

UNIVERSIDADE FUMEC

ANÁLISE DE EFICIÊNCIA NA DETECÇÃO DE
VULNERABILIDADES EM AMBIENTES WEB COM O USO DE
FERRAMENTAS DE CÓDIGO ABERTO

MARCOS FLÁVIO ARAÚJO ASSUNÇÃO

Belo Horizonte - MG

2015

MARCOS FLÁVIO ARAÚJO ASSUNÇÃO

ANÁLISE DE EFICIÊNCIA NA DETECÇÃO DE
VULNERABILIDADES EM AMBIENTES WEB COM O USO DE
FERRAMENTAS DE CÓDIGO ABERTO

Dissertação apresentada ao Programa de Mestrado em Sistemas de Informação e Gestão da Informação da Universidade Fumec como parte dos requisitos para a obtenção do título de Mestre em Sistemas de Informação e Gestão do Conhecimento.

Área de concentração: Gestão de Sistemas de Informação e Gestão do Conhecimento

Linha de pesquisa: Tecnologia e Sistemas de Informação

Orientador: Prof. Dr. Rodrigo Moreno Marques

Coorientador: Prof. Dr. Luiz Cláudio Gomes Maia

Belo Horizonte – MG

2015

Dedico esse trabalho à minha esposa Caroline Araújo Assunção, que sempre esteve do meu lado nos momentos mais difíceis, me apoiando e incentivando a crescer cada vez mais como profissional e ser humano.

“Ter sucesso é avançar de fracasso em fracasso sem perder o entusiasmo”

Winston Churchill

AGRADECIMENTOS

Agradeço novamente à minha esposa Caroline Assunção, pois é o meu pilar de apoio e a pessoa de que me orgulho poder envelhecer lado a lado.

A meus pais, Messias e Ângela, que sempre me incentivaram à ir além do necessário e acreditar que nenhum sonho é impossível.

Aos meus irmãos Marcelo e Sofia, por serem companheiros e sempre amigos. À Eliana Assunção, que também sempre forneceu seu constante apoio.

Ao meu Orientador, Rodrigo Moreno Marques, gostaria de agradecer a paciência e a dedicação para me auxiliar no desenvolvimento desse trabalho.

Aos meus amigos e colegas de trabalho do Senac Minas e do Centro Universitário UNA por me ajudarem a ser um profissional cada vez melhor.

Para finalizar, gostaria de agradecer a meus alunos e leitores por sempre me ensinarem que um bom professor não é necessariamente aquele que detém o maior conhecimento, e sim aquele que ensina apaixonadamente o pouco que sabe e tem a humildade de aceitar que deve aprender um pouco mais a cada novo dia.

RESUMO

Esta pesquisa teve como objetivo identificar as principais vulnerabilidades em aplicações web e avaliar o desempenho de ferramentas *open source* de análise desse tipo de falha. Inicialmente, foi feita uma pesquisa bibliográfica e documental para identificar os principais riscos de segurança em sites da *web*, assim como as soluções *open source* mais recomendadas para detectar essas vulnerabilidades. A seguir, foi desenvolvido um experimento visando à comparação do desempenho das soluções *open source* selecionadas em relação à capacidade de detecção dos principais riscos identificados. Para realização do experimento, cinco softwares (OWASP ZAP, SQLMap, Nikto, Skipfish, W3af) foram instalados em um ambiente virtualizado e testados contra um sistema *web*, contendo falhas propositalmente inseridas para fins de pesquisa. O documento Top Ten OWASP foi tomado como referência para seleção das categorias de risco testadas. Os resultados encontrados mostram que a ferramenta W3af se saiu melhor do que as demais no contexto geral. Entretanto, verifica-se que cada uma das soluções testadas apresentou um desempenho de nível médio a alto, quando consideramos as categorias individuais de teste.

Palavras-chave: *Top Ten OWASP*; Aplicações Web; Ferramentas *Open source*; *PenetrationTest*; Análise de Vulnerabilidades; Segurança da Informação.

ABSTRACT

This paper aimed to identify the main vulnerabilities in web applications and evaluate the performance of open source analysis tools in detecting this type of risks. Initially, a bibliographical and documentary research were carried out to identify the main security risks in Web sites, as well as the open source most recommended solutions to detect these vulnerabilities. Next, an experiment was carried out in order to compare the performance of the selected open source solutions regarding the detection capability of the main risks identified. To perform the experiment, five softwares (OWASP ZAP, sqlmap, Nikto, Skipfish, w3af) were installed in a virtualized environment and tested against a web environment containing deliberately inserted fault for research purposes. The OWASP Top Ten document was used as a reference for selecting the tested risk categories. The results show that the w3af tool performed better than the others in the general context. However, each of solutions tested had an average to high performance level, when considering the individual categories tested.

Keywords: *Top Ten OWASP; Web Applications; Open-Source tools; Penetration Test ; Vulnerability Analysis; Information Security.*

LISTA DE FIGURAS

Figura 1. Comparativo de popularidade entre os servidores	22
Figura 2. Pilares da Segurança da Informação	25
Figura 3. Comparativo da lista <i>Top Ten</i> dos anos de 2010 e 2013.	32
Figura 4. Fluxograma de passos adotado na pesquisa	41
Figura 5. Máquina virtual criada no VirtualBox	42
Figura 6. Tela principal do BackTrack Linux.....	43
Figura 7. Teste de acesso ao <i>Mutillidae</i>	44
Figura 8. Interface gráfica do OWASP ZAP	47
Figura 9. Interface de texto do SQLMAP.....	49
Figura 10. Interface de texto do Nikto	50
Figura 11. Interface de texto do Skipfish	51
Figura 12. Interface gráfica do W3AF.....	52
Figura 13. Execução do processo de crawling	57
Figura 14. Tela de retorno	60
Figura 15. Tela de retorno	61
Figura 16. Tela de Retorno.....	62
Figura 17. Tela de Retorno.....	63
Figura 18. Tela de varredura do programa W3AF.....	65
Figura 19. Falsos Positivos da Ferramenta W3AF	66
Figura 20 Tela de Varredura do Skipfish.....	68
Figura 21. Ferramenta Nikto sendo executada durante os testes	70
Figura 22. Gráfico de falhas corretamente identificadas	73
Figura 23. Gráfico de tempo total de execução	74
Figura 24. Comparativo: total de falhas existentes, detectadas e falsos positivos	75

LISTA DE TABELAS

Tabela 1. Páginas e Vulnerabilidades no <i>Mutillidae</i>	53
Tabela 2. Informações obtidas após a varredura com o OWASP ZAP.....	58
Tabela 3. Informações obtidas após a varredura com o SQLMAP	64
Tabela 4. Informações obtidas após a varredura com o W3AF	67
Tabela 5 Informações obtidas após a varredura com Skipfish	69
Tabela 6. Informações obtidas após a varredura com Nikto	71
Tabela 7. Falhas corretamente detectadas pelos softwares	72
Tabela 8. Tempo de Execução das Ferramentas.....	74
Tabela 9. Total de falhas existentes x detectadas x falso positivo	75

LISTA DE ABREVIATURAS

API	Application Programming Interface
B2B	Business to Business
B2C	Business to Client
CERN	Conseil Européen pour la Recherche Nucléaire
CSRF Cross-Site	Request Forgery
DB	Database
DBMS	Database Management System
HD	Hard Disk
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
IIS	Internet Information Services
IP	Internet Protocol
ISECOM	Institute for Security and Open Methodologies
ISSAF	Information Systems Security Assessment Framework
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
OISSG	Open Information Systems Security Group
OSSTMM	Open Source Security Testing Methodology Manual
OWASP	Open Web Application Security Project
PHP	Hypertext Preprocessor
PKI	Public Key Infrastructure
RFC	Request for Comment
SGBD	Sistema de Gestão de Bancos de Dados
SMTP	Simple Mail Transfer Protocol
SO	Sistema Operacional
SQL	Structured Query Language
SSL	Secure Socket Layer
TLS	Transport Layer Security
UID	User Identification
WASC	Web Application Security Consortium

XML	Extensible Markup Language
XPATH	XML Path Language
XSS	Cross-site Scripting

SUMÁRIO

1. INTRODUÇÃO	14
1.1 Problema de Pesquisa	16
1.2 Justificativa e Relevância do Tema.....	17
1.3 Aderência do Projeto de Pesquisa ao Programa.....	18
1.4 Objetivos.....	18
1.4.1 Objetivo Geral.....	18
1.4.2 Objetivos Específicos	18
2. FUNDAMENTAÇÃO TEÓRICA.....	18
2.1 Ambientes Web	19
2.1.1 – Protocolo HTTP (<i>Hyper Text Transfer Protocol</i>) e HTTPS (<i>Hyper Text Transfer Protocol Secure</i>)	20
2.1.2 – Servidores Web.....	21
2.1.3 – Linguagens de desenvolvimento web	23
2.1.4 – Banco de dados	24
2.2 Segurança da Informação	24
2.3 <i>Penetration Test</i>	27
2.3.1 – Metodologias de <i>Penetration Test</i>	27
2.3.2 – Tipos de testes	28
2.3.3 – Relatório final	30
2.4 Vulnerabilidades no ambiente web	30
2.4.1 Projeto OWASP (<i>Open Web Application Security Project</i>).....	31
2.4.2 As dez principais vulnerabilidades da web	32
2.5 – Ferramentas de análise de vulnerabilidade web	38
3. METODOLOGIA DE PESQUISA.....	39
3.1 Construção do ambiente de testes simulados	42
3.2 Seleção das ferramentas <i>open source</i> para testes de vulnerabilidade web	44
3.2.1 – OWASP ZAP	46
3.2.2 – SQLMap	48
3.2.3 – Nikto	49
3.2.4 – Skipfish.....	50
3.2.5 – W3AF.....	51
4. RESULTADOS.....	53
4.1 Vulnerabilidades existentes no Framework <i>Mutillidae</i>	53
4.2 Análise das falhas através das ferramentas selecionadas	55

4.2.1 – Execução do teste da ferramenta OWASP ZAP	56
4.2.2 – Análise e teste da ferramenta SQLMAP	59
4.2.3 – Análise e teste da ferramenta W3AF	64
4.2.4 – Execução do teste da ferramenta Skipfish.....	67
4.2.5 Análise e teste da ferramenta Nikto	70
4.3 Tabulação dos resultados finais.....	72
5. CONCLUSÃO	76
REFERÊNCIAS.....	80

1. INTRODUÇÃO

Devido à expansão da Internet, aumenta cada vez mais a tendência de migração de aplicações de ambientes de *desktop* locais para versões on-line. Muitas empresas hoje já oferecem um site da *web*, seja para apresentar algum produto ou serviço, ou para oferecer uma solução de comércio eletrônico, que atenda às demandas do usuário final. Além das empresas, vários órgãos governamentais também migraram seus serviços para o ambiente virtual, permitindo a prestação de diversos serviços por meio da Internet, como, por exemplo, emissão de segunda via de documentos, inscrição em concursos, entre outras atividades.

Estas soluções são algumas das facilidades possibilitadas pela popularização da *world wide web* nos últimos anos. Segundo Kurose (2005), a *web* é o conjunto das páginas de hipertexto, ou sites, que pode ser acessado por um usuário através de um navegador de Internet. De acordo com Torres (2014, p. 9), “com o aparecimento da Web 2.0, o utilizador passou a ter um papel dinâmico na Internet, deixando de ser um mero espectador para ser um membro participativo”.

Entretanto, o grande crescimento desse ambiente nem sempre foi acompanhado por uma preocupação adequada com a segurança das soluções desenvolvidas.

Ceron et al. (2008, p. 2) expõem que:

O avanço das tecnologias voltadas para a web e a falta da devida preocupação com requisitos de segurança tornam a Internet um ambiente repleto de vulnerabilidades e alvo de frequentes ataques. O problema torna-se ainda mais grave com a utilização dos mecanismos de busca como uma ferramenta para localizar sites vulneráveis. (CERON et al., 2008, p.2)

Tanto Ceron et al. (2008) quanto Holz (2006) concordam que uma solução web é normalmente mais vulnerável a falhas do que um software tradicional de *desktop*, já que no primeiro caso há diversas camadas de soluções que atuam em conjunto e criam pontos de vulnerabilidade na integração de todo o ambiente. A atuação dos mecanismos de busca permite, inclusive, a localização de determinadas falhas em web sites, tornando-se assim um facilitador para o invasor. Basso (2010)

expõe que a má codificação é o principal fator causador das vulnerabilidades, que pode ter como origem o reuso de código, uso de componentes genéricos ou outras práticas inseguras de codificação. Essa visão também é corroborada por Macedo, Queiroz e Damasceno (2010), quando expõem que, em decorrência do crescimento exponencial do mercado de web, muitas empresas têm recorrido a soluções de *frameworks* terceirizados, que podem conter diversas vulnerabilidades.

Garantir a proteção da informação como um todo é importante para "garantir a continuidade do negócio, minimizar o risco ao negócio, maximizar o retorno sobre os investimentos e as oportunidades de negócio." (ISO 27001, 2013, p.45). Os impactos causados por sistemas web inseguros podem gerar enormes prejuízos para os envolvidos, tanto do âmbito financeiro, quanto em âmbito intelectual.

É nesse contexto que está inserido o problema que norteia a pesquisa realizada, enunciado a seguir.

1.1 Problema de Pesquisa

Os sistemas que têm por base a web são cada vez mais complexos e podem trazer facilidade e boa usabilidade para o usuário. Porém, por serem mais complexos, tornam-se mais vulneráveis a invasões. Para combater esses riscos, surgem as ferramentas para identificação de vulnerabilidades, que examinam toda a estrutura de um site para identificar potenciais brechas de segurança. Entretanto, a maioria das ferramentas de análise de vulnerabilidades são proprietárias e têm um custo muito alto para o uso de uma pequena ou média empresa.

Com o advento do software livre, muitas empresas agora podem ter seus próprios serviços de Internet de forma acessível e com custos decrescentes, podendo utilizar ferramentas com código fonte aberto (*open source*) para identificar vulnerabilidades em seus sistemas. Com respaldo nesses pressupostos, surgem as perguntas que norteiam a presente pesquisa: quais são os softwares de código-aberto mais eficientes para identificação de vulnerabilidades em ambientes web e qual a eficiência das ferramentas de análise de vulnerabilidades neste tipo de ambiente?

1.2 Justificativa e Relevância do Tema

Ao contrário de softwares, que normalmente são projetados para serem executados em apenas uma plataforma computacional, uma aplicação web é criada de modo que possa ser executada em qualquer dispositivo, como *tablets*, computadores pessoais ou *smartphones*. Esse fator de complexidade aumenta o risco de exploração de falhas, pois nem sempre o software é desenvolvido tendo a segurança como um dos focos principais.

Uma pesquisa conduzida pela *Web Application Security Consortium* (WASC)¹, em 2008, sugere que 49% das aplicações web testadas apresentam vulnerabilidades de alto risco (urgentes e críticas), que podem ser detectadas por ferramentas de varredura. Rocha, Kreutz e Turchetti (2012, p.1) complementam dizendo que “o número de vulnerabilidades e ataques é cada vez maior e mais frequente em serviços e sistemas web”.

Torna-se cada vez mais necessário o aprimoramento de novas metodologias de detecção e mitigação de riscos, vulnerabilidades e falhas delas decorrentes, tanto em âmbito privado quanto público, uma vez que, atualmente, empresas e governos de todo mundo tendem a levar seus sistemas para a Internet. Como consequência, caso estes sistemas sejam explorados por um invasor, o banco de dados pode ser comprometido e as informações armazenadas podem ser acessadas por pessoas não autorizadas.

Portanto, esse trabalho se justifica pela importância de discutir e identificar as principais vulnerabilidades de aplicações web por meio do uso de softwares de código aberto. O estudo pretende levantar as principais ferramentas utilizadas para a varredura de brechas e analisar a sua capacidade de detectar falhas em um ambiente web.

Esta pesquisa adere à a linha de pesquisa de Sistemas de Informação, ao mesmo tempo em que estabelece interlocuções com a área de Ciências da Computação. São discutidas temáticas relativas às redes de computadores, segurança da informação e gestão da informação.

¹Disponível em
<<http://projects.webappsec.org/w/page/13246989/Web%20Application%20Security%20Statistics>>. Acesso em 29/08/2015.

1.3 Aderência do Projeto de Pesquisa ao Programa

A pesquisa realizada, de cunho primariamente experimental, tem aderência ao eixo de Sistemas de Informação do curso de mestrado profissional em Sistemas de Informação e Gestão do Conhecimento da Universidade Fumec. As principais temáticas abordadas no trabalho foram as relacionadas às redes de computadores, segurança da informação e serviços da *web*.

1.4 Objetivos

1.4.1 Objetivo Geral

Identificar as principais vulnerabilidades em aplicações web e avaliar o desempenho de ferramentas *open source* de análise desse tipo de falha.

1.4.2 Objetivos Específicos

- Identificar os riscos de segurança mais frequentes que podem ocorrer em ambiente web;
- Preceder a um levantamento das principais ferramentas com código fonte aberto, utilizadas para varredura de vulnerabilidades em ambientes web; e
- Testar as ferramentas selecionadas em um ambiente virtual simulado, comparando-as em relação à eficácia na detecção dos riscos de segurança mais frequentes.

2. FUNDAMENTAÇÃO TEÓRICA

O presente trabalho apresenta sua fundamentação teórica, dividindo-a em quatro seções, que estão correlacionadas pela sequência do conteúdo apresentado. Primeiramente, serão apresentados o ambiente web e, especialmente, as tecnologias que o compõem, como os servidores web e seus *frameworks*. A seguir, será abordada a segurança da informação, com destaque para alguns conceitos utilizados na pesquisa, como intrusão e vulnerabilidade. Na terceira seção, voltada para a discussão do *Penetration Test*, serão expostos as etapas e os processos existentes em um procedimento de análise de vulnerabilidades. Na última seção, as vulnerabilidades da web serão abordadas pela discussão das principais falhas desse tipo de ambiente e apresentados o projeto OWASP (*Open Web Application Security Project*) e as principais ferramentas empregadas para verificação das vulnerabilidades.

A divisão em quatro seções visa a tornar mais clara a exposição da fundamentação teórica que sustenta o trabalho de pesquisa, buscando discutir o que é a web, suas principais vulnerabilidades e as ferramentas que podem ser potencialmente utilizadas para detectá-las.

2.1 Ambientes Web

Com o uso cada vez mais frequente de recursos voltados para a web, as empresas têm cada vez mais levado seus produtos e serviços para a esfera da Internet. Por essa razão, soluções do tipo *Business to Business* (B2B) e *Business to Client* (B2C) utilizam sistemas de gerenciamento de bancos de dados que são parcialmente ou totalmente integrados à *World Wide Web*.

Existem atualmente dezenas de diferentes padrões e tecnologias que podem ser utilizados para a construção de um *website*, ou um portal – uma versão mais complexa de um site web, que integra normalmente um grande número de recursos para acesso do cliente.

A maioria dos *websites* integra a criação de recursos visuais (*webdesign*), o uso de *scripts* de programação e sua integração com um banco de dados. Entretanto, segundo Stallings (2005), independentemente dos padrões utilizados para essa integração, os protocolos de acesso utilizados pelo cliente são o *HTTP* e o *HTTPS*.

2.1.1 – Protocolo HTTP (*Hyper Text Transfer Protocol*) e HTTPS (*Hyper Text Transfer Protocol Secure*)

O HTTP, abreviação de *Hyper Text Transfer Protocol*, é um dos principais protocolos de aplicação utilizado na web. Os protocolos de aplicação são usados em uma rede de computadores para permitir a comunicação entre o software usado no sistema do usuário final e o servidor que hospeda o recurso que será acessado.

Kurose (2005, p.7) expõe que “um protocolo define o formato e a ordem das mensagens trocadas entre duas ou mais entidades comunicantes, bem como as ações realizadas na transmissão e/ou no recebimento de uma mensagem ou outro evento”.

Desde a criação da Web, o HTTP é utilizado pelos navegadores de Internet para permitir aos clientes acesso a sites da Internet, baixando e processando os recursos gráficos das páginas (como imagens e vídeos) para visualização dos usuários finais. De acordo com Assunção (2014, p.38), “o HTTP é quem define como as páginas são formatadas e transmitidas e que ações os servidores web e browsers devem tomar ao responder a certos métodos”.

No entanto, com a rápida expansão da web como meio para troca de informações, tornou-se necessário que, em determinados acessos, houvesse um meio para garantir o sigilo e a identidade do remetente e do receptor. Para esse tipo de aplicação, foi criado o protocolo HTTPS, que resulta da utilização do HTTP em conjunto com protocolos que oferecem recursos de criptografia, provendo uma camada de novas funções. Atualmente, os dois protocolos de criptografia mais usados para esse fim são o SSL e o TLS.

O protocolo TLS (*Transport Layer Security*) e seu predecessor SSL (*Secure Socket Layer*) utilizam o padrão de certificados X.509 (ITU-T, 2001) e criptografia assimétrica para proteger os dados que trafegam na Internet. Eles foram desenvolvidos com o objetivo de tornar a comunicação na Internet mais segura, já que muitos protocolos da camada de aplicação não têm nenhum tipo de criptografia nativa, o que poderia levar a diversos tipos de ataques.

De acordo com Stallings (2008):

Três protocolos de camada superior são definidos como parte do SSL: o Protocolo de Estabelecimento de Sessão (Handshake Protocol), o Protocolo de Mudança de Especificação de Cifra (Change Cipher Spec Protocol) e o Protocolo de Alerta (Alert Protocol). Esses protocolos são usados no gerenciamento de trocas SSL.(STALLINGS, 2008, p. 185)

O Secure Socket Layer se baseia em dois conceitos complementares, a conexão SSL e a sessão SSL. As sessões SSL são criadas pelo Protocolo de Estabelecimento de Sessão e funcionam como uma associação entre um servidor e seus clientes. As sessões são usadas para evitar a renegociação de novos parâmetros seguros a cada nova conexão, definindo assim um conjunto de parâmetros criptográficos, que podem ser compartilhados entre múltiplas conexões.

Basso (2010) afirma que a segurança de uma aplicação web não é determinada apenas pela criptografia aplicada ao protocolo de comunicação, mas por um conjunto de configurações seguras feitas em um servidor web.

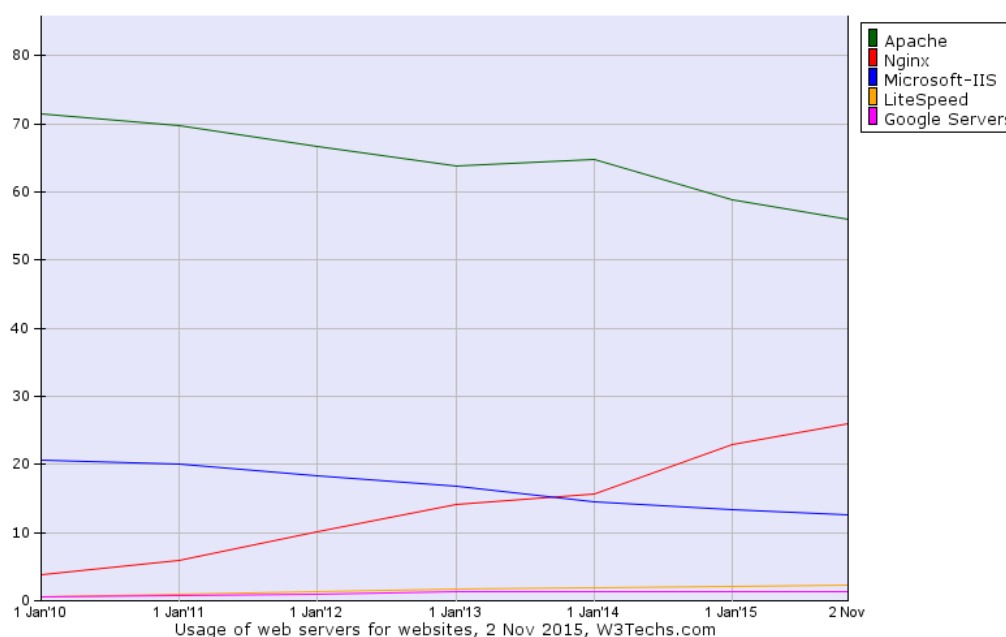
2.1.2 – Servidores Web

Servidores Web são servidores responsáveis por aceitar requisições HTTP e HTTPS de clientes web e entregar respostas destes protocolos, geralmente em forma de páginas web que tenham conteúdos estáticos (como textos e imagens) e/ou dinâmicos (*scripts*) (LIMA, 2003). O conteúdo disponibilizado pelo servidor é visto pelo cliente por meio de navegadores tais como Internet Explorer, Netscape, Google Chrome, Firefox.

O servidor que provê o serviço *web*, ou seja, hospeda o conteúdo para consulta, executa um software específico para realizar essa função. Como exemplo, podemos destacar os servidores IIS (*Internet Information Services*), que são proprietários da Microsoft, e o Apache, que é uma das soluções de código fonte livre, que é executada em diversas plataformas. O Apache é um servidor web de alta performance, leve e robusto, usado tanto por pequenas empresas quanto grandes corporações. Tem código fonte aberto e oferece uma gama de recursos como autenticação, controle de acesso, logs de acesso e extensões de *scripts*. Por ser um software livre, novas funcionalidades podem ser adicionadas por meio da sua API (*Application Programming Interface*).

Assunção (2009) alega que o Apache em conjunto com a linguagem PHP (*Hypertext Preprocessor*) são uma combinação de soluções mais comum em servidores da Internet. Uma pesquisa realizada pela W3Techs² corrobora essa afirmação e revela que, apesar da queda de popularidade nos últimos anos, o software Apache é ainda muito utilizado em vista de outros softwares web e apesar da crescente popularidade do Nginx. A Figura 1 mostra um gráfico comparativo de popularidade dos servidores web.

Figura 1. Comparativo de popularidade entre os servidores



Fonte: http://w3techs.com/technologies/history_overview/web_server/ms/y . Acesso em 02 de Novembro de 2015.

Além de garantir disponibilidade de dados, os servidores são responsáveis por, muitas vezes, interpretar linguagens de programação web. Torres (2014) indica que parte das vulnerabilidades existentes é proveniente da codificação insegura de scripts, feita pelo programador.

² Disponível em: <<http://w3techs.com/about.>>. Acesso em 02 de Novembro de 2015.

2.1.3 – Linguagens de desenvolvimento web

Schildt (2013) classifica a linguagem de programação como responsável por definir os elementos específicos do código fonte do programa. Frente à necessidade de produzir conteúdo dinâmico, é preciso utilizar linguagens de programação que permitam maior flexibilidade que o HTTP consegue executar. Nesse campo, surgem linguagens como o Java, PHP, Javascript, Python, Ruby, entre várias outras. Muitas das linguagens utilizadas na web são baseadas em script.

Scripts podem ser definidos como um conjunto de instruções que realizam tarefas geralmente repetitivas. Costa (2010) afirma que não há um consenso quanto a essa classificação. O uso da linguagem de *scripts* evoluiu a ponto de esse conceito ser aplicado também a programas de computador interpretados.

As linguagens baseadas em *scripts* apresentam características que as diferem das outras linguagens de programação convencionais, como a não geração de um programa executável a partir de um código fonte. Uma delas é o PHP, que é uma linguagem de programação de software livre, baseada em *scripts* com múltiplas funções, que foi concebida originalmente por Rasmus Lerdorf, no ano de 1995.

Ao contrário do HTML, que é executado pelo browser quando uma página é aberta, o PHP é pré-processado pelo servidor. Assim, todo o código PHP incluso num arquivo é processado pelo servidor antes de enviar algo para o cliente, através do browser. Por esse motivo, muitas vezes é referido como uma linguagem *server-side*.

De acordo com Sica e Real (2007, p. 6), “ao acessar alguma página desenvolvida em PHP, o servidor HTTP direciona a requisição para o interpretador PHP, que, por sua vez, executa o *script* contido na página solicitada e devolve ao cliente.”

A integração de uma linguagem de script com o banco de dados deve ser planejada por boas práticas de programação segura, caso contrário, a aplicação pode ficar exposta à vulnerabilidades de injeção de código sql (TORRES, 2014).

2.1.4 – Banco de dados

Um banco de dados (sua abreviatura é BD, e em inglês DB, *database*) é uma entidade na qual é possível armazenar dados de maneira estruturada e com a menor redundância possível. Estes dados poderão ser então acessados e utilizados por usuários distintos, através de softwares específicos. A linguagem mais comumente utilizada para a comunicação entre os usuários e o banco é a SQL (*Structured Query Language*) (LIMA, 2003).

Para poder controlar os dados e os usuários, utiliza-se um sistema chamado de SGBD (sistema de gestão de bancos de dados) ou em inglês DBMS (*Database Management System*). A solução do SGBD compreende um conjunto de serviços que permitem gerenciar os bancos de dados, portanto, são responsáveis por:

- Permitir o acesso aos dados de maneira simples;
- Autorizar um acesso às informações a múltiplos usuários; e
- Manipular os dados presentes no banco de dados (inserção, supressão, modificação).

Existem atualmente diversas soluções para gerenciamento e acesso a banco de dados como Microsoft SQL Server, MySQL, PostgreSQL e diversas outras.

Welling e Thompson (2004) concordam com Pessoa (2007) ao afirmarem que um banco de dados bem planejado e estruturado, levando em consideração boas práticas de desenvolvimento seguro e tratamento de dados sensíveis, pode evitar problemas de redundância de dados que podem ocasionar sua perda ou seu corrompimento. Muitos ataques a ambientes web têm como objetivo alcançar os dados no banco, portanto, a proteção desse sistema, pela implementação e cumprimento de políticas bem definidas de Segurança da Informação, é vital para uma solução segura.

2.2 Segurança da Informação

A informação pode ser considerada o principal bem de qualquer empresa ou instituição, seja pública ou privada. A Segurança da Informação é a área do conhecimento responsável por proteger os dados e os ativos de uma instituição contra vários tipos de ameaças, como o vazamento de informações internas. Torres

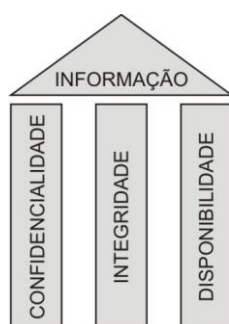
(2014) expõe que a norma de padronização internacional ISO/IEC 27002 define a confidencialidade, integridade e disponibilidade de informações digitais ou físicas, como documentos impressos.

Um dos principais papéis dessa área é proteger todo o conhecimento institucional gerado pelo uso das informações. Campos (2014, p.34) afirma que “a informação é o elemento essencial para todos os processos de negócio da organização, sendo, portanto, um bem ou ativo de grande valor”. Martinelo e Bellezi (2014, p.35) afirmam que “a Segurança da Informação tem que ser vista pelas empresas como algo estratégico, pois elas estão muito dependentes dos serviços de tecnologia da informação”.

Os ativos de uma empresa são os elementos físicos, humanos ou tecnológicos que fazem parte do fluxo de informações dentro da companhia. A Segurança da Informação utiliza-se de padrões, processos e regras para controlar e monitorar o uso destes ativos, a fim de proteger o conhecimento organizacional.

Um sistema de segurança da informação se baseia em três princípios ou pilares básicos: confidencialidade, integridade e disponibilidade. Somente alcançaremos um nível de segurança satisfatório se todos os pilares forem corretamente implementados. A Figura 2 mostra os pilares da Segurança da Informação.

Figura 2. Pilares da Segurança da Informação



Fonte: <http://www.helviojunior.com.br/it/security/clareza-e-productividade-na-gestao-do-firewall-aker-6-1/>. Acesso em março de 2015.

Stallings (2010, p.10) afirma que “a confidencialidade é a proteção dos dados transmitidos contra ataques passivos”. Esse pilar da segurança é responsável pela

codificação dos dados com o uso de técnicas criptográficas, de modo que somente o receptor possa decodificar corretamente a mensagem enviada.

O segundo pilar é o que se refere à integridade. Campos (2014) explica que o pilar da integridade se refere à proteção dos dados em trânsito em uma rede de computadores, ou outro sistema de transmissão, contra alterações impróprias. Outra função da integridade é prevenir que os dados sejam corrompidos durante o processo de envio da mensagem ao receptor final.

O pilar da disponibilidade é responsável por manter os dados sempre à disposição quando forem requisitados. “A RFC 2828 define a disponibilidade como a propriedade de um sistema ou de um recurso do sistema ser acessível e utilizável sob demanda por uma entidade autorizada do sistema.” (STALLINGS, 2010, p.10)

A vulnerabilidade é o fator que, quando não devidamente detectado e corrigido a tempo, pode levar à intrusão e ao total comprometimento do sistema. De acordo com Campos (2014, p.23), “os ativos de informação, que suportam os processos de negócio, têm vulnerabilidades. É importante destacar que essas vulnerabilidades estão presentes nos próprios ativos, ou seja, que são inerentes a eles, e não de origem externa”. Martinelo e Bellezi (2014, p.35) indicam que “as vulnerabilidades são originadas de falhas na maioria das vezes não intencionais”, e complementam que as origens desses problemas podem ser de naturezas diferentes, como problemas físicos, de hardware, de software, naturais ou falha humana.

De outro lado, o processo de intrusão é o comprometimento do sistema pelo invasor, o que, por consequência, permite ao invasor ter acesso a dados privilegiados que não estão disponíveis ao público. Campos (2014) e Stallings (2010) alegam que a intrusão é resultante da descoberta e exploração de uma vulnerabilidade, seja ela proveniente de um problema técnico ou humano. Souza (2002) também aborda o comprometimento de um sistema ao expor que a intrusão e/ou o ataque podem ser definidos como um conjunto de ações que visam a comprometer a confidencialidade, a integridade ou a disponibilidade de um sistema ou recurso computacional.

Assunção (2014) expõe que, para um invasor efetuar um ataque com sucesso, normalmente serão percorridas as etapas de reconhecimento e coleta de informações, varredura por sistemas disponíveis, identificação e exploração de falhas, instalação de portas dos fundos e limpeza de rastros. Além de essas etapas

serem utilizadas para acessos não autorizados e tentativas de invasão a sistemas, também são utilizadas por profissionais de segurança para fazer um *penetration test*, ou teste de invasão.

2.3 Penetration test

Penetration test é uma técnica em que são utilizadas ferramentas de hackers para fazer a análise de um sistema ou rede em busca de falhas de segurança. Souza et al. (2014, p4) afirmam que “os testes de penetração correspondem a uma técnica que procura fazer uma tentativa de invasão legal”. Assunção (2014) refere-se ao profissional treinado para descobrir, identificar e explorar novas falhas em sistemas como hacker ético, indivíduo contratado pela empresa testada, portanto, detentor de uma autorização para executar tais testes.

Um *Penetration Test* exige um planejamento cuidadoso do escopo do que será testado, assim como a duração do processo e os itens contidos no relatório final. No início do planejamento, o hacker ético irá colher as informações necessárias para a execução do teste. Entre estas informações, podemos citar o escopo do projeto, seu objetivo, duração, tarefas a serem realizadas, custo do serviço, forma de pagamento e as metas que devem ser alcançadas.

As fases de realização de um teste de invasão são planejamento, execução e pós-teste.

O cronograma da realização de um *Penetration Test* varia de acordo com três fatores: a quantidade de tarefas a serem realizadas, a quantidade de equipamentos que serão testados e o número de horas diárias demandadas para realização do teste. O tempo médio é de duas a quatro semanas para cada uma das três fases, o que resulta em uma duração média total entre seis a doze semanas para um teste completo do tipo caixa preta (*black-box*).³

2.3.1 – Metodologias de Penetration Test

Assim como ocorre com outras áreas de TI (tecnologia da informação), além da Segurança da Informação, diversas entidades desenvolveram normas para uma

³ Tipos de testes serão abordados no item 2.3.2.

maior padronização das tarefas a serem realizadas em um *penetration test*. Estes padrões são importantes, pois facilitam a organização de um processo de análise de vulnerabilidades em larga escala.

Os dois padrões mais conhecidos são o OSSTMM (*Open Source Security Testing Methodology Manual*), desenvolvido pela ISECOM (*Institute for Security and Open Methodologies*), e o padrão ISSAF (*Information Systems Security Assessment Framework*), criado pelo OISSG (*Open Information Systems Security Group*).

O *Open Source Security Testing Methodology Manual* (ISECOM, 2015), ou simplesmente OSSTMM, é a metodologia mais popular para a padronização de um *penetration test*. Os testes padronizados são explicados em um alto nível de detalhes e incluem também a análise de diversos tipos de tecnologias de redes sem fio, como *bluetooth*, infravermelho e Wi-Fi.

A documentação do OSSTMM não é considerada um ferramental, portanto, as técnicas sugeridas podem ser utilizadas com qualquer ferramenta de análise de vulnerabilidades compatível com o teste proposto. Por este motivo, apesar de oferecer uma maior flexibilidade na realização dos processos, o OSSTMM tem uma maior curva de aprendizado.

O *Information Systems Security Assessment Framework*, ou ISSAF, é outro manual de padronização desenvolvido para a realização de *checklists* de auditoria em sistemas, abordando também técnicas para a análise de vulnerabilidades. Sua documentação é dividida em duas partes: Gerenciamento geral do processo (ISSAF0.2.1A) e *Penetration Testing* (ISSAF0.2.1B).

Uma característica do ISSAF é o fato de não exigir conhecimento prévio em ferramentas e sistemas operacionais. A segunda parte de seu manual (ISSAF0.2.1B) aborda, de forma detalhada, com imagens e exemplos, o uso do ferramental necessário para a realização dos testes. A documentação do ISSAF propõe a divisão do Penetration Test em cinco fases distintas: planejamento, análise, tratamento, resultados e manutenção.

2.3.2 – Tipos de testes

Stuart, Scambray e Kurtz (2014) concordam com Assunção (2009) e Engebretson (2013), ao afirmarem que a definição do tipo de teste a ser realizado

afeta diretamente os resultados do processo, sendo, portanto, muito importante compreender e escolher o teste correto para cada situação.

Tanto o OSTMM quanto o ISSAF definem três tipos de *penetration test* que podem ser feitos. São os testes de caixa preta (*Black-box*), caixa branca (*White-box*) e caixa cinza (*Graybox*). Todos os tipos apresentam muitas similaridades, sendo diferenciados por seus objetivos finais e o escopo.

Vieira, Antunes e Madeira (2009) explanam o funcionamento do teste do tipo *black box*:

[...]consists in the analyses of the execution of the application in search for vulnerabilities. In this approach, also known as penetration testing, the scanner does not know the internals of the web application and it uses fuzzing techniques over the web HTTP requests (VIEIRA; ANTUNES; MADEIRA. 2009, p. 2)⁴

O invasor ou aquele que simula a invasão deve obter todo o tipo de informação para conseguir descobrir os pontos fortes e fracos de um sistema. A maior vantagem desse teste é a simulação, em modo real, da visão de um atacante externo sobre o sistema, e a identificação de possíveis vazamentos de informação. As desvantagens incluem um maior tempo de realização e o fato de mostrar apenas a visão de um invasor externo quando, na realidade, a maioria dos ataques é originada por pessoas que estão dentro do ambiente.

Testes de caixa branca (*White-box*) são feitos com total conhecimento preliminar da rede. Seus objetivos são mais específicos, geralmente são utilizados para descobrir vulnerabilidades no sistema e não estão focados na visão de um invasor externo e nem na descoberta de vazamentos de informações.

O teste *gray-box* é utilizado para identificar possíveis métodos de ataque que poderiam ser feitos por alguém que está dentro da corporação. O exemplo comum é o de um funcionário que tem conhecimento parcial da rede na qual ele se encontra e tenta um ataque contra outro departamento. Atualmente, muitos ataques são iniciados dentro das empresas, o que torna o *Gray-box* uma opção de teste importante para consideração.

⁴ Tradução do autor: [...] consiste na análise da execução da aplicação à procura de vulnerabilidades. Nessa abordagem, também conhecida como teste de penetração, a varredura é iniciada sem o conhecimento interno da aplicação web e usa técnicas de mascaramento para requisições HTTP na web.

2.3.3 – Relatório final

Após a conclusão da análise das vulnerabilidades, é necessário reunir e organizar todas as informações obtidas durante o processo, como, por exemplo, os tipos de problemas encontrados e o nível de gravidade de cada um. Em geral, utiliza-se uma classificação de acordo com a gravidade da falha encontrada: se é de baixo risco, médio risco ou alto risco. Essas informações devem estar contidas no relatório final.

O relatório final é o documento que será composto pelas informações e falhas encontradas durante o *penetration test*. Deve seguir um modelo claro, de preferência com base em tópicos, sem conter informações confusas. Assunção (2014) recomenda quatro itens essenciais que devem estar contidos no relatório: introdução, detalhes do trabalho, resultados obtidos e recomendações.

Além do relatório final, McClure, Scambray e Kurtz (2014) recomendam a criação de um termo de responsabilidade, que deverá ser assinado pelo prestador do serviço de *Penetration Test*, assim como pela empresa responsável pela contratação do teste. Engebretson (2013) e Assunção (2014) também concordam com a importância deste documento, de forma a proteger legalmente o hacker ético, já que o *Penetration Test* pode levar ao comprometimento do sistema pela exploração de vulnerabilidades, principalmente em ambientes web, o que pode ocasionar acesso a informações confidenciais.

2.4 Vulnerabilidades no ambiente web

O ambiente web é composto pelo servidor que hospeda as páginas, a linguagem de programação utilizada, além de gerenciadores de conteúdo como o Wordpress e outras soluções. É um ambiente heterogêneo que precisa trabalhar como uma única entidade.

Doupé (2012) explica que a ocorrência das vulnerabilidades de uma aplicação web pode ser diminuída por meio de uma melhor educação do desenvolvedor ou pelo uso de ferramentas que permitem a realização de testes de segurança. A entidade OWASP (*Open Web Application Security Project*) tem projetos que visam

tanto a educar os programadores, quanto a ajudá-los no processo de identificação das falhas web.

2.4.1 Projeto OWASP (*Open Web Application Security Project*)

O projeto OWASP resulta de esforços da entidade sem fins lucrativos de mesmo nome, tendo como objetivo encontrar e combater a causa da insegurança em aplicativos web. Silva et al. (2014) expõem que a OWASP não é afiliada com nenhuma empresa de tecnologia, portanto, as informações de segurança geradas pela comunidade do projeto são imparciais e práticas.

De acordo com Oliveira (2012, p. 49), “os projetos OWASP são divididos em duas categorias: desenvolvimento e documentação”. Curphey et al. (2005) e Meucci (2008) citam alguns dos principais projetos desenvolvidos:

- *Development Guide* - é o projeto de documentação que foca na segurança durante o desenvolvimento de aplicações web. Ele mostra vários aspectos pertinentes à segurança das soluções
- *Testing Guide* - é um guia prático que norteia as seguintes questões: o que testar, por que testar, quando, onde e como testar as aplicações web em busca de vulnerabilidades.
- *Code Review Guide* - é um guia que mostra boas práticas para a revisão segura de códigos com o objetivo de encontrar vulnerabilidades. É um pouco mais técnico e extenso do que o *Testing Guide*.
- *Top Ten* - é o guia mais popular da OWASP. O *Top Ten* é basicamente um documento que referencia e explica as dez principais vulnerabilidades das aplicações web. Mostra as consequências da exploração de cada uma dessas falhas, assim como algumas técnicas para se proteger contra elas.

Basso (2010, p.20) observa que, através da OWASP, “é possível encontrar informações sobre as vulnerabilidades; explicações sobre como funcionam os ataques a elas”. Já Carvalho (2014) expõe que o documento Top Ten não lista necessariamente as falhas mais críticas de um ambiente web, e sim as mais

frequentes. Por esta razão, ao longo do tempo, a lista vai sofrendo alterações: algumas vulnerabilidades são removidas e outras colocadas em seu lugar.

2.4.2 As dez principais vulnerabilidades da web

De acordo com a OWASP, a maioria das vulnerabilidades da lista ocorre durante a etapa de desenvolvimento da aplicação. Muniz e Lakhani (2013) concordam com Pauli (2014), ao afirmarem que, atualmente, a falha de injeção de código continua sendo a mais frequente a ser encontrada em aplicações web. Devido ao caráter dinâmico das vulnerabilidades, que podem ganhar maior ou menor ênfase ao longo do tempo, a lista *Top Ten* da OWASP é alterada a cada um ou dois anos, pois novas vulnerabilidades surgem, algumas outras perdem a sua eficácia ou até mesmo deixam de existir. Na Figura 3 há uma comparação da versão 2010 e 2013 da Lista *Top Ten*.

Figura 3. Comparativo da lista *Top Ten* dos anos de 2010 e 2013.

OWASP Top 10 – 2010 (Anterior)	OWASP Top 10 – 2013 (Novo)
A1 – Injeção de código	A1 – Injeção de código
A3 – Quebra de autenticação e Gerenciamento de Sessão	A2 – Quebra de autenticação e Gerenciamento de Sessão
A2 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Referência Insegura e Direta a Objetos	A4 – Referência Insegura e Direta a Objetos
A6 – Configuração Incorreta de Segurança	A5 – Configuração Incorreta de Segurança
A7 – Armazenamento Criptográfico Inseguro – Agrupado com A9 →	A6 – Exposição de Dados Sensíveis
A8 – Falha na Restrição de Acesso a URL – Ampliado para →	A7 – Falta de Função para Controle do Nível de Acesso
A5 – Cross-Site Request Forgery (CSRF)	A8 – Cross-Site Request Forgery (CSRF)
<Removido do A6: Configuração Incorreta de Segurança>	A9 – Utilização de Componentes Vulneráveis Conhecidos
A10 – Redirecionamentos e Encaminhamentos Inválidos	A10 – Redirecionamentos e Encaminhamentos Inválidos
A9 – Proteção Insuficiente no Nível de Transporte	Agrupado com 2010-A7 criando o 2013-A6

Fonte: www.owasp.org/images/9/9c/OWASP_Top_10_2013_PT-BR.pdf. Acesso em 01 de setembro de 2015.

Percebe-se que algumas das vulnerabilidades mudaram sua ordem de classificação, enquanto outras deixaram de existir, caso do antigo A6 – “Configuração Incorreta de Segurança.”. Em seu lugar, temos a falha de “Exposição

de dados sensíveis”. Na presente pesquisa, adota-se a versão de 2013 como base, haja vista que este é o padrão mais atual em vigor.

O primeiro objetivo motivador deste trabalho foi pesquisar e descrever quais os riscos de segurança mais frequentes em ambientes web. Tendo como referência os documentos da OWASP Foundation (OWASP, 2013)⁵, os principais riscos são discutidos a seguir.

A1 – Injeção de Código

Falhas de injeção de código resultam do processo de envio de informações não confiáveis, que são enviados para o interpretador da página web, geralmente por meio de uma requisição ou consulta. A aplicação web pode executar comandos inesperados em razão dos dados maliciosos enviados pelo invasor, levando-o a acessar dados não autorizados. De acordo com a OWASP, a injeção de código é ainda o tipo de falha mais frequente de ser encontrada e tem alto impacto nos sistemas computacionais, tais como aplicações em datacenters.

As vulnerabilidades associadas à injeção podem surgir em todo tipo de lugar na aplicação web em que se permita que o usuário forneça dados de entrada maliciosos. Alguns dos ataques mais comuns de injeção têm como alvo as funcionalidades:

- Comandos executados no sistema operacional (SO);
- XPATH (*XML Path Language*);
- LDAP (*Lightweight Directory Access Protocol*);
- SQL (*Structured Query Language*);
- Analisadores XML (*Extensible Markup Language*); e
- Cabeçalhos SMTP (*Simple Mail Transfer Protocol*)

Basso (2010, p.19) alerta que “vulnerabilidades do tipo injeção de SQL permitem que atacantes consigam, por meio de manipulação de entrada de dados de uma aplicação, interagir com as respectivas bases de dados”.

⁵ Disponível em <https://www.owasp.org/images/9/9c/OWASP_Top_10_2013_PT-BR.pdf>. Acesso em 01 de Setembro de 2015.

Diante dessa perspectiva, sempre que a entrada do usuário for aceita pela aplicação web e processada sem a sanitização adequada, a injeção pode ocorrer. Isso significa que o hacker pode influenciar o modo como as consultas e os comandos da aplicação web são compostos e os dados a serem incluídos nos resultados.

A2 – Quebra de Autenticação e Gerenciamento de Sessão

Silva et al. (2014) afirmam que a quebra de autenticação é o processo no qual o invasor consegue acesso indevido ao sistema sem ter necessariamente descoberto credenciais de usuário, como o login e a senha. Em páginas de login que contêm falhas de injeção de SQL, é possível se logar ao sistema apenas manipulando as requisições do banco de dados de forma maliciosa.

As sessões são uma maneira de manter o acesso do usuário durante o tempo de uso da aplicação web, correspondendo a identificadores únicos atribuídos após a autenticação. Existem diversos ataques que se utilizam desses identificadores para serem realizados, como o sequestro de sessão.

Gerenciamento de sessões mal planejadas após a autenticação do usuário é comum em muitas aplicações web. Isso permite que invasores comprometam diversas informações importantes como identificadores de sessão ou mesmo senhas de acesso. Também é possível sequestrar a sessão com o objetivo de assumir a identidade de outros usuários.

A3 - Cross-Site Scripting

As falhas do tipo XSS (*Cross-site Scripting*) acontecem sempre que uma aplicação recebe dados não confiáveis e devolve essas informações ao navegador do usuário sem uso de um filtro ou outro tipo de validação. O *Cross Site Scripting* permite que os invasores rodem código *javascript* no *browser* da vítima, o que permite realizar diversos tipos de ações maliciosas, entre elas o sequestro de sessão causado pelo roubo do *cookie* utilizado para autenticação. O XSS está diretamente ligado às falhas de quebra de autenticação e gerenciamento de sessão.

Torres (2014) e Silva et al (2014) concordam que o *Cross Site Scripting* pode ser do tipo armazenado (*stored*) ou refletido (*reflected*). O tipo refletido é mais

comum de ser encontrado e oferece um perigo menor do que o armazenado. Isso se deve ao fato de uma URL ter de ser enviada para a potencial vítima, fazendo com que o ataque tenha um alcance geralmente limitado, pois somente quem clicar no link da URL enviada será afetado.

Em determinadas ocasiões, o invasor pode injetar o script dentro do código de um site e assim infectar todos os que visitarem a página. Esse ataque seria o XSS armazenado, que é uma ameaça maior do que o refletido pela quantidade de pessoas que ele pode afetar de uma só vez. Os códigos maliciosos podem ser os mesmos do refletido, podendo, por exemplo, redirecionar os usuários para um site falso ou capturar os *cookies* de sessão.

A4 – Referência Insegura e Direta a Objetos

Se um arquivo, registro de banco de dados ou diretório é referenciado de forma direta por uma aplicação, sem nenhum tipo de filtro ou controle, há uma vulnerabilidade de referência insegura e direta a objetos.

Muitas ferramentas atualmente permitem que o invasor possa manipular os parâmetros do site de forma a ter acesso a informações sensíveis que não estão disponíveis ao público. Isso permite, por exemplo, que um atacante tenha acesso ao carrinho de compras de outra pessoa em um site de *e-commerce*. Em alguns casos, o invasor pode ter acesso a arquivos no sistema operacional, o que permite que ele obtenha os usuários e senhas locais do servidor.

Uma das recomendações feitas pela OWASP é que sejam utilizadas referências indiretas para os objetos, ao invés das diretas.

A5 – Configuração Incorreta de Segurança

Esse risco de segurança se refere a todo o ambiente externo, ao software que serve como apoio à aplicação. Essa ambiente é composto pelos elementos necessários para uma aplicação web funcionar. Portanto, temos o servidor web, ao sistema operacional e aos sistemas de gerenciamento de bancos de dados utilizados pelas aplicações web. Existem vários riscos na configuração incorreta de segurança. Os mais comuns são:

- Permissões de pastas malfeitas;

- Softwares desatualizados que podem ser explorados;
- Serviços não utilizados e ainda habilitados; e
- Serviços que são executados no contexto de usuários privilegiados.

Basicamente todo problema de segurança que afete um elemento de software que não seja a aplicação web entra nesta categoria. Uma segurança eficiente exige uma configuração segura definida e implantada na aplicação, no servidor web, no sistema operacional e no sistema gerenciador de banco de dados. Todos os elementos devem estar protegidos pelo cumprimento de boas práticas e normas de segurança.

A6 – Exposição de Dados Sensíveis

Diversas aplicações web não protegem adequadamente dados sensíveis, como números de cartão de crédito, registros CPF ou mesmo credenciais de autenticação. Dados confidenciais de clientes devem ser criptografados quando armazenados em um banco de dados, mesmo que eles não sejam diretamente acessíveis pelo aplicativo da web. O mesmo se aplica a dados sensíveis transmitidos de e para um aplicativo da web, tais como credenciais ou detalhes de pagamento. Essa informação deve ser transmitida por uma camada criptografada de modo que, mesmo que ocorra interceptação dos dados, eles estarão ilegíveis para o invasor que os capturou.

A7 – Falta de Função para Controle do Nível de Acesso

Quando uma aplicação não verifica os direitos de acesso em nível de função, ela possibilita que um invasor explore esse tipo de risco de segurança, alterando a URL no navegador ao acessar um aplicativo da web para tentar acessar uma função a que ele não tem acesso. Se o aplicativo da web não executar verificações de controle de acesso adequadas especificamente para esse objeto em particular, o atacante é capaz de acessar recursos a que ele não deveria ter acesso.

A8 – Cross-Site Request Forgery (CSRF)

O CSRF ocorre quando o invasor consegue enviar uma requisição maliciosa a um usuário autenticado, a qual permite forçar o navegador da vítima a executar tarefas no sistema em que ela esteja logada. Isso acontece porque essas requisições são consideradas legítimas, já que estão se aproveitando do contexto em que o usuário está logado e autenticado. Torres (2014) exemplifica o perigo da vulnerabilidade citando seu impacto em uma aplicação bancária.

Esse tipo de vulnerabilidade é similar ao *Cross Site Scripting* do tipo refletido, no sentido de que o invasor deverá fazer o usuário executar uma ação na aplicação. Entre algumas possibilidades do CSRF, estão a criação de novo usuário, alteração de senha de acesso e inserção de conteúdo (uma postagem em um blog, por exemplo). A solicitação será feita de forma legítima caso o invasor saiba exatamente que parâmetros ele deve manipular para enviar ao navegador da vítima.

A9 – Utilização de Componentes Vulneráveis Conhecidos

As vulnerabilidades dessa categoria são causadas pela exploração de componentes falhos do software, que não foram corrigidos pela atualização de segurança, podendo ser um *framework*, *plug-in* ou um gerenciador de conteúdo.

Se o software desatualizado estiver sendo utilizado, ele pode ser facilmente encontrado por ferramentas de análise de vulnerabilidades e, conseqüentemente, estar comprometido pelo invasor que poderá ganhar acesso privilegiado a recursos do sistema.

A10 – Redirecionamentos e Encaminhamentos Inválidos

Os visitantes do site são frequentemente redirecionados e encaminhados para diferentes páginas e até mesmo outros sites de terceiros, dependendo da localização do visitante, o tipo de navegador que está sendo usado, comportamento dentro do website, para fornecer ao usuário um tratamento diferenciado, como, por exemplo, um *banner* de promoção que leva a outro website para a compra do produto.

Se as funções que analisam esses dados não validam adequadamente os dados, invasores mal-intencionados podem explorar essas funções e utilizar sites

legítimos para redirecionar seus visitantes para um site falso. Ou mesmo utilizar os encaminhamentos para acessar páginas não autorizadas ou bloqueadas.

2.5 – Ferramentas de análise de vulnerabilidade web

Para detectar essas e outras falhas presentes em aplicações web, existem hoje no mercado dezenas de ferramentas. As grandes problemáticas envolvendo o uso dessas ferramentas é a forma de acesso, licenciamento e quantidade de funcionalidades presentes nas aplicações:

Most of these scanners are commercial tools (Acunetix Web Vulnerability Scanner, IBM Rational AppScan, and HP Web Inspect), but there are also some free application scanners (Foundstone WSDigger and wsfuzzer) with limited use, as they lack most of the functionalities of their commercial counterparts. (VIEIRA, ANTUNES, MADEIRA. 2009, p.1)⁶

Como afirmam Vieira, Antunes e Madeira (2009), algumas das ferramentas comerciais com versões gratuitas disponíveis costumam apresentar limitações de recursos, e as ferramentas comerciais pagas têm valores que variam de algumas dezenas a milhares de dólares, muitas vezes com alto custo em comparação aos benefícios agregados. Entre algumas das ferramentas disponíveis, estão o *Acunetix*, o *Nessus* e o *Burp Suite*. Entretanto, alguns fatores devem ser considerados para a escolha da ferramenta que será utilizada para a verificação das falhas.

De acordo com Assunção (2014), um dos fatores relevantes na escolha de uma ferramenta de análise de vulnerabilidades em ambientes web é a quantidade de falsos positivos e falsos negativos gerados após o processo de varredura ser finalizado. Um falso positivo ocorre quando a falha não existe no sistema, mas o software de detecção a identifica erroneamente. Já o falso negativo é quando a vulnerabilidade existe, mas não é detectada pela ferramenta de análise.

⁶ Tradução do autor: “A maioria desses scanners são ferramentas comerciais (Acunetix Web Vulnerability Scanner, IBM Rational AppScan, e HP WebInspect), mas também existem algumas aplicações gratuitas (Foundstone WSDigger e wsfuzzer) com uso limitado, faltando a maioria das funcionalidades das suas contrapartes comerciais.”

Outro fator a ser considerado é a implementação de um perfil para a verificação das falhas citadas na lista *OWASP Top Ten*. Como periodicamente este documento sofre modificações, em virtude dos estudos que identificam a frequência das vulnerabilidades web, a ferramenta deve oferecer atualizações frequentes em sua relação de falhas detectáveis, para se manter relevante ao longo do tempo.

O site *SecTools.org* mantém um ranking das ferramentas de segurança e análise de vulnerabilidades mais populares, segundo especialistas da área (MARTINELO e BELLEZI, 2014).

3. METODOLOGIA DE PESQUISA

A pesquisa realizada teve cunho qualitativo, utilizando os métodos bibliográficos e experimental. De acordo com Marconi e Lakatos (2003, p.201), “metodologia de pesquisa é aquela que abrange o maior número de itens, pois responde, a um só tempo, às questões: Como? Com quê? Onde? Quanto?”.

Em relação ao delineamento da pesquisa, ela tem caráter experimental, sendo comparadas cinco ferramentas de análises de vulnerabilidades para conhecer a eficiência de cada uma delas na detecção das dez principais falhas da lista *Top Ten OWASP*, na versão 2013 do documento.

A lista *Top Ten OWASP*, periodicamente atualizada, apresenta os dez riscos de segurança mais frequentes em um servidor web. De acordo com Carvalho (2014, p. 5), a lista de vulnerabilidades *OWASP Top Ten* “é desenvolvida e atualizada a partir de consultas feitas a especialistas em Segurança da Informação e organizações de portes e ramos diversos, na esfera governamental ou privada, em vários países”. Por esta razão, optou-se por realizar o experimento de análise de falhas tendo como referência a lista dos dez riscos mais frequentes da OWASP, considerada referência na área de Segurança da Informação.

As ferramentas foram selecionadas inicialmente através ranking do site “SecTools.Org”⁷, que faz uma listagem das cento e vinte cinco ferramentas de segurança utilizadas e reconhecidas pela comunidade de Segurança da Informação. O ranking se baseia na opinião de mais de dois mil profissionais da área de Segurança da Informação, que concedem notas às ferramentas de acordo com seus

⁷ Disponível em <<http://sectools.org/about>>. Acesso em 01 de Setembro de 2015.

recursos e popularidade. Após o filtro inicial das soluções que sejam de código fonte livre e especializadas em vulnerabilidades web, foi realizada uma consulta aos artigos selecionados para identificar quais das ferramentas foram mais citadas pelos autores pesquisados.

A parte experimental deste trabalho se baseou nos experimentos de Basso (2010), que conduziu experimentos com três ferramentas em diferentes cenários, testando apenas as vulnerabilidades de *Injection*, *XSS* e *CSRF*, e nos testes conduzidos por Torres (2014), que executou um teste básico em trinta ferramentas diferentes de análise de varredura para desenvolver um processo de análise tradicional de risco. Também foram utilizados como base da pesquisa os artigos de Martinelo (2014), que compara a utilização de uma ferramenta comercial e outra de código fonte livre para análise de falhas em um ambiente web e o artigo de Silva et al (2014), que utiliza seis soluções de softwares com código fonte livre para analisar as vulnerabilidades da aplicação web Moodle, rodando em um ambiente simulado.

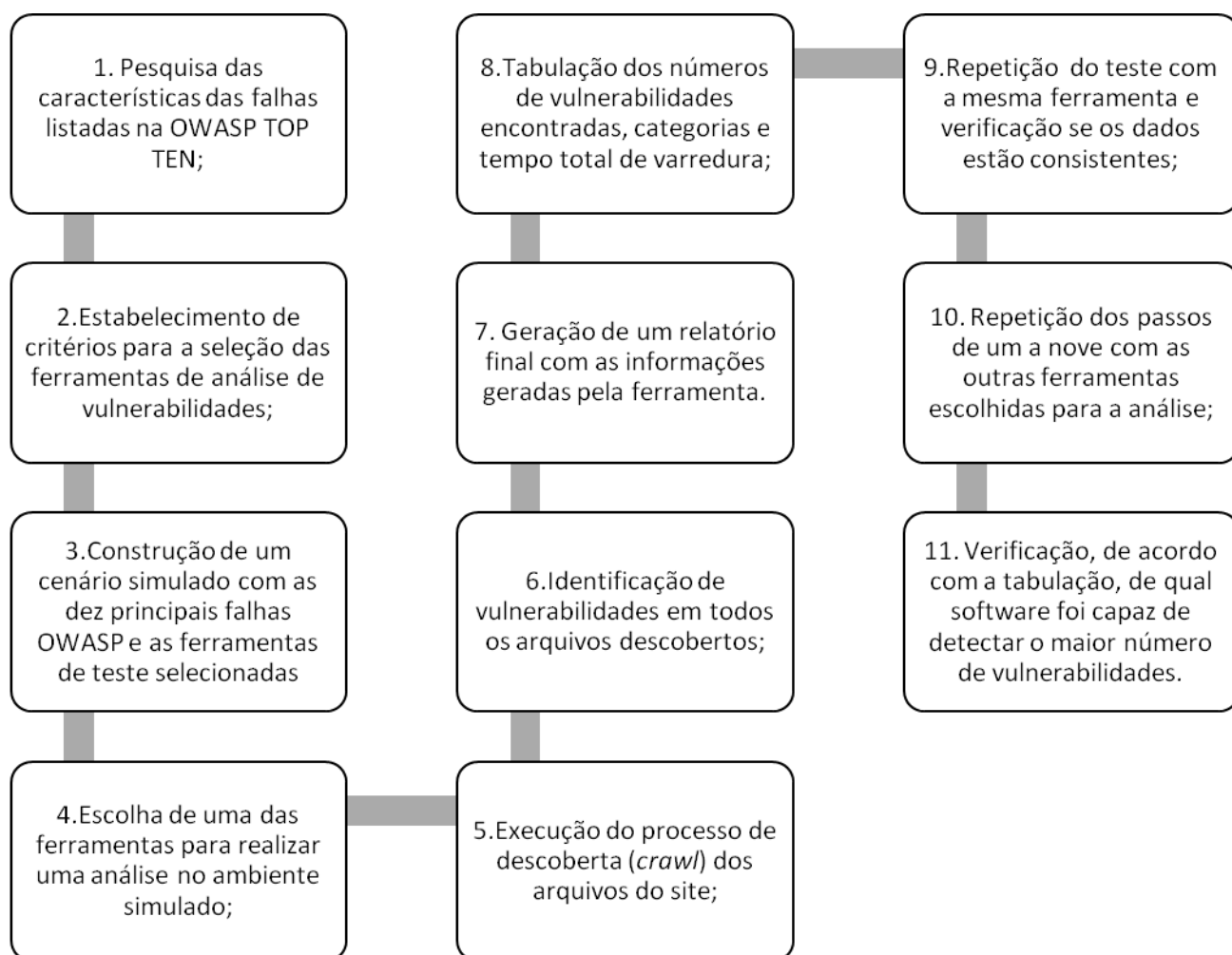
Para o experimento, foram escolhidas cinco ferramentas para a análise de vulnerabilidades das falhas do OWASP Top Ten. A seleção levou em consideração apenas as ferramentas *open source* e seu ranking no site SecTools. A preferência de ferramentas de *open source* é justificada pela possibilidade de auditoria do software por qualquer profissional, visto seu caráter aberto.

Durante o experimento, além dos softwares de teste selecionados, foi utilizado o framework *Mutillidae*⁸, que permite simular as vulnerabilidades descritas no documento *OWASP Top Ten* para fins de teste. O *Mutillidae* é um projeto da própria OWASP, que fornece um site completo com diversas páginas com falhas propositalmente inseridas para mostrar o perigo dessas vulnerabilidades em um ambiente real.

Foram coletados os seguintes dados sobre as ferramentas: quantidade de páginas descobertas automaticamente, falhas detectadas e o tempo total levado para a detecção. Para fins do trabalho, a ferramenta mais eficiente será considerada aquela que identificar o maior número total de vulnerabilidades no ambiente, descartando-se os falso positivos. A pesquisa experimental seguiu o fluxograma exposto na Figura 4.

⁸ Disponível em <<http://sourceforge.net/projects/mutillidae/>>. Acesso em 06 de Julho de 2015.

Figura 4. Fluxograma de passos adotado na pesquisa



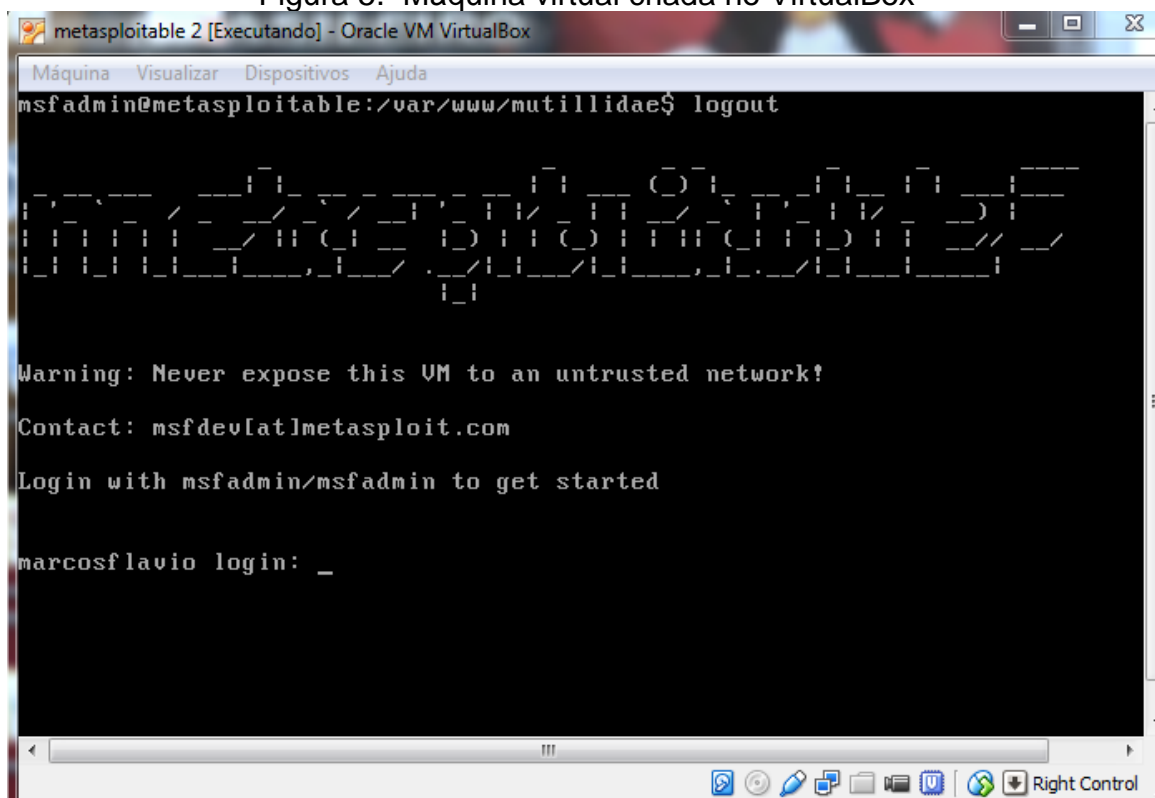
Fonte: Elaborado pelo autor.

As análises dos resultados obtidos na etapa de testes experimentais da pesquisa verificaram os arquivos de páginas descobertos automaticamente pela aplicação, através do processo de *crawling* (descoberta, ou seja, identificação automática de recursos), do número total de falhas detectadas e do tempo total de varredura gasto por ferramenta. Entretanto, o critério da pesquisa é a comparação das soluções com base no total de vulnerabilidades identificadas, excluindo-se os falsos positivos. Os dados obtidos foram comparados e permitiram a formulação de um ranking de ferramentas como uma resposta à questão inicial motivadora dessa pesquisa.

3.1 Construção do ambiente de testes simulados

Para a construção do ambiente para realização dos testes, uma máquina virtual foi preparada com todos os recursos necessários para a utilização do framework *Mutillidae*, que fornece um conjunto de páginas com diversas vulnerabilidades com o propósito de oferecer um aprendizado prático para profissionais da área de Segurança de Redes. O software de gerenciamento de máquinas virtuais escolhido foi o VirtualBox. Foi utilizada a imagem da máquina virtual chamada de “Metasploitable”, Figura 5, distribuída gratuitamente pelo site *Offensive Security*, já contendo o software *Mutillidae* instalado.

Figura 5. Máquina virtual criada no VirtualBox

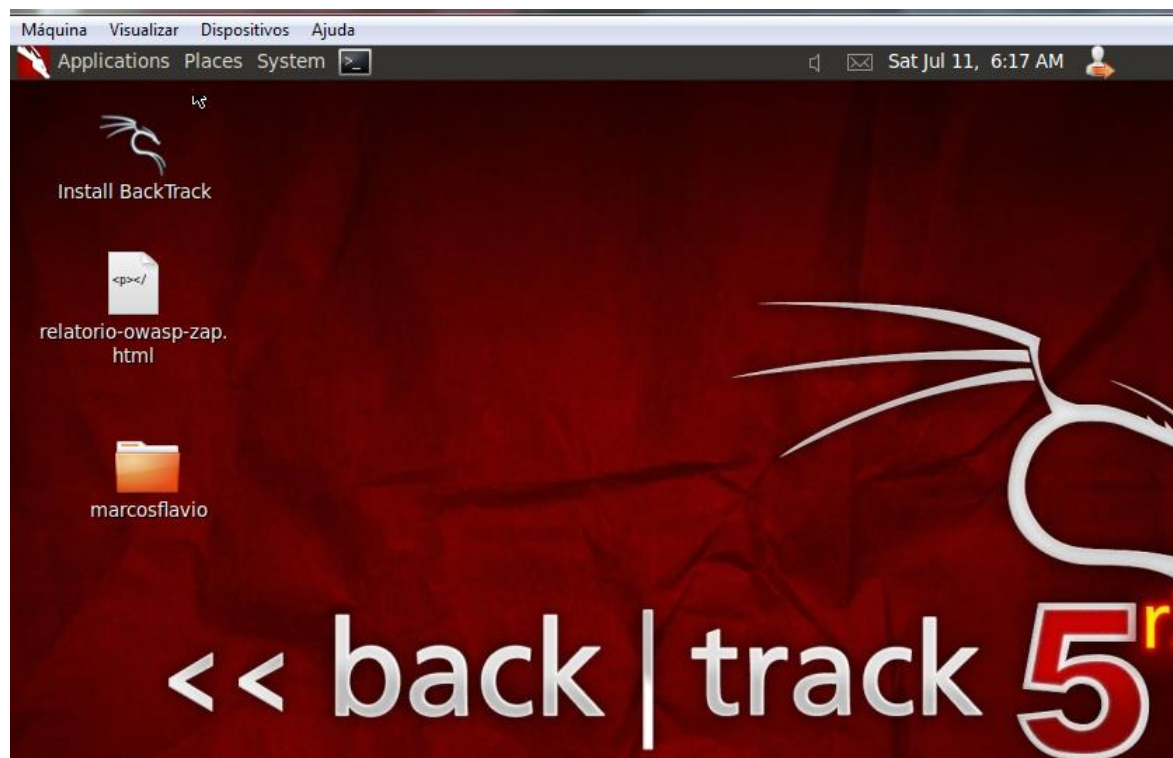


Fonte: *Print screen* gerado pelo autor.

O *BackTrack* foi a distribuição *Linux* escolhida para a criação da máquina virtual que contém as ferramentas para os testes das falhas. Todas as ferramentas utilizadas nesse trabalho já vêm pré-instaladas com a distribuição. Apesar de existirem outras distribuições mais recentes voltadas a Penetration Test, como o Kali Linux, a preferência pela escolha pelo BackTrack se deu pela experiência pessoal

do autor, que considera esse sistema mais estável. Todas as ferramentas utilizadas nos testes foram devidamente atualizadas para a versão mais recente. A Figura 6 mostra a tela principal do BackTrack Linux.

Figura 6. Tela principal do BackTrack Linux



Fonte: *print screen* gerado pelo autor.

Primeiramente, foi feito um teste de *ping* entre as duas máquinas para verificar se ambas estão se comunicando corretamente. O resultado foi positivo: ambas responderam, e a comunicação foi estabelecida com sucesso. Em seguida, foi tentado o acesso ao *Mutillidae*, Figura 7, pela máquina BackTrack. Esse passo é essencial pois o acesso é imprescindível para a realização dos testes. O endereço para acesso configurado foi <http://192.168.10.3/mutillidae>. O navegador Firefox instalado na máquina BackTrack acessou com sucesso o site hospedado no Linux Debian.

Figura 7. Teste de acesso ao *Mutillidae*

Fonte: *Print screen* gerado pelo autor.

A próxima seção descreve como foi definido e feito o processo de filtro e de seleção das ferramentas utilizadas pelo trabalho, também apresentando as características básicas e os principais recursos de cada uma das soluções de software livre escolhidas para a análise de vulnerabilidades no ambiente web proposto.

3.2 Seleção das ferramentas *open source* para testes de vulnerabilidade web

O processo de seleção dos *scanners* de vulnerabilidades foi feito segundo três critérios básicos para garantir soluções adequadas para o propósito e o escopo do experimento proposto.

O primeiro critério de seleção utilizado foi a citação das ferramentas *open source*. Durante a pesquisa, notou-se existência de uma gama de ferramentas para análise de falhas em ambientes web, muitas delas de *open source*, sendo, por este motivo, desconsideradas. Tomou-se como referência o trabalho desenvolvido por Basso (2010), que cita a utilização dos *scanners* *Acunetix*, *AppScan* e *HP WebInspect*. Como não são soluções livres, elas foram descartadas. É a mesma situação do *scanner* *Nessus*, que, a partir de sua terceira versão, passou a ter o seu código fonte fechado, sendo, por essa razão, também desconsiderado.

Dos autores pesquisados que citaram ferramentas *open source*, temos os trabalhos de McClure (2014) e Assunção (2014), que sugerem em suas obras o uso das soluções *Nikto* e *W3af* para a verificação de vulnerabilidades em ambientes web, aplicações que também são citadas por Torres (2014). Martinelo (2014) analisa em seu trabalho a utilização da ferramenta *OpenVAS*, assim como os autores Assunção (2014) e Doupé (2014).

Os trabalhos de Pauli (2014) e Engebretson (2013) citam também o uso das ferramenta de código aberto *Skipfish* e *SQLMAP* como uma solução recomendada na análise de vulnerabilidades de injeção de código e ataques de *cross site scripting*, recomendação também compartilhada por Torres (2014). Muniz e Lakhani (2013) concordam com Pauli (2014), McClure (2014) e Torres (2014) ao citarem a utilização do software *open source* *OWASP ZAP* como uma opção recomendada de uma ferramenta proxy para análise de ambientes web. Silva et al. (2014) também concordam com os outros autores quanto às ferramentas *OWASP ZAP*, *Nikto* e *W3AF*, utilizando-as em seu artigo para análise de vulnerabilidades no software *Moodle*.

O segundo critério utilizado foi a verificação da posição das ferramentas através do site *SecTools*⁹, que faz uma enquete quantitativa para estabelecer um ranking das soluções de *software* mais populares para utilização em segurança de redes e sistemas, de acordo com a opinião de profissionais da área. Optou-se por selecionar apenas cinco ferramentas para o teste, visando a um maior aprofundamento da utilização dos recursos de cada uma.

Essa análise revelou que as ferramentas *OWASP ZAP*, *SQLMAP*, *W3AF*, *NIKTO* e *SKIPFISH* foram as que tiveram melhor resultado no ranking, tendo sido

⁹ Disponível em: <<http://sectools.org/tag/web-scanners/>>. Acesso em 17 de novembro de 2015.

as escolhidas para esta pesquisa. O *OpenVAS* não foi escolhido, visto, de acordo com Martinelo (2014), ser considerado um software de análise de vulnerabilidades genérico, que verifica superficialmente falhas em diversos tipos de serviços ao invés de focar apenas no ambiente web e suas falhas relacionadas. No ranking do *SecTools*, o *OpenVAS* também não está listado no ranking de ferramentas especializadas em análise de vulnerabilidades web.

3.2.1 – OWASP ZAP

O *OWASP Zap*¹⁰ é uma ferramenta *open source* criada pela OWASP, que age como um servidor *proxy* com a finalidade de detectar vulnerabilidades em aplicações web. Este é um dos projetos mais ativos da OWASP, já tendo sido traduzido para mais de 25 linguagens diferentes.

De acordo com Silva et al. (2014, p.6), a ferramenta “fornece scanners automatizados, bem como um conjunto de ferramentas que permitem encontrar vulnerabilidades de segurança manualmente”. Torres (2014, p.76) complementa que o OWASP ZAP é “muito semelhante ao programa Burpsuite, que permite fazer testes de segurança”.

A ferramenta foi desenvolvida em Java, o que permite que ela rode em diversos sistemas operacionais como Windows, Linux ou Mac OS X.

Entre os recursos oferecidos pelo *OWASP Zap*, destacam-se:

- **Proxy ativo** – O recurso de servidor Proxy do Zap permite capturar e editar o tráfego em tempo real, modificando as requisições e respostas enviadas do cliente para o servidor e vice-versa.
- **Web Crawler** – Funcionalidade do programa que permite, a partir da página principal fornecida com o endereço do site, seguir links encontrados para identificar automaticamente todas as outras páginas que compõem o ambiente.
- **Scanner automático** – Módulo do Zap que permite detectar automaticamente vulnerabilidades comuns em ambientes web, como *SQL*

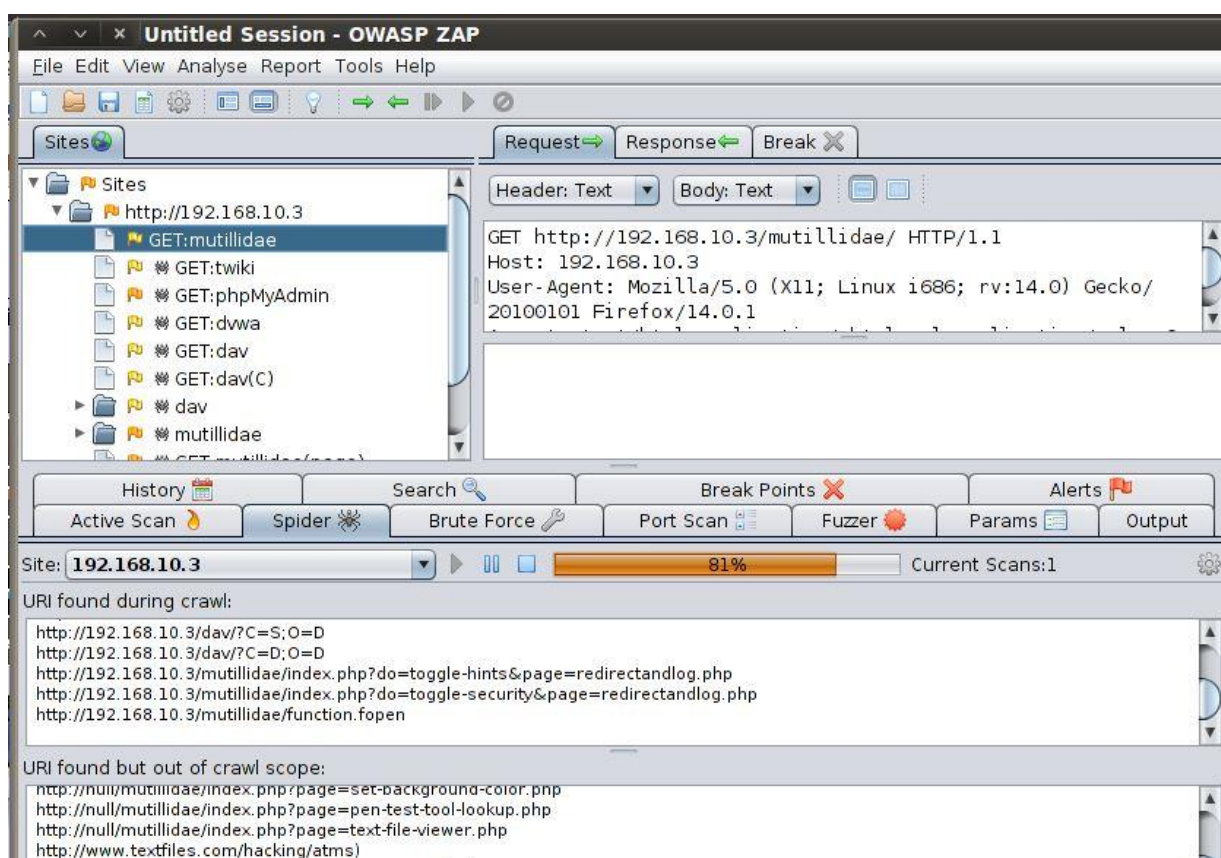
¹⁰ Disponível em: <http://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project>. Acesso em 03 de Setembro de 2015.

Injection, XSS e outras. Ele detecta primariamente as falhas descritas no documento *Top Ten* da OWASP.

- **Fuzzer** – Recurso que permite gerar tráfego aleatório e enviar para a aplicação web e depois analisar o resultado obtido. O objetivo é entender como a aplicação se comportará quando receber tipos diferentes de dados.

A Figura 8 mostra a interface principal do programa OWASP ZAP em execução.

Figura 8. Interface gráfica do OWASP ZAP



Fonte: *Print screen* gerado pelo autor.

3.2.2 – SQLMap

O SQLMAP¹¹ é uma solução *open source*, desenvolvida com a linguagem de programação Python, e de acordo com Torres (2014, p76), a ferramenta “automatiza o processo de detecção e exploração de falhas de SQL Injection”. O objetivo da utilização do software é a obtenção de acesso direto a bancos de dados por meio de aplicações web vulneráveis. O programa suporta diversos tipos de SGBD (Sistema Gerenciador de Banco de dados), como MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Firebird e outros.

Entre outros recursos do software, destacam-se:

- Suporte à enumeração de bancos de dados, tabelas e colunas;
- Suporte otimizado à enumeração de usuários e senhas;
- Suporte à quebra de senhas armazenadas em *hash*;
- Suporte à realização de conexão direta ao SGBD, sem necessidade da aplicação web; e
- Capacidade de pesquisa por informações específicas em nível de banco de dados, tabelas ou colunas.

A figura 9 mostra uma consulta realizada pela interface de texto do SQLMAP.

¹¹ Disponível em: <<http://sqlmap.org>>. Acesso em 03 de Setembro de 2015.

Figura 9. Interface de texto do SQLMAP

```
[07:21:16] [INFO] the SQL query used returns 4 entries
Database: owasp10
Table: credit_cards
[4 columns]
+-----+-----+
| Column | Type |
+-----+-----+
| ccid    | int(11) |
| ccnumber | text |
| ccv     | text |
| expiration | date |
+-----+-----+
```

Fonte: *print screen* gerado pelo autor.

3.2. 3 – Nikto

Nikto¹² é uma solução *open source* que identifica mais de 6 mil tipos diferentes de falhas em ambientes web, além de verificar problemas específicos de mais de 200 plataformas baseadas na web. De acordo com Silva et al. (2014, p.5), o Nikto “também verifica quais são os itens de configuração do servidor, tais como a presença de arquivos de índice, opções do servidor HTTP, tentando identificar os servidores web instalados e quais softwares fazem parte”. Apesar de ser um dos *scanners* mais antigos do mercado, ainda é regularmente atualizado. Os itens e *plug-ins* utilizados pelo Nikto são atualizados frequentemente e podem ser atualizados automaticamente.

Entre as principais características do Nikto, têm destaque:

- Suporte a SSL;
- Suporte a proxy HTTP;
- Verificação de componentes desatualizados;
- Geração de relatórios personalizados;
- Verificação de diversas classes de vulnerabilidades, o que inclui as falhas contidas no documento *Top Ten* da OWASP; e

¹² Disponível em <<http://cirt.net/Nikto2>>. Acesso em 03 de setembro de 2015.

- Integração com a ferramenta *Metasploit* para exploração posterior das falhas identificadas.

A figura 10 demonstra a sintaxe do comando e a interface de texto do Nikto.

Figura 10. Interface de texto do Nikto

```
root@bt:/pentest/web/nikto# ./nikto.pl -host http://192.168.10.3/mutillidae/index.php?page=user-info.php
- Nikto v2.1.5
-----
+ Target IP:          192.168.10.3
+ Target Hostname:    192.168.10.3
+ Target Port:        80
+ Start Time:         2015-07-11 00:51:59 (GMT-3)
-----
+ Server: Apache/2.2.8 (Ubuntu) DAV/2
+ Retrieved x-powered-by header: PHP/5.2.4-2ubuntu5.10
```

Fonte: *Print screen* gerado pelo autor.

3.2.4 – Skipfish

Skipfish¹³ é uma ferramenta *open source* de análise de vulnerabilidades em ambientes web que permite a identificação de arquivos através de um *crawl* recursivo e a criação de um mapa completo de todos os recursos encontrados no alvo analisado. Pauli (2014) expõe que o programa permite a geração de relatórios personalizados com todas as informações descobertas em diversos formatos de arquivos diferentes.

Entre seus principais diferenciais, destacam-se:

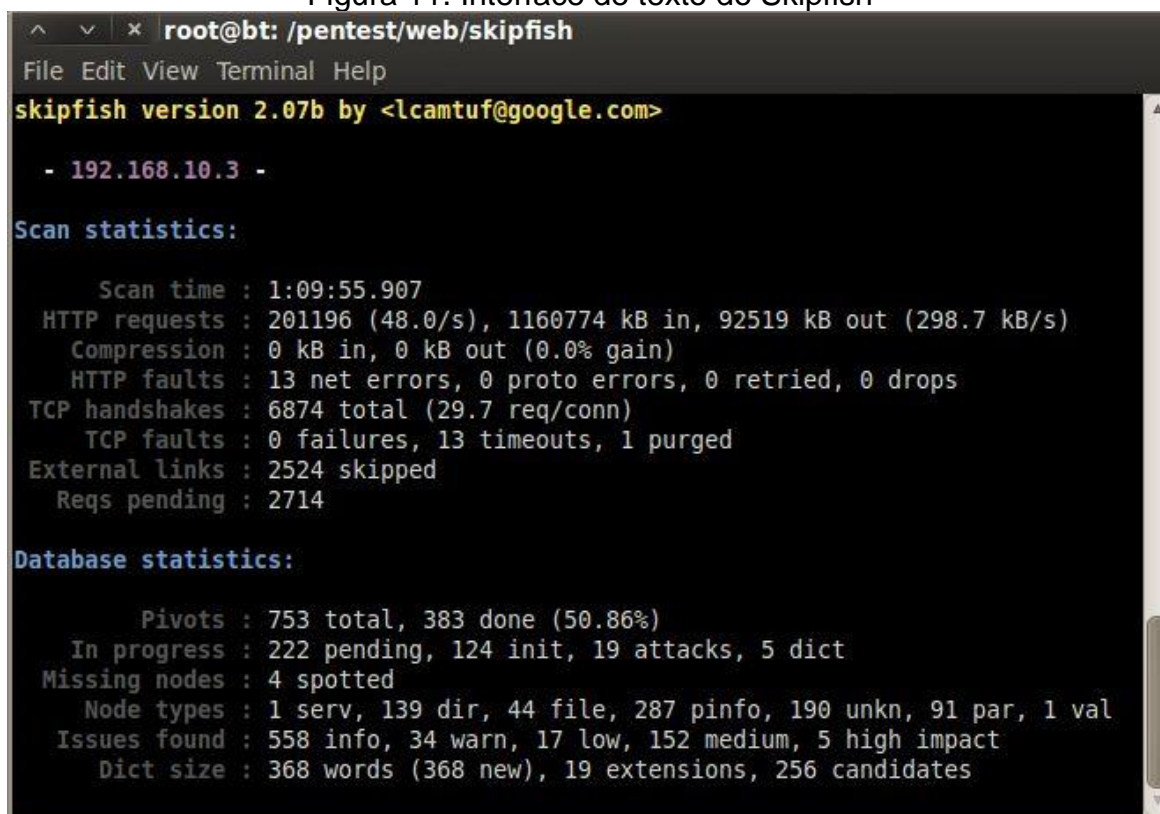
- Facilidade de utilização dos recursos extras, como criação de listas de palavras para testes de força-bruta de senhas;
- Alta velocidade na realização do teste. Otimiza o uso dos recursos da máquina e da rede, o que permite executar milhares de requisições em um curto intervalo de tempo; e

¹³ Disponível em <<http://code.google.com/p/skipfish/>>. Acesso em 06 de setembro de 2015.

- Permite detectar diversos tipos de vulnerabilidades, incluindo as da lista OWASP Top Ten, utilizando algoritmos avançados que visam a diminuir a quantidade de erros na detecção.

A figura 11 mostra o Skipfish em execução através da interface de texto.

Figura 11. Interface de texto do Skipfish



```

root@bt: /pentest/web/skipfish
File Edit View Terminal Help
skipfish version 2.07b by <lcamtuf@google.com>

- 192.168.10.3 -

Scan statistics:

  Scan time : 1:09:55.907
  HTTP requests : 201196 (48.0/s), 1160774 kB in, 92519 kB out (298.7 kB/s)
  Compression : 0 kB in, 0 kB out (0.0% gain)
  HTTP faults : 13 net errors, 0 proto errors, 0 retried, 0 drops
  TCP handshakes : 6874 total (29.7 req/conn)
  TCP faults : 0 failures, 13 timeouts, 1 purged
  External links : 2524 skipped
  Reqs pending : 2714

Database statistics:

  Pivots : 753 total, 383 done (50.86%)
  In progress : 222 pending, 124 init, 19 attacks, 5 dict
  Missing nodes : 4 spotted
  Node types : 1 serv, 139 dir, 44 file, 287 pinfo, 190 unkn, 91 par, 1 val
  Issues found : 558 info, 34 warn, 17 low, 152 medium, 5 high impact
  Dict size : 368 words (368 new), 19 extensions, 256 candidates
  
```

Fonte: *Print screen* gerado pelo autor.

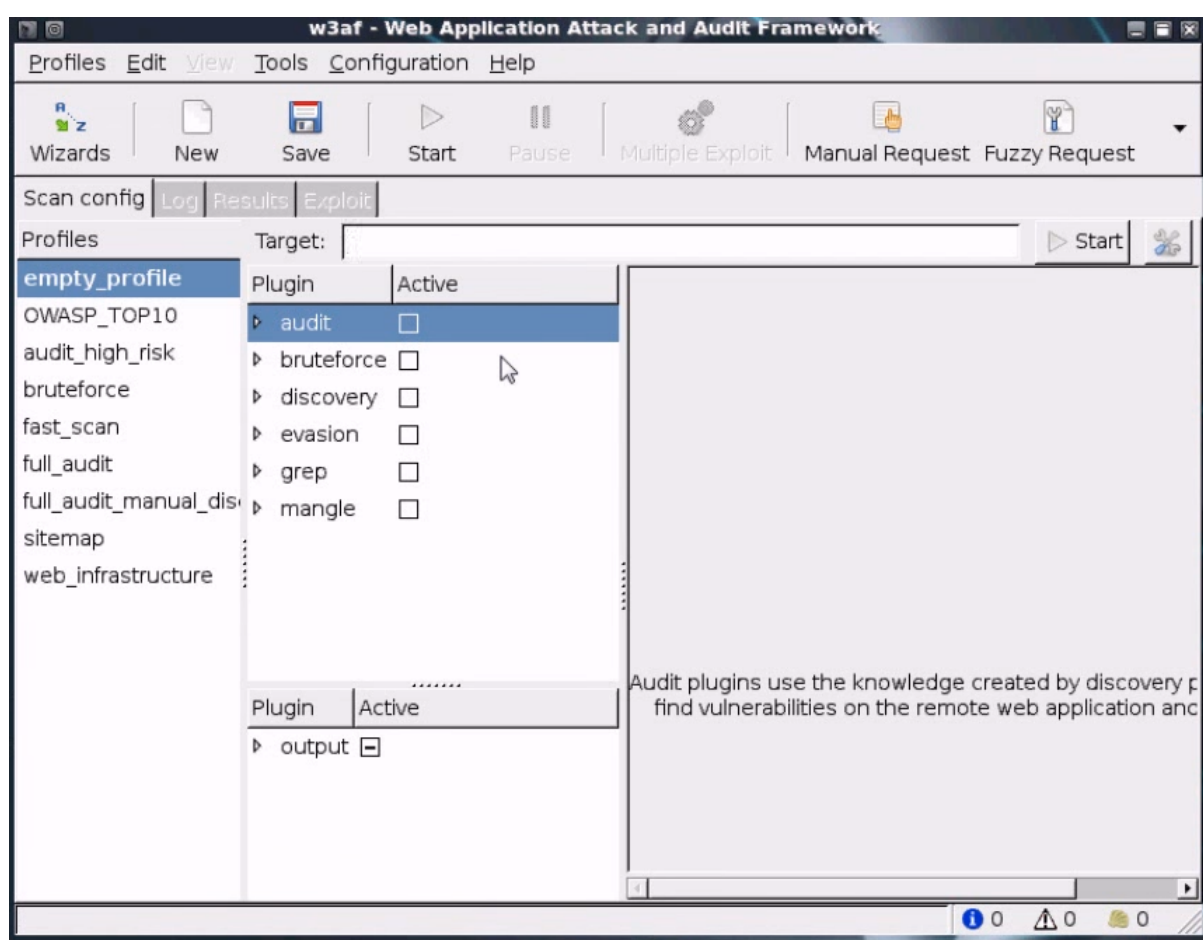
3.2.5 – W3AF

W3aF¹⁴ é a abreviação de *Web Application Attack and Audit Framework*. De acordo com Torres (2014, p. 76), o W3aF “é uma ferramenta *open source* que faz testes de vulnerabilidades contra servidores web”. Assim, o software tem como objetivo fazer a varredura e identificação de falhas em um ambiente web. Além de permitir o *crawling* para identificação de diversos tipos de arquivos e páginas do site, o programa tem mais de 200 *plug-ins* para análise de diversas categorias

¹⁴ Disponível em <<http://w3af.org>>. Acesso em 17 de Setembro de 2015

diferentes de falhas, como *SQL injection*, *XSS*, *XSRF* e inclusão de arquivos remotos. Silva et al. (2014) também expõem que a ferramenta também é utilizada para verificação de testes de injeção de código SQL. A Figura 12 demonstra a interface do W3AF.

Figura 12. Interface gráfica do W3AF



Fonte: *Print screen* gerado pelo autor.

4. RESULTADOS

Esta seção apresenta as informações obtidas após a execução dos testes experimentais propostos no ambiente do Mutillidae. No item 4.1, estão todas as vulnerabilidades existentes no cenário simulado e, nos itens posteriores, são apresentados os resultados obtidos por cada uma das ferramentas selecionadas para o teste.

4.1 Vulnerabilidades existentes no Framework *Mutillidae*

Para oferecer um procedimento padronizado que permita uma comparação mais eficiente do resultado final gerado pelas ferramentas, foram compiladas informações sobre todas as páginas que constituem o *framework* do *Mutillidae* e as vulnerabilidades contidas em cada uma delas. Essas informações estão contidas no próprio *framework* (em *Documentation/Vulnerabilities*) e foram tabuladas para facilitar a visualização e a comparação posterior com os testes realizados. O objetivo é que a Tabela 1 sirva como referência para a análise dos resultados obtidos.

Tabela 1. Páginas e Vulnerabilidades no *Mutillidae*

Página do <i>Mutillidae</i>	Parâmetros afetados	Total de falhas
Add-to-your-blog.php	<ul style="list-style-type: none"> • Injeção de SQL (blog, username) • XSS (blog, username) • CSRF • Injeção de HTML (blog) • Erro visível da aplicação 	7
Arbitrary-file-inclusion.php	<ul style="list-style-type: none"> • Comprometer arquivos do sistema • Carregar qualquer página do site 	2
Browser-info.php	<ul style="list-style-type: none"> • XSS (<i>referer</i> e <i>user-agent</i> HTTP) • <i>JavaScript injection</i> (<i>referer</i> HTTP) 	3
Capture-data.php	<ul style="list-style-type: none"> • XSS (<i>Get</i>, <i>Post</i> e <i>Cookie</i>) 	3
Captured-data.php	<ul style="list-style-type: none"> • XSS (<i>Get</i>, <i>Post</i> e <i>Cookie</i>) 	3
Credits.php	<ul style="list-style-type: none"> • Encaminhamento e Redirecionamento sem validação 	1
Dns-lookup.php	<ul style="list-style-type: none"> • XSS (host, log) • <i>SQL Injection</i> (log) • Injeção de comandos do SO (host) • Forçar GET ao invés de POST 	5
Footer.php	<ul style="list-style-type: none"> • XSS (<i>user-agent</i> HTTP) 	1
Header.php	<ul style="list-style-type: none"> • XSS (<i>username</i>, <i>signature</i>) 	2

Html5-storage.php	<ul style="list-style-type: none"> • Injeção DOM (<i>erro add-key</i>) 	1
Index.php	<ul style="list-style-type: none"> • XSS (<i>hints-enabled</i>) • Injeção SQL (<i>UID cookie</i>) • Elevação de privilégios para admin (<i>valor do UID</i>) • Comentários no HTML • <i>HTTP Response Splitting</i> (divisão das respostas HTTP e seus parâmetros) 	5
Log-visit.php	<ul style="list-style-type: none"> • Injeção SQL (<i>referer e user-agent HTTP</i>) • XSS (<i>referer e user-agent HTTP</i>) 	4
Login.php	<ul style="list-style-type: none"> • Burlar autenticação por injeção (<i>username,password</i>) • Injeção SQL (<i>username, password</i>) • XSS (<i>username</i>) • Burlar validação do <i>JavaScript</i> 	6
Password-generator.php	<ul style="list-style-type: none"> • Injeção de <i>JavaScript</i> 	1
Pen-test-tool-lookup.php	<ul style="list-style-type: none"> • Injeção JSON (<i>JavaScript Object Notation</i>) 	1
Phpinfo.php	<ul style="list-style-type: none"> • Configuração visível do servidor PHP • Acesso ao caminho e plataforma da aplicação 	2
Process-commands.php	<ul style="list-style-type: none"> • Manipulação malfeita de cookies (<i>não o HTTP-only</i>) 	1
Process-login-attempt.php	<ul style="list-style-type: none"> • Burlar autenticação por injeção (<i>username, password</i>) • Injeção SQL (<i>username, password</i>) • XSS (<i>username</i>) • Burlar validação do <i>JavaScript</i> 	6
Redirectandlog.php	<ul style="list-style-type: none"> • Encaminhamento e Redirecionamento sem validação 	1
Register.php	<ul style="list-style-type: none"> • Injeção de SQL (<i>username, signature, password</i>) • XSS (<i>username, signature, password</i>) 	6
Robots.txt	<ul style="list-style-type: none"> • Contém diretórios que supostamente deveriam ser privados 	1
Config.inc	<ul style="list-style-type: none"> • Credenciais de acesso ao banco de dados não criptografadas 	1
Set-background-color.php	<ul style="list-style-type: none"> • XSS (<i>color</i>) • CSS Injetion (<i>color</i>) 	2
Show-log.php	<ul style="list-style-type: none"> • XSS (<i>hostname, IP, date, referer HTTP</i>) 	4
Site-footer-xss-discussion.php	<ul style="list-style-type: none"> • XSS (<i>user-agent HTTP</i>) 	1
Source-viewer.php	<ul style="list-style-type: none"> • Carregamento arbitrário de arquivos do sistema operacional 	1
Text-file-viewer.php	<ul style="list-style-type: none"> • Carregamento arbitrário de páginas na Internet ou arquivos do sistema operacional 	1
User-info.php	<ul style="list-style-type: none"> • Injeção de SQL (<i>username, password</i>) 	4

	<ul style="list-style-type: none"> • XSS (username, password) 	
User-poll.php	<ul style="list-style-type: none"> • Poluição de parâmetros • Forçar GET ao invés de POST • XSS (choice) • CSRF 	4
View-someons-blog.php	<ul style="list-style-type: none"> • XSS (todos os campos) 	3

Fonte: Construído pelo autor

Algumas páginas do *Mutillidae* foram ignoradas durante o teste, pois não contêm nenhum erro ou as vulnerabilidades existentes não se encaixam em nenhuma categoria do documento OWASP Top Ten. São elas:

- Usage-instructions.php
- Set-up-database.php
- Rene-magritte.php
- Php-errors.php
- Notes.php
- Home.php
- Framer.html
- Framing.php
- Authorization-required.php

Após a realização das varreduras usando as ferramentas selecionadas, foram verificados e tabulados o total de vulnerabilidades encontradas, quantas foram corretamente identificadas e o número de falsos positivos gerados pela ferramenta.

4.2 Análise das falhas através das ferramentas selecionadas

Nesta seção do trabalho, são apresentados os resultados dos testes realizados no ambiente web do *Mutillidae* preparado para o cenário. Três categorias foram empregadas para comparação das ferramentas: número de vulnerabilidades detectadas, quantos falsos positivos ocorreram e o tempo total de varredura realizado.

Uma característica comum notada em todas as ferramentas é que o log dos resultados finais é normalmente muito extenso devido a uma característica comum a todas as ferramentas, que é a de reportar avisos (*warnings*) da possibilidade de uma falha ocorrer, além das vulnerabilidades em si. As falhas encontradas são comumente categorizadas em *high risk* (alto risco), *medium risk* (médio risco), *low risk* (baixo risco) e *Informational* (informativo). Normalmente, as aplicações encontram poucas vulnerabilidades e alto e médio risco, e a maior parte de baixo risco, além de dados informacionais (*informational*), como avisos sobre versões de interpretadores de *scripts* desatualizadas. A categoria *Informational* não se refere à vulnerabilidades, e sim às informações obtidas durante a varredura, podendo ser úteis à melhoria da segurança da aplicação. Como exemplo, temos uma entrada de log detectada pela ferramenta OWASP ZAP:

Informational (Warning): *X-Frame-Options header not set*

Description: *X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks*

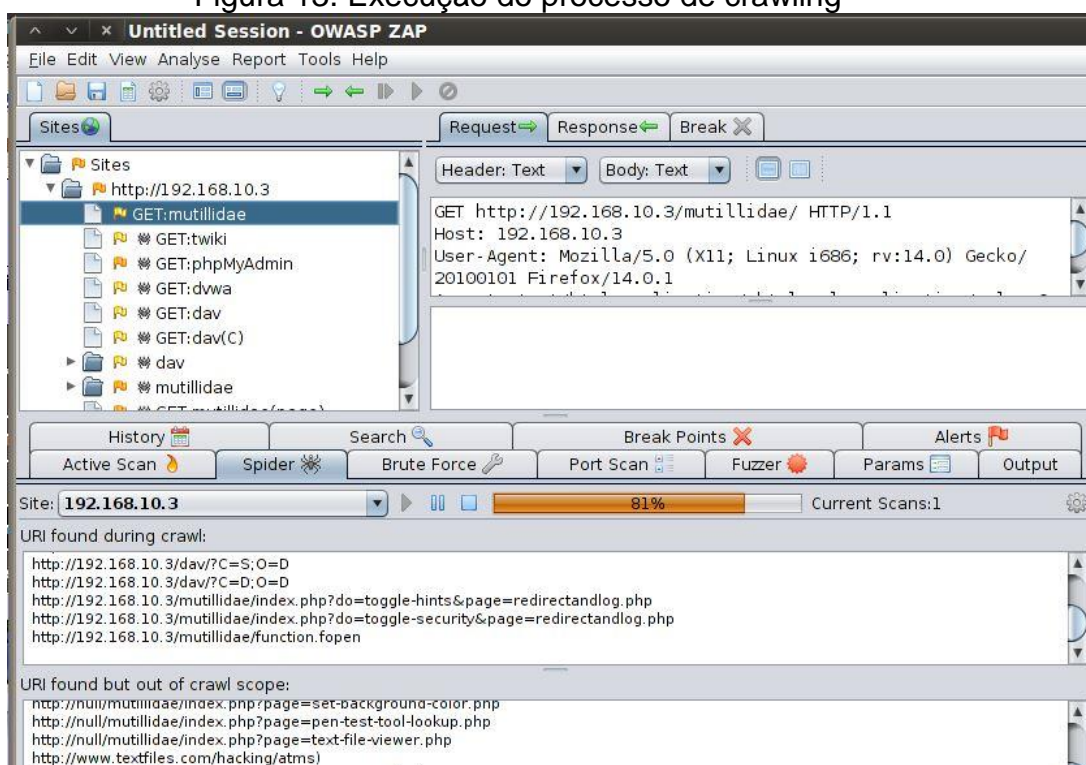
URL: *http://192.168.10.3/mutillidae/styles/global-styles.css*

Como as informações da categoria *informational* não são relacionadas aos riscos descritos no *OWASP Top Ten*, esses dados foram ignorados na análise final. As falhas de baixo risco que também não se encaixaram em nenhuma categoria do *Top Ten* também foram desconsideradas.

4.2.1 – Execução do teste da ferramenta OWASP ZAP

Inicialmente, foi utilizado o módulo Web Crawler para fazer o mapeamento de todas as páginas existentes no ambiente testado através da página principal (*index.php*). Durante essa etapa, as falhas ainda não foram analisadas, pois este primeiro passo é feito apenas para identificação dos recursos. A Figura 14 ilustra a execução do processo de crawling do ambiente do *Mutillidae*.

Figura 13. Execução do processo de crawling



Fonte: *Print Screen* obtido pelo autor

Após a finalização do *crawler*, que levou 8 minutos e 22 segundos para ser completado, verificou-se que todas as páginas do site foram corretamente identificadas. O próximo passo foi a execução do módulo Active Scan, que fez a varredura à procura de vulnerabilidades em todas as páginas do ambiente. O processo de análise das falhas levou 12 minutos e 10 segundos para ser finalizado e retornou um total de 119 falhas identificadas. O teste foi repetido por mais duas vezes para verificar se os resultados seriam diferentes, o que não ocorreu. O processo todo, incluindo as etapas de *crawling* e varredura, durou 20 minutos e 32 segundos.

A ferramenta se saiu muito bem na análise das vulnerabilidades de todas as categorias, com exceção da categoria A10 (*Unvalidated Redirects and Forwards*), em que o software não conseguiu encontrar nenhuma vulnerabilidade. Nas categorias A2 (*Broken Authentication*), A4 (*Insecure Direct Object Reference*), A5 (*Security Misconfiguration*), A8 (*Cross Site Request Forgery*) e A9 (*Known Vulnerable Components*), todas as falhas existentes foram corretamente identificadas.

Outro problema detectado ao final da varredura foi a enorme quantidade de falsos positivos reportados pelo *OWASP ZAP*, especialmente nas categorias de vulnerabilidades que se referem à Injeção de código (A1), manipulação de parâmetros de autenticação (A2) e *Cross Site Request Forgeries* (A8). No final da varredura, 119 falhas foram identificadas pelo software, mas após análise manual verificou-se que 72 delas eram falsos positivos. Portanto, das 83 falhas existentes no ambiente, apenas 47 foram corretamente detectadas. A Tabela 2 apresenta os resultados obtidos ao final do teste de varredura.

Tabela 2. Informações obtidas após a varredura com o OWASP ZAP

ID	TIPOS DE FALHAS	FALHAS EXISTENTES	FALHAS DETECTADAS	FALSOS POSITIVOS	DETECTADAS CORRETAMENTE
A1	Injeção de código	24	43	25	18 75,0%
A2	Quebra de Autenticação e Gerenciamento de Sessão	7	22	15	7 100,0%
A3	Cross-Site Scripting	33	8	2	6 18,2%
A4	Referência Insegura e Direta a Objetos	5	6	1	5 100,0%
A5	Configuração Incorreta de Segurança	4	4	0	4 100,0%
A6	Exposição de Dados Sensíveis	2	1	0	1 50,0%
A7	Falta de Função para Controle do Nível de Acesso	3	2	0	2 66,7%
A8	Cross-Site Request Forgery (CSRF)	2	31	29	2 100,0%
A9	Utilização de Componentes Vulneráveis Conhecidos	2	2	0	2 100,0%
A10	Redirecionamentos e Encaminhamentos Inválidos	1	0	0	0 0,0%

Fonte: Elaborado pelo autor com base nos resultados experimentais obtidos

4.2.2 – Análise e teste da ferramenta SQLMAP

Como o SQLMAP é uma solução específica para identificação e exploração de vulnerabilidade de injeção de SQL, ele só identifica falhas da categoria A1. Não há o recurso de *crawl* para identificação automática das páginas que constituem o site do ambiente testado, portanto, o teste teve de ser feito manualmente em cada uma das páginas, o que demandou muito mais tempo do que nos outros scanners. Um invasor poderia contornar esse problema utilizando outro software de varredura com recursos de *crawling* ou construir um *script* para identificação das páginas. Em ambos os casos, os resultados teriam de ser exportados para uso pelo SQLMAP.

O processo de testes manual para todas as páginas foi o mesmo: verificar se a página é vulnerável a SQL Injection, tentar identificar primeiramente o nome do banco de dados, depois as tabelas que o compõem e, por último, extrair os dados das colunas existentes em uma das tabelas. O tempo despendido para a execução do teste manual em todas as páginas foi de aproximadamente 1 hora e 30 minutos. Todas as funções foram executadas no modo *shell*, com a utilização de comandos específicos para cada uma das etapas. Para exemplificar, será ilustrado o processo de execução passo a passo da análise da página *user-info.php* e alguns dos resultados obtidos.

O primeiro passo foi a identificação da falha por meio da análise das entradas de formulário da página *user-info.php*. O argumento `-dbs` permite listar todos os bancos de dados existentes no SGBD consultado pela exploração da vulnerabilidade. O comando foi:

```
./sqlmap.py -u http://192.168.10.3/mutillidae/index.php?page=user-info.php -forms -batch -dbs
```

A execução do comando retornou o nome de todos os bancos de dados disponíveis no servidor SGBD, conforme apresentado na tela da Figura 14.

Figura 14. Tela de retorno

```

[00:10:53] [INFO] do you want to exploit this SQL injection? [Y/n] Y
[00:10:53] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
back-end DBMS: MySQL 5.0
[00:10:53] [INFO] fetching database names
available databases [7]:
[*] dvwa
[*] information_schema
[*] metasploit
[*] mysql
[*] owasp10
[*] tikiwiki
[*] tikiwiki195

[00:10:53] [INFO] you can find results of scanning in multiple targets
de the CSV file '/pentest/database/sqlmap/output/results-07112015_1210a
[*] shutting down at 00:10:53

```

Fonte: *Print screen* obtido pelo autor

O segundo passo foi a identificação das tabelas do banco de dados escolhido (owasp10) da análise das entradas de formulário da página user-info.php. O argumento `–tables` permitiu a visualização das tabelas existentes no SGBD. O comando executado foi:

```

./sqlmap.py -u http://192.168.10.3/mutillidae/index.php?page=user-info.php –forms –
batch -D owasp10 –tables

```

A execução do comando retornou a relação de tabelas existentes no banco de dados owasp10, conforme o resultado apresentado na tela da Figura 15.

Figura 15. Tela de retorno

```

[00:14:59] [INFO] fetching tables for database: 'owasp10'
Database: owasp10
[6 tables]
+-----+
| accounts      |
| blogs_table   |
| captured_data |
| credit_cards  |
| hitlog        |
| pen_test_tools|
+-----+

[00:14:59] [INFO] you can find results of scanning in multiple
de the CSV file '/pentest/database/sqlmap/output/results-07112
←
[*] shutting down at 00:14:59

```

Fonte: *Print screen* obtido pelo autor

O terceiro passo foi a identificação do nome das colunas existentes na tabela accounts. O argumento `--columns` permitiu a exibição das colunas existentes nesta tabela. O comando executado foi:

```

./sqlmap.py -u http://192.168.10.3/mutillidae/index.php?page=user-info.php --forms --
batch -D owasp10 -T accounts --columns

```

A execução do comando retornou as colunas existentes na tabela account, dentro do banco de dados owasp10, conforme o resultado apresentado na tela da Figura 16.

Figura 16. Tela de Retorno

```
Database: owasp10
Table: accounts
[5 columns]
+-----+-----+
| Column | Type |
+-----+-----+
| cid    | int(11) |
| is_admin | varchar(5) |
| mysignature | text |
| password | text |
| username | text |
+-----+-----+

[00:16:04] [INFO] you can find results of scanning in multiple
de the CSV file '/pentest/database/sqlmap/output/results-07112
[*] shutting down at 00:16:04
```

Fonte: Print screen obtido pelo autor

O quarto passo foi a extração dos dados das informações dos campos das colunas *username* e *password*, permitindo verificar a possibilidade de obter as informações de autenticação de usuários. A opção `-dump` foi usada para extrair os dados. O comando executado foi:

```
./sqlmap.py -u http://192.168.10.3/mutillidae/index.php?page=user-info.php -forms -
batch -D owasp10 -T accounts -C username, password -dump
```

A execução do comando retornou o nome e a senha dos usuários na tabela *account*, através da listagem dos campos *username* e *password*, conforme resultado apresentado na tela da Figura 17.

Figura 17. Tela de Retorno

```
[00:18:32] [INFO] analyzing table dump for possible password hashes
Database: owasp10
Table: accounts
[17 entries]
+-----+-----+
| username | password |
+-----+-----+
| admin    | adminpass |
| adrian   | somepassword |
| john     | monkey    |
| jeremy   | password  |
| bryce    | password  |
| samurai  | samurai   |
| jim      | password  |
| bobby    | password  |
| simba    | password  |
| dreveil  | password  |
| scotty   | password  |
| cal      | password  |
| john     | password  |
| kevin    | 42        |
| dave     | set       |
| ed       | pentest   |
| marcos   | 123456    |
+-----+-----+
```

Fonte: *Print screen* obtido pelo autor

Uma vantagem desta ferramenta identificada pelos testes é a ausência de falsos positivos. Não há como extrair informações do banco de dados se a falha realmente não existir, o que torna o SQLMAP uma excelente opção para testes de *SQL Injection* em comparação com as outras ferramentas. Entretanto, como o foco do programa é apenas na *Structured Query Language*, três falhas existentes no *Mutillidae* não foram identificadas, referentes a tipos diferentes de injeção. Foram elas: Injeção de HTML, Injeção de comandos do SO e Injeção de Javascript. Nenhuma vulnerabilidade das outras categorias de 2 a 10 foram identificadas, devido ao foco da ferramenta ser apenas na categoria A1.

Tabela 3. Informações obtidas após a varredura com o SQLMAP

ID	TIPOS DE FALHAS	FALHAS EXISTENTES	FALHAS DETECTADAS	FALSOS POSITIVOS	DETECTADAS CORRETAMENTE
A1	Injeção de código	24	21	0	21 87,5%
A2	Quebra de Autenticação e Gerenciamento de Sessão	7	0	0	0 0,0%
A3	Cross-Site Scripting	33	0	0	0 0,0%
A4	Referência Insegura e Direta a Objetos	5	0	0	0 0,0%
A5	Configuração Incorreta de Segurança	4	0	0	0 0,0%
A6	Exposição de Dados Sensíveis	2	0	0	0 0,0%
A7	Falta de Função para Controle do Nível de Acesso	3	0	0	0 0,0%
A8	Cross-Site Request Forgery (CSRF)	2	0	0	0 0,0%
A9	Utilização de Componentes Vulneráveis Conhecidos	2	0	0	0 0,0%
A10	Redirecionamentos e Encaminhamentos Inválidos	1	0	0	0 0,0%

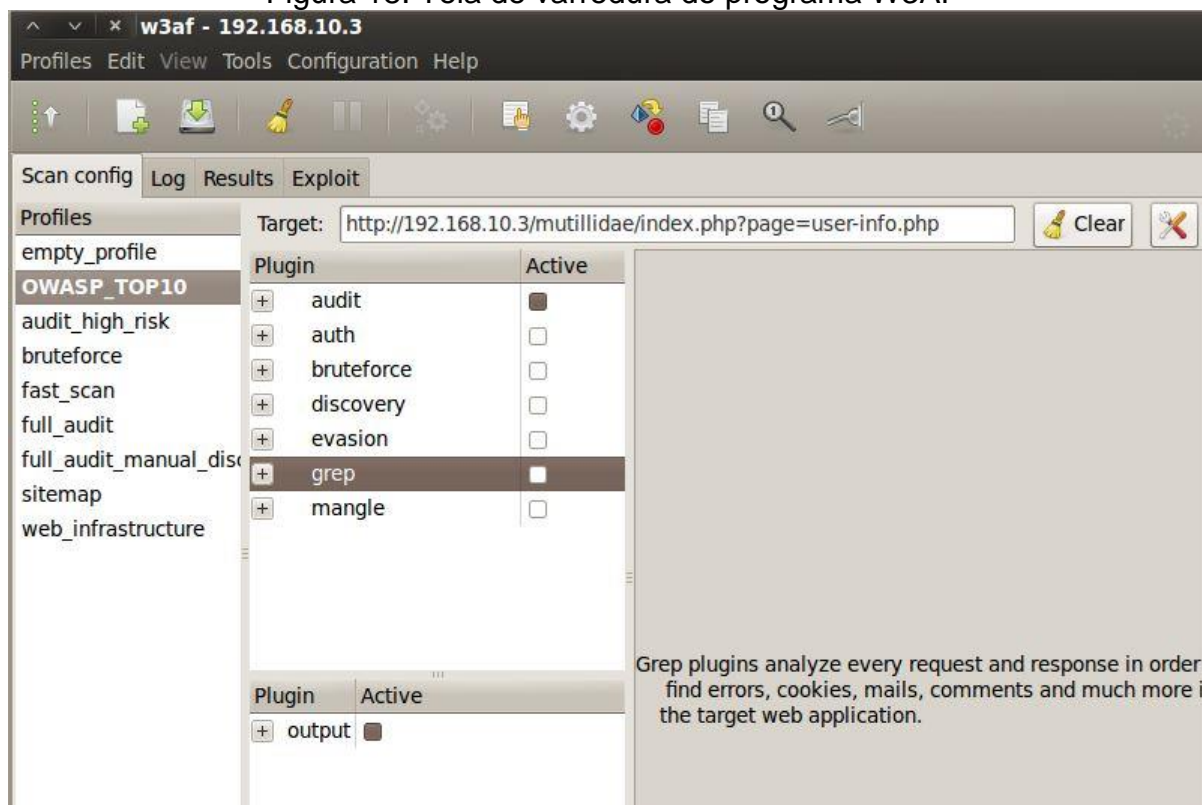
Fonte: Elaborado pelo autor com base nos resultados experimentais obtidos

4.2.3 – Análise e teste da ferramenta W3AF

Para o teste do software W3AF, foi escolhido o perfil *OWASP_TOP10*, que faz a varredura somente das vulnerabilidades categorizadas de acordo com a documentação do *Top Ten*. Primeiramente, foi iniciado o processo de *crawling* para identificação das páginas web que constituem o site. Notou-se que a ferramenta se

comportou de forma instável e chegou a travar algumas vezes durante a etapa de *crawl*, sendo completado apenas na terceira tentativa de execução. A Figura 18 mostra a tela de varredura do W3AF.

Figura 18. Tela de varredura do programa W3AF



Fonte: *Print screen* obtido pelo autor

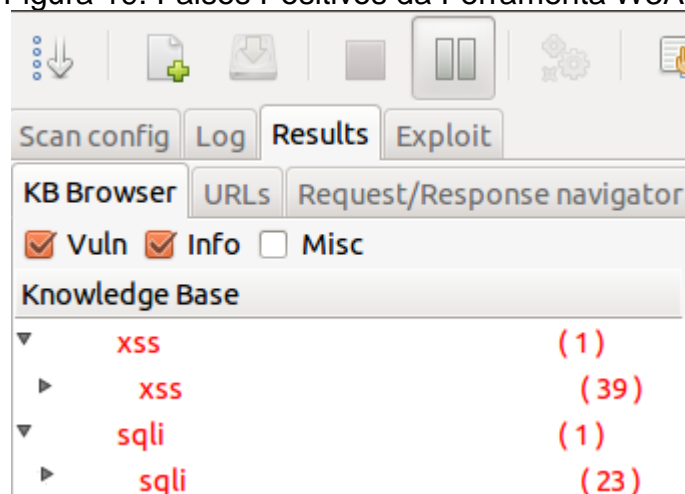
Algumas páginas também não foram identificadas corretamente durante esse processo. Elas tiveram que ser testadas manualmente após a execução do teste principal. Os recursos não identificados foram:

- Html5-storage.php
- Credits.php
- Text-file-viewer.php
- Config.inc
- Robots.txt

O processo de *crawling* levou 14 minutos e 10 segundos para ser completado, e a etapa de identificação das vulnerabilidades durou mais 16 minutos e 39 segundos. No total, foram despendidos aproximadamente 30 minutos com todo o processo. A imagem a seguir mostra algumas das vulnerabilidades identificadas durante o processo de análise de falhas realizado pelo W3AF. Após o problema inicial com o reconhecimento das páginas, a identificação dos problemas ocorreu sem dificuldades.

A ferramenta gerou alguns falsos positivos, especialmente nas categorias de risco A1, A3 e A8, mais especificamente tratando-se de variações de *SQL Injection*, *XSS* e *CSRF*. Parte dos resultados encontrados na varredura, incluindo os falsos positivos, pode ser vista na Figura 19.

Figura 19. Falsos Positivos da Ferramenta W3AF



Fonte: Print screen obtido pelo autor

Mesmo com os falsos positivos, o W3AF foi a ferramenta que conseguiu identificar um maior número de vulnerabilidades de *Cross Site Scripting* (categoria A3), tendo também identificado todas as falhas existentes nas categorias A6 (*Sensitive Data Exposure*), A8 (*Cross Site Request Forgery*), A9 (*Known Vulnerable Components*) e A10 (*Unvalidated Redirects and Forwards*). Das 83 falhas existentes no ambiente, 61 foram corretamente identificadas pelo W3AF. Apenas 28 falsos positivos foram detectados, valor que pode ser considerado razoável em relação ao total de problemas. Todos os resultados obtidos após a análise das vulnerabilidades com a ferramenta podem ser verificados na Tabela 4.

Tabela 4. Informações obtidas após a varredura com o W3AF

ID	NOME DAS FALHAS	FALHAS EXISTENTES	FALHAS DETECTADAS	FALSOS POSITIVOS	DETECTADAS CORRETAMENTE
A1	Injeção de código	24	23	8	15 62,5%
A2	Quebra de Autenticação e Gerenciamento de Sessão	7	3	0	3 42,9%
A3	Cross-Site Scripting	33	39	9	30 90,9%
A4	Referência Insegura e Direta a Objetos	5	5	1	4 80,0%
A5	Configuração Incorreta de Segurança	4	3	0	3 75,0%
A6	Exposição de Dados Sensíveis	2	1	0	1 50,0%
A7	Falta de Função para Controle do Nível de Acesso	3	0	0	0 0,0%
A8	Cross-Site Request Forgery (CSRF)	2	10	8	2 100,0%
A9	Utilização de Componentes Vulneráveis Conhecidos	2	4	2	2 100,0%
A10	Redirecionamentos e Encaminhamentos Inválidos	1	1	0	1 100,0%

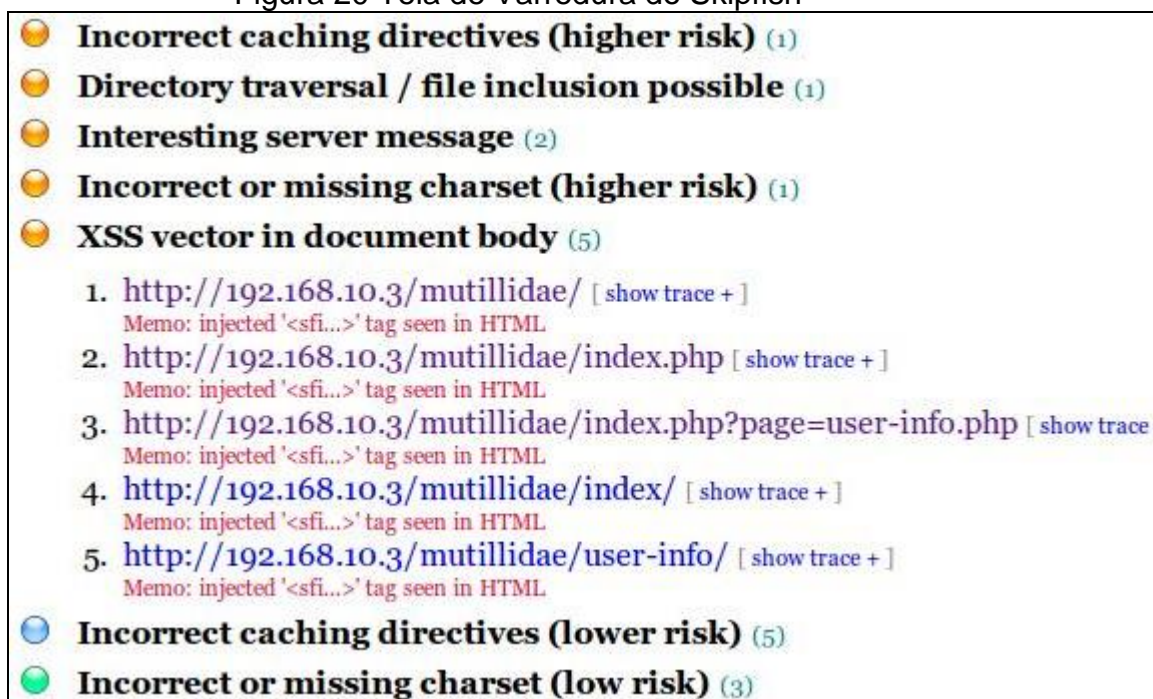
Fonte: Elaborado pelo autor com base nos resultados experimentais obtidos

4.2.4 – Execução do teste da ferramenta Skipfish

Assim como o *W3AF* e o *OWASP ZAP*, o *Skipfish* tem a capacidade de *crawling* do site, permitindo a identificação de todas as páginas existentes no cenário criado. Durante o processo de *crawl*, foram identificados todos os links do *Mutillidae*, com exceção das páginas *php-error.php* e *framing.php*. Como não existem vulnerabilidades nesses dois arquivos, eles não foram inseridos manualmente na

varredura posteriormente. Ao contrário das ferramentas testadas anteriormente, o Skipfish não separa suas etapas de crawling e varredura de falhas. As vulnerabilidades são identificadas simultaneamente à identificação das páginas. O tempo total de varredura foi de 25 minutos e 18 segundos. A Figura 20 mostra uma das seções do resultado gerado pelo Skipfish, exibindo algumas das vulnerabilidades identificadas no ambiente do *Mutillidae*

Figura 20 Tela de Varredura do Skipfish



Fonte: *Print screen* obtido pelo autor

O *SKIPFISH* não detectou nenhuma das falhas relacionadas à categoria A8, que trata de problemas de *Cross Site Request Forgery*, e teve um desempenho ruim nas categorias relacionadas a *Cross Site Scripting* (A3) e de referência insegura a objetos (A4), detectando um baixo percentual das vulnerabilidades existentes. A única categoria que o programa detectou em todas as vulnerabilidades foi a A9 (*Known Vulnerable Components*).

O software gerou um grande número de alertas (*warnings*), indicando problemas de configuração que fogem ao escopo das dez categorias da *OWASP Top Ten*. Após um filtro dos resultados, verifica-se que apenas uma porcentagem razoavelmente baixa de vulnerabilidades foi identificada, apenas 35 falhas (X%). Depois de uma verificação manual nos dados, foram identificados 8 falsos

positivos, divididos entre as categorias A1 (*Injection*), A4 (*Insecure Direct Object Reference*), A7 (*Missing Function Level Access Control*) e A9 (*Known Vulnerable Components*). Portanto, de 83 falhas existentes no ambiente, apenas 26 foram corretamente identificadas.

O conjunto dos resultados obtidos após o teste com a ferramenta *SKIPFISH* pode ser visualizado na Tabela 5.

Tabela 5 Informações obtidas após a varredura com Skipfish

ID	NOME DAS FALHAS	FALHAS EXISTENTES	FALHAS DETECTADAS	FALSOS POSITIVOS	DETECTADAS CORRETAMENTE
A1	Injeção de código	24	10	2	8 33,3%
A2	Quebra de Autenticação e Gerenciamento de Sessão	7	2	0	2 28,6%
A3	Cross-Site Scripting	33	5	0	5 15,2%
A4	Referência Insegura e Direta a Objetos	5	6	4	1 20,0%
A5	Configuração Incorreta de Segurança	4	4	0	4 100,0%
A6	Exposição de Dados Sensíveis	2	2	0	2 100,0%
A7	Falta de Função para Controle do Nível de Acesso	3	3	1	2 66,7%
A8	Cross-Site Request Forgery (CSRF)	2	0	0	0 0,0%
A9	Utilização de Componentes Vulneráveis Conhecidos	2	2	1	1 50,0%
A10	Redirecionamentos e Encaminhamentos Inválidos	1	1	0	1 100,0%

Fonte: Elaborado pelo autor com base nos resultados experimentais obtidos

4.2.5 Análise e teste da ferramenta Nikto

O Nikto assim como o Skipfish realizam o processo de *crawling* em simultâneo com a identificação das vulnerabilidades. Entretanto, a página *user-info.php* não foi identificada durante as três tentativas de execução dos testes, tendo que ser testada manualmente a posteriori. Verificou-se não ser um problema de permissão de acesso, já que outros programas testados encontraram o link da página sem nenhum problema. O tempo despendido pelo processo de identificação das páginas e reconhecimento de vulnerabilidades resultou em um total de 18 minutos e 9 segundos: 17 minutos e 2 segundos para o ambiente todo e 1 minuto e 7 segundos para a análise da página *user-info.php*. A seguir, são analisados os resultados obtidos pela varredura. A Figura 21 mostra a execução do Nikto na página não identificada

Figura 21. Ferramenta Nikto sendo executada durante os testes

```
root@bt:/pentest/web/nikto# ./nikto.pl -host http://192.168.10.3/mutillidae/index.php?page=user-info.php
- Nikto v2.1.5
-----
+ Target IP:      192.168.10.3
+ Target Hostname: 192.168.10.3
+ Target Port:    80
+ Start Time:     2015-07-11 00:51:59 (GMT-3)
-----
+ Server: Apache/2.2.8 (Ubuntu) DAV/2
+ Retrieved x-powered-by header: PHP/5.2.4-2ubuntu5.10
```

Fonte: *Print screen* obtido pelo autor

O NIKTO não conseguiu identificar nenhuma vulnerabilidade nas categorias A6 (*Sensitive Data Exposure*) e A10 (*Unvalidated Redirects and Forwards*). A ferramenta foi capaz de identificar um total de 82 vulnerabilidades, entretanto, 32 delas foram comprovadamente reportadas como falsos positivos após o teste manual de verificação. Portanto, das 83 falhas existentes no sistema, o NIKTO reportou 50 problemas corretamente, considerando apenas o escopo das categorias do *Top Ten*.

Um ponto considerado negativo da ferramenta foi o excesso de informações desnecessárias no *log*, gerado após a varredura. Após detectar a versão do servidor

web e do *php* rodando no *Mutillidae*, o programa gerou uma quantidade enorme de sugestões de melhorias de segurança com base nesses serviços. Portanto, removendo todos esses avisos e vulnerabilidades de baixo nível que não se encaixavam nas categorias *OWASP*, pouco foi aproveitado dos resultados gerados.

As informações encontradas após o processo de crawling e análise de vulnerabilidades com o *NIKTO* podem ser vistas na Tabela 6.

Tabela 6. Informações obtidas após a varredura com Nikto

ID	NOME DAS FALHAS	FALHAS EXISTENTES	FALHAS DETECTADAS	FALSOS POSITIVOS	DETECTADAS CORRETAMENTE
A1	Injeção de código	24	16	4	12 50,0%
A2	Quebra de Autenticação e Gerenciamento de Sessão	7	1	0	1 14,3%
A3	Cross-Site Scripting	33	46	19	27 81,8%
A4	Referência Insegura e Direta a Objetos	5	7	3	4 80,0%
A5	Configuração Incorreta de Segurança	4	3	1	2 50,0%
A6	Exposição de Dados Sensíveis	2	2	2	0 0,0%
A7	Falta de Função para Controle do Nível de Acesso	3	3	2	1 33,3%
A8	Cross-Site Request Forgery (CSRF)	2	1	0	1 50,0%
A9	Utilização de Componentes Vulneráveis Conhecidos	2	3	1	2 100,0%
A10	Redirecionamentos e Encaminhamentos Inválidos	1	0	0	0 0,0%

Fonte: Elaborado pelo autor com base nos resultados experimentais obtidos

4.3 Tabulação dos resultados finais

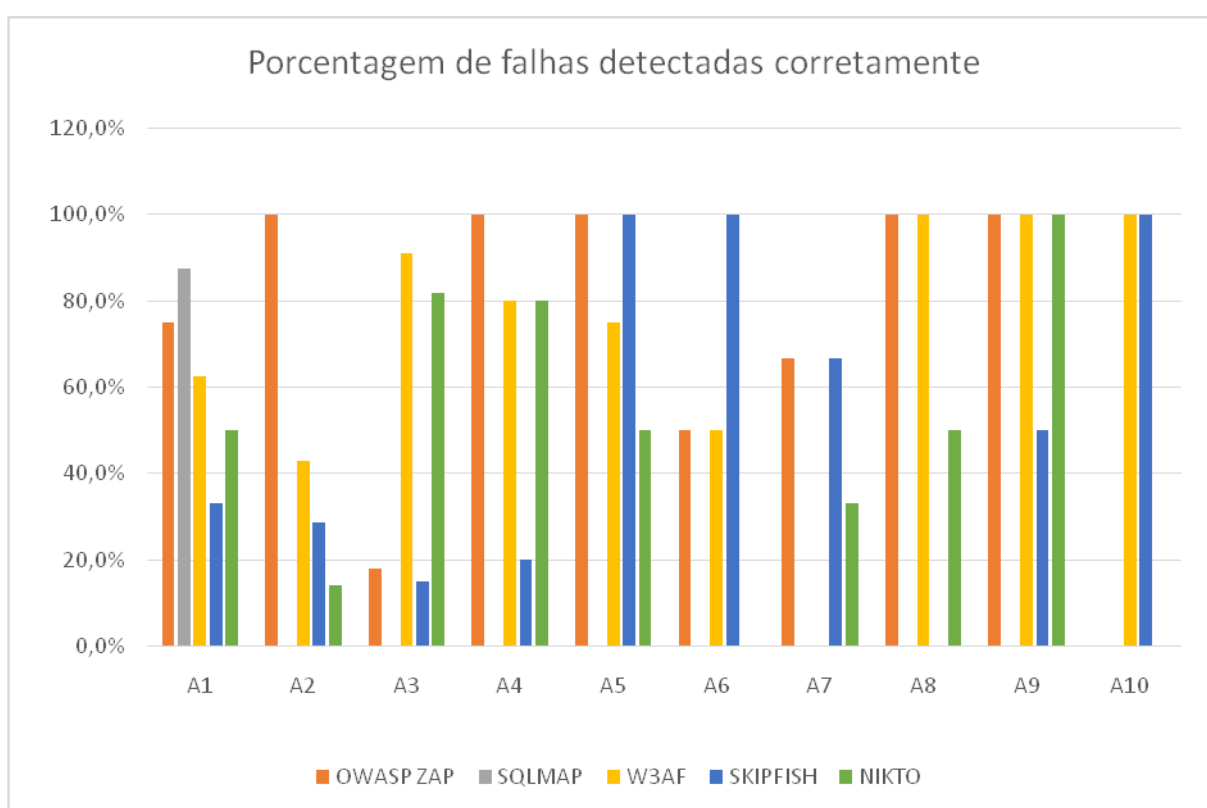
A Tabela 7 exibe o total de falhas existentes e o número de falhas corretamente detectadas, excluindo-se os falsos positivos. A Figura 22 apresenta as falhas identificadas pelos softwares testados em cada uma das categorias verificadas.

Tabela 7. Falhas corretamente detectadas pelos softwares

ID	NOME DAS FALHAS	FALHAS EXISTENTES	OWASP ZAP	SQLMAP	W3AF	SKIPFISH	NIKTO
A1	Injeção de código	24	18	21	15	8	12
A2	Quebra de Autenticação e Gerenciamento de Sessão	7	7	0	3	2	1
A3	Cross-Site Scripting	33	6	0	30	5	27
A4	Referência Insegura e Direta a Objetos	5	5	0	4	1	4
A5	Configuração Incorreta de Segurança	4	4	0	3	4	2
A6	Exposição de Dados Sensíveis	2	1	0	1	2	0
A7	Falta de Função para Controle do Nível de Acesso	3	2	0	0	2	1
A8	Cross-Site Request Forgery (CSRF)	2	2	0	2	0	1
A9	Utilização de Componentes Vulneráveis Conhecidos	2	2	0	2	2	2
A10	Redirecionamentos e Encaminhamentos Inválidos	1	0	0	1	1	0

Fonte: Elaborado pelo autor com base nos resultados experimentais obtidos

Figura 22. Gráfico de falhas corretamente identificadas



Fonte: Elaborado pelo autor com base nos resultados experimentais obtidos

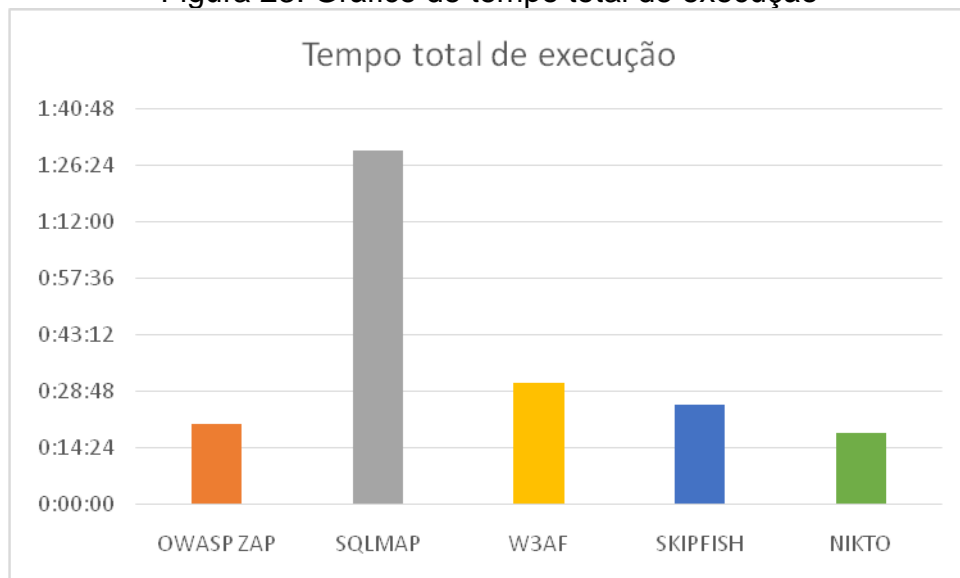
A Tabela 8 e Figura 23 mostram o tempo de varredura total despendido por cada ferramenta para obter os resultados finais apresentados nesse trabalho. Como no caso do SQLMAP, o teste foi totalmente manual, foi a ferramenta que consumiu mais tempo para a realização de todo o processo.

Tabela 8. Tempo de Execução das Ferramentas

Ferramenta	Tempo total de execução
OWASP ZAP	20 minutos e 32 segundos
SQLMAP	1 hora e 30 minutos
W3AF	30 minutos e 49 segundos
SKIPFISH	25 minutos e 18 segundos
NIKTO	18 minutos e 9 segundos

Fonte: Elaborado pelo autor com base nos resultados experimentais obtidos

Figura 23. Gráfico de tempo total de execução



Fonte: Elaborado pelo autor a partir dos resultados experimentais obtidos

A Figura 24 mostra um gráfico construído com os dados da Tabela 9, que contém três colunas que representam os seguintes itens obtidos pela análise das cinco ferramentas:

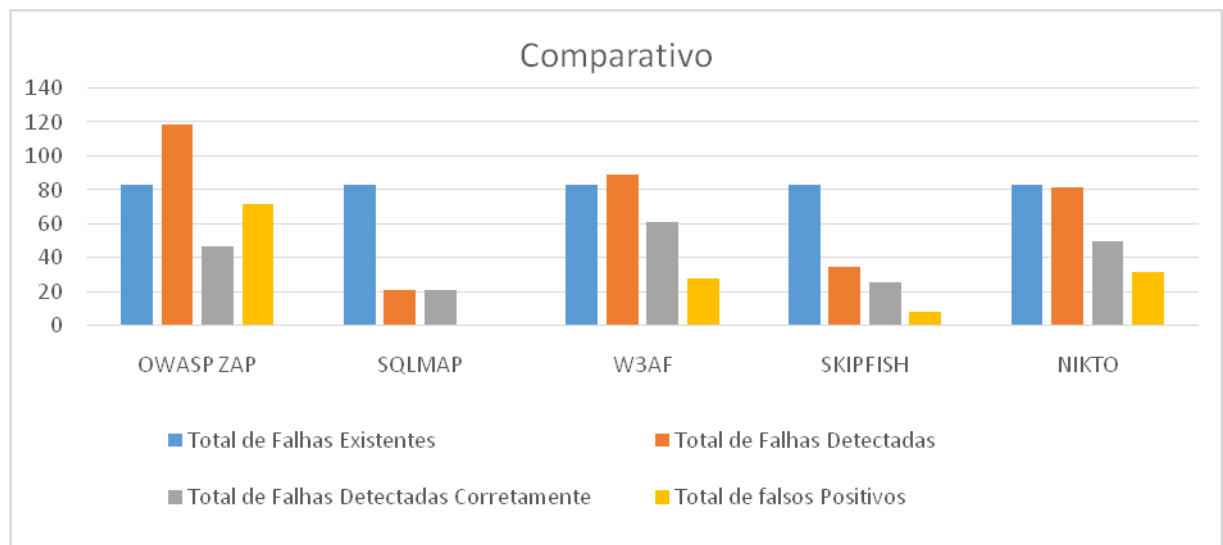
- Total de falhas existentes no *framework Mutillidae*;
- Número de falsos positivos gerados pelas ferramentas;
- Tempo de varredura gasto pela ferramenta.

Tabela 9. Total de falhas existentes x detectadas x falso positivo

Ferramenta	Total de Falhas Existentes	Total de Falhas Detectadas	Total de Falhas Detectadas Corretamente	Total de falsos Positivos
W3AF	83	89	61 (73%)	28
NIKTO	83	82	50 (60%)	32
OWASP ZAP	83	119	47 (56%)	72
SKIPFISH	83	35	26 (31%)	8
SQLMAP	83	21	21 (25%)	0

Fonte: Elaborado pelo autor com base nos resultados experimentais obtidos

Figura 24. Comparativo: total de falhas existentes, detectadas e falsos positivos



Fonte: Elaborado pelo autor com base nos resultados experimentais obtidos

5. CONCLUSÃO

Entre as contribuições dessa pesquisa, destaca-se a identificação dos pontos fortes das ferramentas de código aberto, analisadas pela realização de um teste com base nas dez categorias de risco mais frequentes em aplicações web, de acordo com o documento Top Ten da OWASP.

Após a compilação e interpretação dos resultados referentes às dez categorias de falhas da web que foram contempladas pelo trabalho, identificou-se o padrão de comportamento dos softwares analisados para a detecção de cada um dos diferentes tipos de riscos considerados no teste. A seguir, são apresentados a relação das categorias analisadas e o nome das soluções que identificaram mais falhas em cada uma delas:

A1 (*Injection*): SQLMAP

A2 (*Broken Authentication*): OWASP ZAP

A3 (*Cross Site Scripting*): W3AF

A4 (*Insecure Direct Object Reference*): OWASP ZAP

A5 (*Security Misconfiguration*): OWASP ZAP

A6 (*Sensitive Data Exposure*): SKIPFISH

A7 (*Missing Function Level Access Control*): SKIPFISH e OWASP ZAP

A8 (*Cross Site Request Forgery*): OWASP ZAP e W3AF

A9 (*Known Vulnerable Components*): ZAP, W3AF, SKIPFISH e NIKTO

A10 (*Unvalidated Redirects and Forwards*): W3AF e SKIPFISH

De acordo com as informações acima, pode-se verificar que a única categoria em que todos os softwares testados (com exceção do SQLMAP) foram capazes de identificar todas as vulnerabilidades existentes foi a A9, que trata do uso de componentes vulneráveis conhecidos no ambiente web. Houve empates também em outras categorias: na A7, que trata da falta de funções adequadas de controle de acesso, o SKIPFISH e o OWASP ZAP detectaram um maior número de vulnerabilidades. Na categoria A8, relacionada a CSRF, o W3AF e o ZAP também

empataram. Por último, na categoria A10, que trata de redirecionamentos não validados, o W3AF e o SKIPFISH foram aqueles que tiveram a melhor performance.

A solução SQLMAP não foi capaz de identificar vulnerabilidades em categorias diferentes da A1 (*Injection*), mas foi a que obteve com folga o melhor resultado em falhas de Injeção de SQL, com 87% das vulnerabilidades recomendadas, já que, pelo fato de sua utilização ser manual, a ferramenta não gera falsos positivos. Com base nos resultados apresentados, não é recomendado ter este programa como um recurso principal de análise de vulnerabilidade, já que o software não é capaz de detectar falhas em categorias diferentes da A1. Entretanto, é uma excelente solução para ser utilizada em conjunto com as outras ferramentas testadas como um complemento para verificação de falsos positivos de *SQL Injection*. Como o critério desta pesquisa se baseia no número de vulnerabilidades detectadas corretamente, esta ferramenta obteve a última posição no ranking.

A ferramenta *SKIPFISH* não teve um desempenho satisfatório, por ter identificado corretamente uma percentagem muito pequena das vulnerabilidades existentes (apenas 31% das vulnerabilidades totais existentes no ambiente), além de o tempo de execução ter sido consideravelmente alto quando comparado às outras soluções testadas. Apesar de ter sido a única ferramenta a identificar todas as vulnerabilidades da categoria A6 e ter empatado nos resultados das categorias A7, A9 e A10, foi considerada uma das piores soluções testadas por causa dos resultados no total de falhas obtidas, ficando na quarta posição.

O *OWASP ZAP* foi o software que teve a maior quantidade de falsos positivos. Portanto, mesmo se saindo melhor em mais categorias individuais do que as outras ferramentas (teve o melhor resultado nas categorias A2, A4 e A5 e empatou nas categorias A7 e A8), o total de 72 falsos positivos detectados foi considerado muito elevado quando comparado com as outras soluções testadas. Apenas 56% das vulnerabilidades foram detectadas corretamente (47 de um total de 83). Um ponto muito positivo do software foi o excelente recurso de *crawling*, que não deixou de detectar nenhum arquivo no ambiente, mas este não é um critério para o ranking das ferramentas, assim, o ZAP ficou com a terceira posição em números de vulnerabilidades existentes identificadas no ambiente simulado.

O software *NIKTO* teve uma performance razoavelmente boa, tendo sido capaz de detectar corretamente 50 vulnerabilidades de um total de 83 existentes. Apesar da grande quantidade de falsos positivos (32 no total) e da quantidade de

informações confusas geradas nos logs das vulnerabilidades detectadas - o que levou a um maior tempo de análise manual para filtrar vulnerabilidades de baixo risco ou aquelas que não pertencem às dez categorias da OWASP - o programa foi o segundo a detectar o maior número real de vulnerabilidades, posicionando-se, assim, na segunda posição do ranking das ferramentas testadas.

O W3AF, apesar de ter apresentado alguns problemas durante o processo de *crawling* para detecção dos arquivos do ambiente, foi a ferramenta que identificou o maior número de vulnerabilidades existentes. A quantidade de falsos positivos foi considerada baixa, quando comparada proporcionalmente com os resultados obtidos pelas outras soluções testadas. Foram verificados 28 falsos positivos, das 89 falhas reportadas pelo *log* gerado pelo programa. De todas as 83 falhas existentes, a ferramenta identificou corretamente 61 delas, um total de 73% de todos os problemas.

Portanto, o W3AF foi a solução que apresentou a melhor relação da taxa de falhas identificadas corretamente, falsos positivos e tempo total de análise, quando comparada com as outras soluções, além de ter uma interface amigável de fácil utilização. O único ponto negativo verificado foi a instabilidade da ferramenta durante o processo de *crawl*. Pelos motivos apresentados, este software foi considerado o primeiro da lista entre os programas testados.

A seguir, a apresentação do resultado final do ranking das ferramentas, obtido após a análise e verificação dos dados e dos pontos positivos e negativos de cada uma das opções testadas:

Primeiro: W3AF

Segundo: NIKTO

Terceiro: OWASP ZAP

Quarto: SKIPFISH

Quinto: SQLMAP

Diversos pontos não abordados por essa pesquisa podem ser desenvolvidos em trabalhos futuros, como o teste das ferramentas em cenários diferentes do proposto e a verificação da eficiência dos softwares testados frente à opções equivalentes que não sejam de software-livre, podendo ser ferramentas gratuitas

que não tenham código aberto ou mesmo soluções proprietárias. A inclusão no processo de análise de outras categorias de risco de vulnerabilidades web, além das contempladas pelo documento *Top Ten OWASP*, também merece ser investigada como um desdobramento desta pesquisa.

REFERÊNCIAS

ASSUNÇÃO, Marcos F. **Honeypots e Honeynets**. Florianópolis: Visual Books, 2009.

ASSUNÇÃO, Marcos F. **Segredos do Hacker Ético**. 5 ed. Florianópolis: Visual Books, 2014.

BASSO, Tania. **Uma abordagem para avaliação da eficácia de scanners de vulnerabilidades em aplicações web**. Dissertação e Apresentação de Mestrado, 2010.

CAMPOS, André. **Sistema de Segurança da Informação**. 3 ed. Florianópolis: Visual Books, 2014.

CARVALHO, Alan Henrique Pardo. Segurança de aplicações web e os dez anos do relatório OWASP Top Ten: o que mudou? **Fasci-Tech – Periódico Eletrônico da FATEC-São Caetano do Sul**, São Caetano do Sul, v.1, n. 8, Mar./Set. 2014, p. 6 a 18.

CERON, João; FAGUNDES, Leonardo; LUDWIG, Glauco; TAROUCO, Liane; BERTHOLDO, Leandro. **Vulnerabilidades em Aplicações Web: uma Análise Baseada nos Dados Coletados nos honeypots**. VIII Simposio Brasileiro de Segurança. 2008 Disponível em <http://sbseg2008.inf.ufrgs.br/proceedings/data/pdf/st06_02_resumo.pdf>.

COSTA, D.G. **Administração de Redes com scripts: Bash Script, Python e VBScript**. Rio de Janeiro: Brasport, 2010.

CURPHEY, Mark; ENDLER, David.; HAU, William; TAYLOR, Steve.; SMITH, Tim; RUSSELL, Alex.; MCKENNA, Gene; PARKE, Richard; MCLAUGHLIN, Kevin.; TRANTER, Nigel. **A guide to building secure web applications and web services**. The Open Web Application Security Project, v. 1, 2005.

DOUPÉ, Adam; CAVEDON, Ludovico; KRUEGEL, Christopher; VIGNA. **Enemy of the State: A State-Aware Black-Box Web Vulnerability Scanner**. In: USENIX Security Symposium. 2012. Disponível em <www.usenix.org/system/files/conference/usenixsecurity12/sec12-final225.pdf>. Acesso em 22 de dezembro de 2014.

ENGEBRETSON, Patrick. **Introdução ao Hacking e aos Testes de Invasão**. São Paulo: Novatec, 2013.

- GIL, Antonio Carlos. **Como elaborar projetos de pesquisa**. 5. ed. São Paulo: Atlas, 2010.
- HOLZ, T.; MARECHAL, S; RAYNAL, F. New threats and attacks on the World Wide Web. **IEEE Security & Privacy Magazine**, v. 4, n. 2, p. 45-50, março. 2006.
- ISECOM. **Open Source Security Testing Methodology Manual**. Disponível em < www.isecom.org/osstmm/>. Acesso em 07 de Janeiro de 2015.
- ISO 27001. **ABNT NBR ISO/IEC 27001:2013 – Tecnologia da Informação – Técnicas de segurança – Código de prática para a gestão da segurança da informação**. Associação Brasileira de Normas Técnicas – Rio de Janeiro: ABNT, 2013.
- ITU-T. **Recommendation X.509 - Open Systems Interconnection - The directory: authentication framework**. Disponível em <https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.509-199311-S!!PDF-E&type=items>. Acesso em 02 de junho de 2015.
- KUROSE, James; ROSS, Keith. **Redes de Computadores e a Internet: uma abordagem top-down**. 3. ed. São Paulo: Pearson, 2005.
- LAKATOS, Eva M.; MARCONI, Marina A. **Fundamentos de metodologia científica**. 5. ed. São Paulo: Atlas. 2003.
- LIMA, João P. **Administração de Redes Linux**. Goiânia: Terra, 2003.
- MACÊDO, Márcio A.; QUEIROZ, Ricardo G.; DAMASCENO, Julio C..jShield: **Uma Solução Open Source para Segurança de Aplicações Web**. Universidade Federal de Pernambuco. 2010.
- MARTINELO, Clériston Aparecido Gomes; BELLEZI, Marcos Augusto. **Análise de Vulnerabilidades com OpenVAS e Nessus**. T.I.S. São Carlos, v. 3, n. 1 , p. 34-44, 2014.
- MCCLURE, Stuart; SCAMBRAY, Joel; KURTZ, George. **Hackers Expostos**. 7 ed. São Paulo: Bookman. 2014.
- MEUCCI, M. **Owasp testing guide version 3.0**. OWASP Foundation. 2008.
- MUNIZ, Joseph; LAKHANI, Aamir. **Web Penetration Testing with Kali Linux**. Birmingham: Packt. 2013.
- OISSG. **Information Systems Security Assessment Framework**. Disponível em < www.oissg.org/issaf>. Acesso em 07 de Janeiro de 2015.
- OLIVEIRA, Túlio. **Testes de Segurança em Aplicações Web segundo a metodologia OWASP**. Projeto de TCC. Universidade Federal de Lavras. 2012.

Disponível em <<http://www.bcc.ufla.br/wp-content/uploads/2013/09/TESTES-DE-SEGURAN%C3%87A-EM-APLICA%C3%87%C3%95ES-WEB-SEGUNDO-A.pdf>>

OWASP. **Top 10 Web Vulnerabilities**. Disponível em <www.owasp.org/index.php/Top_10_2013>. Acesso em 15 de Dezembro de 2014.

PAULI, Josh. **Introdução ao WebHacking**. São Paulo: Novatec, 2014.

PESSOA, Márcio. **Segurança em PHP**. São Paulo: Novatec, 2007.

ROCHA, Douglas; KREUTZ, Diego; TURCHETTI, Rogério. **Uma Ferramenta Livre e Extensível Para Detecção de Vulnerabilidades em Sistemas Web**. Disponível em <article.sapub.org/pdf/10.5923.j.computer.20120001.08.pdf>. Acesso em 21 de dezembro de 2014.

SCHILDT, Hebert; SKRIEN, Dale. (2013). **Programação com Java: Uma Introdução Abrangente**. São Paulo: McGraw-Hill.

SECTOOLS. **Top 125 security tools**. Disponível em <www.sectools.org>. Acesso em 07 de Janeiro de 2015.

SICA, Carlos; REAL, Petter. **Programação Segura utilizando PHP**. São Paulo: Ciência Moderna, 2007.

SCHILDT, Hebert; SKRIEN, Dale. **Programação com Java: Uma introdução abrangente**. São Paulo: McGraw-Hill, 2013.

SILVA, Rodrigo; LIMA, Rommel; LEITE, Cicilia; SILVA, Romero. **Investigação de segurança no moodle**. Renote, v. 12, n. 2, 2014.

STALLINGS, William. **Criptografia e segurança em redes**. 4.ed. São Paulo: Person Prentice Hall, 2008.

STALLINGS, William. **Redes e sistemas de comunicação de dados: teorias e aplicações corporativas**. 5 ed. Rio de Janeiro: Elsevier, 2005.

TORRES, André Felipe F. **Os referenciais de segurança da informação e a melhoria contínua: um caso exploratório**. Dissertação e Apresentação de Mestrado. 2014.

VIEIRA, Marco; ANTUNES, Nuno; MADEIRA, Henrique. **Using web security scanners to detect vulnerabilities in web services**. Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on. IEEE, 2009. p. 566-571.

WEB APPLICATION SECURITY CONSORTIUM . Disponível em: <<http://www.webappsec.org/>>. Acesso em: 01 de junho de 2015.

WELLING, Luke; THOMSON, Laura. **Tutorial MySQL**. Rio de Janeiro: Ciência Moderna, 2004.

