Universidade Federal do Pará
Instituto de Ciências Exatas e Naturais
Faculdade de Computação
Curso de Bacharelado Em Sistemas de Informação

PAULO HENRIQUE AMORIM PEREIRA

TESTES DE SEGURANÇA EM APLICAÇÃO WEB

BELÉM

2017

PAULO HENRIQUE AMORIM PEREIRA

TESTES DE SEGURANÇA EM APLICAÇÃO WEB

Trabalho de Conclusão de Curso apresentado no curso de Sistemas de Informação da Universidade Federal do Pará, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Roberto Samarone dos Santos Araújo

BELÉM

2017

PAULO HENRIQUE AMORIM PEREIRA

TESTES DE SEGURANÇA EM APLICAÇÃO WEB

Trabalho de Conclusão de Curso apresentado no curso de Sistemas de Informação da Universidade Federal do Pará, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Aprovada em: ___/___/_____

BANCA EXAMINADORA

Prof. Dr. Roberto Samarone dos Santos Araújo
Universidade Federal do Pará
Orientador

Prof. Dr. Eloi Luiz Favero
Universidade Federal do Pará
Avaliador

Prof. Dr. Nelson Cruz Sampaio Neto
Universidade Federal do Pará
Avaliador

Dedico aos meus pais, Tânia do Socorro Amorim Pereira e Lourival da Conceição Pereira, que sempre deram incentivo ao meu estudo. E para todas as pessoas que passam por momentos de fraqueza, não deixem seus problemas afetarem o que há de melhor em você.

Agradecimentos

Dedico meus sinceros agradecimentos:

Agradeço aos meus familiares e em especial a minha mãe Tânia do Socorro Amorim Pereira e meu pai Lourival da Conceição Pereira por sempre me darem apoio, incentivo e condições ao meu estudo. Sou muito grato por tudo o que fizeram e continuam fazendo por mim, vocês foram muito importante para minha formação.

Agradeço ao Professor Roberto Samarone, pelo acolhimento no laboratório e pela passagem em diversos momentos da minha graduação. Sou grato pelos ensinamentos, ajuda e orientações durante o perído de estudo e para a realização deste trabalho.

Agradeço aos colegas de curso e aos conhecidos na UFPA, pelos momentos de estudo, distrações, conversas e por todos os momentos vividos durante o período do curso.

Agradeço a minha amiga Talita Cari pelos momentos de conversas e aconselhamentos. Agradeço pelo apoio e por estar disposta a me ouvir em alguns momentos.

Agradeço a todos os Professos da Faculdade de Computação que se dedicam para levar seus conheciementos e experiências para os seus alunos. Obrigado por escolherem essa profissão, embora seja desvalorizada mas é de grande relevância para a nossa sociedade e que contribui muito para a realização dos sonhos de muitas pessoas.

Finalizando, agradeço às pessoas que de alguma forma contribuiram para minha formação acadêmica e para a conclusão deste trabalho. Obrigado!

"Então se você sente como se estivesse
em um buraco negro, não desista.
Sempre existe uma forma de sair."
(Stephen Hawking)

Resumo

Diversas transações são realizadas constantemente por aplicações Web. Muitas dessas transações são sigilosas e lidam com diversos dados confidenciais. Devido a isso, a segurança de aplicações Web deve ser corretamente implementada e constantemente testada. A realização de testes de segurança é uma prática importante para a segurança de aplicações Web, visto que constantemente novas vulnerabilidades estão surgindo ou sendo modificadas. O NIST define testes de segurança como sendo uma das formas de avaliar a segurança da informação de uma entidade. Testes de segurança em aplicações Web são procedimentos para encontrar vulnerabilidades de segurança. Diversas metodologias de teste de segurança foram criadas para fornecer melhor planejamento e melhores práticas para a execução dos testes. Desta forma, este trabalho tem o objetivo de abordar a prática de testes de segurança em aplicações Web e utilizar uma metodologia para realizar como estudo de caso testes de segurança em uma aplicação Web em busca de vulnerabilidades de segurança, alertando sobre as vulnerabilidades encontradas.

Palavras-chave: Teste de Segurança; Segurança da Informção; Aplicações Web; Vulnerabilidades de Segurança; Owasp.

Abstract

Many transactions are handled through Web applications. Many of the transactions are sensitive and handle a lot of sensitive data. Because of this, a web application security must be properly implemented and tested constantly. A security test is an important practice for Web application security, as new vulnerabilities are constantly emerging or being modified. NIST defines security testing as one of the ways to evaluate an entity's information security. Security testing on Web applications are procedures for finding security vulnerabilities. Several security testing methodologies have been created to provide better planning and best practices for testing. In this way, this work aims to approach a practice of security testing in Web applications and use a methodology to carry out as a case study security tests in a Web application in search of security vulnerabilities, alerting as vulnerabilities found.

Key words: Security Test. Information security; Web applications; Security Vulnerabilities; Owasp.

Sumário

Lista de Abreviaturas

CERT.br	centro de estudos, respostas e tratamento de incidentes de segurança no Brasil
STJ	Superior Tribunal de Justiça
SGSI	Sistemas de Gerenciamento de Segurança da Informação
XSS	Cross-Site Scripting
CSRF	Cross-Site Request Forgery
NIST	National Institute of Standards and Technology
OSSTMM	Open Source Security Testing Methodology Manual
ISSAF	Information System Security Assessment Framework
PTES	Penetration Testing Execution Standard
ISECOM	Institute for Security and Open Methodologies
PHYSSEC	Physical Security
SPECSEC	Spectrum Security
COMSEC	Communications Security
RAVs	Risk Assessment Values
OISSG	Open Information Systems Security Group
SET	Toolkit de Social-Engineer
DIG	Domain Information Groper
Nmap	Network Mapper
URL	Uniform Resource Locator
HTTP	Hypertext Transfer Protocol
XST	Cross-Site Tracing

HTML	HyperText Markup Language
HTTPS	Hyper Text Transfer Protocol Secure
ZAP	Owasp Zed Attack
captcha	Completely Automated Public Turing test to tell Computers and Humans Apart
NISTIR	NIST Internal/Interagency Report

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Introdução

A *Internet* se transformou em um dos principais meios de comunicação do mundo. Nesta rede trafega grande quantidade de dados confidenciais a todo momento e em tempo real, como senhas de usuários, números de cartões de créditos, documentos privados etc. Devido ao grande tráfego desses dados, é necessário garantir a segurança empregada nos serviços *Web*, pois se um sistema deste tipo falhar, todos esses dados estarão comprometidos e podem ser usados indevidamente. Sendo assim, é importante atestar a segurança nos serviços *Web* e a realização de testes de segurança é uma das práticas fundamentais para isso.

Um dos grandes problemas relacionado às tecnologias *Web* é o fato de ser propícia a diversos tipos de ataques, além do crescimento das ocorrências dos ataques que exploram vulnerabilidades em sistemas *Web*. As organizações na medida que percebem ou são afetadas por esse cenário, passam a demandar por serviços *Web* mais seguros [1]. Outro fator relevante são os riscos relacionados a qualquer aplicação *Web*, em razão desses riscos mudarem constantemente devido aos avanços das tecnologias [2].

Diversas aplicações possuem vulnerabilidades comuns e que geralmente exigem pouco esforço para ser exploradas por atacantes. Dentre as várias vulnerabilidades existentes, existem as que são exploradas com maior frequência e que oferecem grande perigo às aplicações. Alguns desses riscos mais críticos são apresentados em um projeto da comunidade *Owasp (Open Web Application Security Project)* denominado de *Owasp Top* 10 [3]. Esse projeto tem o objetivo de identificar e listar os 10 principais riscos mais críticos em aplicações *Web* que afetam as organizações.

O crescimento dos ataques contra as aplicações *Web* decorrentes da exploração de vulnerabilidades é devido a inexistência ou a falha na implementação dos mecanismos de segurança nessas aplicações. Isso é um fator relevante para segurança da informação, uma vez que um ataque bem sucedido pode comprometer a privacidade de informações confidenciais ou permitir a invasão de redes corporativas [4].

Stallings [1] destaca que os ataques na *Internet* e em sistemas conectados à *Internet* se tornaram mais sofisticados enquanto a habilidade e o conhecimento necessário para executar os ataques diminuiu. Desta forma, é crescente a necessidade da segurança da informação nas aplicações *Web*.

A segurança da informação consiste, basicamente, como sendo um processo para garantir a proteção da informação. De acordo com a norma ISO/IEC 27000 [5], a segurança da informação é alcançada através da implementação de controles de segurança. Esses controles precisam ser especificados, implementados, monitorados, revisados e melhorados para garantir que os objetivos da segurança da informação e do negócio da organização sejam atendidos.

Tendo em vista a importância do tema, a prática de testes de segurança é fundamental para as aplicações *Web*, visto que testes de segurança avaliam se os controles de segurança das aplicações estão sendo corretamente implementados e analisa se possui vulnerabilidades a ser exploradas, assegurando se está ou não vulnerável a ataques.

1.1 Motivação

A *Internet* tem se expandido cada vez mais ao longo do tempo. Este crescente uso da *Web* está acompanhado pela expansão dos negócios das organizações através dela, principalmente por meio das aplicações *Web*. O uso de aplicações comerciais, lojas virtuais, serviços como transações bancárias, bibliotecas digitais, sistemas acadêmicos, entre outros, utilizam aplicações *Web* como principal meio para diversas transações.

Todas essas aplicações e serviços trabalham com operações e informações, os quais muitos são confidenciais. Sendo assim, a *Internet* se tornou propícia para o roubo de informação sensíveis. Desta forma, é fundamental garantir a segurança de tais aplicações.

Testes de segurança em aplicações *Web* é uma ferramenta de grande importância para as organizações que mantém serviços *Web* em funcionamento. Eles permitem verificar e analisar se a aplicação está ou não vulnerável a ataques, conforme é apresentado nos trabalhos de Castanha [13] e Tulio [14].

Uma das principais motivações para realização de testes de segurança é a busca por falhas nos controles de segurança das aplicações, realizando testes no mecanismo de

autenticação, autorização, etc. Caso seja encontrado vulnerabilidades, é importante que seja feita a devida correção para evitar ataques futuros que comprometam a segurança.

De acordo com a cartilha de segurança para *Internet* criada pelo site CERT.br (centro de estudos respostas e tratamento de incidentes de segurança no Brasil), computares, redes e serviços acessados pela *Internet* podem ser vítimas de ataques por diversos motivos, dentre eles, pode-se citar motivações financeiras, ideológicas, comerciais, por demonstração de poder e até por mero prestígio diante de outros atacantes [6].

No trabalho de Ceron et al. [7], é destacado que os ataques contra aplicações *Web* representam uma parcela considerável dos incidentes de segurança. Isso vem ocorrendo devido a falta da preocupação com os requisitos de segurança. Como reflexo disso, se tem a ocorrência de diversos ataques bem sucedidos, como mostra as estatísticas da coleta de incidentes realizada pelo CERT.br que obteve o total de 722.205 mil incidentes notificados no período de janeiro a dezembro do ano de 2015 no Brasil [6]. A Figura 1 apresenta a coleta de indentes reportados desde o ano de 1999 até 2015.

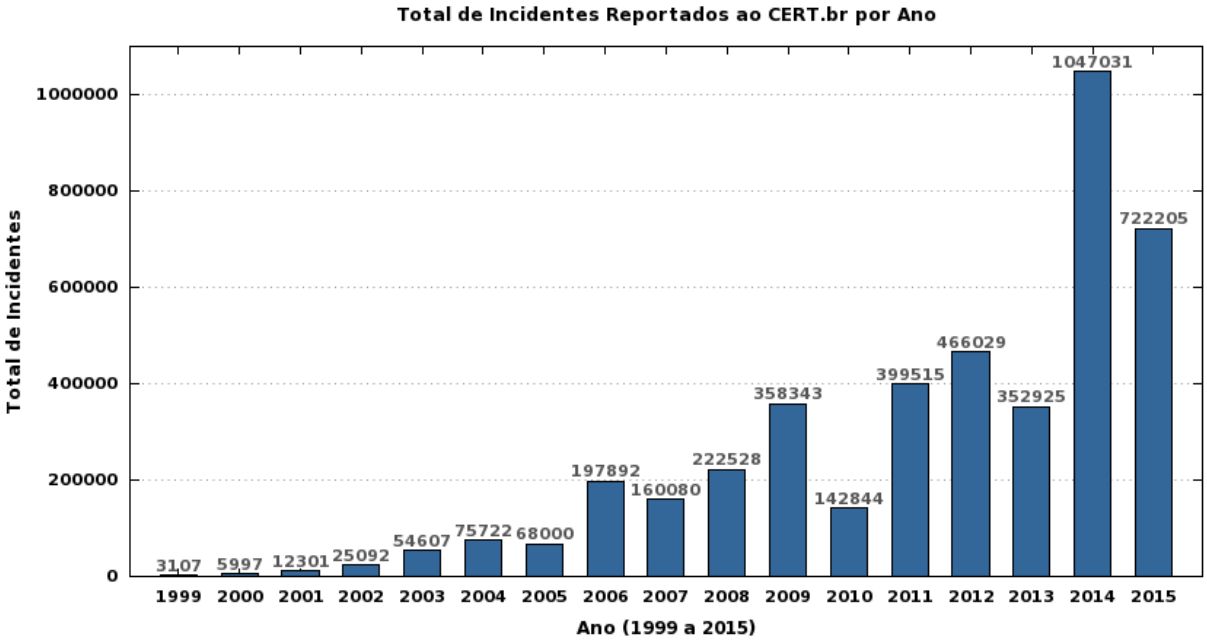

Figura 1: Incidentes reportados ao CERT.BR
Fonte: CERT.BR[6]

As informações expostas anteriormente são alguns dos principais fatores que sustentam a motivação para se realizar testes de segurança em aplicações *Web* e sustentam a importancia deste trabalho contribuindo para a discussão do tema diante do cenário atual.

1.2 Objetivos

Este trabalho tem como objetivo geral realizar um estudo sobre testes de segurança e utilizar uma metodologia reconhecida internacionalmente para realizar testes de segurança em uma aplicação *Web*.

Os objetivos específicos deste trabalho serão:

- Estudar conceitos relacionados à segurança da informação;
- Apresentar as principais vulnerabilidades em aplicações *Web*;
- Estudar as principais metodologias, técnicas e ferramentas para testes de segurança;
- Empregar a metodologia *Owasp Testing Guide* versão 4 para realizar testes de segurança em uma aplicação *Web*;
- Identificar e apresentar possíveis vulnerabilidades de segurança encontradas na aplicação.

1.3 Trabalhos Relacionados

Diversos trabalhos relacionou-se com o aqui proposto. A procura se deu principalmente por estudos e pesquisas com atuação na área de testes de segurança. Os principais trabalhos relacionados encontrados são apresentados abaixo.

Nos trabalhos de Saad *et al.* [8] e Austin *et al.*[9] foram realizados pesquisas sobre técnicas de testes de segurança. Saad *et al.*[8] apresenta uma pesquisa mostrando as principais técnicas de teste de segurança. As técnicas analisadas foram: revisão de código, análise estática automatizada, analise de código binário, teste *fuzz*, analise de risco, varredura de vulnerabilidade e teste de penetração.

Austin *et al.* [9] apresenta uma pesquisa com o objetivo de ajudar na escolha de técnicas de descoberta de vulnerabilidades, através da comparação das vulnerabilidades encontradas por cada técnica e comparando suas eficiências. Para atingir esse objetivo, foi realizado três estudos de caso em três sistemas de registro de saúde. As técnicas de descoberta de vulnerabilidades analisadas foram: teste de penetração por exploração manual, teste de penetração manual sistemática, testes de penetração automatizada e análise estática automatizada. O trabalho teve como conclusão a ideia de que apenas uma técnica é insuficiente para encontrar todos os tipos de vulnerabilidades possíveis. Foi sugerido que seja utilizado mais de uma técnica para se encontrar a maior variedade de tipos de vulnerabilidades.

Avramescu *et al.* [10] apresenta uma análise de ameaças de segurança em aplicações *Web*. Foram aplicados testes de penetração em um cenário real contra a rede, a aplicação *Web* e o banco de dados. O trabalho contribuiu para a discussão do estado da arte de segurança da informação e ilustra os valores de testes de penetração. Os principais objetivos do trabalho foram descrever os procedimentos de teste de penetração, as técnicas de ataques e descrever algumas contra medidas e melhores práticas, a fim de prevenir brechas de segurança em aplicações *Web*.

Primesh e Alex [11] apresenta um estudo de caso com o objetivo de demonstrar os procedimentos e métodos de *pentest* utilizando a metodologia PTES (*Penetration Test Execution Standard*). Foram apresentadas as etapas de execução do processo de *pentest* em servidores em produção de uma empresa multinacional. Realizaram também a demonstração da exploração de uma das vulnerabilidades encontradas e provaram a existência da falha de segurança.

Borges [12] apresenta uma comparação das principais metodologias de *pentest*, a fim de identificar qual a mais abrangente. As metodologias comparadas são OSSTMM, ISSAF, OWASP e NIST. Foi apresentado também uma pesquisa de campo feita com empresas que aplicam testes de penetração no Brasil, a qual apontou as metodologias proprietárias, desenvolvidas pelas empresas que aplicam testes, como as mais utilizadas no mercado nacional.

Castanha [13] apresenta uma auditoria de segurança através de testes de invasão, aplicados em alguns sistemas e aplicações que estão em funcionamento no ambiente da Uniriotec, bem como nos recursos humanos do departamento.

Tulio [14] apresenta testes de segurança em aplicações *Web* utilizando a metodologia *Owasp Testing Guide*. Foram testadas três aplicações *Web* da categoria *e-commerce*. Nos testes, uma das aplicações foi considerado a mais insegura e nas outras aplicações não foram encontrados falhas relevantes.

1.4 Organização do Trabalho

O restante do trabalho está dividido em mais quarto capítulos. Os capítulo 2 e 3 abordam o embasamento teórico pertinentes à segurança da informação e testes de segurança. Enquanto que o capítulo 4 apresenta como estudo de caso testes de segurança em uma aplicação Web. Os capítulos são:

Capítulo 2 – Referencial Teórico

Este capítulo apresenta conceitos relacionados sobre segurança da informação e suas

características, conceitos sobre testes de segurança, principais vulnerabilidades para aplicações *Web* e sobre os principais riscos.

Capítulo 3 – Testes de Segurança

Neste capítulo, são descritas as principais metodologias de testes de segurança reconhecidas e utilizadas internacionalmente, as técnicas de testes de segurança mais usadas e recomendadas e as principais ferramentas que dão suporte aos testes de segurança.

Capítulo 4 – Estudo de Caso: Teste de Segurança em uma aplicação *Web*

Este capítulo apresenta o estudo de caso realizado em uma aplicação *Web*. Será aplicado testes de penetração na aplicação para verificar a existência de vulnerabilidades que podem comprometer a segurança da informação.

Capítulo 5 – Considerações Finais e trabalhos futuros

Neste último capitulo são apresentados as considerações finais sobre o trabalho e possíveis trabalhos futuros.

CAPÍTULO 2

Referencial Teórico

Este capítulo apresenta a base teórica para melhor entendimento do tema proposto no trabalho, e fornece informações relacionadas ao que é tratado nos capítulos seguintes. É apresentado conceitos relacionados com segurança da informação e vulnerabilidades de segurança.

Desta forma, na Seção 2.1 é descrito, o conceito de segurança da informação e as suas principais características. E na Seção 2.2, é apresentado os principais riscos existentes em aplicações *Web* e as formas de exploração através de ataques.

2.1 Segurança da Informação

Antes da ampla utilização da *Internet*, a *Web* foi utilizada principalmente por pesquisadores de universidades. A segurança não era um requisito essencial. Porém, com o passar dos anos e o acelerado avanço das tecnologias *Web*, milhões de usuários comuns passaram a utilizar a *Internet* para realizar operações dos mais diversos tipos. Assim, a segurança das redes passou a ser considerado um problema potencial [15].

A *Internet* propiciou e estimulou a proliferação de inúmeros serviços e informações. As organizações passaram a produzir muitas informações na *Internet*, utilizando a *Web* como meio de comunicação em diversas transações.

A cartilha de segurança da informação elaborada pelo STJ [16] (Superior Tribunal de Justiça), órgão máximo do poder judiciário do Brasil que lida com muitas informações importantes diariamente, refere-se o conceito de informação como sendo um ingrediente

básico que alimenta os processos de decisão e apoio nas organizações. Considera também que quanto maior a importância da informação, maior será a necessidade de manter a confidencialidade, a integridade e a disponibilidade de tal informação [16].

A relevância das informações, principalmente quando trafegada pela *Internet*, fez com que houvesse a necessidade de garantir a proteção da informação. Consequentemente a isso, se tem a crescente quantidade de ameaças e vulnerabilidades às informações trafegadas. Dentro deste contexto se destaca o conceito de segurança da informação.

A norma NISTIR [17] define segurança da informação como sendo a proteção da informação e dos sistemas contra acesso não autorizado, utilização, divulgação, interrupção, modificação ou destruição, a fim de garantir a confidencialidade, integridade e disponibilidade.

A segurança da informação envolve o emprego e a gestão de medidas de segurança apropriadas, com o objetivo de garantir o sucesso e a continuidade do negócio sustentado, e a minimização dos impactos de incidentes de segurança da informação [5].

A definição de segurança da informação envolve a tríade: confidencialidade, integridade e disponibilidade. Mas também é importante destacar outros atributos como a irretratabilidade ou não repúdio e autenticidade. A seguir a descrição desses atributos conforme apresentado por Stallings [18]:

- **Confidencialidade** – É uma propriedade que tem o objetivo de garantir que informações não sejam visualizadas ou divulgadas sem a devida autorização, ou seja, garantir que apenas pessoas autorizadas tenha acesso a tal informação.
- **Integridade** – É uma propriedade que tem o objetivo de garantir que as informações não sejam modificadas ou destruídas de maneira não autorizada, ou seja, garantir que a informação continue íntegra e sem alterações.
- **Disponibilidade** – É uma propriedade que tem o objetivo de garantir que as informações estejam sempre disponíveis para o acesso quando requisitadas.
- **Autenticidade** – É uma propriedade que tem o objetivo de verificar a identidade de quem está enviando uma informação, ou seja, garantir que seja verificado se o usuário é quem diz ser.
- **Irretratabilidade ou Não Repúdio** – É uma propriedade que tem o objetivo de impedir que o emissor ou receptor negue uma ação realizada, ou seja, garantir uma ação seja atribuída especificamente ao autor dela.

A segurança da informação atinge seu objetivo por meio dos controles de segurança. Segundo a norma ISO/IEC 27000 [5], os controles são selecionados e gerenciados

usando um SGSI (Sistemas de Gerenciamento de Segurança da Informação) incluindo políticas, processos, procedimentos, estruturas organizacionais, *software e hardware* com a finalidade de proteger os ativos de informação identificados.

Para a segurança da informação é importante que seja identificado todos os riscos que ameacem as informações. Quando ameaças são exploradas podem gerar vulnerabilidades e permitir a realização de ataques e tem como consequência os incidentes que comprometem as informações, ocasionando a perda da confidencialidade, disponibilidade e integridade [5].

2.2 Riscos de Segurança em Aplicações Web

As aplicações *Web* estão sendo grandes alvos de ataques de segurança, como visto na Seção 1.1 que apresenta a quantidade de incidentes reportados ao CERT.br. Elas estão constantemente sendo ameaçadas por novos tipos de ataques de acordo com o avanço das tecnologias *Web*. A norma RFC 4949 [19] define ameaça como sendo uma possível chance de se explorar uma vulnerabilidade existente, enquanto que um ataque é uma ação sistemática que viola a segurança atravez da exploração de uma vulnerabilidade.

A seguir são apresentadas as principais ameaças voltadas às aplicações *Web*. A documentação *Owasp Top* 10 [3] é usada como base. As ameaças mais comuns de acordo com o documento são as seguintes:

2.2.1 Injeção de Código

Os ataques por injeção de códigos são ataques que visam os *sites Web* que fazem uso de banco de dados para o armazenamento de informações. A injeção de código é uma prática maliciosa que tem como objetivo obter ou deletar informações dos bancos de dados relacionais das aplicações [3].

A injeção de código ocorre quando o atacante insere um código em campo de entrada de dados para que seja executado pelo SGBD (sistema de gerenciamento de banco de dados) da aplicação. Esses dados podem enganar a aplicação para que seja executado o código inserido com comandos maliciosos para ter acesso a dados no banco de dados.

O ataque de injeção SQL ou SQL *injection* é um tipo específico de ataque de injeção, em que é utilizado características da linguagem SQL. Este ataque ocorre quando o atacante envia dados concatenados com instruções SQL através da entrada de dados que será executado pelo sistema de gerenciamento de banco de dados da aplicação [20]. O impacto desse ataque pode resultar, por exemplo, no roubo de dados.

Um possível cenário para a realização de um ataque de SQL *injection* é uma aplicação *Web* que não realiza o tratamento adequado dos dados enviados por um cliente HTTP. Ao ser enviado, esses dados são concatenados em uma *string* de consulta HTTP, resultando no seguinte comando SQL:

```
String query ="SELECT * FROM conta WHERE clienteID= "+
        request.getParameter ("id") + "a";
```

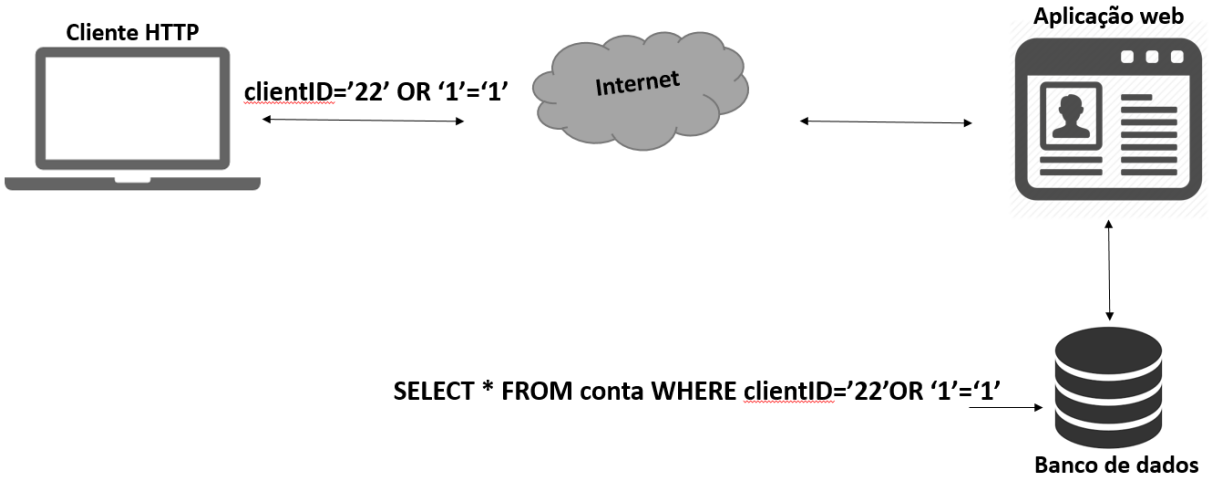

Figura 2: Ataque SQL Injection.
Fonte: Criação do autor.

No comando, uma *string* com o SQL a ser executado pelo sistema gerenciador de banco de dados é criada através da junção entre *SELECT * FROM conta WHERE clienteID= 22*, com o valor do id cliente passado como parâmetro na *query* da URL. O atacante, em vez de passar apenas o clienteID, encaminha como *query* de entrada os dados: *clienteID= 22 OR ´1´=´1´*, conforme a Figura 2.

O uso do parâmetro recebido, sem tratamento adequando, produziria a seguinte consulta no banco de dados:

```
SELECT * FROM conta WHERE clientID= 22 OR ´1´=´1´
```

A consulta contém uma expressão lógica que é sempre verdadeira, visto que a primeira condição *clienteID = 22* pode ser falsa, mas a segunda condição é verdadeira, então a consulta retornará todos os registros de usuários da tabela conta, e não apenas o registro da pessoa de *clienteID = 22*. Uma possível consequência desse ataque seria a perda da confidencialidade dos dados manipulados pela aplicação, pois o atacante obteria acesso aos dados sem a devida autorização no sistema.

2.2.2 Quebra de Autenticação e Gerenciamento de Sessão

Esse tipo de vulnerabilidade possibilita ao atacante utilizar um sistema se passando por outro usuário. Isso ocorre devido a falhas na autenticação ou no gerenciamento de sessão [3].

Isso ocorre quando as funções da aplicação relacionadas à autenticação e ao gerenciamento de sessão apresentam falhas e permitem que a mesma seja explorada por atacantes. A *Owasp* descreve uma lista com alguns motivos que tornam os ativos de gerenciamento de sessão e autenticação vulneráveis. Os principais motivos de acordo com a *Owasp* [3] são:

- Se as credenciais de autenticação são armazenados sem a utilização de *hash* ou criptografia.
- Se é possível descobrir as credencias por meio de funções fracas de gerenciamento de contas, por exemplo, no cadrato de usuário, no mecanismo para recuperar ou alterar senha e na utilização de IDs de sessão fracos.
- Se for possível a reescrita de URL.
- Se for possível ataques de fixação de sessão.
- Se a sessão não expirar corretamente.
- Se credenciais são enviadas através de conexões não criptografadas.

Um exemplo de cenário propício para a exploração da vulnerabilidade é uma aplicação *Web* que possibilita a reescrita de URL, colocando o ID de sessão na própria URL, como o exemplo abaixo:

```
http://exemplo.com/itens;jsessionid=KGJBKJGLKBML2KNW232KJNKJ4?obj=11
```

O id de sessão (*jsessionid*) está inserido na própria URL, isso possibilita no caso de um usuário malicioso na posse desta URL realizar operações indevidas utilizando a identidade de outro usuário.

2.2.3 *Cross-Site Scripting* (XSS)

O XSS é um tipo de vulnerabilidade que permite ao atacante executar *scripts* maliciosos no navegador *Web* da vítima, ou seja, permite a injeção de códigos no lado do

cliente. Ele ocorre quando a aplicação recebe dados não confiáveis e os envia ao navegador da vítima sem realizar uma verificação adequada desses dados [3] . Através de um ataque XSS, o atacante insere códigos malicioso ocultos em um campo na página *Web* já existente da aplicação e este campo é apresentado às vítimas.

A partir de um ataque XSS é possível comprometer a confidencialidade e integridade das transferências de dados entre a aplicação *Web* e o cliente, e fazer sequestro de sessão ou redirecionar o usuário para uma aplicação maliciosa.

Esta vulnerabilidade é subdividida em três categorias, sendo elas do tipo refletido, armazenado e baseado em DOM.

XSS Refletido

O ataque do tipo XSS Refletido ocorre quando o servidor reflete o que é enviado sem verificar os dados inseridos. Por exemplo, ao ser enviado determinado parâmetro para o servidor, não é feito nenhuma verificação e o servidor executa o parâmetro e exibe na página resultante, causando essa vulnerabilidade [20].

Um exemplo de cenário para esse ataque é uma aplicação que reflete dados inseridos em um campo de entrada de dados, conforme a Figura 3.

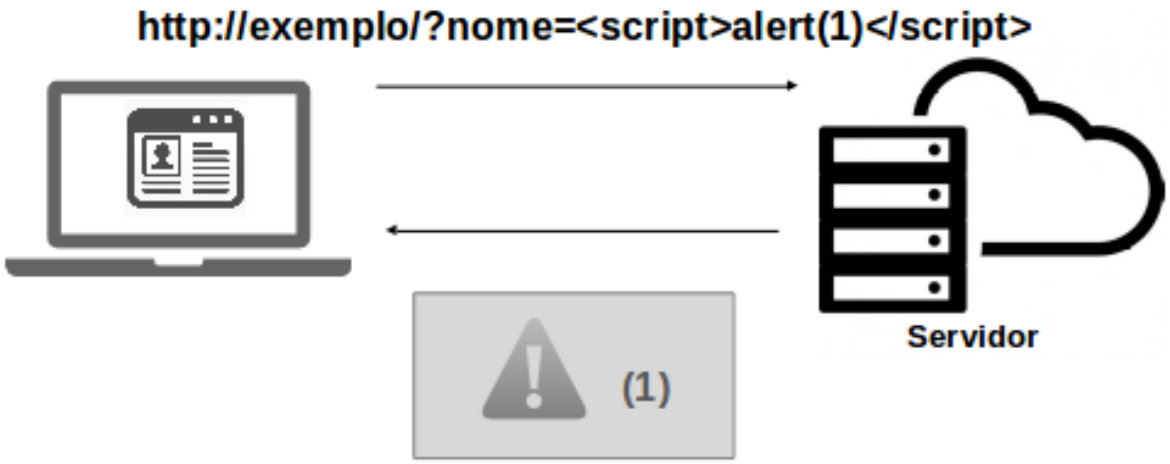

Figura 3: Esquema ataque XSS refletido.
Fonte: Criação do autor.

O funcionamento do cenário da Figura 3 da vulnerabilidade XSS refletido ocorre basicamente da seguinte maneira:

1. O usuário acessa a página;
2. A página solicita os dados;

3. O usuário preenche com um *script* que emite um alerta;

4. A página imprime na tela uma mensagem e gera o alerta com o valor (1).

Desta forma, como a aplicação refletiu o *script* inserido, então é possível que seja executado outros tipos de *scripts* com intenções maliciosas.

XSS Armazenado

O ataque XSS do tipo Armazenado é quando o código malicioso a ser injetado não foi filtrado e está armazenado, persistido em algum componente da aplicação, que normalmente é no banco de dados [20]. Quando alguma página for imprimir na tela o conteúdo armazenado, caso não filtre os caracteres nocivos, o XSS Armazenado é disparado. O usuário é vítima ao acessar o campo afetada pelo armazenamento do código malicioso.

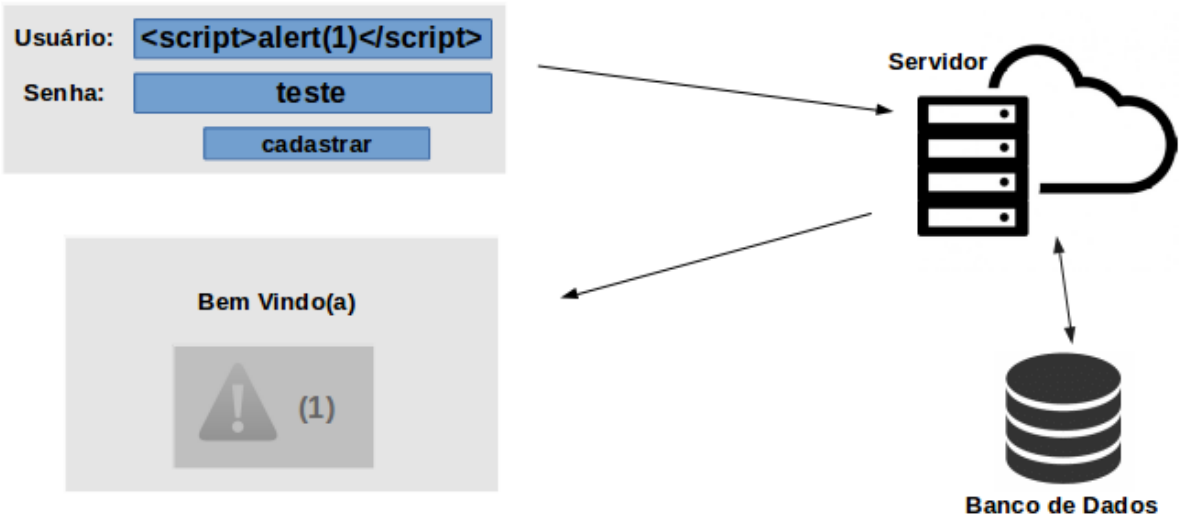

Figura 4: Esquema de ataque XSS armazenado.
Fonte: Criação do autor.

No cenário da Figura 4 é possível verificar a existência da vulnerabilidade XSS por armazenamento. O funcionamento ocorre da seguinte maneira:

1. O usuário acessa a página http://www.exemplo.com/cadastro;

2. A página solicita usuário e senha para cadastrar;

3. O usuário preenchido com "≪script≫alert(1)≪/script≫" no campo usuário e "teste"no compo senha;

4. Após o cadastro o usuário é redirecionado para uma tela de boas vindas e gera uma caixa de alerta com o conteúdo "(1)".

XSS Baseado em DOM

Este é o tipo mais difícil de ser encontrado entre os três porque depende de uma vulnerabilidades em algum dos componentes da página *Web* (geralmente códigos *javascript*) caracterizado por ser disparado no momento de execução ao invés de vir embutido no código fonte.

2.2.4 Referência Insegura e Direta a Objetos

Essa vulnerabilidade ocorre quando a aplicação expõe uma referência à implementação interna de um objeto da aplicação, por exemplo, um arquivo ou um diretório. Isto acontece, por exemplo, quando a aplicação utiliza o nome real de um objeto para gerar determinada página *Web*. Além do mais, a aplicação não verifica se o acesso a determinado objeto é autorizado para o usuário [3]. Sendo assim, a aplicação apresenta uma falha de referência insegura e direta ao objeto.

Devido a uma falha de referência insegura e direta a um objeto, acompanhado da ausência de verificação do controle de acesso, um atacante é capaz de explorar essa vulnerabilidade para acessar dados não autorizados.

Para saber se uma aplicação está vulnerável a isso, é preciso verificar se todos os objetos referenciados estão devidamente protegidos apropriadamente. Para isso, são considerados duas verificações de acordo com a *Owasp* [3]:

- Verificar se é possível o acesso direto a um objeto restrito da aplicação por um usuário não autorizado.
- Verificar se o acesso restrito a um objeto falha para um usuário autorizado.

Um exemplo de cenário de ataque é uma aplicação que utiliza dados não verificados em uma chamada SQL que está acessando as informações de conta dos usuários. Sendo assim, o atacante modifica um parâmetro na URL para enviar qualquer número de conta:

```
http://exemplo.com/accountInfo?acct=outraConta
```

É possível manipular valores de parâmetros para detectar a falha. Se não verificado adequadamente, o atacante pode acessar qualquer conta de usuário, em vez de somente a conta permitida.

2.2.5 Configurações Incorreta de Segurança

É comum encontrar aplicações com configurações incorretas de segurança. Diversas partes da aplicação pode ser configurada incorretamente, como o servidor *Web* utilizado, o banco de dados ou até em partes do código [3].

É necessário que seja garantido que todos os níveis da aplicação estejam configurados corretamente. Falhas como o servidor *Web* desatualizado e o uso de contas padrão podem ser fatores que contribuam para que atacantes consigam o acesso não autorizado a alguns dados ou funcionalidades do sistema.

Para saber se o sistema apresenta algum risco devido a incorreta configuração ou falta de proteção, algumas verificações devem ser feitas. Ferramentas automatizados podem ser utilizados para detectar esses erros. A *Owasp* [3] recomenda as seguintes verificações:

- Verificas a versão dos *softwares* utlizados, como o sistema operacional, o servidor *Web*, o SGBD.
- Verificar a existência de recursos desnecessários ativados ou instalados, por exemplo, portas, serviços, páginas, contas e privilégios.
- Verificar a existência de contas e senhas padrões.
- Verificar o tratamento de erros que revelam informações sensíveis.
- Verificar as configurações de segurança do *framework* de desenvolvimento e bibliotecas.

2.2.6 Exposição de Dados Sensíveis

Ocorre quando as aplicações não protegem devidamente os dados sensíveis, tais como senhas, números de cartão de crédito e credenciais de autenticação. A criptografia é um dos principais mecanismo para a proteção de dados sensíveis, mas é preciso que seja corretamente implementada [3].

Inicialmente, no desenvolvimento da aplicação, é necessário definir quais dados são sensíveis e que precisam de maior proteção. Posteiormente, para todos esses dados, a *Owasp* [3] recomenda as seguintes verificações:

- Verificar se o armazenamento de dados sensíveis é realizado em texto plano.
- Verificar se a trasmissão de dados sensíveis é feita em texto plano.

- Verificar se os algoritmos de criptografia utilizados são fracos ou defasados.
- Verificar se as chaves criptográficas geradas são fracas.

2.2.7 Falta de Função para Controle de Nível de Acesso

A falta de função para controle de nível de acesso resulta que a aplicação permita o acesso a determinada funcionalidade sem a devida autorização. Isto acontece, pois a aplicação não verifica as permissões de acesso na requisição à função alvo [3].

A verificação de permissão de acesso deve ocorrer em nível de função e também no servidor, porém não é o que acontece em determinadas aplicações, o que possibilita que um atacante consiga explorar essa falha.

Para verificar a falta de função para controle de nível de acesso em uma aplicação, a *Owasp* [3] considera as seguintes verifiações:

- Verificar se aplicação restringe o acesso à funções não autorizadas.
- Verificar se no lado do servidor é realizado verificação de autenticação e autorização.

Um exemplo de cenário de ataque é forçar a navegação pelas URLs alvo da aplicação. Por exemplo, as duas URLs abaixo exigem autenticação para acesso. A segunda URL "admin_getInfo", além de autenticação, restringe o acesso apenas para usuários com permissão de administrador.

```
http://exemplo.com/getInfo
http://exemplo.com/admin_getInfo
```

O mínimo para acessar as páginas é estar autenticado. Se for possível acessa-las sem a devida autenticação, então o controle de acesso é falho. E se a página "admin_getInfo"permitir o acesso por usuário sem permissão de administrador, então a falha ainda é mais grave, pois o atacante poderá assumir a página de administrador.

2.2.8 *Cross-Site Request Forgery* (CSRF)

O CSRF ocorre quando o navegador da vítima é forçado a executar uma ação maliciosa a favor do atacante. É um tipo de ataque que se utiliza a sessão ativa para realizar operações sem o consentimento da vítima [20]. Diferentemente do XSS *Injection*, o CSRF *Injection* explora a confiança que o servidor tem no cliente para liberar recursos e funcionalidades.

O CSRF é um ataque que explora a relação de confiança entre a aplicação *Web* e o usuário. Para isso, o atacante induz o usuário a executar atividades arbitrárias na aplicação, permitindo que ações específicas sejam realizadas no sistema sem o consentimento da vítima.

Um possível cenário de ataque CSRF é um sistema bancário que permite um usuário submeter uma requisição para ação maliciosa, conforme a Figura 5. No exemplo, a vítima irá realizar uma operação de transferência.

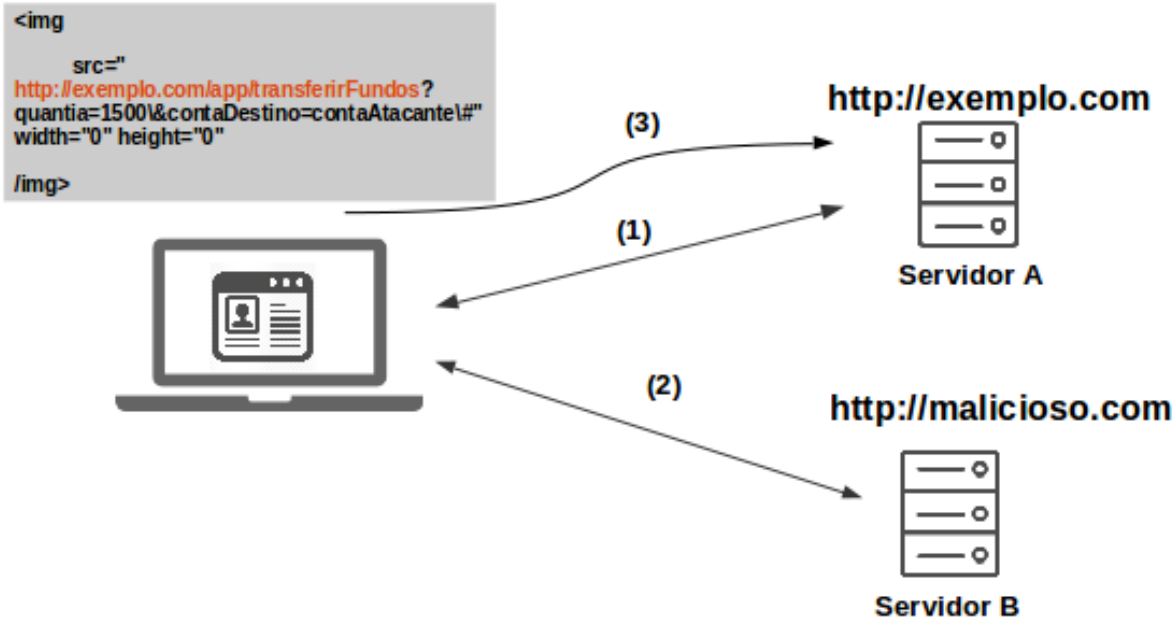

Figura 5: Esquema ataque CSRF.
Fonte: Criação do autor.

Primeiramente, o usuário acessa o sistema bancário e se autêntica inserindo suas credenciais. Após isso, requisita a operação para realizar uma transferência de dinheiro para outra conta. A URL gerada para a operação é a seguinte:

```
http://exemplo.com/transferir?quantia=1500&contaDestino=8236624
```

Na URL, é possível ver as variáveis *"quantia=1500"* e *"contaDestino=8236624"* que serão utilizadas na operação. Para mudar o valor e o destino basta alterar os valores das variáveis.

Sendo assim, na segunda etapa para se realizar o ataque, o atacante constrói uma requisição que irá transferir dinheiro da conta da vítima para a conta do atacante, e então incorpora este ataque em uma requisição armazenada em uma imagem ou *iframe* em vários *sites* sob seu controle. Por exemplo, o atacante pode armazenar a requisição maliciosa em uma imagem, conforme é mostrado no código a seguir:

```
<img
  src="http://malicioso.com/transferirFundos?quantia=1500&
  contaDestino=contaAtacante#"width="12"height="10"
/img>
```

Por fim, se a vítima acessar qualquer um dos *sites* do atacante enquanto estiver autenticado em exemplo.com, essas requisições forjadas irão incluir automaticamente informações de sessão do usuário, autorizando o pedido do atacante. Sendo assim, o usuário inconscientemente efetua a operação que está indicado na URL que o atacante forneceu na imagem, transferindo a quantia indicada pelo atacante.

2.2.9 Utilização de Componentes Vulneráveis Conhecidos

As aplicações *Web* geralmente utilizam alguns componentes externos, tais como bibliotecas, *frameworks*, e outros módulos de *software* que em alguns casos são executados com privilégios elevados [3].

Isto é um problema, pois em muitos casos são utilizados componentes ou bibliotecas externas que estão desatualizados. A situação é crítica quando a aplicação mantém relação de dependência com esses componentes vulneráveis.

A melhor forma de determinar se a aplicação apresenta alguma vulnerabilidade com relação a utilização de componentes vulneráveis é por meio de pesquisas em banco de dados de vulnerabilidades como o CVE [3]. O objetivo é pesquisar se algum componente que faz parte da aplicação tem alguma vulnerabilidade já conhecida publicamente.

2.2.10 Redirecionamento e Encaminhamento Inválidos

O redirecionamento e encaminhamento inválidos ocorrem quando o atacante consegue alterar a página de destino em um redirecionamento. Isso acontece, pois a aplicação especificou a página de destino com parâmetros que não são validados [3]. Sem essa validação, os atacantes podem redirecionar as vítimas para *sites* de *phishing* ou usar encaminhamentos para acessar páginas não autorizadas.

Para saber se a aplicação possui redirecionamentos ou encaminhamentos não validados, a *Owasp*[3] recomenda as seguintes verificações:

- Verificar se é feito a validação da URL de destino dos redirecionamentos;
- Testar todos os redirecionamentos.

2.3 Conclusão

Inicialmente, foi apresentado de forma pontual o contexto sobre a relação do uso da *Internet* com a segurança da informação. Foi enfatizado o grande tráfego de dados privados e inúmeros serviços através da *Web*, o que influenciou no aumento dos ataques visando o roubo de informações. Conceitos foram apresentados afim de compreender os objetivos e importância da segurança da informação nesse contexto e para enfatizar a importância de testes de segurança.

Em seguida, foram apresentados as principais ameaças que estão constantemente colocando em risco as aplicações *Web* e que são os principais fatores para o roubo de informações confidenciais na *Internet*. As ameaças apresentadas têm grande ocorrência de exploração, pois são as que mais afetam as organizações. As vulnerabilidades apresentadas servirá como base para o trabalho, pois os testes de segurança tem o objetivo de encontrar possíveis vulnerabilidades.

CAPÍTULO 3

Testes de Segurança

O NIST (*National Institute of Standards and Technology*) define avaliação de segurança da informação como sendo um processo para verificar se uma entidade atente aos requisitos de segurança necessários. A avaliação pode ser realizada através de testes, análises ou entrevistas [21].

Testes de segurança são procedimentos para encontrar vulnerabilidades de segurança. Possibilitam às organizações assegurar que os sistemas estão devidamente protegidos e identificar se os requisitos de segurança foram implementados corretamente[21].

Para a realização de testes de segurança, é conveniente a criação de um plano de testes contendo todo o escopo dos testes. O plano também define a metodologia, as técnicas e as ferramentas que serão utilizadas.

Este capítulo apresenta as principais metodologias, técnicas e ferramentas para a realização de testes de segurança. Ele está organizado da seguinte forma: a Seção 3.1 apresenta as principais metodologias de testes de segurança; a Seção 3.2 apresenta os tipos de técnicas utilizadas em testes de segurança e a Seção 3.3 apresenta ferramentas de testes de segurança.

3.1 Metodologias de Testes de Segurança

Uma metodologia fornece o planejamento e melhores práticas para a execução dos testes de segurança. Para Stouffer *et al.* [21], o uso de uma metodologia em relação a testes de segurança é importante para ajudar a identificar os recursos necessários e

planejar todo o processo a fim de contornar limitações de recursos (*e.g.,* limitação de pessoal, *hardware, software* e tempo).

Conforme os estudos realizados por Bertoglio e Zorzo [22] e o de Santos e Nunes [23], as principais metodologias adotadas para a realização de testes de segurança são: OSSTMM (*Open Source Security Testing Methodology Manual*) [24], ISSAF (*Information System Security Assessment Framework*) [25], PTES (*Penetration Testing Execution Standard*) [26], NIST SP 800-115 *(National Institute of Standards and Technology*) [21] e *OWASP Testing Guide* [27]. A seguir são apresentados estas metodologias.

3.1.1 OSSTMM (*Open Source Security Testing Methodology Manual)*

A metodologia OSSTMM [24] é atualmente mantida pela ISECOM (*Institute for Security and Open Methodologies*) e está na versão 3 que foi disponibilizada no ano de 2010. A metodologia é livre para utilização.

O objetivo da metodologia é fornecer um padrão sistemático para a definição da segurança operacional, ou seja a segurança do meio físico, das interações humanas e dos meios de comunicação, através da análise e da comparação dos resultados de testes realizados, por exemplo teste de penetração e avaliação de vulnerabilidade [24].

O tamanho da organização ou a tecnologia utilizada não influência na aplicação da metodologia. Além disso, ela se adequa para a maioria dos tipos de auditorias, incluindo teste de penetração, avaliação de segurança, avaliação de vulnerabilidade, entre outros.

O OSSTMM, a partir da versão 3, tem o propósito de englobar em seu escopo de testes todo o ambiente da segurança operacional possível, para atingir qualquer tipo de ativo (*i.g* qualquer informação ou serviço com valor agregado). Isso se deve ao fato do avanço de algumas áreas tecnológicas (*e.g.* computação em nuvem, operações remotas e virtualização) e também por novos tipos de infraestruturas.

A metodologia é dividida em canais (*channel*), módulos (*module*) e tarefas (*task*). O escopo do teste é dividido em cinco canais, classificados em humano, físico, comunicação sem fio, redes sem fio, redes de dados e telecomunicação. Os canais, por sua vez, são classificados em três classes chamados de PHYSSEC (*Physical Security*), SPECSEC (*Spectrum Security*) e COMSEC (*Communications Security*).

Hertzog [24] divide a execução da metodologia em quatro fases que contempla 17 módulos para testes:

1. Fase de Indução – É realizado um estudo dos requisitos, do escopo e das limitações

do escopo. Essa fase envolve os módulos: revisão de estado, logística e verificação de detecção ativa.

2. Fase de Interação – É efetivada a criação do escopo. Essa fase envolve os módulos: visibilidade de auditoria, verificação de acesso, verificação de confiança e verificação de controles.

3. Fase de Inquérito – É realizada a coleta de informações. Essa fase envolve os módulos: verificação de processo, verificação de configuração, validação de propriedade, revisão da segregação, verificação da exposição e inteligência competitiva.

4. Fase de Intervenção – É realizado os testes práticos utilizando as informações adquiridas. Essa fase envolve os módulos: verificação de quarentena, auditoria de privilégios, validação de sobrevivência, alerta e revisão de *logs*.

A classificação dos testes de segurança é dividida em seis tipos que se diferem em alguns aspectos, como a quantidade de informação que o testador tem do alvo de teste, a legitimidade e o conhecimento que o alvo tem em relação ao testador. A divisão é realizada de acordo com a Figura 6.

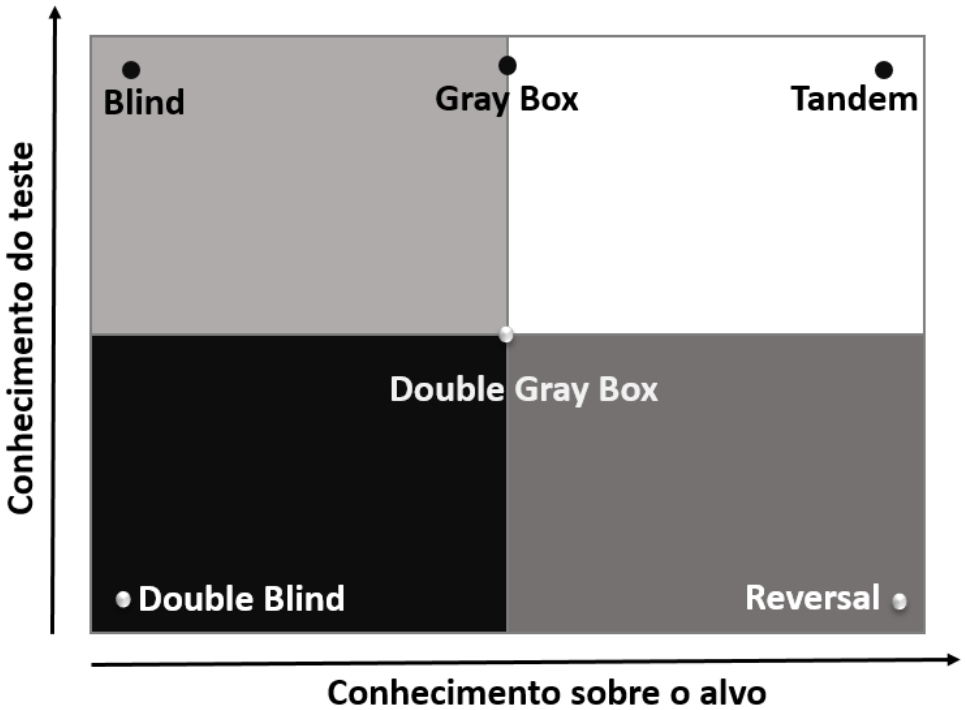

Figura 6: Tipos de Testes de Segurança
Fonte: Adaptado de OSSTMM[24]

A Figura 6 apresenta a divisão dos tipos de testes de segurança a partir da relação entre o conhecimento do atacante sobre o alvo e o conhecimento do alvo em relação aos testes que serão realizados. Os seis tipos de testes são:

- *Bind* – O testador não tem nenhum conhecimento prévio do alvo, porém o alvo tem consciência de todos os detalhes do teste.
- *Double Bind* – É semelhante ao *bind*, porém tanto o testador quanto o alvo não tem nenhum conhecimento prévio do teste. É chamado também de teste caixa-preta (*black-box*) ou teste de penetração.
- *Gray Box* – O testador tem conhecimento parcial do alvo, mas tem pleno conhecimento dos seus canais. Pode ser chamado também de teste de vulnerabilidade e geralmente é realizado pelo próprio alvo para um auto avaliação.
- *Double Gray Box* – É semelhante ao *gray box*, porém o alvo tem conhecimento sobre o escopo e a duração do teste e nenhum conhecimento dos canais testados. É chamado também de teste caixa-branca (*white-box*).
- *Tandem* – O testador e o alvo são preparados para ter conhecimento dos detalhes do teste. É chamado também de Teste cristal-box (*crystal-box*).
- *Reversal* – O testador tem pleno conhecimento sobre o alvo, porém o alvo não tem nenhum conhecimento sobre o que, como e quando o teste será realizado. É chamado também de teste *red-team*.

A metodologia também disponibiliza uma classificação dos tipos de erros encontrados em um teste e um conjunto de métricas, chamado RAVs (*Risk Assessment Values*), para mensurar o estado de segurança e para demonstrar alterações do nível de segurança ao longo do tempo.

O OSSTMM 3.0 apresenta uma documentação organizada que facilita a compreensão do que é proposto. É muito abrangente, pois a preocupação com toda a segurança operacional o torna completa. Outra vantagem é a sua neutralidade de utilização, pelo fato de se adequar a diferentes cenários, organizações e tecnologias.

Destaca-se como desvantagem a falta de modelagem para casos de testes de descoberta de vulnerabilidades e a utilização de ferramentas que podem auxiliar nos testes.

3.1.2 ISSAF (*Information System Security Assessment Framework*)

O ISSAF [25] é uma metodologia utilizada para a realização de análise e testes de segurança com ênfase ao teste de penetração. Ela é mantida pela OISSG (*Open Information Systems Security Group*) e está na versão 0.2.1B que foi disponibilizada no ano de 2006.

Conforme o ISSAF [25] a metodologia é dividida em três fases e nove passos de avaliação realizados em sua respectiva fase. As três fases são:

1. Planejamento e Preparação (*Planning and Preparation*) – É realizado o planejamento do teste com a definição do escopo, dos aspectos legais, do tempo de duração e de outras definições.
2. Avaliação (*Assessment*) – Fase em que é realizado os testes de penetração. Usa uma abordagem em camada no qual representa um nível de acesso aos ativos, as camadas são classificadas em:
   - Coleta de informações;
   - Mapeamento de Rede;
   - Identificação de Vulnerabilidades;
   - Penetração;
   - Acesso e Escala de Privilégio;
   - Enumeração;
   - Comprometer usuários remotos;
   - Manutenção de acesso;
   - Cobrindo rastros.
3. Relatório, Limpeza e Destruição de artefatos (*Reporting, Clean-up and Destroy Artefacts*) – É criado um relatório final dos testes realizados e todos os artefatos e informações criadas e armazenados no sistema criados durante a avaliação devem ser apagados.

O ISSAF possui uma documentação bastante extensa com um nível de detalhamento alto. Sua abordagem em camadas deixa a metodologia bem organizada e objetiva. É abrangente nos tipos de testes e descreve com clareza os testes, relacionando com as ferramentas que podem ser utilizados e os procedimentos a serem realizadas.

A principal desvantagem da metodologia é a falta de atualização, pois a última publicação foi realizada no ano de 2006. Isso prejudica e limita a cobertura dos testes, pelo fato de que novas vulnerabilidades e ataques são criados ao longo do tempo.

3.1.3 NIST SP 800–115

A metodologia intitulada de *"Technical Guide to Information Security Testing and Assenssment"* é mantida pelo NIST (*National Institute of Standards and Technology*) e teve sua última publicação no ano de 2008 [21].

O objetivo da metodologia é fornecer orientações das técnicas para a realização de testes e avaliações de segurança, análise de resultados e estratégias de mitigação. Ela fornece apenas uma visão geral dos principais elementos das técnicas [21].

A metodologia propõe quatro recomendações para a realização de testes de segurança nas organizações:

- Estabelecer uma política de avaliação de segurança da informação;
- Implementar uma metodologia de avaliação repetível e documentado;
- Determinar os objetivos de cada avaliação de segurança;
- Analisar os resultados e desenvolver técnicas de mitigação de risco para corrigi-las.

A estrutura para a realização de avaliação de segurança recomendada pela metodologia é dividida em três fases:

1. Planejamento – Deve ser realizado a coleta de informações que serão necessários para a avaliação, que inclui os principais ativos que serão avaliados, as possíveis ameaças e os principais controles de segurança implementados.
2. Execução – Aborda os métodos da avaliação de segurança com o objetivo identificar possíveis vulnerabilidades.
3. Pós Execução – Tem o objetivo de criar o relatório final com as vulnerabilidades encontradas, fornecendo as causas e recomendações de mitigação.

O NIST SP 800–115 é uma metodologia com uma documentação objetiva e padronizada, com recomendações concretas para os testes e análises de segurança. A desvantagem da metodologia é que ela fornece apenas uma visão geral do que é proposto, não apresenta um direcionamento mais claro para a realização dos processos de testes. Outra desvantagem é que ela não considera o fator humano para os casos de testes.

3.1.4 PTES (*Penetration Testing Execution Standard*)

A metodologia PTES [26] está na versão v1 e é composta por sete sessões relacionadas a um teste de penetração. Ela tem o objetivo de obter a melhor compreensão da segurança do alvo testado, através da busca por vulnerabilidades, da exploração e da pós-exploração. As sessões exploradas pela metodologia são:

1. Interações Iniciais (*Pre-engagement interactions*) – Definição e apresentação das técnicas e ferramentas que serão utilizados no teste.

2. Coleta de Inteligência (*Intelligence Gathering*) – É realizado coleta de informações para ser usadas na fase de avaliação.

3. Modelagem de Ameaças (*Theat Modeling*) – Define uma abordagem para a modelagem de ameaças, porém a metodologia dá a liberdade para que seja usado outros modelos.

4. Análise de Vulnerabilidades (*Vulnerability Analysis*) – É feito a identificação e avaliação dos riscos de segurança.

5. Exploração (*Exploitation*) – Tem o objetivo de ter acesso ao alvo, fraudar os controles de segurança.

6. Pós-execução (*Post Exploitation*) – Tem o objetivo de identificar e documentar dados importantes que poderão ser utilizados em uma análise posterior.

7. Relatório (*Reposting*) – Define critérios necessários para o testador criar o relatório de teste.

O PTES apresenta um catálogo com ferramentas comerciais e open source que podem ser usados para auxiliar em um teste de penetração e demonstra a utilidade das ferramentas e das técnicas. Outra vantagem é a modelagem dividida em fases que deixa a execução da metodologia organizada. Sua desvantagem é que a docomentação está desatualizada.

3.1.5 *Owasp Testing Guide*

A metodologia *Owasp Testing Guide* [27] é mantida pela *OWASP Fundation*, teve sua primeira versão publicada em 2003. Atualmente é liderado por Meucci e Muller, além de outros colaboradores. Está na versão 4.0 publicado em 2014 e é distribuído sob a licença *Creative Commons 2.5*.

É abordado na metodologia assuntos sobre pré-requisitos de segurança em aplicações *Web*, princípios de técnicas de testes, técnicas e tarefas para realizar um ciclo de vida de desenvolvimento de *software* seguro com ênfase ao *framework* proposto pelo guia e sobre os testes de vulnerabilidades por inspeção de código e testes de penetração.

Para Meucci e Muller[27] o propósito da metodologia é disponibilizar uma visão geral sobre testes em aplicações *Web* de forma a ajudar a entender o quê, o porquê, quando, onde e como executar os testes.

A metodologia sustenta a ideia de que a melhor forma de evitar erros de segurança em uma aplicação é melhorar o ciclo de vida do desenvolvimento de *software*, com a

inclusão de segurança em cada uma das fases. Sustenta também que um programa de teste de segurança eficaz deve incluir as componentes: pessoas, processos e tecnologia.

Para os testes de segurança é apresentado quatro técnicas que podem ser usados em um plano de teste. As técnicas são: inspeção manual e revisões, modelagem de ameaças, revisões de código fonte e teste de penetração.

O *framework* de testes proposto na metodologia define quais as técnicas e tarefas mais apropriadas para cada fase do ciclo de vida do desenvolvimento de *software*.

A metodologia é baseada na abordagem caixa-preta ou *black-box*, ou seja, o testador assume que não há nenhum conhecimento a priori sobre o ambiente o qual será aplicado o teste [27]. Ao contrário da abordagem caixa-branca ou *white-box* em que o testador detém o completo conhecimento sobre a infraestrutura a ser testada.

A metodologia é dividida em duas fases. Na primeira fase chamada de modo passivo é realizado um levantamento de informações sobre a aplicação alvo do teste para o testador entender melhor a lógica da aplicação. Na segunda fase chamado de modo ativo é feito a aplicação dos testes segundo a metodologia. Os testes realizados na segunda fase foram divididos em 11 categorias:

1. Coleta de Informação (*Information Gathering*)
2. Gerenciamento de configuração (*Configuration and Deployment Management*);
3. Gerenciamento de Identidade (*Identity Management*);
4. Autenticação (*Authentication*);
5. Autorização (*Authorization*);
6. Gerenciamento de sessão (*Session Management*);
7. Validação de dados de entrada (*Input Validation*);
8. Erro de manipulação (*Error Handling*);
9. Criptografia (*Cryptography*);
10. Lógica de negócios (*Business Logic*);
11. Client Side (*Client Side*).

O metodologia *Owasp Testing Guide* apresenta uma documentação clara, organizada e estruturada, o que facilita a sua compreensão. A descrição dos processos de testes

é claramente apresentada e demonstrada com as devidas ferramentas e técnicas. É apresentado também um aparato completo para os possíveis resultados, soluções e medidas de mitigação.

3.2 Técnicas de Testes de Segurança

Uma metodologia provê técnicas e ferramentas para testes. Técnicas de testes de segurança descrevem o estado da arte para a execução dos testes, ou seja, a forma como será conduzido os procedimentos dos testes. Enquanto que as ferramentas são os meios que os testes podem ser realizados.

Liu et al. [28] destaca a transição das técnicas realizadas manualmente ou com o auxílio do computador, para técnicas totalmente automatizadas. Um plano de teste pode empregar diferentes tipos de técnicas e de análises de segurança para alcançar os objetivos propostos. O NIST [21] categoriza em três tipos de técnicas:

- Técnicas de Revisões – Avaliação realizado através da análise manual para descobrir vulnerabilidades de segurança. A revisão pode ser realizada através da documentação, dos *logs*, das configurações do sistema e da rede.
- Técnicas de Análise – É realizado manualmente ou principalmente utilizando ferramentas automatizadas. Tem o objetivo de identificar portas, serviços e vulnerabilidades do sistema alvo. Inclui técnicas de descoberta de rede, identificação de portas e serviços, varredura de vulnerabilidades, varredura da rede sem fio e análise de segurança.
- Técnicas de Validação de Vulnerabilidades – É realizado manualmente ou principalmente utilizando ferramentas automatizadas. Tem o objetivo de confirmar a presença de vulnerabilidades. Inclui técnicas de quebra de senha, testes de penetração e engenharia social.

Um Teste de Penetração ou *Pentest*, corresponde a uma técnica que procura realizar uma tentativa de invasão legal. Seu objetivo é explorar uma vulnerabilidade existente em uma aplicação e evidenciar os problemas aos quais uma determinada aplicação estar suscetível, possibilitando que as falhas descobertas através da realização dos testes, possam ser resolvidas antes que uma pessoa maliciosa consiga explorá-las [12].

A *Owasp* recomenda a agregação de segurança para todas as fases do desenvolvimento de *software*, ou seja, integrar diferentes tipos de técnicas de testes em cada fase do ciclo de vida de desenvolvimento de *software* seguro [27].

3.3 Ferramentas de Testes de Segurança

Para a realização de testes de segurança, é necessário um conjunto de ferramentas para a exploração de potenciais vulnerabilidades sobre a aplicação a ser testada. A utilização de ferramentas automatizadas para a execução de testes facilita o trabalho do testador e diminui o tempo de realização dos testes.

De acordo com Uto [4], os tipos de ferramentas utilizadas em testes de segurança são classificados em:

- Navegadores *Web* – Principal meio de acesso às aplicações *Web* e por isso é fundamental para a realização de testes de penetração. Os principais navegadores são: *Internet Explore, Mozila Firefox, Google Chrome e Safari.*
- *Proxies* de Interceptação – Permite inspecionar e alterar requisições e respostas HTTP em tempo real;
- *Web Spiders* – Tem o objetivo de montar automaticamente o mapa da aplicação;
- *Fuzzers* – Utilizados para identificar falhas no tratamento e validação de informações;
- Varredores de Portas e Serviços – Utilizados para analisar e identificar portas e serviços disponíveis em determinada máquina;
- Varredores de Vulnerabilidades – Utilizados para encontrar automaticamente vulnerabilidades em determinado alvo.

O trabalho de Bertoglio e Zorzo [22] identifica 13 ferramentas que tiveram destaque em citações de artigos científicos relacionadas a testes de penetração. As ferramentas apresentadas foram: *Acunetix WVS, Burp Suite, WebInspect, AppScan, Metasploit, Metasploit, NeXpose, Nikto, Nmap, Paros, QualysGuard, WebScarab e Wireshark.*

A seguir, ferramentas utilizadas em testes de segurança são apresentadas. As ferramentas estão subdivididas nas categorias "Coleta de informações"e "Análise e exploração de vulnerabilidade".

Ferramentas Para Coleta de Informações

Como visto nas princípais metodologias para teste de segurança apresentadas na Seção 3.1, geralmente a primeira fase a ser realizada em um teste de segurança de uma aplicação *Web* é a coleta de informações. É preciso reunir informações sobre a aplicação que será avaliada. Para isso, diversas ferramentas podem ser usadas, tais como:

mecanismos de busca, *scanners*, ferramentas para envio de requisições HTTP, entre outras. Estas ferramentas têm seu foco de utilização para as etapas iniciais de um *Pentest*.

A Tabela 1 apresenta ferramentas relevantes para a coleta de informações bem como uma breve descrição.

Tabela 1: Ferramentas para coleta de informações.

Ferramenta	**Descrição**
Netcat [29]	O netcat é uma ferramenta de rede *open source*. É usada para ler e escrever dados em conexões de rede usando o protocolo TCP/IP. É considerado o canivete suíço do TCP/IP, pois pode ser usado para fazer desde análise de porta até ataque por força bruta.
Httprecon [30]	Httprecon é uma ferramenta para impressões digitais de servidor *Web*. O objetivo é a identificação de determinadas informações do httpd.
SET [31]	O SET (*Toolkit de Social-Engineer*) Trata-se de uma ferramenta *open-source Python* para testes de penetração. É utilizado para a coleta de informações por meio da engenharia social.
Whois [32]	Ferramenta para a coleta de informações sobre o proprietário de um domínio e outras informações.
DIG [33]	O DIG (*Domain Information Groper*) é uma ferramenta para coletar os nomes de DNS de determinado domínio ou *host*.
Maltego [34]	O Maltego é uma ferramenta que coleta informações de varias fontes publicas e relaciona os dados coletados para análise. É uma plataforma única que mostra todo o ambiente de uma organização ou rede.

Ferramentas Para Análise e Exploração de Vulnerabilidades

Após realizar a coleta de informações na primeira fase, é realizado uma análise das informações que podem ser importantes para os próximos testes. Em seguida, é posto em prática os testes que farão a identificação e exploração de possíveis vulnerabilidades.

A tabela 2 apresenta ferramentas que tem seu foco de utilidade para a análise, identificação e exploração de vulnerabilidades em aplicações *Web*.

Tabela 2: Ferramentas para análise e exploração de vulnerabilidades.

Ferramenta	**Descrição**
Nmap [35]	O Nmap (*Network Mapper*) é uma ferramenta para varredura de portas e serviços em uma rede ou em um *host*. Ele lista uma tabela de portas interessantes com o número das portas, o estado e os serviços disponíveis.
Nessus [36]	É um *scanner* de vulnerabilidades para aplicações *Web*. Ele realiza a varredura de portas, detectando servidores ativos e simulando invasões para detectar vulnerabilidades. Ele é composto por um cliente e um servidor, sendo que o *scan* propriamente dito é feito pelo servidor.
Nikto [37]	É uma ferramenta desenvolvida em *Perl*, cujo objetivo é fazer varredura de vulnerabilidades em servidores *Web*. Foi desenvolvida para encontrar diversos tipos de arquivos, configurações e programas padrões ou inseguros, em servidores *Web*.
Acunetix WVS [38]	É um scanner de vulnerabilidades para aplicações *Web*. Ele verifica automaticamente vulnerabilidades como injeções SQL, *cross site scripting*e senha fraca em páginas de autenticação.
AppScan [39]	É um *scanner* de vulnerabilidades para aplicações *Web* e *mobile*. Ele permite identificar vulnerabilidades de segurança e gerar relatórios e recomendações de correção.
NeXpose [40]	Um *scanner* de vulnerabilidades que acompanha todo o ciclo de vida de gerenciamento de vulnerabilidades, incluindo descoberta, detecção, verificação, classificação de risco, análise de impacto, relatórios e mitigação.
Burp Suite [41]	É uma plataforma para a realização de testes de segurança em aplicações *Web*. Ele possui ferramentas do tipo *Proxie* de interceptação, *Web Spider*, Varredor de vulnerabilidades, Ferramenta de penetração, Ferramenta de sequenciamento para teste de identificadores de sessãos.
WebInspect [42]	Uma ferramenta para testes de penetração em aplicações *Web*. Realiza a análise de vulnerabilidades de segurança em aplicações Web.

Continua na próxima página.

Tabela 2 – *Continuação da página anterior.*

Ferramenta	**Descrição**
Metasploit [43]	É um *Framework open source* para testes de penetração. O *metasploit* é uma ferramenta com o objetivo de analisar vulnerabilidades de segurança, facilitar testes de penetração e desenvolver assinaturas para sistemas de detecção de intrusos.
Paros [44]	É um *proxy* HTTP/HTTPS criado em *Java*, utilizado para analisar vulnerabilidade em aplicações *Web*. A Ferramenta apresenta funcionalidades de *Web Spider*, Varredor de vulnerabilidades, *Proxie* de Interceptação e injeções SQL, entre outras funcionalidades.
WiresShark [45]	É uma Ferramenta do tipo *Proxie* de interceptação. O *Wireshark* analisa o tráfego de rede e organiza os pacotes por protocolos. As funcionalidades são apresentadas por uma interface gráfica. Através dele é possível controlar o tráfego de uma rede e monitorar a entrada e saída de dados do computador, em diferentes protocolos, ou da rede à qual o computador está conectado.
WebScarab [46]	Ferramenta do tipo *Proxie* de interceptação, criado pela *Owasp*. O WebScarab é um *framework* para analisar aplicações *Web*. Ele é escrito em Java, e portanto, multiplataformas. O WebScarab possui vários modos de operação implementados por *plugins*. Opera como um *proxie* de interceptação, permitindo que visualize e modifique solicitações criadas pelo navegador *Web* antes de serem enviadas para o servidor e vice-versa. WebScarab é capaz de interceptar a comunicação HTTP e HTTPS.

3.4 Conclusão

Este capítulo, abordou de forma pontual o conteúdo referente a testes de segurança. Foram apresentadas as principais metodologias, técnicas e ferramentas para testes de segurança.

Entre as metodologias apresentadas a mais atual e com maior abrangência nos

testes é a metodologia da *Owasp*. Por ser a metodologia mais atualizada e pela sua completude na descrição de cada caso de teste, a *Owasp Testing Guide* é a mais adequada para a utilização em testes de segurança em aplicações *Web* e é a que será utilizada neste trabalho.

CAPÍTULO 4

Estudo de Caso: Testes de Segurança em uma Aplicação Web

Este capítulo apresenta como estudo de caso testes de segurança realizados em uma aplicação Web. Ele objetiva identificar vulnerabilidades na aplicação. Para isso foram realizados testes de segurança sobre a aplicação utilizando ferramentas de testes de segurança.

Este capítulo está organizado da seguinte forma: A Seção 4.1 apresenta a aplicação objeto de teste. A Seção 4.2 especifica a metodologia do teste, com informações gerais sobre os testes, a metodologia de teste de segurança utilizada, as ferramentas utilizadas, os ataques realizados e o ambiente de teste. A Seção 4.3 apresenta os resultados dos testes realizados na aplicação.

4.1 A Aplicação Web Objeto de Teste

Para este estudo de caso, será testada uma aplicação *Web* que está em pleno funcionamento no ambiente de produção de uma organização. O nome e o endereço de acesso à aplicação serão mantidos em sigilo por questão de segurança.

A aplicação *Web* objeto de teste é um portal que tem o objetivo de promover o acesso e maior visibilidade às revistas científicas publicadas. O portal utiliza o sistema eletrônico de editoração de revistas (SEER) na versão 2.4.8.0, utilizado para a construção e gestão de publicação periódica eletrônica.

A aplicação apresenta páginas *Web* para visualizar e cadastrar trabalhos em revistas cientificas. Os trabalhos submetidos passam por um processo de avaliação para serem publicados no portal. A submissão é realizada por preenchimento de formulários e o *upload* de arquivos na aplicação. No cadastro, são definidos metadados para serem utilizados na indexação do arquivo e facilitar a pesquisa dentro do portal.

Para submeter trabalhos, o usuário deve estar cadastrado no sistema. O cadastro de usuário é realizado por meio de um formulário http. O usuário pode escolher entre três opções de perfis: leitor (notificado via *e-mail* da publicação de novos trabalhos), autor (pode submeter à revista) e avaliador (pode avaliar as submissões).

A aplicação apresenta mecanismos para o usuário, como alterar senha, lembrar senha, editar perfil e pesquisar revistas. No processo para alterar ou lembrar senha, a aplicação enviar um *e-mail* ao usuário para poder realizar a operação.

4.2 Metodologia

A aplicação objeto de teste foi submetida a testes de penetração. Um teste de penetração, conforme descrito na Seção 3.2, é um método de teste de segurança que tem a finalidade de buscar por possíveis vulnerabilidades em potencial para que sejam corrigidas a tempo de evitar a exploração por atacantes reais com intenções maliciosas.

Os testes foram aplicados em uma aplicação *Web* real, conforme apresentado na Seção 4.1. A aplicação foi autorizada para os testes pelo gestor de tecnologia da informação.

Foi concedido apenas a URL *(Uniform Resource Locator)* de acesso à página inicial da aplicação para a realização dos testes. Sendo assim, a categoria do teste é do tipo caixa-preta ou *black-box* (Seção 3.1.5), pois nenhuma informação a mais sobre a aplicação foi fornecida.

Em geral, os testes de segurança são realizados tanto para garantir a adequação aos requisitos mínimos de segurança, quanto para assegurar que a aplicação está protegida contra ataques conhecidos. No entanto, não foram disponibilizados os requisitos de segurança da aplicação. Sendo assim, os testes aqui realizados terão basicamente os seguintes objetivos:

- Realizar o levantamento de informações sobre a aplicação.
- Aplicar testes de segurança em busca por possíveis vulnerabilidades de segurança.
- Identificar e apresentar possíveis vulnerabilidades encontradas nos testes.

Metodologia de teste de segurança utilizada

A metodologia utilizada para os testes será a *Owasp Testing Guide* versão 4, conforme apresentada na Seção 3.1.5. A metodologia possibilita o atendimento dos objetivos proposto neste estudo de caso. Como apresentado na Seção 3.1.5, a metodologia *Owasp* é direcionada para testes de segurança em aplicações *Web* e utiliza a técnica de teste de penetração para a validação de vulnerabilidade.

Os testes são conduzidos através de duas fases, conforme a Figura 7.

Figura 7: Fases dos testes.
Fonte: Criação do autor.

No modo passivo o objetivo é entender o funcionamento da aplicação e suas principais características. Nesta etapa, os principais pontos de acesso da aplicação podem ser conhecidos, tais como: servidor *Web*, parâmetros, *cookies*, entre outras informações.

Enquanto que no modo ativo, é realizado a identificação e exploração de possíveis vulnerabilidades. Nesta segunda etapa, a metodologia de testes de penetração da *Owasp* é utilizada.

Testes Realizados

O propósito deste estudo de caso é identificar as principais vulnerabilidades que as aplicações *Web* enfrentam, incluindo as apresentadas na Seção 2.2: Injeção SQL, quebra de autenticação e sessão, ataques *XSS* e *CSRF*, referências inseguras a objetos, exposição de dados sensíveis, entre outras. Os testes farão uso de ferramentas de testes de segurança.

Os testes decorrem sobre algumas das categorias de teste descritas na metodologia da *Owasp* [27], conforme a Figura 8. As categorias testadas na aplicação são:

- Levantamento de Informações – Testes para coletar informações sobre a aplicação, tais como versão do servidor, metadados, *cookies*, etc.
- Gerenciamento de Configuração – Testes para verificar informações de configuração da aplicação, tais como o *framework* utilizado, o banco de dados, o tipo de servidor *Web*. Tem o objetivo de analisar a configuração dos componentes tecnológicos utilizados, afim de identificar possíveis vulnerabilidades.

Figura 8: Testes realizados.
Fonte: Criação do autor.

- Gerenciamento de Identidade – Testes para verificar as funcionalidades que manipulam as identidades de autenticação de usuários. O objetivo é validar as funções de identidade definidas no sistema, tais os perfis de usuários permitidos, o mecanismo de cadastro, o processo de alteração de senha, etc.

- Autenticação – Testes para verificar os mecanismos de autenticação da aplicação. O objetivo é testar o transporte das credenciais de autenticação, o mecanismo de bloqueio de conta, a força das senhas, a utilização de credenciais padrões, o armazenamento das credenciais, etc.

- Autorização – Testes para verificar o processo de autorização na aplicação. O objetivo é testar se as definições de autorização são corretamente aplicadas para os usuários com a devida permissão.

- Gerenciamento de Sessão - Testes para verificar o processo de gerenciamento de sessão da aplicação.

- Validação de dados de entrada – Testes para verificar a validação da entrada de dados na aplicação. O objetivo é testar se a aplicação valida e trata os dados de entrada antes de usá-los.

Ferramentas Utilizadas nos Testes

Na Seção 3.3, foram apresentas diversas ferramentas para teste de segurança. Para a realização deste estudo de caso, algumas ferramentas foram utilizadas. As ferramentas foram escolhidas de acordo com a demanda de cada caso de teste. Elas tiveram grande importância para este trabalho, pois possibilitaram atingir o objetivo proposto nos testes.

As seguintes ferramentas foram utilizadas:

- *Netcat* – Identificação de servidor *Web*;
- *Nmap* e *Nikto* – Enumeração de aplicações no servidor;
- *Wget* – Análise de comentário em páginas *Web*;
- *Whatweb* – Coletar informações da aplicação;
- *ZAP* – Scaneamento de vulnerabilidades;
- *Wirehark* – Análise de tráfego de rede;
- *THC Hydra* – Ataque de força bruta.

Ambiente de Teste

Os testes foram realizados em rede externa do servidor da aplicação objeto de teste. Para este estudo de caso, foram utilizados os itens descritos na Tabela 3:

Tabela 3: Ambiente de teste.

Item	**Finalidade**
Notebook Asus *(Intel Core i3-2365M CPU 1.40GHz x 4)* com sistema operacional *Ubuntu* 16.04 LTS	Máquina utilizada para realizar os testes de segurança.
Mozilla Firefox versão 52.0.2 (64-bit)	Navegador *Web* utilizado para acessar a aplicação.

4.3 Resultados

Informações detalhadas sobre a aplicação e dados privados colhidas ao longo dos testes, ficarão sobre sigilo para evitar a exposição da aplicação. Apenas dados públicos e que não fazem relação direta com a identidade da aplicação serão descritos para dar sustento ao trabalho, por exemplo, as possíveis vulnerabilidades encontradas, mensagens de erros, os testes realizados etc. Nas próximas seções são apresentados os testes realizados.

4.3.1 Primeira Fase: Modo Passivo

A primeira fase para os testes de segurança está focada em obter o máximo de informações possíveis sobre a aplicação a ser testada. A coleta de informações é um passo necessário para um teste de penetração, visto que, além de obter informações sensíveis da aplicação, também pode ser possível descobrir antecipadamente possíveis vulnerabilidades.

Esta tarefa pode ser realizada de diversas maneiras. A principal forma de se realizar é por meio utilização de ferramentas. Por exemplo, ferramentas públicas (buscadores *Web*), *proxy* HTTP, varredores ou por solicitações HTTP criadas para forçar o vazamento de informações, divulgando mensagens de erros ou obtendo as versões e tecnologias utilizadas. A coleta de informações inclui os seguintes testes:

Teste de Identificação do Servidor *Web*

Uma das primeiras tarefas ao conduzir um teste de penetração em aplicações *Web* é tentar identificar a versão do servidor *Web* no qual está hospedada. A identificação do servidor utilizado permite descobrir possíveis vulnerabilidades já conhecidas que podem afetar o servidor em questão [27].

O objetivo deste teste é realizar a identificação do servidor *Web* em execução para que o testador conheça a versão e o tipo do servidor. Essa informação, aparentemente inofensiva, pode ser útil para o atacante, que pode buscar por potenciais vulnerabilidades e ataques específico para versão utilizada.

Esse teste pode ser feito automaticamente por ferramentas de segurança ou manualmente. Na forma manual, pode ser realizado com a utilização de diferentes utilitários, tais como a Telnet ou o Netcat. Na forma automatizada pode ser realizado com a utilização de ferramentas como *HTTPrint, HTTPrecon, Netcraft, Desenmascarame.*

Foi utilizada a ferramenta *Netcat* (Seção 3.3). Ela é uma ferramenta simples e de fácil utilização por meio de linha de comando no *Linux*. Foi possível também visualizar a

informação da versão do servidor através da tela de erro na aplicação, como apresentado na Figura 9.

Figura 9: Versão do servidor web.
Fonte: Print da aplicação.

O teste obteve como resultado o tipo do servidor *Web* utilizado pela aplicação. Ela está hospedada em um servidor *Web Apache*/2.4.10 com sistema operacional *Linux*. A versão estável mais atual do servidor *Apache* é a 2.4.25 (*released* 2016-12-20). Sendo assim, o servidor da aplicação está desatualizado e que, por sua vez, apresenta vulnerabilidades de segurança, o que caracteriza uma configuração incorreta de segurança, como apresentado na Seção 2.2.5.

Teste de Análise de Meta Arquivos do Servidor *Web*

Este teste decorre sobre o arquivo *robots.txt* com o propósito de identificar possível vazamento de informações sobre diretórios ou endereço de pastas da aplicação *Web*. O arquivo *robots.txt* é um padrão usado por *sites* para se comunicar com robôs de busca. Os robôs de busca (também conhecido como *Web Wanderers, Crawlers ou Spiders*), são programas que trafegam pela *Web* automaticamente [47].

O arquivo *robots.txt* tem o objetivo de fazer um filtro do que pode ser indexado pelos motores de buscas, por exemplo o Google. Nem todos os robôs de busca seguem o padrão do arquivo, pois *spam bots, malware* e varredores de vulnerabilidades de segurança podem não seguir o padrão. O arquivo *robots.txt* se encontra na raiz do *site*, disponível ao acesso público. Portanto, não é recomendável usar *robots.txt* para esconder informações.

No teste, foi utilizado apenas um navegador *Web* e foi encontrado o arquivo, mas não continha informações relevantes. No arquivo, foi definido apenas as duas linhas abaixo, o que significa que somente o diretório */cache* está habilitado.

```
User-agent:  *
Disallow:  cache/
```

Teste de Enumeração de Aplicações no Servidor *Web*

Esse teste visa identificar as aplicações hospedadas no mesmo servidor *Web*. Isso é importante, visto que pode ser encontrado versões antigas de arquivos ou artefatos não excluídos, *scripts* obsoletos criados durante a fase de desenvolvimento ou em alguma manutenção. Além de poder revelar aplicações de uso administrativos que podem ser exploradas pelo atacante.

A ferramenta utilizada para o teste foi o *Nikto*. No resultado, foi identificado 31 *hosts* ativos hospedados. Foi realizado a verificação de todas as 31 aplicações ativas, porém elas não fazem relação direta com a aplicação em teste e não revelam nenhum tipo de informações sensíveis.

Teste de Análise de Comentários e metadados em Páginas *Web*

Esse teste tem o objetivo de revisar os comentários e metadados na aplicação *Web* para encontrar vazamento de informações sensíveis. Ferramenta como o *Wget* e a visualização de código fonte pelo navegador podem ser utilizadas para este teste.

Sendo assim, para realizar o teste foi utilizado a ferramenta *Wget* para baixar os arquivos da aplicação e ser feito a revisão nos arquivos em busca de informações sensíveis, como comentários com senhas. Nenhuma informação sensível foi encontrada.

Teste *Fingerprint* da Aplicação *Web*

Esta verificação é para definir o tipo de *framework Web* e a versão utilizada e determinar possíveis vulnerabilidades conhecidas. As ferramentas que podem ser utilizadas para este teste são *WhatWeb, BlindElephant e Wappalyzer*. Entretanto, a ferramenta utilizada foi a *WhatWeb*.

As informações obtidas, como visto na Figura 10, são que a aplicação é composta por *Jquery 1.4.4, Jquery UI 1.8.6*, Linguagem de programação PHP e Sistema de gestão de documentos *Open Journal Systems 2.4.8.0.*.

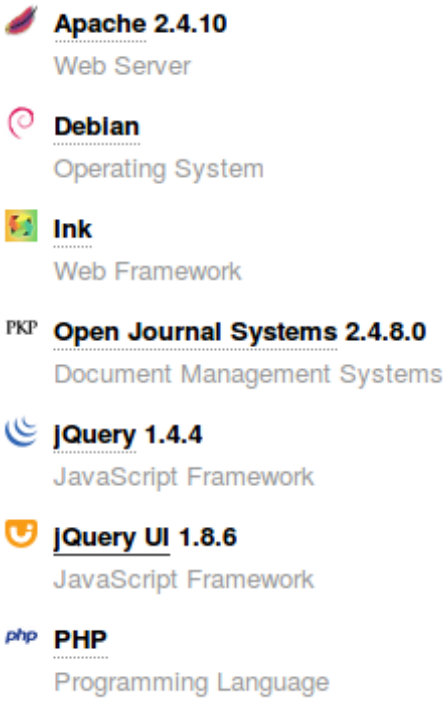

Figura 10: Fingerprint da aplicação *Web*.
Fonte: Print da aplicação.

4.3.2 Segunda Fase: Modo Ativo

A fase no modo ativo dá início aos testes utilizando a metodologia *Owasp* [27] para a identificação e possível exploração de vulnerabilidades encontradas. Os testes ativos estão divididos em subcategorias apresentadas a seguir:

Gerenciamento de Configuração

A infraestrutura em que a aplicação *Web* se encontra pode conter informações importantes sobre ela. Nesta categoria de teste deve ser criado um mapa da aplicação, que reflita a estruturação dos arquivos componentes, as funcionalidades ofertadas e as tecnologias utilizadas para checar tais configurações [27].

Segundo o *Owasp Top 10* [3], configurações incorretas podem acontecer em qualquer nível da aplicação, incluindo a plataforma, servidor *Web*, *framework* e códigos personalizados. Desenvolvedores e administradores de sistemas precisam trabalhar juntos para garantir que toda a infraestrutura esteja configurada corretamente. *Scanners* automatizados são úteis para detectar falta de atualizações, erros de configuração, uso de contas padrão, serviços desnecessários, etc.

O primeiro teste a ser realizado é analisar os problemas de segurança da configuração da infraestrutura e plataforma da aplicação. A análise de configuração decorre

sobre informações obtidas na fase do modo passivo e tem o objetivo de verificar se algum *software* está desatualizado. Dentre as configurações analisadas a mais relevante, que merece atenção foi a versão do servidor *Web*, visto que está desatualizada. Tal versão do servidor apresenta vulnerabilidades já conhecidas publicamente e pode ser explorada por atacantes reais. É recomendável realizar a atualização do servidor para evitar problemas futuros.

Outra análise importante é a verificação da existência de recursos desnecessários ativados ou instalados, como a presença de arquivos velhos, *backup*, arquivos, serviços ou páginas não referenciados na aplicação que podem expor informações sensíveis. A presença de um recurso desse tipo caracteriza a utilização de componentes vulneráveis conhecidos, como apresentado na Seção 2.2.9.

Foi utilizado a ferramenta *Wget* que obteve uma cópia dos arquivos da aplicação. A Figura 11 mostra o mapa desses arquivos. Na verificação não foram identificados arquivos que comprometam a aplicação, nem dados sensíveis nos arquivos.

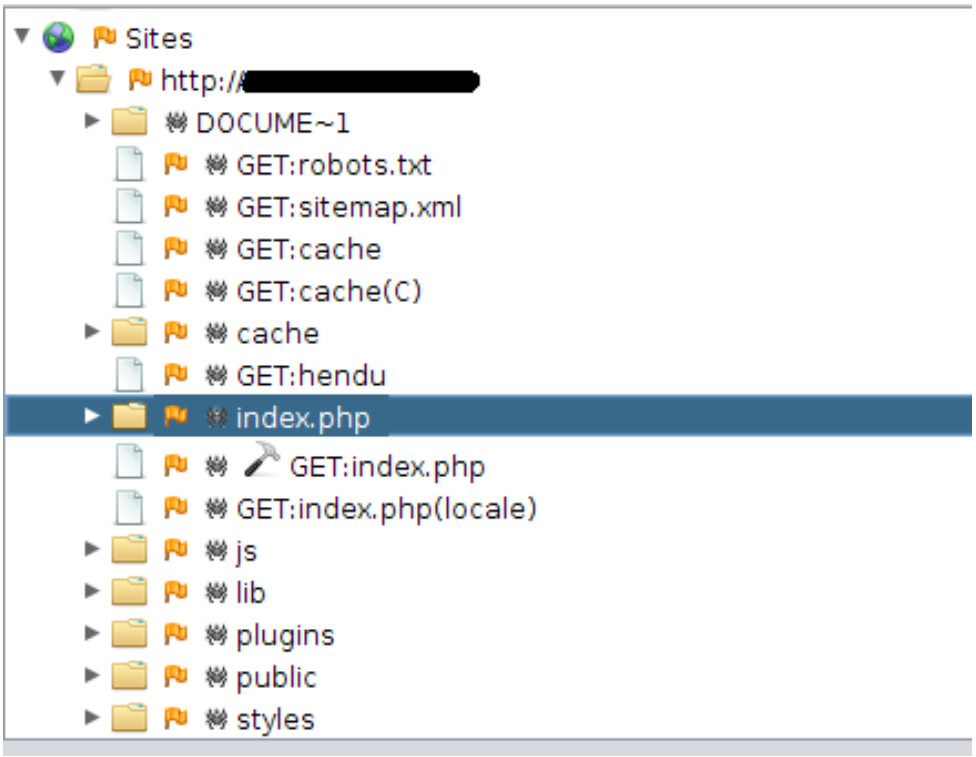

Figura 11: Mapa da Aplicação.
Fonte: Print da aplicação.

A última verificação é testar se a utilização de métodos HTTP está devidamente limitada a usuários confiáveis e com condições de segurança. Os métodos do protocolo HTTP são usados para realizar ações no servidor *Web*. Os mais comuns são os métodos *GET e POST*, que são usados para acessar informações. Recomendações de segurança apontam que os métodos *PUT, DELETE, CONNECT e TRACE* devem ser desabilitados.

Para esse teste foi utilizada a ferramenta *Nikto* e um navegador *Web*. O resultado identificou que o método HTTP *TRACE* está ativo. Este método retorna ao cliente qualquer conteúdo que foi enviado ao servidor. Esse método faz com que o *host* se torne vulnerável a ataque XST (*Cross-Site Tracing*), o que carateriza uma configuração incorreta de segurança (Seção 2.2.5). O ataque consegue obter o conteúdo de um *cookie* com a *flag HttpOnly* graças ao método *TRACE* habilitado no servidor *Web*.

Gerenciamento de identidade

Inicialmente, é realizado a verificação dos perfis válidos no sistema definidos dentro da aplicação para separar cada perfil de usuários do sistema para gerenciar acesso adequado a funcionalidade e informações do sistema. Esta análise é realizada de forma manual através dos tipos de perfis aceitos no cadastro de novos usuários. Foi identificado três opções de perfil de cadastro, conforme a Tabela 4.

Tabela 4: Perfis de usuários da aplicação

Perfil	Descrição
Leitor	Apenas leitor na aplicação.
Autor	Pode submeter artigos à aplicação.
Avaliador	Pode realizar avaliação das submissões de artigos.

Após verificado os perfis permitidos de usuários, é testado o processo de registro de novos usuários. O objetivo é verificar se os requisitos de identidade para registo são alinhados com requisitos de segurança. Para isso, é preciso analisar manualmente o processo de registo da aplicação.

Verificou-se que o processo de registro solicita dados satisfatórios para o cadastro de novos usuários. Além disso, foi observado que a aplicação não permite a utilização duplicidade de nome de *login* e do endereço de *e-mail* já cadastrados no sistema, o que garante um processo de registro seguro.

A terceira análise feita acerca do gerenciamento de identidade foi em relação a possibilidade de identificar nomes de usuários válidos através da enumeração pelo mecanismo de autenticação. O objetivo é verificar se é possível obter nomes de usuários válidos, interagindo com o processo de autenticação da aplicação, o que pode ser utilizado para burlar o mecanismo (Seção 2.2.2) e obter o acesso não autorizado. Este teste pode ser útil para testar a força bruta, em que o dispositivo de teste verifica se, dado um nome de utilizador válido, é possível encontrar senha.

A análise se deu de forma manual através da tentativa de *login* incorreto para forçar a visualização da mensagem de erro. O mecanismo não emite informações diferentes

quando recebe credenciais inteiramente erradas ou quando recebe apenas a senha errada. A única mensagem exibida pode ser vista na Figura 12. Foi verificado que a mensagem está de acordo com o padrão de segurança, pois não permite a enumeração de usuário.

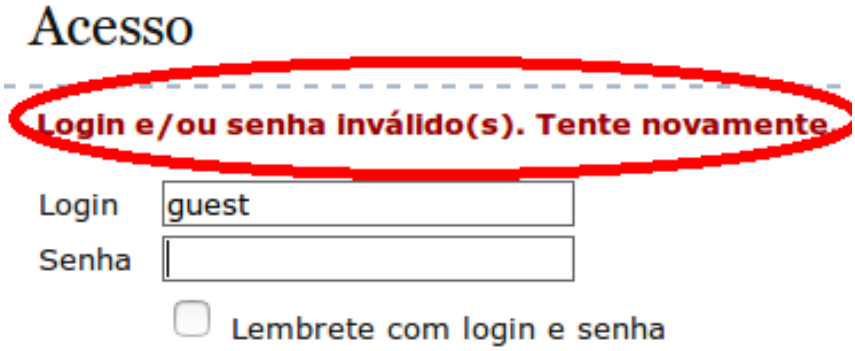

Figura 12: Mensagem de erro no login.
Fonte: Print da aplicação.

Porém, foi possível verificar que outros mecanismos permitem a enumeração. O mesmo teste foi realizado no mecanismo de autenticação. O objetivo é o mesmo, determinar se as mensagens de erro permitem enumerar identidades de usuários.

Com o teste, constatou-se que é possível enumerar identificador de usuário (*e-mail*) através do mecanismo de recuperação de senha e pelo o mecanismo de cadastro de novos usuários, o que pode colaborar com a quebra de autenticação da aplicação, como apresentado na Seção 2.2.2. A Figura 13 ilustra as mensagens de erro que são mostradas no cadastro de novos usuários.

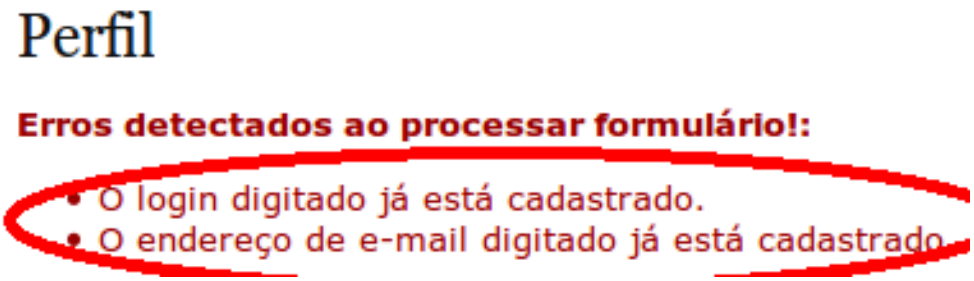

Figura 13: Mensagem de erro no mecanismo de cadastro.
Fonte: Print da aplicação.

Autenticação

Um dos principais requisitos de segurança da informação em aplicações *Web* é o mecanismo de autenticação de usuários. O mecanismo é responsável por garantir que a comunicação é autêntica, isto é, assegurar que a entidade se comunicando é aquela que ela afirma ser [1].

De acordo com Uto [4], o mecanismo de autenticação mais utilizado em aplicações *Web* é através da verificação do identificador de usuário e senha, em que a autenticação é feita pelo banco de dados da aplicação. A principal abordagem utilizada para implementar esse tipo de autenticação é por meio da autenticação de formulário. As credenciais são submetidas por formulário HTML e são verificadas pelo servidor.

Pelo o fato das credenciais serem enviadas por meio de formulário ao servidor, se tem a possibilidade de serem transportadas para um destino incorreto ou serem visualizadas ou capturadas no canal de comunicação entre o usuário e o servidor de destino.

Os principais motivos dessas falhas ocorrem devido o transporte das credencias em texto claro ou apenas codificada, através de um canal de comunicação inseguro. Como consequência se tem o risco de a mensagem ser interceptada em qualquer ponto entre a origem e o destino por um atacante mal intencionado.

É importante fazer a verificação do transporte das credencias para verificar se estão sendo enviadas de forma segura. Testar o transporte das credênciais de usuários significa verificar como os dados de autenticação são enviados para o servidor da aplicação. Usar uma conexão criptografada evita que os dados sejam interceptados por usuários maliciosos.

O primeiro teste tem o objetivo de verificar se os dados são transportados sem criptografia a partir do navegador *Web* para o servidor, ou se a aplicação *Web* toma as medidas de segurança apropriadas usando um protocolo como HTTPS (*Hyper Text Transfer Protocol Secure*). No teste, é verificado se os dados inseridos e enviados em formulários *Web* são transmitidos usando protocolos seguros.

O teste é realizada utilizando um *Web proxy* para analisar as mensagens de requisição da aplicação *Web*. É possível identificar inicialmente se a URL está especificando o protocolo HTTP, se sim, então pode-se entender que os dados serão enviados sem criptografia. Na Figura 14 o navegador *Web* identificou automaticamente que a página não é segura, pois não apresenta o protocolo HTTPS. Para um transporte seguro é recomendado o uso de tal protocolo.

Para comprovar e testar esse tipo risco a *Owasp* [27] recomenda o uso das ferramentas como o *WebScarab*, ZAP (*Owasp Zed Attack Proxy*) ou outra ferramenta com função de analisar o tráfego da rede. Por conformidade, foi utilizada a ferramenta *Wireshark*.

No teste, foi verificado e comprovado que o envio das credencias da aplicação é através do método *POST* via HTTP. Os dados são transportados em texto claro pelo o canal de comunicação sem criptografia, o que colabora para a quebra de autenticação (Seção 2.2.2), o que caracteriza a exposição de dados sensíveis (Seção 2.2.6). Um atacante

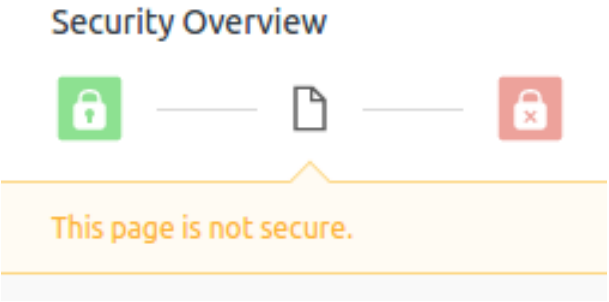

Figura 14: Página insegura.
Fonte: Print da aplicação.

mal intencionado pode interceptar o canal e roubar as informações transportadas. A Figura 15 mostra como é feito o tráfego dos dados na rede, mostrando o método de envio e as credenciais do usuário.

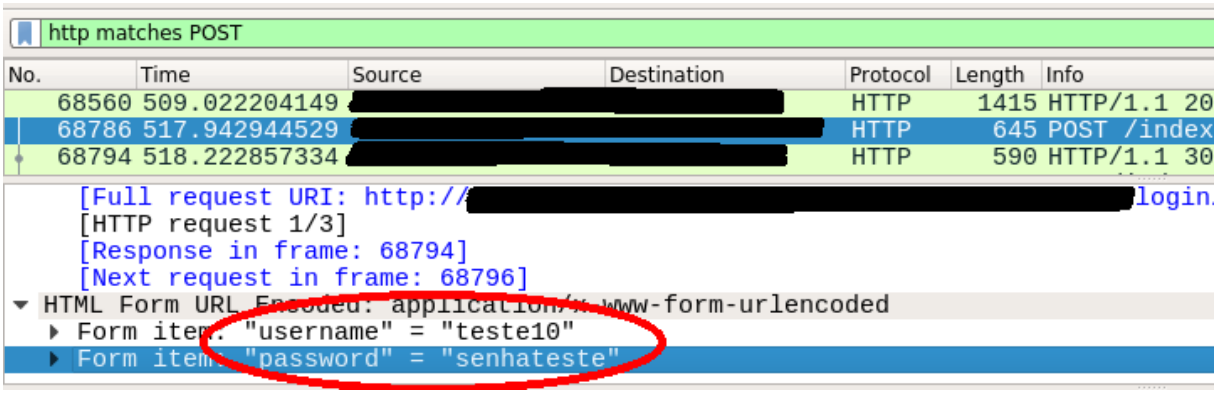

Figura 15: Analise do trafego de rede com Wireshark.
Fonte: Print da aplicação.

Outro teste importante realizado no mecanismo de autenticação é a verificação da existência do mecanismo de bloqueio de conta e verificar a possibilidade realizar ataques de força bruta. O mecanismo de bloqueio de conta tem como objetivo checar se um atacante consegue bloquear uma conta de usuário válida enviando repetidamente senhas incorretas para a aplicação.

O mecanismo de bloqueio de contas está diretamente relacionado com a descoberta de senhas usando ataques de força bruta. A ação tomada pela aplicação é bloquear a conta após um certo número de tentativas erradas. Sendo assim, mesmo se o usuário autêntico da conta fornece as credenciais corretas, será impossível se autenticar até que a mesma seja desbloqueada.

Este teste é feito de forma manual. Com os testes verificou-se que o sistema não apresenta mecanismo de bloqueio de conta, visto que foi feito diversas tentativa de *login* com senha incorreta e mesmo assim após isso foi realizado o *login* normalmente com o mesmo nome de usuário, ou seja, a conta do usuário não foi bloqueada.

Além disso, o mecanismo de autenticação não utiliza *captcha* para evitar ataques. Isso torna o mecanismo de autenticação vulnerável a ataque de força bruta, visto que não apresenta um mecanismo de bloqueio de conta nem a utilização de *captcha*.

Seguindo com os testes, é importante determinar a resistência da aplicação contra a quebra de senha por força bruta usando dicionários de senhas disponíveis. Avaliando o comprimento, complexidade, reutilização e requisitos de senhas. O ataque por força bruta consiste em conseguir credenciais de usuário válida para acessar áreas restritas da aplicação.

A ferramenta *THC Hydra* é utilizada para executar ataques de força bruta em esquemas de autenticação usando formulários HTML. No teste, a ferramenta foi utilizada e obteve como resultado a identificação da credencial dentro de um dicionário de senhas. O único requisito das senhas cadastradas na aplicação é ter o mínimo de 6 caracteres apenas, como pode ser visto na Figura 16, o que facilita o ataque por força bruta.

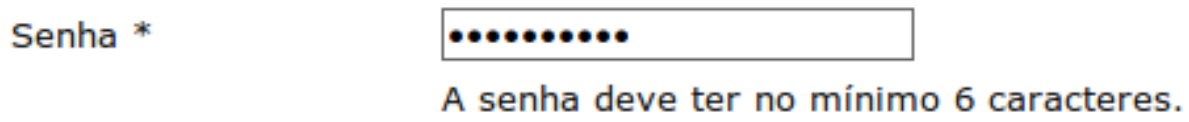

Figura 16: Tamanho da senha.
Fonte: Print da aplicação.

É importante verificar também credenciais padrão fornecidas para autenticação e configuração na aplicação. Frequentemente, as aplicações utilizam credenciais que são fornecidas para uma autenticação inicial, mas não são devidamente alterados e as combinações *login*/senha padrões são mantidas ativas mesmo com a aplicação estando no ambiente de produção.

O teste para este tipo de falha é realizado para encontrar combinações de credencias usando padrões como: *"admin", "administrator", "root"e "guest"*. Nos testes realizados, não se obteve êxito com as tentativas com credencias padrões.

O próximo teste será analisar a funcionalidade de relembrar a senha do usuário no navegador e relembrar possíveis dados sensíveis. Estas funcionalidades são convenientes para os usuários, mas não são consideradas seguras, pois permitem acessos posteriores sem a necessidade de digitar a senha novamente. Estes recursos podem tornar o processo de autenticação vulnerável.

Essas funcionalidades são implementadas com a utilização de *cookies* persistentes que podem conter o identificador do usuário. Eles são armazenados no navegador *Web* e enviados toda vez que a aplicação é acessada. Sendo assim, é necessário avaliar o uso dos *cookies* que são armazenados e verificar se contêm as credenciais do usuário ou outras informações sensíveis.

A análise foi feita de forma manual e verificou-se que apenas o *cookie "OJSSID"* é armazenado pelo mecanismo e não foi identificado a exposição de outros dados sensíveis, como visto na Figura 17.

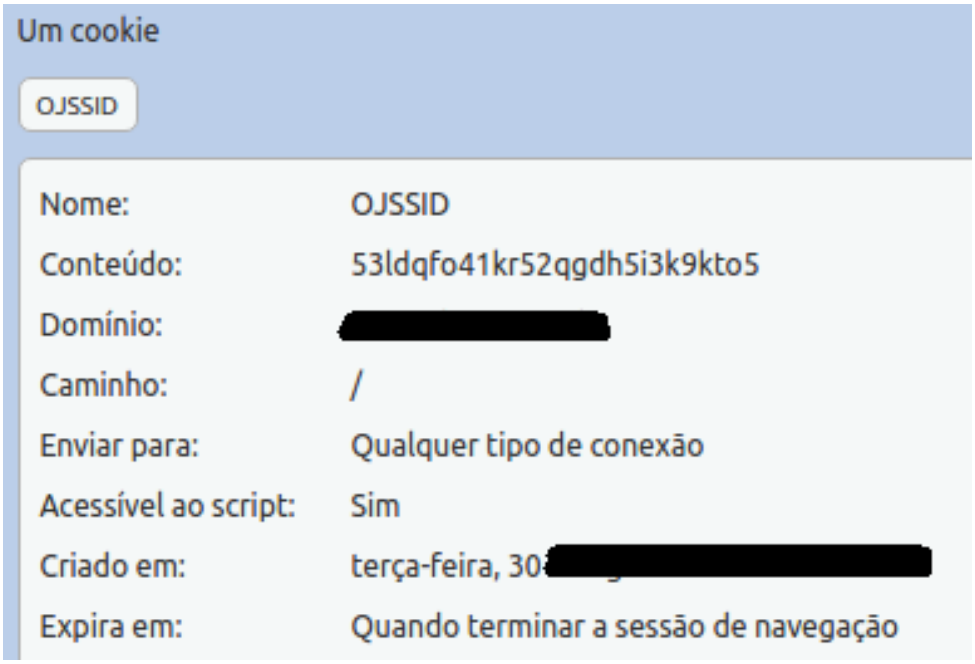

Figura 17: Cookie da sessão.
Fonte: Print da aplicação.

O *cookie* se expira quando terminar a sessão de navegação. Outras credenciais não são armazenas. Ou seja, se o usuário marcar "lembrar usuário e senha"no *login*, então o usuário poderá fechar o navegador e abri-lo posteriormente, e ainda será logado. Nesse caso, o *cookie* expirará após 30 dias desde sua última visita, ou até que seja feito o *logout* e fechar o navegador.

O perigo de ativar a funcionalidade de relembrar senha é que caso seja ativado em um computador de uso público, a conta da última pessoa que usou a aplicação pode ser revelada e utilizada por outras pessoas.

Autorização

A autorização tem o propósito de permitir o acesso à recursos ou informações apenas para aqueles com a devida permissão. O processo de autorização sempre acontece após o autenticação, pois o usuário necessita ter credenciais válidas associada às definições de privilégios [27].

Os testes no processo de autorização têm o objetivo de verificar se é possível burlar o mecanismo de autorização da aplicação, e encontrar potenciais vulnerabilidades nos tipos de privilégios permitidos.

O primeiro teste relacionado a autorização se chama teste de passagem de diretório. O ataque de passagem de diretório permite que atacantes visualizem diretórios e arquivos que por segurança não deviam visualizar, o que caracteriza a falta de função para controle de nível de acesso (Seção 2.2.7).

O teste é feito verificando partes da aplicação que aceitam conteúdos vindos do usuário e posteriormente verificando se tal conteúdo é validado de forma segura. No teste realizado, foi constatado não ser possível realizar a escalada de diretório, caracterizando o correto controle de nível de acesso.

O segundo teste tem o propósito de avaliar como o esquema de autorização foi implementado para cada regra de acesso à funções e recursos da aplicação *Web*. O objetivo é verificar se é possível acessar recursos e funções que supostamente eram para ser acessadas apenas por um tipo de usuário.

É preciso verificar o que acontece se um usuário sem privilégios envia requisição que apenas usuários com tais privilégios podem realizar. Caso seja possível a aplicação é vulnerável e seu mecanismo de autorização é falho. Os testes foram feitos de forma manual e não foi constatado tal falha.

Gerenciamento de Sessão

O gerenciamento de sessão é fundamental para as aplicações *Web*. O principal objetivo é atribuir um identificador diferente para cada usuário quando interage com a aplicação [4]. Esse identificador é utilizado em outras requisições para que seja identificada cada interação. Uto [4] apresenta três tipos de sessões:

- Sessão por *Cookies* – Define o identificador de sessão por um cabeçalho *Set-cookie*.
- Sessão por Parâmetros de URL – Adiciona o identificador de sessão como um parâmetro da URL.
- Sessão por Campos escondidos – Adiciona o identificador de sessão em um campo escondido de formulário.

As vulnerabilidades no gerenciamento de sessão têm o objetivo de obter um identificador de sessão válido, para que seja utilizado em ataques de sequestro de sessão.

A aplicação alvo do teste define o identificador de sessão por meio de um *cookie*, chamado OJSSSID. O OJSSSID pode ser utilizado para realizar o sequestro de sessão de outro usuário autenticado na aplicação. Para isso, é necessário que o identificador de sessão seja válido. Será realizado o teste para verificar a transmissão em claro do

identificador de sessão. O objetivo é capturar o identificador de sessão, por meio da escuta dos pacotes de rede.

Para o teste, foi utilizado a ferramenta *Wireshark* e a partir do monitoramento da rede se obteve o identificador de sessão após o usuário se autenticar na aplicação, o que pode ser utilizado para o sequestro de sessão do usuário.

A partir do scaneamento da aplicação com a ferramenta ZAP, foi notificado a presença de *cookies* sem a *flag httpOnly*. A flag mitiga ataques que tentam realizar sequestro de sessão através de CSRF (Seção 2.2.8).

Validação de dados de entrada

A validação de dados de entrada tem o objetivo de validar toda entrada proveniente do usuário antes que seja utilizada. Não validar os dados de entrada torna a aplicação *Web* vulnerável a ataques, por exemplo, SQL *injection* e XSS [27].

Os dados de entrada na aplicação não são confiáveis e devem ser adequadamente validados, pois podem ser manipulados por um atancante para fim malicioso. Devido a isso, deve se fazer testes de validação de dados. Para isso, será realizado teste para verificar ataque do tipo SQL *injetion* e XSS.

Os testes para os dois tipos de ataques foram realizados manualmente sem a utilização de ferramentas. Para o ataque XSS foi explorado do tipo XSS armazenado, apresentado na Seção 2.2.3.

Foi testado campos que permitem edição HTML onde o usuário insere informações para ser salvado pela aplicação e posteriormente ser exibido em uma página *Web*. Para o teste foi utilizado códigos, conforme apresentado abaixo, para ser salvo pela aplicação e posteriormente ser disparado na página onde é visualizado as informações gravadas, porém a aplicação trata esse tipo de código para não ser executado e grava apenas a *string*.

```
<script > alert("XSS") </ script>
<img src="teste"onerror="alert(1)"'>
```

Diversas tentativas com alteração do *script* foram realizadas na tentativa de burlar o tratamento, porém a aplicação aplica artifícios para contornar o ataque. Entretanto, em uma das tentativas o tratamento é realizado de forma incorreta, porém nos testes não foi encontrado uma forma de explorar o erro.

Com os testes se comprovou que a aplicação realiza tratamento contra códigos inseguros, valida os campos para evitar conteúdo não confiável e implementa política de segurança de contéudo para entrada de dados.

Os testes para o ataque do tipo SQL *Injection* (Seção 2.2.1) não obtiveram sucesso também, por causa do tratamento da entrada de dados. Os valores originados da coleta de dados externos, são validados e tratados, o que impede a execução de instruções destrutivas ou operações com propósito de injetar código SQL.

Scanneamento da Aplicação

O escanamento realizado tem o objetivo de identificar vulnerabilidades na aplicação. Para isso, foi utilizada a ferramenta *Owasp ZAP (Zed Attack Proxy)* e se obteve como resultado os seguintes alertas, conforme a Figura 18. Dentre os alertas identificados, a ferramenta rotulou como dois de risco médio e seis de risco baixo.

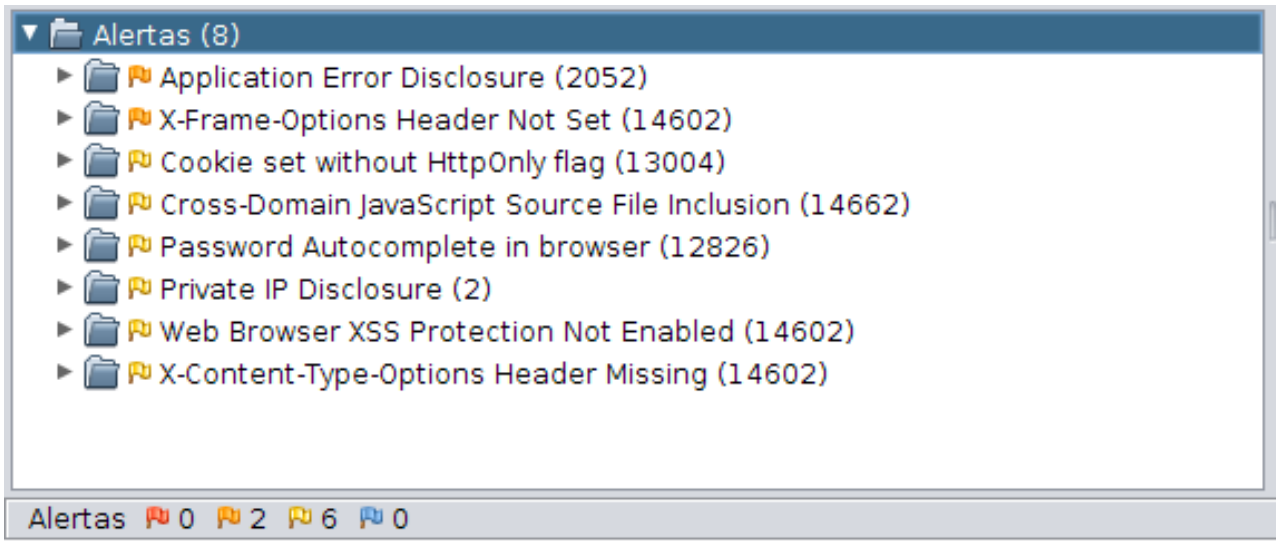

Figura 18: Mapa da Aplicação.
Fonte: Print da aplicação.

- *Application Erro Disclosure*: Página que contém uma mensagem de erro/alerta que pode expor informações sensíveis (Seção 2.2.6), como a localização do arquivo que produziu a exceção não tratada. Esta informação pode ser usada para realizar outros ataques contra a aplicação. O alerta pode ser um falso positivo se a mensagem de erro é encontrada dentro de uma página referente a um documento. A solução para isso seria rever o código-fonte das páginas. Implementar páginas de erro personalizadas.

- *X-Frame-Options Header Not Set:* O cabeçalho *X-Frame-Options* não é incluído na resposta HTTP para proteger contra ataques *clickjacking*. A solução é definir o cabeçalho em todas as páginas *Web* retornados pela aplicação.

- *Cookie set without httpOnly flag*: Foram encontradas *cookies* sem a *flag httpOnly*, que está relacionada a quebra de autenticação (Seção 2.2.2). A *flag* mitiga ataques que tentam realizar sequestro de sessão através de CSRF (Seção 2.2.8).

- *Password Autocomplete in Browser:* O atributo "autocomplete"não está desativada no elemento *FORM/INPUT HTML* contendo campo de entrada do tipo senha. As senhas podem ser armazenados em navegadores *Web* e recuperados em seguida. A Solução é desabilitar o atributo "autocomplete", no formulário ou individualmente no campo de senha.

- *Private IP Disclosure*: Um IP privado foi encontrado no corpo da resposta HTTP. Esta informação pode ser útil para outros ataques contra os sistemas internos.

- *Web Browser XSS Protection not Enabled*: Foram detectadas 14602 vulnerabilidades no que diz respeito à XSS (Seção 2.2.3). Umas das principais consequências da vulnerabilidade de XSS é a exposição negativa do sistema e a possibilidade de utilização da falha para a distribuição de *phishing* e facilitação de fraudes.

- *X-Content-Type-Options Header Missing*: Foram encontradas 14602 configurações incorretas de segurança (Seção 2.2.5). Isso é ocasionado pela falta de configuração no cabeçalho das páginas *Web*.

- *X-Content-Type-Options*: onde a opção *Type-Options* não foi definida como "*nosniff*". Esse resultado permite que versões mais antigas do *Internet Explorer* e *Chrome* possam executar *MIME-Sniffing* no corpo da resposta, podendo causar respostas no corpo a ser interpretadas e exibidas como um tipo de conteúdo que não seja o tipo declarado.

4.3.3 Resumo dos Resultados

A partir dos testes apresentados comprovou-se pelos resultados que a aplicação possui vulnerabilidades simples e que são comuns em muitas aplicações. O resumo do resultado obtido com os testes são apresentas na Tabela 5.

Tabela 5: Resumo do Resultado dos testes.

Teste	**Ferramenta**	**Descrição**	**Vulnerável**
Identificação do Servidor *Web*	*Netcat*	Servidor *Web* desatualizado.	Sim
Análise de meta arquivos do servidor *Web*	Navegador *Web*	Não apresenta informações sensíveis	Não
Enumeração de aplicações no servidor *Web*	*Nikto*	Não apresenta informações sensíveis	Não

Continua na próxima página.

Tabela 5 – *Continuação da página anterior.*

Teste	Ferramenta	Descrição	Vulnerável
Análise de comentários e metadados em páginas *Web*	*Wget*	Não apresenta informações sensíveis	Não
Configuração da Infraestrutura	–	Servidor *Web* desatualizado.	Sim
Verificação da existência de recursos desnecessários ativados ou instalados	*Wget*	Não identificado.	Não
Métodos HTTP	*Nikto*	Método HTTP *TRACK* está ativo. É um risco para ataques XST.	Sim
Definição de perfis	Manual	Permite três tipos de perfis de usuários.	–
Processo de registro de usuários	Manual	Processo satisfatório e seguro.	Não
Enumeração de usuários no *login*	Manual	Não permite a enumeração de usuário.	Não
Enumeração de usuários no cadastro	Manual	Permite a enumeração de identificador de usuário.	Sim
Transporte seguro das credencias	*Wireshark*	São transportados em texto plano pelo o canal de comunicação sem criptografia.	Sim
Mecanismo de bloqueio de conta e possibilidade de ataques de força bruta.	Manual	Não apresenta mecanismo de bloqueio de conta.	Sim
Mecanismo para avaliar força da senha	*THC Hydra*	Política de senha fraca.	Sim
Relembrar a senha do usuário no navegador	Manual	Permite	Sim
Credenciais padrões	Manual	Não apresenta conta ativa com credencias padrões.	Não
Armazenamento de dados sensíveis no navegador	Navegador *Web*	Não armazena nenhum dados sensível no navegador.	Não
Passagem de diretório	Navegador *Web*	Não é permitido realizar a escalada de diretório.	Não

Continua na próxima página.

Tabela 5 – *Continuação da página anterior.*

Teste	**Ferramenta**	**Descrição**	**Vulnerável**
Quebra do esquema de autorização	Navegador *Web*	Não é permitido	Não
Scaneamento da aplicação	*ZAP*	Foram encontrado 8 alertas de riscos de segurança.	Sim
Transporte seguro de identificador de sessão	*Wireshark*	Transporte inseguro do cookie, pode ser explorado para sequestro de sessão.	Sim
Utilização da *flag HTTPOnly*	*ZAP*	Não utiliza	Sim
Ataque XSS	Manual	A aplicação realiza o devido tratamento.	Não
Ataque SQL *Injection*	Manual	Não é possível	Não

4.4 Conclusão

Este capítulo, abordou a realização de testes de segurança em uma aplicação *Web*, com o propósito de encontrar por possíveis vulnerabilidades que possa comprometer a aplicação.

Com os testes foram encontradas vulnerabilidades na aplicação. No entanto, não foram encontradas vulnerabilidades quanto às falhas de segurança que incluem injeção SQL, XSS, referência insegura e direta a objetos, e redirecionamentos e encaminhamentos inválidos. Ataques do tipo XSS e SQL *Injection* não é possível, pois a aplicação realiza tratamento eficiente na entrada de dados.

CAPÍTULO 5

Considerações Finais e Trabalhos Futuros

Em geral, as aplicações Web estão sendo bastantes utilizada pelos usuários. Muitas dessas aplicações realizam transações sigilosas e lidam com diversos dados confidenciais. Devido a esse fato, a segurança de tais aplicações deve ser corretamente implementada e constantemente testada.

Este trabalho abordou testes de segurança como uma prática importante para verificar a segurança em aplicações *Web*. Mostrou-se que uma aplicação *Web*, está suscetível a diversos tipos de ataques e que novos tipos de vulnerabilidades surgem constantemente que podem comprometer até mesmo as aplicações consideradas seguras. Sendo assim, foi realizado um estudo sobre testes de segurança e utilizado uma metodologia para aplicar testes de segurança em uma aplicação *Web*.

A aplicação escolhida apresenta uma base de dados extensa. Todos os dados dos usuários e arquivos podem ser roubados, modificados, ou excluídos caso a aplicação seja atacada. O comprometimento da aplicação afetaria, além dos dados, a sua reputação diante da comunidade de usuários.

Desta forma, o objetivo deste trabalho foi testar a aplicação *Web* em relação à sua segurança. Este objetivo foi alcançado após estudo e a apresentação das principais metodologias, técnicas e ferramentas para testes de segurança.

Com uso de uma metodologia, a *Owasp Testing Guide* versão 4, a aplicação *Web* foi usada como caso de uso para testes de segurança. A partir dos testes e discussões feitas sobre os testes aplicados, pode se verificar que existem vulnerabilidades na aplicação que podem ser exploradas por atacantes reais.

Com o resultado dos testes realizados se concluiu que é necessária a correção desses problemas para evitar problemas futuros. O estudo de caso abrangeu os testes proposto na metodologia da *Owasp*, todavia não abrangeu todas as ameaças conhecidas, visto que o processo de testes de segurança é uma tarefa abrangente em alguns aspectos e alguns ataques são mais recentes e robustos. Como trabalho futuro, é sugerido uma análise no código fonte da aplicação e a correção das falhas com base nas melhores práticas de programação segura da *Owasp*.

O trabalho mostrou-se importante na área de segurança da informação, pois a partir do estudo de caso realizado, foi possível verificar a presença de vulnerabilidades e descuidos em relação a segurança. De forma geral, a principal contribuição foi chamar atenção para a correta implementação dos controles de segurança, nas possíveis consequências das vulnerabilidades e também da importância de se realizar constantemente testes de segurança em aplicações *Web*.

Referências

[1] W. Stallings, *Criptografia e segurança de redes: princípios e práticas.* Pearson Prentice Hall, 2008.

[2] F. Campos, N. S. Neto, and W. F. Canto, "Web 2.0 sob a perspectiva da segurança," vol. 202, 2009.

[3] D. Wichers, "The 2013 owasp top 10: The ten most critical web application security vulnerabilities." Owasp, 2013.

[4] N. Uto, "Vulnerabilidades em aplicações web," *Rede Nacional de Ensino e Pesquisa*, 2009.

[5] ISO, "Information technology - security techniques - information security management systems - overview and vocabulary." 2014.

[6] CERT, "Centro de Estudos Respostas e Tratamento de incidentes de Segurança no Brasil," 2015.

[7] J. M. Ceron, L. L. Fagundes, G. A. Ludwig, L. Tarouco, and L. Bertholdo, "Vulnerabilidades em aplicações web: uma análise baseada nos dados coletados em honeypots," in *VIII Simposio Brasileiro de Segurança*, 2008.

[8] A. Saad, "A Survey on Software Security Testing Techniques," 2013.

[9] A. Austin, C. Holmgreen, and L. Williams, "A comparison of the efficiency and effectiveness of vulnerability discovery techniques," *Information and Software Technology*, vol. 55, no. 7, pp. 1279–1288, 2013.

[10] G. Avramescu, M. Bucicoiu, D. Rosner, and N. Tapus, "Guidelines for discovering and improving application security," in *Control Systems and Computer Science (CSCS), 2013 19th International Conference on.* IEEE, 2013, pp. 560–565.

[11] P. Parmar and A. Feleol, "Aplicação de pentest em sistemas computacionais para analise de vulnerabilidades: Um estudo de caso," *Anais do Encontro Regional de Computação e Sistemas de Informação*, 2013.

[12] C. Borges, "Estudo comparativo de metodologias de pentests," *Universidade Lutera-rana do Brasil (Ulbra)*, 2011.

[13] D. Castanha, "Auditoria em segurança da informação no ambiente uniriotec," *Universidade Federal do Estado do Rio de Janeiro*, 2014.

[14] T. Oliveira, "Testes de seguranÃ§a em aplicações web segundo a metodologia owasp," *Universidade Federal de Lavras*, 2012.

[15] A. Tanenbaum, "Redes de computadores, 4ª. edição traduzida, editora campus," 2008.

[16] STJ, "Cartilha segurança da informação." *Superior Tribunal de Justiça*, 2015.

[17] R. Kissel, "Glossary of key information security terms," *NIST Interagency Reports NIST IR*, vol. 7298, no. 3, 2013.

[18] W. Stallings, *Criptografia e segurança de redes: princípios e práticas.* Pearson Prentice Hall, 2014.

[19] R. W. Shirey, "Internet security glossary, version 2," 2007.

[20] N. Uto and S. Melo, "Vulnerabilidades em aplicações web e mecanismos de proteção," *Minicursos SBSeg*, 2009.

[21] K. Stouffer, J. Falco, and K. Scarfone, "Nist sp 800-115: Technical guide to information security testing and assessment," *National Institute of Standards and Technology*, 2008.

[22] D. D. Bertoglio and A. F. Zorzo, "Um mapeamento sistemático sobre testes de penetração," 2015.

[23] E. dos Santos and R. C. Nunes, "Avaliando a importância das metodologias para aplicação de testes de segurança em sistemas de informação," 2008.

[24] P. Hertzog, "Osstmm-open source security testing methodology manual," *Institute for Security and Open Methodologies, ISECOM. Disponivel: http://www. isecom. org/osstmm*, 2010.

[25] M. Brunner, M. Dilaj, O. Herrera, P. Brunati, R. Subramaniam, S. Raman, U. Chavan, and B. Rathore, "Information systems security assessment framework (issaf) draft 0.2.1," *ISSAF. Disponivel em: http://www.oissg.org/downloads/issaf-0.2/information-systems-security-assessment-framework-issaf-draft-0.2.1/view.html*, 2006.

[26] PTES, "Penetration testing execution standard," *PTES, Disponivel em: http://www.pentest-standard.org/index.php/Main_Page.*

[27] M. Meucci and A. Muller, "The owasp testing guide 4.0," *Open Web Application Security Project*, 2014.

[28] B. Liu, L. Shi, Z. Cai, and M. Li, "Software vulnerability discovery techniques: A survey," in *Multimedia Information Networking and Security (MINES), 2012 Fourth International Conference on.* IEEE, 2012, pp. 152–156.

[29] Netcat, "Netcat," *Disponivel em: http://netcat.sourceforge.net/ Acesso em: 01/04/2017.*

[30] Httprecon, "httprecon," *Disponivel em: https://w3dt.net/tools/httprecon Acesso em: 01/04/2017.*

[31] Set, "set," *Disponivel em: http://www.social-engineer.org/framework/general-discussion/ Acesso em: 01/04/2017.*

[32] Whois, "whois," *Disponivel em: https://registro.br/2/whois Acesso em: 01/04/2017.*

[33] Dig, "dig," *Disponivel em: ftp://ftp.isc.org/isc/bind9/cur/9.10/doc/arm/man.dig.html Acesso em: 01/04/2017.*

[34] Maltego, "maltego," *Disponivel em: https://www.paterva.com/web7/ Acesso em: 01/04/2017.*

[35] Nmap, "nmap," *Disponivel em: https://nmap.org/ Acesso em: 01/04/2017.*

[36] Nessus, "nessus," *Disponivel em: https://www.tenable.com/products/nessus-vulnerability-scanner Acesso em: 01/04/2017.*

[37] Nikto, "nikto," *Disponivel em: https://www.cirt.net/Nikto2 Acesso em: 01/04/2017.*

[38] Acunetix, "Acunetix wvs," *Disponivel em: https://www.acunetix.com/ Acesso em: 01/04/2017.*

[39] AppScan, "Appscan," *Disponivel em: http://www-03.ibm.com/software/products/pt/appscan Acesso em: 01/04/2017.*

[40] Nexpose, "Nexpose," *Disponivel em: https://www.rapid7.com/products/nexpose/ Acesso em: 01/04/2017.*

[41] Burp, "Burp suite," *Disponivel em: https://portswigger.net/burp/ Acesso em: 01/04/2017.*

[42] Webinspect, "Webinspect," *Disponivel em: http://www8.hp.com/br/pt/software-solutions/webinspect-dynamic-analysis-dast/ Acesso em: 01/04/2017.*

[43] Metasploit, "Metasploit," *Disponivel em: https://www.metasploit.com/ Acesso em: 01/04/2017.*

[44] Paros, "Paros," *Disponivel em: http://sectools.org/tool/paros/ Acesso em: 01/04/2017.*

[45] Wireshark, "Wireshark," *Disponivel em: https://www.wireshark.org/ Acesso em: 01/04/2017.*

[46] WebScarab, "Webscarab," *Disponivel em: https://www.owasp.org/index.php/ Acesso em: 01/04/2017.*

[47] robotstxt, "The web robots pages," *Disponivel em: http://www.robotstxt.org/ Acesso em: 01/04/2017.*

APÊNDICE A – Primeiro apêndice

A.1 Glossário

A área de testes de segurança é uma atividade que demanda o conhecimento de diversos conceitos da área de segurança da informação e das sub-áreas como a criptografia. Também é recomendável ter o conhecimento básico de conceitos da computação em geral, como em sistemas operacionais, redes de computadores, bancos de dados, lógica de programação entre outros.

Por se tratar de uma atividade que requer um certo grau de conhecimento para a prática de testes de segurança, alguns conceitos e termos mais comuns em segurança da informação são fundamentais para um bom entendimento deste trabalho.

A seguir são apresentas alguns desses principais conceitos e termos que são utilizados no decorrer do trabalho. As definições estão baseadas na norma NISTIR 7298 (NIST Internal/Interagency Report), que trata de um glossário dos principais termos utilizados na área de segurança da informação [17].

Acesso

É a capacidade de fazer uso de qualquer recurso do sistema de informação, ou seja, usar os recursos do sistema para lidar com informações, para adquirir conhecimento das informações, ou para controlar componentes e funções do sistema.

Controle de Acesso

É o processo de conceder ou negar solicitação para obter e utilizar informações e serviços ou obter acesso a lugares específicos. Tem o objetivo de assegurar que o acesso aos ativos é autorizada e limitados com base em requisitos de negócios e de segurança.

Criptografia

A disciplina que incorpora os princípios, meios e métodos para a transformação de dados, a fim de esconder o seu conteúdo semântico, impedir a sua utilização não autorizada, ou impedir a sua modificação não autorizada. Além de fornecer confidencialidade, integridade dos dados, não-repúdio, e autenticidade.

Incidente

Uma ocorrência que ponha em risco real ou potencialmente, a confidencialidade, integridade ou disponibilidade de um sistema de informação ou a informação dos processos do sistema, armazenamento ou transmissão ou que constitua uma violação ou iminente ameaça de violação das políticas de segurança, procedimentos de segurança, ou políticas de uso aceitável.

Vulnerabilidade

Fraqueza em um sistema de informação, procedimentos de segurança do sistema, controles internos, ou na implementação que pode ser explorada por uma ameaça.

Ataque

É uma tentativa de obter acesso não autorizado aos serviços, recursos, ou informação do sistema, ou uma tentativa de comprometer a integridade do sistema. É Qualquer tipo de atividade maliciosa que tenta coletar, interromper, negar, degradar ou apagar informação do sistema.

É uma ação em que alguma vulnerabilidade é explorada através de uma ameça, comprometendo algum ativo, ou seja, qualquer ação que pode comprometer a segurança de uma organização.

Auditoria

Uma revisão e análise independente dos registos e atividades para avaliar a adequação dos controles do sistema, com o proposito de garantir a conformidade com as políticas estabelecidas e procedimentos operacionais, e recomendar as alterações necessárias nos controles, nas políticas ou nos procedimentos.

Autenticação

Processo para verificar a identidade de um utilizador, processo ou dispositivo, como pré-requisito para permitir o acesso aos recursos num sistema de informação. Esse processo permite o estabelecimento de confiança de autenticidade, ou seja, confirmar a identidade de uma entidade.

Autorização

Procedimento que concede privilégios de acesso a um usuário, programa ou processo para acessar um determinado recurso do sistema.

Risco

É o nível de impacto sobre as operações de organização, ativos organizacionais, ou indivíduos resultantes do funcionamento de um sistema de informação, dado o impacto potencial de uma ameaça e a probabilidade dela ocorrer e de suas consequências para a organização.

Ameaça

Qualquer circunstância ou evento com o potencial de afetar negativamente as operações organizacionais, os ativos da organização, os indivíduos, através do acesso não autorizado, destruição, divulgação, modificação de informações e/ou por negação de serviço. Qualquer evento que explore vulnerabilidades.

Atacante

É um usuário não autorizado que tem o objetivo de tentar ganhar acesso a um sistema de informação. Pessoa responsável por um ataque ou de uso não autorizado a um sistema.