* (Calvi e Viganò, 2016) |
| *An automated vulnerability scanner for injection attack based on injection point* (Chen e Wu, 2010) |
| *Analysis and Assessment of Web Application Security Testing Tools* (Gupta e Sharma, 2011) |
| *Analyzing the Accuracy and Time Costs of Web Application Security Scanners* (Suto, 2010) |
| *Assessment of open source web application security scanners* (Abbas Saeed e Abed Elgabar, 2014) |
| *Automated black-box detection of access control vulnerabilities in web applications* (Li et al., 2014a) |
| *Benchmarking of Pentesting Tools* (Vega e Villalba, 2017) |
| *Black Hat Python, Python Programming for Hackers* (Fallis, 2013) |
| *Black-box detection of XQuery injection and parameter tampering vulnerabilities in web applications* (Deepa et al., 2018) |
| *BLOCK: a black-box approach for detection of state violation attacks towards web applications* (Li e Xue, 2011) |
| *Comparison of penetration testing tools for web applications* (van der Loo, 2011) |
| *Deemon: Detecting CSRF with Dynamic Analysis and Property Graphs* (Pellegrino et al., 2017) |
| *Designing vulnerability testing tools for web services: approach, components, and tools* (Antunes e Vieira, 2017) |
| *Detecting Vulnerabilities in Web Applications Using Automated Black Box and Manual Penetration Testing* (Awang e Manaf, 2013) |
| *Detection of Web Vulnerabilities via Model Inference assisted Evolutionary Fuzzing* (Duchene, 2015) |
| *Effectiveness of Automated Application Penetration Testing Tools* (Ferreira e Kleppe, 2011) |
| *Enemy of the State: A State-Aware Black-Box Web Vulnerability Scanner* (Doupé et al., 2012) |
| *Enlargement of vulnerable web applications for testing* (Román Muñoz et al., 2017) |
| *Evaluation and Testing of Several Free / Open Source Web* (Suteva et al., 2013) |
| *Evaluation of web vulnerability scanners* (Makino e Klyuev, 2015) |
| *Exploiting Software How to Break Code* (Hoglund e Mcgraw, 2004) |
| *Hacking Exposed Web Applications, 2nd Ed. (Hacking Exposed)* (Scambray et al., 2006) |
| *Kali Linux Web Penetration Testing Cookbook* (Nájera-Gutiérrez, 2016) |

**Tabela B.2 continuação da página anterior**

| **Título** |
|---|
| *KameleonFuzz* (Duchene et al., 2014) |
| *Knowledge-based security testing of web applications by logic programming* (Zech et al., 2017) |
| *LigRE: Reverse-engineering of control and data flow models for black-box XSS detection* (Duchene et al., 2013) |
| *LogicScope: automatic discovery of logic vulnerabilities within web applications* (Li e Xue, 2013) |
| *Mastering Kali Linux for Web Penetration Testing* (McPhee, 2017) |
| *Mastering Metasploit* (Jaswal, 2014) |
| *Mastering Modern Web Penetration Testing* (Prasad, 2016) |
| *Open Source Web Vulnerability Scanners : The Cost Effective Choice?* (Mcquade, 2014) |
| *OWASP Testing Guide 4.0* (OWASP, 2014) |
| *Penetration testing for web services* (Antunes e Vieira, 2014) |
| *Penetration testing tool for web services security* (Mainka et al., 2012) |
| *Penetration Testing with BackBox* (Uygur, 2014) |
| *Pentesting on web applications using ethical - Hacking* (De Jimenez, 2017) |
| *Performance evaluation of web application security scanners for prevention and protection against vulnerabilities* (El Idrissi et al., 2017) |
| *Scanning of real-world web applications for parameter tampering vulnerabilities* (Fung et al., 2014) |
| *SecuBat: A Web Vulnerability Scanner* (Kals et al., 2006) |
| *Security Evaluation of Web Application Vulnerability Scanners Strengths and Limitations Using Custom Web Application* (Martirosyan, 2012) |
| *Semi-automatic security testing of web applications from a secure model* (Büchler et al., 2012a) |
| *SPaCiTE - Web application testing engine* (Büchler et al., 2012b) |
| *SQL-injection vulnerability scanning tool for automatic creation of SQL-injection attacks* (Ali et al., 2011) |
| *SSOScan: Automated Testing of Web Applications for Single Sign-On Vulnerabilities* (Zhou e Evans, 2014) |
| *State of the art: Automated black-box web application vulnerability testing* (Bau et al., 2010) |
| *TestREx: a framework for repeatable exploits* (Dashevskyi et al., 2017) |
| *The Art of Exploitation* (Jon Erikson, 2007) |
| *The Web Application Vulnerability Scanners Benchmark* (Chen, 2014) |
| *Toward automated detection of logic vulnerabilities in web applications* (Felmetsger e Cavedon, 2010) |
| *Toward Black-Box Detection of Logic Flaws in Web Applications* (Pellegrino e Balzarotti, 2014) |
| *Using a Web Server Test Bed to Analyze the Limitations of Web Application Vulnerability Scanners* (Shelly, 2010) |
| *Using WASSEC to Evaluate Commercial Web Application Security Scanners* (Saeed, 2014) |
| *Using Web Security Scanners to Detect Vulnerabilities in Web Services* (Vieira et al., 2009) |
| *VERA: A flexible model-based vulnerability testing tool* (Blome et al., 2013) |

**Tabela B.2 continuação da página anterior**

| **Título** |
|---|
| *Vulnerability Factors in New Web Applications: Audit Tools, Developer Selection & Languages* (Bau et al., 2012) |
| *Web Application Vulnerability Prediction Using Hybrid Program Analysis and Machine Learning* (Shar et al., 2015) |
| *Web Penetration Testing using Nessus and Metasploit Tool* (Mukhopadhyay et al., 2014) |
| *Web Penetration Testing with Kali Linux* (Muniz e Lakhani, 2013) |
| *Web Security A WhiteHat Perspective* (Hanqing Wu, 2015) |
| *Why Johnny Can't Pentest: An Analysis of Black-Box Web Vulnerability Scanners* (Doup et al., 2010) |
| *WIVET - Benchmarking Coverage Qualities of Web Crawlers* (Tatli e Urgun, 2017) |
| *ZigZag: Automatically Hardening Web Applications Against Client-side Validation Vulnerabilities* (Weissbacher et al., 2015) |