UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS I
CENTRO DE CIÊNCIA E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE COMPUTAÇÃO

THIAGO HENRIQUE DOMINGOS DE SÁ

TESTES DE SEGURANÇA EM APLICAÇÕES WEB: UM ESTUDO COMPARATIVO ENTRE FERRAMENTAS *OPEN SOURCE*

CAMPINA GRANDE
2023

THIAGO HENRIQUE DOMINGOS DE SÁ

TESTES DE SEGURANÇA EM APLICAÇÕES WEB: UM ESTUDO COMPARATIVO ENTRE FERRAMENTAS *OPEN SOURCE*

Trabalho de Conclusão de Curso ao Programa de Graduação em 2023 da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de Bacharel em Ciência da Computação.

Área de concentração: Engenharia de Software.

Orientadora: Profa. Dra. Sabrina de Figueirêdo Souto

CAMPINA GRANDE
2023

S111t Sá, Thiago Henrique Domingos de.
Testes de segurança em aplicações web [manuscrito] : um estudo comparativo entre ferramentas *open source* / Thiago Henrique Domingos de Sa. - 2023.
61 p. : il. colorido.

Digitado.
Trabalho de Conclusão de Curso (Graduação em Computação) - Universidade Estadual da Paraíba, Centro de Ciências e Tecnologia, 2023.
"Orientação : Prof. Dr. Sabrina de Figueirêdo Souto, Departamento de Computação - CCT. "

1. Teste de software. 2. Teste de segurança. 3. Vulnerabilidade de software. 4. Ferramentas de scan. I. Título

21. ed. CDD 005.3

Elaborada por Talita M. A. Tavares - CRB - CRB 15/971 BC/UEPB

THIAGO HENRIQUE DOMINGOS DE SÁ

TESTES DE SEGURANÇA EM APLICAÇÕES WEB: UM ESTUDO COMPARATIVO ENTRE FERRAMENTAS OPEN SOURCE

Trabalho de Conclusão de Curso de Graduação em Ciência da Computação da Universidade Estadual da Paraíba, como requisito à obtenção do título de Bacharel em Ciência da Computação.

Aprovada em 03 de Julho de 2023.

Sabrina de F. Souto.

Profa. Dra. Sabrina de Figueiredo Souto (CCT/UEPB)
Orientador(a)

Fabio Luiz Leite Junior

Prof. Dr. Fábio Luiz Leite (CCT/UEPB)
Examinador(a)

Lucas Barbosa Oliveira

Me. Lucas Barbosa Oliveira (NUTES/UEPB)
Examinador(a)

À minha mãe, pela dedicação, companheirismo e amizade, DEDICO.

AGRADECIMENTOS

À professora Sabrina pelas leituras sugeridas ao longo dessa orientação e pela dedicação.

A minha mãe, pelo constante apoio e pela oportunidade me dada para me dedicar exclusivamente aos estudos.

Aos professores do Curso de Especialização da UEPB que contribuíram ao longo de trinta meses, por meio das disciplinas e debates, para o desenvolvimento desta pesquisa.

Aos funcionários da UEPB, pela presteza e atendimento quando nos foi necessário.

Aos colegas de classe pelos momentos de amizade e apoio.

RESUMO

O teste de segurança é uma peça importante no atual cenário do desenvolvimento e teste de software já que proporciona uma camada a mais de segurança e pode evitar problemas futuros como vazamento de dados e brechas de segurança que podem acarretar, por exemplo, prejuízo financeiro. Este trabalho possui caráter qualitativo e exploratório com o objetivo de comparar os resultados obtidos através da análise de aplicações web por ferramentas de escaneamento automático de código aberto. Estas ferramentas performam testes de caixa-preta na aplicação alvo e buscam identificar vulnerabilidades de segurança. As ferramentas foram escolhidas e filtradas levando em consideração quatro aspectos: Implementarem análises de vulnerabilidades listadas nos sites OWASP e CWE, foram atualizadas no ano de 2021 ou posteriormente, são licenciados como software de código aberto e podem ser usadas para testes de aplicações web. As análises foram feitas utilizando o escaneamento padrão de cada ferramenta para garantir uma consistência de configuração. As ferramentas OWASP e Arachni obtiveram os melhores resultados entre as testadas, tanto em questão de identificação de vulnerabilidades quanto em relação ao relatório gerado. Ambas conseguiram identificar a possível presença de falhas *SQL-Injectio*n e *Cross-Site Request Forgery (CSRF)*. As demais tiveram um desempenho abaixo do esperado em ambos os quesitos. O trabalho mostra a importância de uma vasta gama de ferramentas de análise de vulnerabilidade, pois usar apenas uma ferramenta não é suficiente para ter uma mínima segurança da ausência de falhas.

Palavras-Chave: Teste de software; Teste de segurança; Vulnerabilidade de software; Ferramentas de scan.

ABSTRACT

Security testing is an important part of today's software development and testing landscape as it provides an extra layer of security and can prevent future trouble such as data leaks and security breaches that can lead to, for example, financial loss. These tools perform black-box tests on the target application and aim to identify security vulnerabilities. This work has a qualitative and exploratory character and aims to compare the results obtained through the analysis of web applications by open-source scanner tools. The tools were chosen and filtered taking into consideration four aspects: They implement vulnerability analyzes listed on the OWASP and CWE websites, they were updated in the year 2021 or later, they are licensed as open-source software and can be used for web application testing. Analyzes were performed using the standard scan method of each tool to ensure configuration consistency. The OWASP and Arachni tools obtained the best results among those tested, both in terms of vulnerability identification and the generated text report. The other tools analyzed performance was below expectations in both aspects. The work shows the importance the use of a wide range of vulnerability analysis tools, because using just one tool is not enough to have a consistent test scenario.

Keywords: Software testing; Security testing; Software vulnerability; Scan tools.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CLI	*Command Line Interface*
CSRF	*Cross Site Request Forgery*
CVE	*Common Vulnerability and Exposures*
CWE	*Common Weakness Enumeration*
DOM	*Document Object Model*
HTML	*HyperText Markup Language*
OS	*Operating System*
OWASP	*Open Web Application Security Project*
SQL	*Structured Query Language*
SSL	*Secure Socket Layer*
TLS	*Transport Layer Security*
XSS	*Cross Site Scripting*

SUMÁRIO

1 INTRODUÇÃO

No início da internet existiam apenas sites estáticos. Estes sites desempenhavam um papel apenas de repositórios de informações que continham documentos estáticos. A maioria dos sites não utilizava sistemas de autenticação de usuários porque não havia esta necessidade, já que a mesma informação era exibida para todos os usuários de forma aberta.

Com o avanço dos sistemas de informação e da internet em si, os sites se tornaram um híbrido de informações abertas e fechadas, havendo a necessidade de proteger e autenticar o acesso ao conteúdo ali disponibilizado. A área de segurança da informação vem se tornando cada vez mais importante, já que a invasão de um servidor corporativo pode levar desde a um acesso indevido ao conteúdo da página até a exposição de dados de milhares de pessoas físicas e/ou jurídicas, resultando na maioria das vezes em um prejuízo monetário considerável.

No mundo de teste de software existem diversas definições relacionadas a problemas de segurança e/ou funcionamento de sistemas e com isso se faz necessário o esclarecimento de certos termos que se fazem presente na área. Os termos *bug*, erro, falha e defeito são amplamente utilizados na computação, mas suas definições podem variar entre as subáreas da computação.

Segundo o IEEE (2009), erro é a diferença entre o resultado gerado pelo software e o resultado esperado, onde a causa advém de uma ação humana. O defeito é a manifestação de um ou mais erros no software e pode apontar possíveis erros de requisitos. A falha ocorre quando o software não executa a sua função, ou seja, não cumpre o propósito para o qual foi criado. Já o *bug* é uma maneira genérica de se referir aos termos anteriores. (IEEE, 2009)

Na parte de cibersegurança, tais problemas são normalmente nomeados como fraquezas e vulnerabilidades. Fraquezas são falhas que podem gerar vulnerabilidades. (CVE, 2022a)

Já uma vulnerabilidade de software é um erro na aplicação que pode ser usada diretamente para ganhar acesso a um sistema ou rede. (CVE, 2022b)

Com isso, surgiram organizações e sistemas que visam educar os profissionais de TI sobre os principais problemas de segurança e formas de solucioná-los. Entre os principais estão o OWASP e o CWE/CVE. Estes órgãos organizam e catalogam as

mais diversas fraquezas e vulnerabilidades encontradas nos mais diversos softwares disponíveis.

A *Open Web Application Security Project* – OWASP atua com foco na colaboração para o fortalecimento da segurança de software em todo o mundo. A organização disponibiliza uma lista anual com as dez principais categorias de vulnerabilidades de softwares reportadas naquele ano.

Os projetos CVE e CWE também atuam catalogando as mais diversas falhas e vulnerabilidades encontradas nos mais diversos sistemas. O CWE cataloga as falhas de software em si, como por exemplo o *SQL-Injection*. Já o CVE cataloga as vulnerabilidades encontradas em um produto de software, como por exemplo a grave vulnerabilidade CVE-2021-44228 descoberta na biblioteca de logs do Apache, amplamente utilizada em vários sistemas.

Outro recurso criado para educar e treinar profissionais da área é a ideia de uma aplicação que é explicitamente vulnerável. Ou seja, possui falhas intencionais para que pesquisadores, especialistas da área e pessoas que estão iniciando na área de cibersegurança possam praticar a identificação das principais vulnerabilidades e falhas. Estas aplicações normalmente se baseiam nas informações divulgadas através da CWE, OWASP e outros órgãos semelhantes para implementar os problemas mais atuais dentro destes sistemas explicitamente vulneráveis.

Aliado a todos estes recursos, existe o teste de segurança. Este teste visa descobrir vulnerabilidades e falhas dentro de um aplicativo de software identificando as lacunas e possíveis fraquezas presentes na aplicação afim de evitar perda, acesso indevido ou vazamento de dados contidos ali.

Segundo o estudo do *CheckPoint Research* (2022), no ano de 2021 houve um aumento de 50% nas tentativas de invasão semanais a redes corporativas em comparação ao ano de 2020. No Brasil, esse aumento foi de 77%. Os cinco setores que mais sofreram tentativas de invasão foram: educação e pesquisa, governo/militar, comunicação, ISP/MSP e saúde com um aumento de 1.605, 1.131, 1.079, 1.068 e 830 casos respectivamente. Estes ataques buscam explorar vulnerabilidades em softwares que são executados em servidores para que o atacante obtenha controle e/ou acesso indevido as informações contidas no servidor. (*CheckPoint Research*, 2022)

Nessa perspectiva, diante do constante aumento de tentativas de invasão, principalmente a aplicações web, percebe-se a necessidade de avaliar possíveis

soluções voltadas para identificação e resolução das vulnerabilidades encontradas nos sistemas que são executados nestes servidores.

Neste contexto, o teste de segurança provê métodos e ferramentas a fim de facilitar o trabalho do testador de software em localizar possíveis vulnerabilidades que normalmente já são conhecidas pelos profissionais de segurança da informação e são implementadas nestas ferramentas, fazendo com que a o tempo gasto nestas análises seja mais bem aproveitado.

Atualmente, a cultura de testes nas empresas de desenvolvimento está se tornando regra. Porém, os testes que são performados visam a revisão e teste do código fonte para identificar possíveis falhas no software, não problemas de segurança mais amplos que são explorados após a aplicação ser colocada em produção.

Uma das abordagens que visa reduzir e/ou mitigar as invasões em sistemas de informação é o uso das ferramentas de scanners automatizados que fazem uso de componentes como *spiders* e *fuzzers* que vasculham os links presentes nas aplicações web e os testam através de listas de entradas com elementos inválidos, aleatórios e que estão fora do escopo de entradas esperadas pela aplicação.

Essas ferramentas agilizam o teste manual através da detecção automática de possíveis vulnerabilidades presentes na aplicação, fazendo com que o trabalho manual seja aplicado especificamente nas partes do sistema onde o scanner encontrou estas possíveis vulnerabilidades.

1.1 Objetivos

1.1.1 Geral

Apresentar um comparativo entre ferramentas de teste de software para profissionais da área da computação afim de identificar problemas que envolvem a localização de vulnerabilidades em aplicações WEB.

1.1.2 Específicos

- Descrever e exemplificar as principais vulnerabilidades e falhas de software que acontecem em aplicações web;

- Identificar e selecionar ferramentas de scanner automático open source que englobam as vulnerabilidades relacionadas a aplicações web;

- Apresentar uma análise comparativa dos resultados de cada ferramenta;

1.1.3 Estrutura do documento

No capítulo 2 são discutidos alguns conceitos relevantes para o trabalho, como uma variedade de tipos de testes e as organizações que atuam na área de segurança da informação.

No capítulo 3 é explicado como acontece o processo de teste de segurança e o funcionamento dos scanners de vulnerabilidades e o detalhamento de etapas do processo.

No capítulo 4 são levantadas as questões da pesquisa e os passos utilizados para selecionar as ferramentas e um detalhamento mais aprofundado no funcionamento de cada uma, assim como o seu desempenho nos testes.

No capítulo 5 são feitas as considerações finais e avaliação dos resultados.

2 REFERENCIAL TEÓRICO

Nesta seção são apresentadas as referências bibliográficas utilizadas como embasamento para a fundamentação teórica das áreas e ferramentas abordadas neste trabalho.

Inicialmente são apresentadas definições de teste de software e suas categorias mais utilizadas no ambiente de testes de forma geral. Em seguida, discutimos as organizações mais atuantes no cenário de segurança da informação e são definidas as principais ameaças e vulnerabilidades de softwares catalogadas por estas organizações, assim como uma distinção entre certos termos utilizados no âmbito de testes.

Também é definido o que é teste de segurança e são mostradas aplicações que auxiliam na identificação de vulnerabilidades em aplicações. Por fim, é feita uma contextualização da evolução das aplicações web e são apresentadas algumas aplicações que foram desenvolvidas contendo algumas das vulnerabilidades discutidas anteriormente com o intuito de servirem como ferramentas de aprendizagem.

2.1 O teste de software

Sistemas de Informação modernos baseados em conceitos como computação em nuvem, serviços baseados em localização ou redes sociais estão permanentemente ligados a outros sistemas e constantemente manipulam dados sensíveis. Estes sistemas interligados estão sujeitos a ataques de segurança que podem resultar em incidentes de segurança de elevada gravidade que podem afetar a infraestrutura técnica ou seu ambiente. (FELDERER et al., 2016)

O teste tem como objetivo mostrar que um programa faz o que se propõe a fazer e descobrir defeitos do programa antes de colocá-lo em uso. Quando se testa um software, o programa é executado usando dados artificiais. Então os resultados da execução de teste são analisados em busca de erros, anomalias ou informações sobre os atributos não funcionais do programa. (SOMMERVILE, 2011)

2.1.1 Teste funcional

Teste funcional é uma técnica utilizada para se projetarem casos de teste no qual o programa ou sistema é considerado uma caixa-preta. No teste de caixa-preta, os detalhes de implementação não são considerados e o software é avaliado segundo o ponto de vista do usuário. (DELAMARO et al., 2016)

Também existe o teste caixa-branca. Este é o contrário do teste caixa-preta. É um teste que envolve a análise da codificação interna e infraestrutura de uma solução de software. Concentra-se principalmente no fortalecimento do salvaguardar, o fluxo de entradas e saídas através da aplicação e a melhoria do design e usabilidade com conhecimento total do código fonte.

Durante o ciclo de desenvolvimento do software podem ser aplicados alguns tipos de testes, como teste de unidade, teste de componente e teste de sistema.

2.1.1.1 Teste de unidade

No teste de unidade são testadas unidades do programa ou classes de objetos. Esse teste foca na funcionalidade dos métodos presentes nessas partes do programa. (SOMMERVILE, 2011)

2.1.1.2 Teste de componente

O teste de componente aglutina várias dessas unidades de programa e visa o teste da interface do componente que dá acesso às funções do componente. (SOMMERVILE, 2011)

2.1.1.3 Teste de sistema

O teste de sistema visa testar a integração de alguns ou todos os componentes do sistema como um todo, focando na interação que ocorre entre esses componentes. (SOMMERVILE, 2011)

2.1.2 Teste não-funcional

Outra classificação existente no mundo de testes de software é a de teste não-funcional. Este visa a verificação de aspectos não funcionais da aplicação como performance, usabilidade e confiabilidade. Os tipos mais comuns de testes não funcionais são o teste de carga, teste de desempenho e o teste de estresse.

2.1.2.1 Teste de desempenho

Tem como objetivo avaliar o desempenho de um sistema no que diz respeito à sua capacidade de resposta e à sua disponibilidade. Trata-se então da execução de uma requisição por parte de um cliente e a avaliação do tempo de resposta recebido.

Para este tipo de teste ser realizado, devem ser definidos alguns parâmetros, tais como, por exemplo, o número médio de utilizadores esperados em simultâneo, o tempo entre pedidos distintos, o tempo máximo aceitável para que o servidor responda a uma requisição etc. (MOLYNEAUX, 2014)

2.1.2.2 Teste de carga

Este tipo de teste é no fundo uma subcategoria do teste de Desempenho. Define, no entanto, que um sistema seja avaliado em condições de carga pré-definidas, ou seja, o sistema será testado para uma quantidade de pedidos que imponha alguma carga significativa ao sistema. Visa avaliar o tempo necessário à execução de uma ou mais tarefas em simultâneo. (MOLYNEAUX, 2014)

2.1.2.3 Teste de estresse

Um teste de estresse é em certa medida muito semelhante aos já descritos testes de Desempenho e Carga. Funciona sensivelmente da mesma forma, mas o desempenho será avaliado com o sistema a funcionar em situações de sobrecarga, muito para além do seu ponto de ruptura, ou seja, o sistema será obrigado a responder a quantidades massivas de tráfego HTTP e a suportar quantidades de carga muito para além das suas capacidades. Ou seja, o teste de estresse visa simular uma execução exaustiva do sistema para permitir encontrar problemas relacionados ao ambiente de execução (máquina executora do processo, sistema operacional, partes

além do próprio sistema, mas que influenciam diretamente o seu funcionamento). (MOLYNEAUX, 2014)

2.2 Teste de segurança

O teste de segurança identifica se as propriedades de segurança especificadas ou pretendidas são, para um determinado conjunto de ativos de interesses, implementadas corretamente. Isso pode ser feito para mostrar conformidade com as propriedades de segurança, semelhante aos testes baseados em requisitos; ou tentando resolver vulnerabilidades conhecidas, que são semelhantes aos testes tradicionais baseados em falhas ou destrutivos.

2.2.1 OWASP

A sigla OWASP é a abreviação para "*Open Web Application Security Project*". Trata-se de uma entidade sem fins lucrativos e com reconhecimento internacional, atuando com foco na colaboração para o fortalecimento da segurança de software em todo o mundo.

A entidade é responsável por um dos scanners mais reconhecidos entre os profissionais de segurança da informação, o OWASP Zed Proxy. A aplicação *OWASP Zed Attack Proxy*[1] é um scanner de aplicações web grátis e de código aberto mantido pela comunidade de segurança da informação. Ele é constantemente atualizado para abranger todas as ameaças contidas na lista divulgada pela própria OWASP.

A organização também mantém uma lista com as 10 categorias de vulnerabilidades de segurança de software mais perigosas, juntamente com os métodos mais eficazes de como lidar com elas. Esta lista é divulgada anualmente, sendo a divulgada em 2021 a versão mais atual da lista no momento. A seguir serão definidas cada categoria contida na lista. (OWASP, 2022b)

[1] *ZAPROXY. In: GitHub, 2023. Disponível em: https://github.com/zaproxy/zaproxy. Acesso em: 5 jul. 2023.*

2.2.1.1 A01:2021- Broken access control

As falhas nesta categoria geralmente levam à divulgação, modificação ou destruição de informações não autorizadas de todos os dados ou à execução de uma função comercial fora dos limites do usuário. Cross-Site Request Forgery (CWE-352) é a falha mais conhecida desta categoria.

2.2.1.2 A02:2021-Cryptographic failures

Se refere à proteção dos dados em trânsito (que estão sendo enviados de um local para outro) e em repouso (que estão armazenado em um disco rígido, banco de dados etc.). Alguns dados relevantes e suscetíveis a este problema são senhas, números de cartão de crédito, registros de saúde, informações pessoais, segredos comerciais etc. Uso de senhas forçadas diretamente no código (CWE-259) e entropia insuficiente (CWE-331) são as principais falhas dessa categoria.

2.2.1.3 A03:2021-Injection

Esta categoria se destina a ameaças de segurança que, através de um comando passado ao servidor, tenta extrair informações utilizando uma injeção de código por meio de campos de texto disponíveis na aplicação ou alterando diretamente a URL do site. *SQL-Injection* (CWE-79), *Cross-Site Scripting* (CWE-89) e *External control of filename or path* (CWE-73) são as falhas mais conhecidas dessa categoria.

2.2.1.4 A04:2021-Insecure design

Design inseguro é uma categoria ampla que representa diferentes pontos fracos, expressos como “projeto de controle ausente ou ineficaz”. Não é necessariamente má implementação. Elementos como a exibição de mensagens de erros contendo informações sensíveis (CWE-209) e armazenamento de credenciais de forma não criptografada (CWE-256) englobam essa categoria.

2.2.1.5 A05:2021-Security misconfiguration

Normalmente se dá por conta de certas características que foram incorporadas, instaladas ou ativadas no projeto de forma desnecessária, erros exibem o *stack trace* de forma não controlada, controle de segurança não adequado, software desatualizado etc.

2.2.1.6 A06:2021-Vulnerable and outdated components

É normalmente encontrada quando não se tem controle das versões dos componentes usados no desenvolvimento da aplicação. O uso de bibliotecas desatualizadas ou que não estão mais em desenvolvimento (CWE-1104) estão inclusas nessa categoria.

2.2.1.7 A07:2021-Identification and authentication failures

Se refere a confirmação de identidade de usuários e gerenciamento de sessões. Alguns pontos que contribuem para isso são falta de autenticação multifator, o atacante consegue acesso a lista de usuários e senhas válidos, sistema desprotegido de recuperação de Email e senha etc. Fixação de sessão (CWE-384) é uma das principais falhas dessa categoria.

2.2.1.8 A08:2021-Software and data integrity failures

As falhas de integridade de software e dados estão relacionadas a código e infraestrutura que não protegem contra violações de integridade. Um exemplo disso é quando um aplicativo depende de plugins, bibliotecas ou módulos de fontes não confiáveis, repositórios e redes de entrega de conteúdo etc. Inclusão de funcionalidades de fontes não confiáveis (CWE-829) e descarga de código sem verificação de integridade (CWE-494) são as principais falhas contidas nessa categoria.

2.2.1.9 A09:2021-Security logging and monitoring failures

Esta categoria serve para ajudar a detectar, escalar e responder a violações ativas. Sem registro e monitoramento, as violações não podem ser detectadas. Log Insuficiente (CWE-778), Inserção de informações sensíveis no arquivo de log (CWE-532) fazem parte dessa categoria.

2.2.1.10 A10:2021-Server-side request forgery

As falhas do SSRF ocorrem sempre que um aplicativo da Web está buscando um recurso remoto sem validar a URL fornecida pelo usuário. Ele permite que um invasor force o aplicativo a enviar uma solicitação criada para um destino inesperado, mesmo quando protegido por um firewall, VPN ou outro tipo de lista de controle de acesso à rede (ACL). Como é uma categoria recente, ainda não existiam CWEs sendo destacadas como pertencente a esta categoria.

2.2.2 CVE e CWE

A MITRE é uma organização financiada pelo governo dos Estados Unidos que publica padrões a serem usados pela comunidade de segurança da informação. Dois dos mais populares projetos são CWE e CVE, e muitas vezes podem ser confundidos pelos profissionais de segurança.

O objetivo do programa CVE é identificar, definir e catalogar vulnerabilidades de segurança cibernética divulgadas publicamente. Há um registro CVE para cada vulnerabilidade no catálogo. As vulnerabilidades são descobertas, atribuídas e publicadas por organizações de todo o mundo que fizeram parceria com o Programa CVE. (CVE, 2022a)

Os parceiros publicam registros no CVE para comunicar descrições consistentes de vulnerabilidades. Os Profissionais de tecnologia da informação e segurança cibernética usam os Registros CVE para garantir que estejam discutindo o mesmo problema e para coordenar seus esforços para priorizar e resolver as vulnerabilidades.

Common Weakness Enumeration (CWE) é uma lista desenvolvida pela comunidade de tipos comuns de fraquezas de software e hardware que tem

ramificações de segurança. A Lista CWE e a taxonomia de classificação associada servem como uma linguagem que pode ser usada para identificar e descrever esses pontos fracos em termos de CWEs. (CWE, 2022a)

O CWE disponibiliza algumas linguagens que possuem listas próprias com as principais falhas naquela linguagem, que são PHP, C, Java e C++. As demais não possuem tal lista.

Voltado para as comunidades de desenvolvedores e profissionais de segurança, o principal objetivo do CWE é interromper as vulnerabilidades na origem, educando arquitetos, designers, programadores e adquirentes de software e hardware sobre como eliminar os erros mais comuns antes que os produtos sejam entregues.

Em última análise, o uso do CWE ajuda a evitar os tipos de vulnerabilidades de segurança que atormentam os setores de software e hardware e colocam as instituições que os usam em risco.

Em resumo, o CWE tem a ver com a fraqueza – não com a instância dentro de um produto ou sistema. Já o CVE tem a ver com a instância específica dentro de um produto ou sistema – não com a falha subjacente. A seguir serão definidas as falhas contidas no TOP 10 CWE.

2.2.2.1 CWE-787: Out-of-bounds write

O software grava dados em uma posição anterior ou posterior do buffer. A falha implica em corrompimento dos dados, *crashs*, modificação de memória ou execução de código.

A causa dessa falha se dá normalmente pela manipulação errônea de indexes e ponteiros que tentam acessar endereços inválidos ou fora do buffer de memória. Essa falha afeta principalmente as linguagens C, C++ e Assembly. (CWE, 2022c)

2.2.2.2 CWE-79: Improper neutralization of input during web page generation ('cross-site scripting')

O software não neutraliza ou neutraliza incorretamente a entrada controlável pelo usuário antes de ser colocada na saída que é usada como uma página Web que é fornecida a outros usuários. (CWE, 2022d)

É causada principalmente por dados não confiáveis que entram na aplicação através de uma requisição web, ou durante a geração da página a aplicação não

verifica se os dados usados na geração possuem algum conteúdo que pode ser executado pelo browser que foi injetado através de dados não confiáveis, ou ainda o software gera dinamicamente uma página da Web que contém esses dados não confiáveis.

Esse tipo de fraqueza independe da linguagem de programação e afeta principalmente aplicações web. Existem três tipos principais de XXS, são eles:

- **XSS Refletivo (Não persistente): Tipo 1** - O servidor lê dados diretamente da requisição HTTP e reflete para a resposta HTTP. O método mais comum utilizado para explorar esse tipo de XSS é incluindo o conteúdo malicioso como um parâmetro da URL.

- **XSS armazenado (Persistente): tipo 2** - A aplicação armazena o conteúdo perigoso na base de dados, log de um visitante etc. Posteriormente, esses dados são lidos pela aplicação e incluído em algum processo gerado dinamicamente.

- **XSS *DOM-Based*: tipo 0** - O cliente realiza a injeção de XSS na página; nos demais tipos, o servidor realiza a injeção. O XSS baseado em DOM geralmente envolve um script confiável controlado pelo servidor que é enviado ao cliente, como um código Javascript que executa verificações de integridade em um formulário antes que o usuário o envie. Se o script fornecido pelo servidor processa os dados fornecidos pelo usuário e os injeta de volta na página da Web (como HTML dinâmico), o XSS baseado em DOM é possível.

2.2.2.3 CWE-125: Out-of-bounds read

O software lê dados em uma posição anterior ou posterior do buffer. A falha implica em travamentos, modificação de memória ou acesso a outros endereços de memória que contém dados. (CWE, 2022e)

É causada principalmente pela manipulação errônea de indexes e ponteiros que tentam acessar endereços inválidos ou fora do buffer de memória e erro na manipulação de sentinelas. Afeta principalmente as linguagens C e C++.

2.2.2.4 CWE-20: Improper input validation

O software recebe dados ou entradas e não válida, ou valida erroneamente. Causada principalmente por falta ou falha de validação das entradas e dados recebidos pelo software. Esse erro independe de linguagem. (CWE, 2022f)

2.2.2.5 CWE-78: Improper neutralization of special elements used in an os command ('OS command injection')

Tratamento insuficiente de comandos enviados ao SO por um software, levando a um ambiente vulnerável. É causada por entradas que não são verificadas antes da execução do comando e independe de linguagem. (CWE, 2022g)

2.2.2.6 CWE-89: Improper neutralization of special elements used in an sql command ('sql-injection')

O Software executa um comando SQL a partir de uma entrada influenciada por um usuário. A causa é a não verificação ou verificação incorreta de elementos perigosos que podem modificar o comando original. É uma falha que independe da linguagem e afeta principalmente servidores de banco de dados. (CWE, 2022h)

2.2.2.7 CWE-416: Use after free

O software referência um endereço de memória após o endereço ter sido desalocado causando erros no programa, como o uso de valores errados. É causado por uma lógica de programação errônea e/ou confusão na identificação de que parte do programa é responsável por limpar a memória afetando principalmente as linguagens C e C++. (CWE, 2022i)

2.2.2.8 CWE-22: Improper limitation of a pathname to a restricted directory ('path traversal')

O software usa entrada externa para construir um nome de caminho destinado a identificar um arquivo ou diretório localizado abaixo de um diretório pai restrito. (CWE, 2022j)

O problema ocorre quando o software não neutraliza adequadamente elementos especiais dentro do nome de caminho que podem fazer com que o nome do caminho seja resolvido para um local que está fora do diretório restrito. Este é um erro que independe de linguagem.

2.2.2.9 CWE-352: Cross-Site request forgery (CSRF)

O servidor web recebe uma requisição e a trata como legítima. Pode ser feito por meio de um URL, carregamento de imagem etc. e pode resultar na exposição de dados ou execução de código não intencional. (CWE, 2022k)

Ocorre quando o aplicativo web não verifica ou não verifica suficientemente se uma solicitação bem formada, válida e consistente foi fornecida intencionalmente pelo usuário que enviou a solicitação. Essa falha é independence de linguagem.

2.2.2.10 CWE-434: Unrestricted upload of file with dangerous type

O software permite que o invasor carregue ou transfira arquivos de tipos perigosos que podem ser processados automaticamente pela aplicação. É causado pela falta de verificação no tipo ou conteúdo do arquivo. Este erro independe de linguagem. (CWE, 2022l)

2.2.3 Termos relacionados a problemas em softwares

Um ponto importante na área de teste de software é o entendimento correto das várias terminologias que existem quando se trata de problemas no sistema. Por isso, é necessário definir o que cada termo significa de forma clara para que não haja confusão. Dentro da área de softwares existem algumas definições que tratam de problemas no software como por exemplo erro, falha e bugs.

Segundo Delamaro et al. (2016)**,** não existe uma unanimidade em relação ao significado dos termos que se referem a problemas no software, pois o termo “erro” é generalizado no cotidiano. Então, os seguintes termos sobre problemas em softwares serão seguidos. (DELAMARO et al., 2016)

As fraquezas são defeitos, falhas, bugs ou outros erros na implementação de software ou hardware, código, design ou arquitetura que, se não forem resolvidos, podem resultar em sistemas, redes ou hardware vulneráveis a ataques.

A vulnerabilidade é uma fraqueza no software que pode ser diretamente usada por um agente mal-intencionado para obter acesso a um sistema ou rede.

2.2.4 Teste de penetração

O teste de penetração é um teste que visa analisar, na visão de um atacante, possíveis falhas no software afim de obter informações sigilosas através da aplicação de técnicas de invasão. (WEIDMAN, 2014)

Testes de penetração são difíceis de modelar porque normalmente não produzem problemas de segurança visíveis. Também exige que o testador pense como um *hacker*, o que acarreta a necessidade de conhecimentos específicos de segurança e dos componentes que fazem parte daquele sistema.

2.2.5 Automatização de testes

Dependendo do tamanho do projeto do software que está sendo desenvolvido, testá-lo pode ser muito desgastante e consumir muito tempo. Segundo Delamaro et al. (2016)**,** um ponto importante para o sucesso no teste de um software é a automatização.

Diversos tipos de ferramentas de teste têm sido utilizados para buscar aumentar a produtividade nessa atividade, que tende a ser extremamente dispendiosa. (DELAMARO et al., 2016)

2.2.6 Ferramentas de scanners

Ferramentas de escaneamento automático são softwares desenvolvidos com o intuito de identificar possíveis vulnerabilidades em programas de computador. Este tipo de análise não envolve o código da aplicação, é todo feito utilizando a ideia do teste de caixa preta. O uso deste tipo de ferramenta ajuda a mitigar riscos que possíveis problemas no software podem gerar, além de agilizar o trabalho do profissional de segurança da informação.

Com o objetivo de facilitar o uso das várias ferramentas utilizadas no mundo dos testes de penetração, a comunidade de software livre desenvolveu o sistema Kali Linux. É um sistema de código aberto é baseado no Debian e oferece uma variedade de ferramentas pré-instaladas por padrão. Além das ferramentas pré-instaladas, o sistema ainda conta com um grande repositório online com mais uma gama de ferramentas que podem ser utilizadas por profissionais de segurança da informação. O escopo do sistema abrange teste de penetração, computação forense, engenharia reversa, gerenciamento de vulnerabilidades etc. (KALI, 2022)

2.2.6.1 FindBugs

Ferramenta de código aberto para análise estática especificamente para códigos escritos em JAVA. O software requer o uso da JDK na versão 1.7 ou superior, mas consegue analisar códigos escritos em versões anteriores do JAVA. (FINDBUGS, 2022)

2.2.6.2 OWASP Zed Proxy Attack

É um scanner de segurança de aplicativos da web de código aberto. Ele pode ser usado tanto por aqueles que são novos em segurança de aplicativos quanto por testadores de penetração profissionais. É um dos projetos mais ativos do OWASP e recebeu o status de carro-chefe da entidade. É uma das ferramentas mais completas disponíveis até o momento e possui várias ferramentas para procurar vulnerabilidades listadas pela OWASP. Quando usado como um servidor de proxy, permite que o usuário manipule todo o tráfego que passar por ele, incluindo tráfego HTTP. (ZAPROXY, 2022b)

2.2.6.3 w3af

É uma aplicação de código aberto para ataques web e quadro de auditoria, usada para identificar vulnerabilidades em aplicações web através de requisições HTTP. Verifica, entre outras vulnerabilidades, *SQL-Injection* e XSS. (W3AF, 2022)

2.2.6.4 Arachni scanner

É uma aplicação de código aberto desenvolvida em rubi, modular e disponibiliza vários recursos para testadores e administradores com o intuito de avaliar a segurança de aplicativos da web modernos. É multiplataforma e cobre uma grande quantidade de casos de uso, desde um simples utilitário de scanner de linha de comando, até uma grade global de scanners de alto desempenho. (ARACHINI/GITHUB, 2022)

2.2.6.5 Nikto

É uma web *scanner* de código aberto que realiza testes abrangentes em servidores *web* para vários itens, incluindo mais de 6700 arquivos potencialmente perigosos Algumas das verificações feitas pela ferramenta são: Má configuração de software e servidores, problemas em programas e arquivos padrões, arquivos e programas inseguros, programas e servidores com versões antigas de software etc. (NIKTO, 2022)

2.2.6.6 Wfuzz

Ferramenta de código aberto utilizada para testes de segurança em aplicações web. Envolve testes de LDAP *Injection, SQL-Injection, XSS Injection* etc. Alguns pontos fortes da aplicação são o seu suporte a autenticação, *Cookies fuzzing Multi-threading* Pontos de injeção múltiplos e Suporte a *proxy* e *SOCK*. (WFUZZ, 2022)

2.2.6.7 Wapiti

Ferramenta de código aberto que permite auditar sites e aplicações web através de scans black-box. Procura por páginas e formulários com possíveis vulnerabilidades. Uma vez catalogada a lista de URLs, formulários e seus respectivos inputs, injeta payloads para testar a falha. A aplicação está disponível para vários sistemas. (WAPITI, 2022)

2.2.6.8 SQLMap

O SQLMap é uma das aplicações mais conhecidas quando se trata de testes de invasão. É uma ferramenta de código aberto usada para testes de segurança em servidores de banco de dados. Suporta praticamente todas as bases de dados mais famosas, como: SQLServer, MySQL, SQLite, H2, Amazon Redshift etc. Utiliza seis tipos de técnicas de SQL injection: boolean-based blind, time-based blind, error-based, UNION query-based, stacked queries and out-of-band. (SQLMAP, 2022)

2.2.6.9 NoGoToFail

É uma ferramenta de código aberto desenvolvida pelo Google para encontrar vulnerabilidades e falhas de configuração em certificados TLS/SSL. Procura por vulnerabilidades como: *MiTM attacks, SSL Certificate verification issues, SSL injection e TLS Injection*. (NOGOTOFAIL, 2022)

2.3 Aplicações WEB

No início da internet existiam apenas sites estáticos. Os sites existentes desempenhavam papel apenas de repositórios de informações que continham documentos estáticos, e os navegadores web foram criados como uma forma de visualização desses documentos. A maioria dos sites não autenticava usuários porque não havia necessidade já que a mesma informação era exibida para todos os usuários. (HABIB, 2013)

Neste cenário, o teste das aplicações era pouco utilizado pelo fato de as páginas web da época não serem softwares complexos e/ou com dados sensíveis que precisavam ser protegidos. No entanto, com o amadurecimento no processo de desenvolvimento de sistemas, o teste de software vem se provando uma peça fundamental no ciclo de desenvolvimento de aplicações. Boa parte dos grandes softwares que antes eram feitos exclusivamente para plataformas desktop, hoje estão recebendo versões mobile e web devido a evolução de vários setores da indústria de tecnologia.

2.3.1 Aplicações web explicitamente vulneráveis

Aplicações explicitamente vulneráveis são sites e aplicativos da web projetados para serem inseguros por padrão. Profissionais de segurança podem treinar técnicas de ataque, aumentar seus conhecimentos e encontrar novas vulnerabilidades. (GEEKFLARE, 2022). O OWASP mantém em seu site um diretório contendo várias destas aplicações de forma gratuita. (OWASP, 2022a)

2.3.1.1 Aplicações em PHP

- **Acuart**: Aplicação online com vulnerabilidades como SQL Injection, Cross-site Scripting (XSS), Cross-site Request Forgery (CSRF), entre outras.

2.3.1.2 Aplicações em python

- **Gruyere**: Aplicação online com vulnerabilidades como cross-site scripting, cross-site request forgery, information disclosure, denial of service, and remote code execution.
- **PyGoat**: Aplicação online com 10 categorias que seguem o top10 OWASP

2.3.1.3 Aplicações em JavaScript

- **Broken Crystals**: Aplicação que roda localmente e oferece uma vasta lista de vulnerabilidades a serem exploradas

- **Cyber Scavenger Hunt**: Aplicação online que oferece uma espécie de capture the flag mostrando vulnerabilidades e dicas de como encontrá-las

- **OWASP Juice Shop**: Aplicação que pode ser executada online ou localmente e oferece uma série de vulnerabilidades a serem exploradas pertencentes ao top 10 OWASP.

3 METODOLOGIA

Nesta seção serão expostas as abordagens e estratégias utilizadas para seleção das ferramentas de scanner automático para aplicações web e os testes feitos utilizando estes scanners.

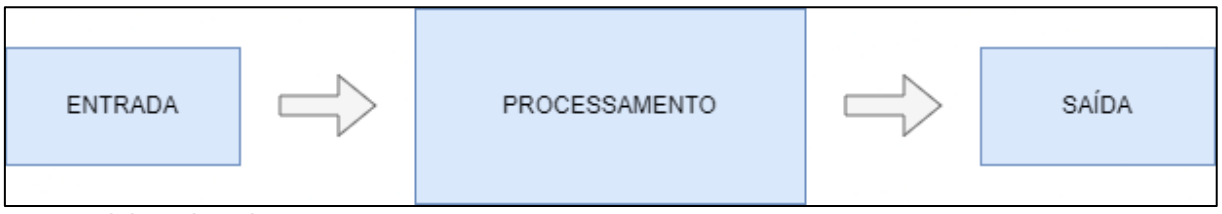
A avaliação tem caráter qualitativo, pois busca avaliar o tempo gasto no escaneamento das aplicações, a identificação de vulnerabilidades consideradas relevantes (alto risco) e a qualidade dos relatórios gerados pelas ferramentas.

Também possui caráter exploratório por conta da procura por diversas ferramentas que servem ao propósito do trabalho.

3.1 Abordagem

Para realizar uma análise de uma aplicação, seja ela nativa de um sistema ou hospedada em um servidor para ser acessada através de um navegador, assemelha-se bastante as etapas de um programa de computador assim como mostrado na Figura 1. Existe uma ou mais entradas com a(s) informação(es) inicial(s) necessária(s) para dar início a execução, a partir disso é feito um processamento interno, e em seguida é disponibilizada uma saída contendo a entrada processada e/ou dados complementares derivados desta.

Figura 1 - Etapas de uma operação geral de processamento de dados

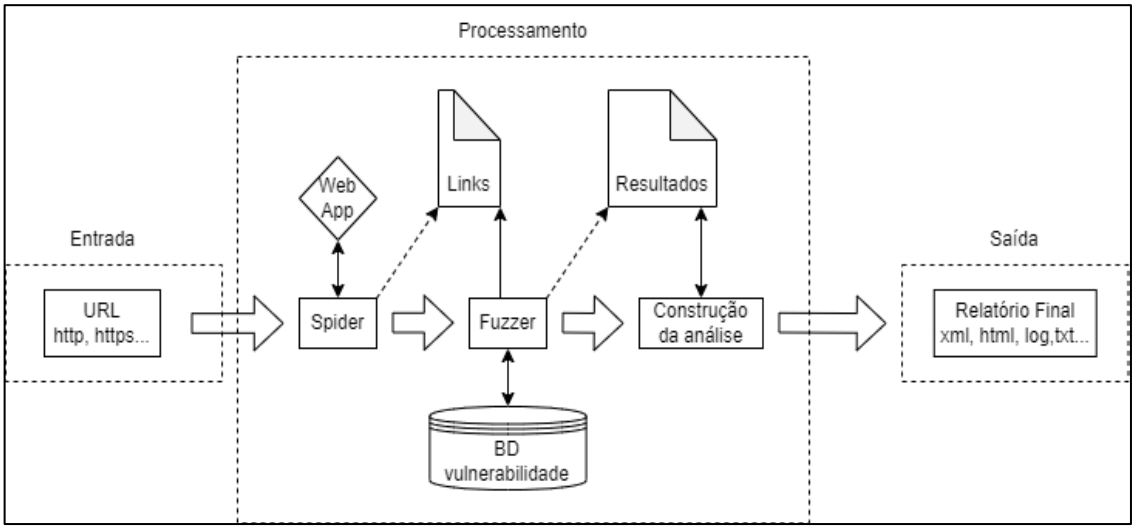


Fonte: Elaborado pelo autor, 2022.

Os scanners de aplicações web funcionam de forma bem parecida de modo geral. É recebido um URL como entrada para o scanner, então os componentes *spider* e *fuzzer* atuam no processamento dos sub links presentes naquela URL. Ao identificar todos os sub links, são injetadas as mais diversas técnicas e valores contidos na base da dados da aplicação de scanner afim de tentar identificar as vulnerabilidades da aplicação. Por fim, é gerado um relatório contendo informações relevantes sobre o

teste como os tipos de vulnerabilidades encontradas, o nível de severidade de tal vulnerabilidade e a melhor forma de remover esse problema. A representação visual pode ser vista na figura 2.

Figura 2 – Visão geral do funcionamento de uma ferramenta automática de escaneamento



Fonte: Elaborado pelo autor, 2022.

3.1.1 Scanner e Spider

Aplicações web possuem uma infinidade de links, diretórios e caminhos que podem ser acessados das mais diversas formas. Fazer isso manualmente é trabalhoso e demanda muito tempo como já foi explicitado. É neste processo que entra o Spider.

O Spider nada mais é que um vasculhador de informações. Essa ferramenta é usada para, de forma automática, procurar URLs em um site específico. O *spider* é iniciado recebendo como entrada um ou mais sites, chamados de *seeds*, e a partir desses sites é feita uma busca por novas URLs. (ZAPROXY, 2022a)

A Ferramenta visita essa URL, identifica todos os hyperlinks naquela página, adiciona esses links a uma lista de sites a serem visitados, e de forma recursiva, continua procurando novos links.

O scanner é, de modo geral, a junção das ferramentas que atuam na coleta e processamento das informações.

A partir das URLs encontradas, o scanner inicia o chamado *Active Scan*, que é onde essas URLs serão testadas pelo *fuzzer* para verificar se elas possuem elementos falhos que podem acarretar vulnerabilidades no software. Os tipos de vulnerabilidades que serão testadas variam de *scanner* para *scanner*.

3.1.2 Fuzzer

Outro componente importante para a automatização dos testes de penetração é o *fuzzer*. Este componente é responsável por tratar da lista de entradas que serão passadas para a aplicação alvo. A lista consiste em entradas inválidas, aleatórias e que estão fora do domínio esperado pelo sistema alvo afim de verificar se aquela entrada inválida irá permitir a exploração de alguma fraqueza presente na aplicação, como *SQL-injection, cross site scripting, remote command execution* etc.

As ferramentas mais modernas de teste de invasão já incluem esse módulo ao serem instaladas, assim como uma lista pré-definida de entradas a serem usadas no teste.

3.1.3 Exemplo ilustrativo

Nesta seção será feita uma demonstração dos passos necessários para a execução de uma varredura através das ferramentas de escaneamento automático. Um site explicitamente vulnerável e uma ferramenta de escaneamento serão escolhidas para a demonstração. As descrições de ambos estão presentes nas seções 2.3.1 e 2.2.6 deste mesmo documento. Posteriormente, será discutido como funciona o tratamento do resultado pelo testador a partir do relatório gerado pela ferramenta.

O primeiro passo para a execução do procedimento é identificar a aplicação que será escaneada. É possível que a aplicação seja varrida em sua totalidade ou apenas partes específicas. Isso fica a critério do testador, que deve também levar em conta a capacidade de customização da ferramenta escolhida. Nesta demonstração a aplicação escolhida foi a *Gruyere* (2.3.1.2). É possível visualizar a interface da aplicação sendo executada via navegador na figura 3.

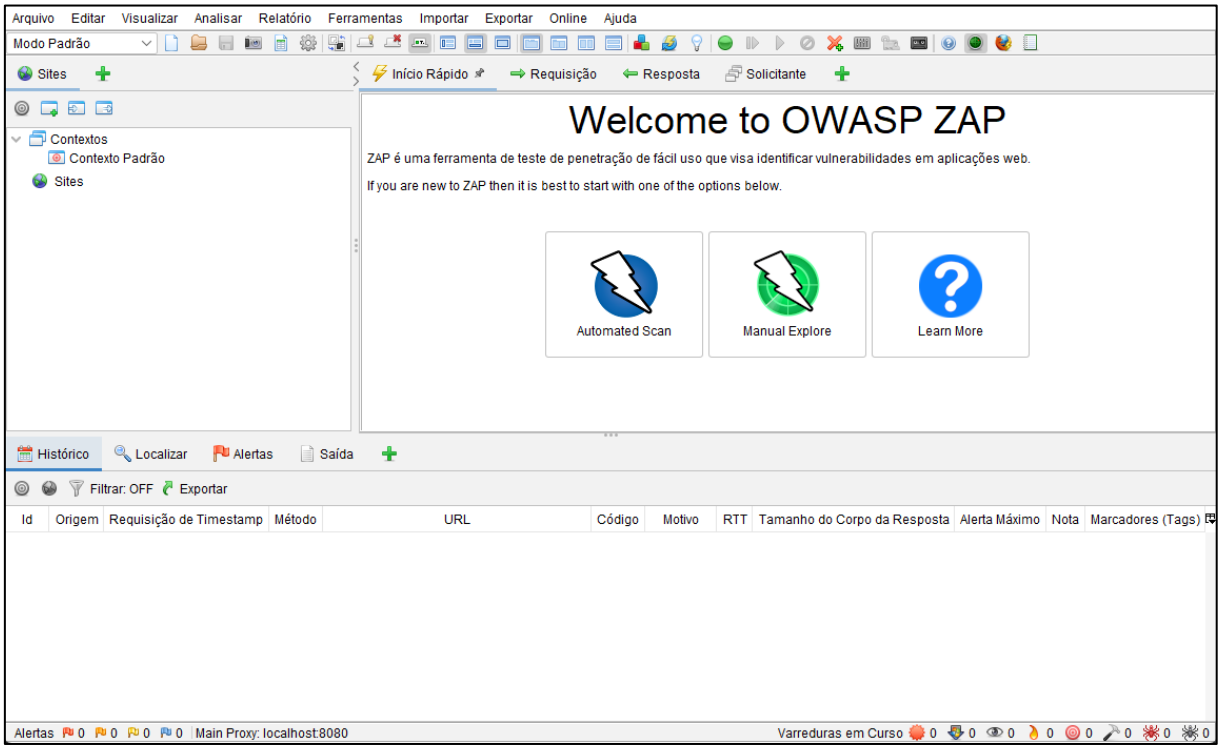
Figura 3 – Página inicial da aplicação web Gruyere

Fonte: Elaborado pelo autor, 2022.

Em seguida vem a escolha da ferramenta. Para essa simulação, a ferramenta escolhida foi a OWASP Zed Attack Proxy (2.2.6.2). O ZAP é uma das ferramentas mais completas disponíveis atualmente. É altamente customizável e possui vários ajustes finos para atender vários tipos de demanda. Para esta análise, vamos usar o escaneamento padrão da ferramenta. A entrada para dar início a análise é justamente o endereço da aplicação web.

Para este exemplo, será utilizada a opção “Automated Scan”. Esta opção pode ser acessada diretamente pela página inicial da aplicação (Figura 4). A ferramenta também possibilita usar um navegador embutido (normalmente o Mozilla Firefox) e ao iniciar este navegador, também é mostrada a opção de escaneamento automático.

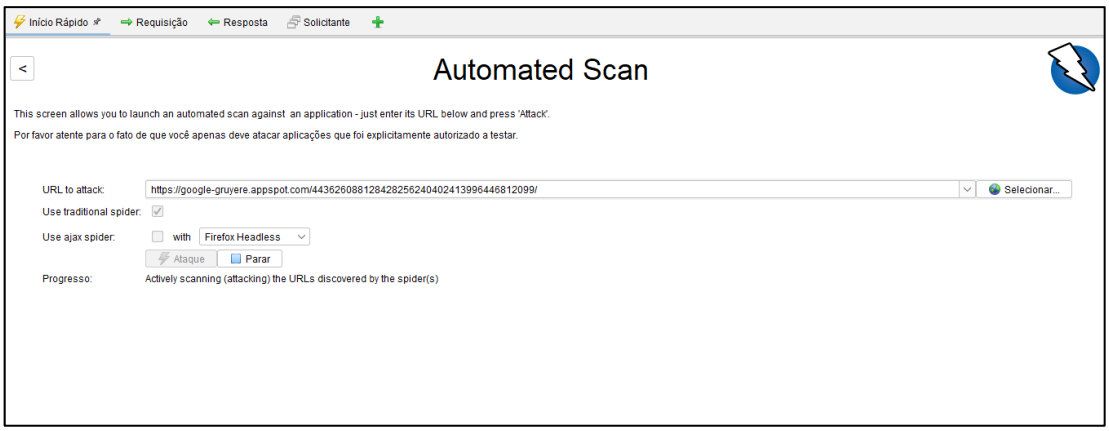
Figura 4 – Interface inicial da ferramenta *Zed Attack Proxy*



Fonte: Elaborado pelo autor, 2022.

Ao clicar na opção “Automated Scan”, é apresentado o campo para inserir o endereço da aplicação web que será escaneada (Figura 5), assim como algumas configurações relacionadas ao *spider*. Para iniciar o escaneamento padrão basta clicar no botão “Ataque”.

Figura 5 – Informações de escaneamento automático do Zed Attack Proxy

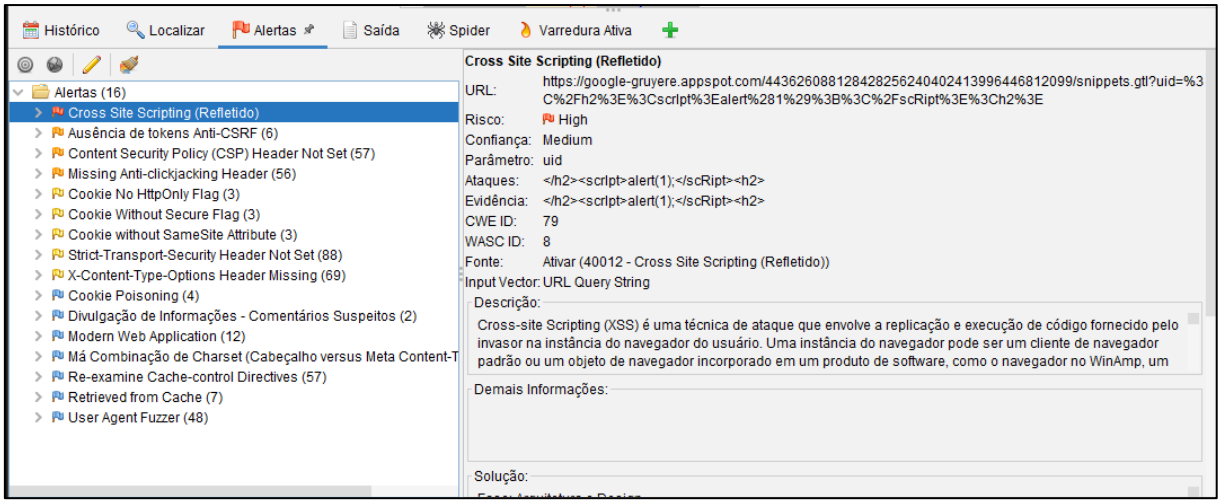


Fonte: Elaborado pelo autor, 2022.

Durante o escaneamento, é possível acompanhar o progresso através da seção inferior da ferramenta. São mostradas informações como o método utilizado para acessar a página, a URL, código HTML de resposta, tamanho do cabeçalho etc.

Ainda na seção inferior, existe uma aba chamada “Alertas” (Figura 6). Nela é possível visualizar os tipos de ameaças que estão sendo detectadas no processo de varredura. É uma aba interessante para melhor entendimento da ameaça, pois conta com algumas informações relevantes sobre ela. No caso desse teste, o tempo gasto para escanear completamente a aplicação Gryuere foi de sete minutos e 56 segundos.

Figura 6 – Aba “Alerta” do Zed Attack Proxy



Fonte: Elaborado pelo autor, 2022.

O próximo passo é a geração do relatório (Figura 7). Apesar da ferramenta permitir uma boa visualização e apresentar opções para tratamento dos problemas dentro da própria interface, ela também permite a exportação de um relatório em diversos formatos, sendo o formato HTML um dos mais intuitivos e fáceis de se analisar. Essa opção gera uma página web contendo um índice para cada tipo de ameaça e suas respectivas informações. Para gerar o relatório, foi utilizado o menu “Relatório” > “Generate Report” localizado na barra de ferramentas da aplicação.

Por fim, é feita a análise manual do relatório para identificar e corrigir os possíveis problemas que foram identificados e filtragem dos falsos positivos.

Figura 7 – Relatório em HTML gerado através do Zed Attack Proxy

Fonte: Elaborado pelo autor, 2022.

4 AVALIAÇÃO

Nesta seção serão apresentadas as métricas para escolha das ferramentas, bem como a descrição das mesmas e algumas características mais especificas de cada uma. Também são discutidas as listas da CWE e OWASP que trazem as vulnerabilidades e fraquezas mais relevantes no atual momento e que estão inclusas no escopo de aplicações web.

4.1 Questões da pesquisa

Considerando o objetivo desse trabalho de identificação das melhores ferramentas de código aberto através de um teste comparativo, foram elaboradas as seguintes questões de pesquisa:

- **As ferramentas escolhidas desempenham a função que foram atribuídas?**
- **O relatório gerado por tais ferramentas é relevante?**
- **O tempo gasto necessário para que as ferramentas finalizem de forma eficiente o teste é palpável e é menor do que uma avaliação manual?**

4.1.1 Seleção das ferramentas

A princípio, foram feitas pesquisas sobre as ferramentas disponíveis para identificação de vulnerabilidades em softwares web, buscando principalmente as ferramentas código aberto e gratuitas. Durante a pesquisa, foram encontradas tanto ferramentas para testes caixa-preta quanto caixa-branca.

Caixa-branca é um teste que envolve a análise da codificação interna e infraestrutura de uma solução de software. Concentra-se principalmente no fortalecimento do salvaguardar, o fluxo de entradas e saídas através da aplicação e a melhoria do design e usabilidade com conhecimento total do código fonte.

Já o caixa-preta é o oposto de caixa-branca. O teste é feito de uma perspectiva externa ou do usuário final. Quando é feito esse tipo de teste, a priori não se tem conhecimento do código fonte do software.

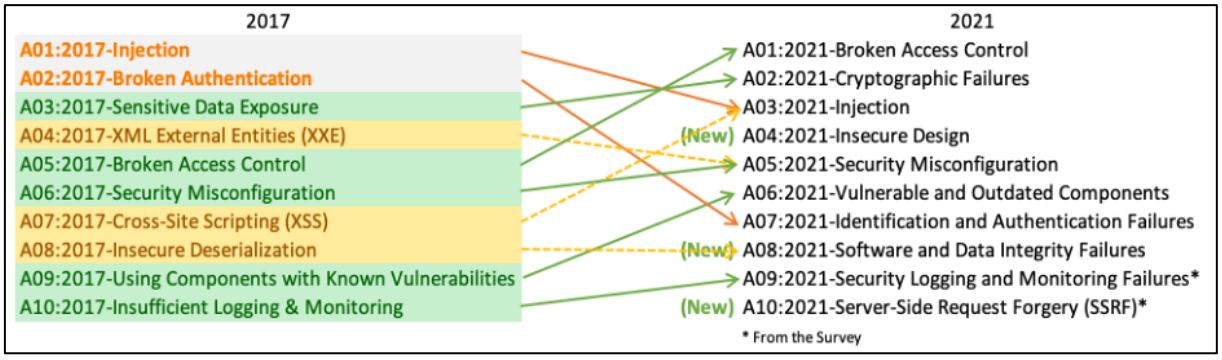
Foram identificadas oito possíveis ferramentas para serem usadas: FindBugs, OWASP Zed Proxy Attack, w3af, Arachni Scanner, Nikto, Wfuzz, Wapiti, SQLMap e NoGoToFail.

4.1.2 OWASP TOP 10: 2021

Após a busca por ferramentas, foi iniciada a pesquisa pelas vulnerabilidades que estão mais presentes nas aplicações no ano de 2021. Como base foi utilizada a lista OWASP top 10: 2021. (OWASP, 2022b)

Na lista de 2021 ocorreram algumas mudanças em relação aos anos anteriores e foram adicionadas três novas categorias. Além disso, quatro categorias sofreram mudanças de nome e escopo como mostrado na figura 8.

Figura 8 – Alteração na ordem das vulnerabilidades e incorporação de novas categorias



Fonte: OWASP, 2021. Disponível em: https://owasp.org/Top10/. Acesso em: 16 fev. 2022.

Dentro de cada categoria listada na imagem acima, existem várias fraquezas. Então a partir daí, foram utilizadas as que estavam contidas tanto na lista da OWASP quanto na lista da CWE, onde a lista da OWASP também faz referência.

4.1.3 CWE TOP 25

O CWE top 25 *Most Dangerous Software Weaknesses* é um projeto que lista as principais fraquezas encontradas em softwares. (CWE, 2022b)

Dentre as vinte e cinco, foram escolhidas as dez primeiras colocadas para uma comparação de onde tais fraquezas se encaixariam quando relacionadas com as

categorias da lista publicada pela OWASP. Com isso, foram identificados alguns fatores relevantes relacionados a estas falhas, como por exemplo qual a tecnologia é mais afetada por essa falha, as linguagens predominantes (ou se independe da linguagem) e o escopo da fraqueza. Estas informações podem ser vistas na tabela 1.

Tabela 1 – Top 10 fraquezas retiradas da lista CWE top 25

Posição	Fraquezas	Independe de linguagem	Afeta aplicações web
1	CWE-787	x	x
2	CWE-79	✓	✓
3	CWE-125	x	x
4	CWE-20	✓	✓
5	CWE-78	✓	✓
6	CWE-89	✓	✓
7	CWE-416	x	x
8	CWE-22	✓	✓
9	CWE-352	✓	✓
10	CWE-434	✓	✓

Fonte: Elaborado pelo autor, 2022.

Sendo o foco do trabalho as aplicações web, foram removidas as fraquezas CWE-787, CWE-125 e CWE-416, pois estas fraquezas afetam linguagens como C e Assembly, linguagens estas que não são amplamente usadas para o desenvolvimento de aplicações web. A relação entre as fraquezas da lista da CWE e as categorias da lista da OWASP pode ser vista na tabela 2.

Tabela 2 – Relacionamento entre as fraquezas CWE e categorias OWASP

Posição CWE	CWE	Categoria OWASP
2	CWE-79	A03-Injection
4	CWE-20	A03-Injection
5	CWE-78	A03-Injection
6	CWE-89	A03-Injection
8	CWE-22	A01-Broken Access Control
9	CWE-352	A01-Broken Access Control
10	CWE-434	A04-Insecure Design

Fonte: Elaborado pelo autor, 2022.

4.1.4 Escolha dos scanners

Para a escolha dos scanners, foram definidos os seguintes critérios: foram atualizadas em 2021 ou mais recentemente, cobrem totalmente ou em parte as vulnerabilidades escolhidas e que possuem ligação com as falhas listadas pelas publicações do CWE e OWASP.

Das nove ferramentas inicialmente listadas, foram escolhidas quatro: OWASP Zed Proxy Attack, Arachni, Wapiti e Nikto.

4.1.4.1 OWASP Zed attack proxy

É um scanner de segurança de aplicativos da web de código aberto. Ele pode ser usado tanto por aqueles que são novos em segurança de aplicativos quanto por testadores de penetração profissionais. É um dos projetos mais ativos do OWASP e recebeu o status de carro-chefe da entidade. É uma das ferramentas mais completas disponíveis até o momento e possui várias ferramentas para procurar vulnerabilidades listadas pela OWASP. Quando usado como um servidor de proxy, permite que o usuário manipule todo o tráfego que passar por ele, incluindo tráfego HTTP.

A aplicação é desenvolvida utilizando majoritariamente a linguagem de programação Java e possui mais de 80 contribuintes no GitHub. Seu código pode ser

visualizado por completo no repositório público e pode ser utilizado em Windows, Linux e MacOS.

Possui características como um servidor de interceptação de proxy, web *spiders* embarcado, escâner automático, escâner passivo, fuzzer embarcado, suporte a *WebSocket* etc.

4.1.4.2 Arachni

É uma aplicação de código aberto desenvolvida majoritariamente em rubi, modular e disponibiliza vários recursos para testadores e administradores a avaliar a segurança de aplicativos da web modernos.

É multiplataforma e cobre uma grande quantidade de casos de uso, desde um simples utilitário de scanner de linha de comando, até uma grade global de scanners de alto desempenho. Possui mais de 15 contribuintes no GitHub e seu código está disponível de forma pública no repositório do projeto. Possui uma versão web que tem menos funções que a versão desktop e é uma opção viável dependendo do objetivo pretendido. Está disponível para Linux e Windows.

A aplicação treina a si mesma enquanto percorre a aplicação e pode detectar mudanças enquanto a análise está sendo feita. Tem suporte a SSL, sistemas de autenticação, manipulação de cookies, suporte a proxy, sistema automático que detecta caminhos que causarão a desconexão de conta (log out) acidental etc.

4.1.4.3 Wapiti

Ferramenta de código aberto que permite auditar sites e aplicações web através de scans black-box. Procura por páginas e formulários com possíveis vulnerabilidades. Uma vez catalogada a lista de URLs, formulários e seus respectivos inputs, utiliza-se de métodos e entradas específicos para testar a falha.

A aplicação está disponível para vários sistemas. Possui mais de 20 contribuintes e seu código está disponível no GitHub do projeto. É escrito majoritariamente em Python e possui funcionalidades como: Restrição de escopo da análise, autenticação, suporte a proxy, proteção contra loops infinitos, controle avançado do *web crawler* etc.

4.1.4.4 Nikto

É uma web scanner de código aberto que realiza testes abrangentes em servidores web para vários itens, incluindo mais de 6700 arquivos potencialmente perigosos. Algumas das verificações feitas pela ferramenta são: Má configuração de software e servidores, problemas em programas e arquivos padrões, arquivos e programas inseguros, programas e servidores com versões antigas de software etc.

Possui poucos contribuintes no GitHub e seu código está disponível para análise em seu repositório público. É escrito majoritariamente em Perl e funciona em qualquer ambiente que dê suporte a linguagem. Suporta funções como SSL, proxies, autenticação de host etc.

4.1.5 Execução dos escaneamentos

Todas as ferramentas possuem uma opção padrão de escaneamento que aplica todos os tipos de análise que eles possuem. Portanto, a fim de manter um padrão, apenas foram adicionados comandos relacionados a autenticação e geração dos relatórios. Então os testes foram executados utilizando todas as funcionalidades de cada ferramenta.

4.1.6 Sites utilizados

Foram utilizados três endereços distintos: Dois que foram desenvolvidos pelo Laboratório NUTES da UEPB(https://portal.regnutes.tk/ e https://web.regnutes.tk/) e o site do Simples Nacional (https://www8.receita.fazenda.gov.br/SimplesNacional/Default.aspx).

A escolha desse site do governo federal se deu pelo teor de relevância da aplicação, pois é uma página que contém informações relevantes, além de um grande volume de acessos.

Os dois sites do NUTES também são páginas que possuem um caráter crítico semelhante ao da aplicação anterior, mas estes se tratando de informações médicas. Essa escolha também adiciona mais um tipo de teste sobre a aplicação do NUTES servindo para identificar possíveis problemas.

4.2 Resultados e Discussões

Nesta seção serão discutidos os resultados obtidos através dos relatórios gerados pelas ferramentas de scanner automático, mostrando as vulnerabilidades encontradas por cada ferramenta e outros detalhes técnicos das análises.

4.2.1 Análises

Algumas ferramentas disponibilizam um ranqueamento da severidade para as vulnerabilidades encontradas, sendo os riscos classificados como: Alto, Médio, Baixo e Informacional.

Esses riscos são calculados levando em conta diversos fatores, como por exemplo a facilidade de utilização da falha, dificuldade de identificação da fraqueza na aplicação, frequência que a vulnerabilidade aparece, o impacto causado pelo seu uso (financeiro, privacidade etc.) entre outros. A partir do cruzamento de todos esses fatores com seus respectivos pesos é gerado o nível de risco. A forma como esse risco é calculado e implementado varia entre as ferramentas.

Por exemplo, a vulnerabilidade SQL-Injection é tida como uma vulnerabilidade de alto risco, pois, aparece com muita frequência em aplicações web, sua utilização é de dificuldade moderada e o impacto que ela pode causar é muito grande por se tratar de uma brecha que permite acesso a dados privados que podem ser de alto sigilo.

Os pontos principais que serão utilizados para fazer a comparação das ferramentas são:

- Qualidade do relatório gerado
- Tempo gasto para finalizar as análises
- Identificação de vulnerabilidades consideradas de alto risco
- Tipo de interface de usuário

É possível visualizar um rápido comparativo entre as ferramentas através da tabela 3, e nos tópicos seguintes uma análise mais detalhada de cada ferramenta. É importante ressaltar que a duração das análises pode variar de acordo com os componentes de hardware da máquina e sistema operacional onde o programa está sendo executado. Todos os testes deste comparativo foram feitos utilizando o mesmo computador e sistema operacional.

Tabela 3 – Comparativo dos principais pontos de análise das ferramentas

Ferramentas	Qualidade do Relatório	Duração das análises			Identificou Vulnerabilidade de alto risco	Interface
		Reg Nutes	Web Nutes	Simples Nacional		
ZAP	Bom	< 1h	< 1h	aprox. 34h	Sim	Gráfica
Arachini	Bom	< 1h	< 1h	aprox. 48h	Sim	CLI
Wapiti	Razoável	< 1h	< 1h	< 1h	Não	CLI
Nikto	Insuficiente	< 1h	<1h	< 1h	Não	CLI

Fonte: Elaborado pelo autor, 2022.

4.2.1.1 Análise da ferramenta OWASP Zed Attack Proxy

As varreduras performadas pelo OWASP Zed Proxy ocorreram utilizando a configuração padrão da aplicação. A interface gráfica da ferramenta facilita bastante a manipulação e visualização do que está sendo feito durante a varredura e análise da aplicação web. É possível acompanhar o progresso e observar em tempo real as ameaças que estão sendo encontradas e analisadas. A ferramenta gera um relatório detalhado nos mais diversos formatos contendo a quantidade de ameaças detectadas e o seu nível de severidade. Também oferece uma descrição dos problemas encontrados além de várias soluções e links úteis para que o usuário consiga entender melhor tal vulnerabilidade.

A busca por vulnerabilidades ocorreu de forma rápida nas aplicações do NUTES por se tratar de uma aplicação com um número pequeno de páginas e demorou cerca de 34 horas para finalizar a análise do site do Simples Nacional. A ferramenta identificou dois tipos de vulnerabilidades de alto risco que estão contidas nas listas CWE e OWASP. É possível visualizar a lista de vulnerabilidades identificadas pela ferramenta através das tabelas 4, 5 e 6.

Tabela 4 – Vulnerabilidades encontradas pela ferramenta OWASP Zed Attack Proxy no site SimplesNacional

Alerta	Ocorrências	Risco
Blind SQL Injection (differential analysis)	7	Alto
Cross-Site Request Forgery (CSRF)	102	Alto
Absence of Anti-CSRF Tokens	21	Médio
Content Security Policy Header Not Set	20	Médio
Missing Anti-clickjacking Header	20	Médio
Cookie No HttpOnly Flag	2	Baixo
Cookie Without Secure Flag	2	Baixo
Cookie without SameSite Attribute	2	Baixo
Cross-Domain JavaScript Source File Inclusion	20	Baixo
Private IP Disclosure	20	Baixo
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	32	Baixo
Timestamp Disclosure - Unix	6873	Baixo
X-AspNet-Version Response Header	20	Baixo
X-Content-Type-Options Header Missing	30	Baixo
Re-examine Cache-control Directives	20	Informacional

Fonte: Elaborado pelo autor, 2022.

Tabela 5 – Vulnerabilidades encontradas pela ferramenta OWASP Zed Proxy no site portal RegNUTES

Alerta	Ocorrências	Risco
Content Security Policy (CSP) Header Not Set	3	Médio
Cross-Domain Misconfiguration	67	Médio
Cross-Domain JavaScript Source File Inclusion	3	Baixo
Timestamp Disclosure - Unix	254	Baixo
X-Content-Type-Options Header Missing	2	Baixo
Information Disclosure - Suspicious Comments	14	Informacional
Re-examine Cache-control Directives	37	Informacional

Fonte: Elaborado pelo autor, 2022.

Tabela 6 – Vulnerabilidades encontradas pela ferramenta OWASP Zed Proxy no site Web RegNUTES

Alerta	Ocorrências	Risco
Content Security Policy (CSP) Header Not Set	1	Médio
Cross-Domain Misconfiguration	43	Médio
Vulnerable JS Library	1	Médio
Cross-Domain JavaScript Source File Inclusion	1	Baixo
Timestamp Disclosure - Unix	347	Baixo
X-Content-Type-Options Header Missing	2	Baixo
Information Disclosure - Suspicious Comments	13	Informacional
Re-examine Cache-control Directives	30	Informacional

Fonte: Elaborado pelo autor, 2022.

4.2.1.2 Análise da ferramenta Wapiti

Com a ferramenta Wapiti, as análises ocorreram de forma rápida. A ferramenta é utilizada através de uma interface de linha de comando, o que pode dificultar a manipulação para um usuário que não está habituado com este ambiente. O feedback durante a análise é precário, sendo possível a visualização da análise apenas após o fim da execução.

Ao fim da execução, é gerado um relatório contendo o nome de todas as vulnerabilidades que foram analisadas e a quantidade encontrada. Também apresenta uma breve descrição de cada problema e sugestões para solucioná-lo. O relatório pode ser gerado em diversos formatos, como HTML, que foi o usado neste estudo. As vulnerabilidades encontradas não estavam contidas na lista da OWASP e nem nas dez primeiras colocações da CWE, e foi a ferramenta que menos identificou vulnerabilidades. Um ponto importante é baixa quantidade de vulnerabilidades encontradas em comparação as outras ferramentas. É possível visualizar a lista de vulnerabilidades identificadas pela ferramenta através das tabelas 7, 8 e 9.

Tabela 7 – Vulnerabilidades encontradas pela ferramenta Wapiti no site SimplesNacional

Alerta	Ocorrências	Risco
Content Security Policy Configuration	1	-
HTTP Secure Headers	4	-
HttpOnly Flag cookie	1	-
Secure Flag cookie	1	-

Fonte: Elaborado pelo autor, 2022.

Tabela 8 – Vulnerabilidades encontradas pela ferramenta Wapiti no site portal RegNUTES

Alerta	Ocorrências	Risco
Content Security Policy Configuration	1	-

Fonte: Elaborado pelo autor, 2022.

Tabela 9 – Vulnerabilidades encontradas pela ferramenta Wapiti no site web RegNUTES

Alerta	Ocorrências	Risco
Content Security Policy Configuration	1	-

Fonte: Elaborado pelo autor, 2022.

4.2.1.3 Análise da ferramenta Nikto

A ferramenta Nikto conseguiu identificar uma quantidade razoável de vulnerabilidades e a varredura se deu de forma rápida. Assim como outras ferramentas de linha de comando, pode afastar alguns usuários pouco habituados a este tipo de interação com o usuário. O relatório pode ser gerado nos mais diversos formatos. A análise não identificou nenhuma das vulnerabilidades consideradas relevantes que foram discutidas anteriormente.

Em relação ao relatório gerado pela ferramenta, este se provou ser o menos detalhado e informativo. A descrição da vulnerabilidade é apenas o seu nome e não oferece nenhuma sugestão ou detalhamento mais aprofundado. Além disso, não existe um contador para cada vulnerabilidade e estas acabam sendo apresentadas de forma aleatória, o que dificulta muito a identificação e contabilização delas.

Outro problema identificado é a inserção de informações pouco relevantes entre uma vulnerabilidade e outra, o que agrava ainda mais o problema de não agregar as vulnerabilidades de mesmo tipo em apenas uma seção. É possível visualizar a lista de vulnerabilidades identificadas pela ferramenta através das tabelas 10, 11 e 12.

Tabela 10 – Vulnerabilidades encontradas pela ferramenta Nikto no site SimplesNacional

Alerta	Ocorrência	Risco
Retrieved x-aspnet-version header: 4.0.30319	1	-
Retrieved x-powered-by header: ARRAY(0x561317e02e68)	1	-
The anti-clickjacking X-Frame-Options header is not present.	1	-
The X-XSS-Protection header is not defined	1	-
The site uses SSL and the Strict-Transport-Security HTTP header is not defined.	1	-
The site uses SSL and Expect-CT header is not present.	1	-
The X-Content-Type-Options header is not set.	1	-
Cookie ARRAffinity created without the secure flag	1	-
Cookie ARRAffinity created without the httponly flag	1	-
Multiple index files found	20	-
Potentially interesting archive/cert file found	200+	-
Allowed HTTP Methods: OPTIONS, TRACE, GET, HEAD, POST	1	-

Fonte: Elaborado pelo autor, 2022.

Tabela 11 – Vulnerabilidades encontradas pela ferramenta Nikto no site Portal RegNUTES

Alerta	Ocorrência	Risco
access-control-allow-origin header	1	-
Uncommon header 'x-dns-prefetch-control' found, with contents: off	1	-
The site uses SSL and Expect-CT header is not present.	1	-
Hostname 'portal.regNUTES.tk' does not match certificate's names: regNUTES.tk	1	-

Fonte: Elaborado pelo autor, 2022.

Tabela 12 - Vulnerabilidades encontradas pela ferramenta Nikto no site web RegNUTES

Alerta	Ocorrência	Risco
access-control-allow-origin header	1	-
Uncommon header 'x-dns-prefetch-control' found, with contents: off	1	-
The site uses SSL and Expect-CT header is not present.	1	-
Hostname 'web.regNUTES.tk' does not match certificate's names: regNUTES.tk	1	-

Fonte: Elaborado pelo autor, 2022.

4.2.1.4 Análise da ferramenta Arachni

Essa ferramenta de análise de vulnerabilidades permite que as análises sejam feitas através de uma interface gráfica diretamente pelo navegador ou através da linha de comando. A interface gráfica não é muito intuitiva e oferece menos opções em relação a interface de linha de comando. Neste estudo foi utilizado a interface em

linha de comando, onde é possível acompanhar o que está sendo analisado, mesmo que de forma não muito intuitiva.

O relatório é gerado em uma extensão própria do programa e para fazer a leitura do relatório é preciso utilizar um dos módulos do programa, o arachni_report. Este módulo decodifica o conteúdo gerado e permite que o conteúdo seja aberto em um editor de texto comum.

Arachni foi a ferramenta que melhor performou dentre as quatro analisadas. O que mais chamou a atenção foi a duração do *scan* que levou mais de 48 horas para ser finalizado. Este tempo foi resultado, em parte, pelo uso do *scan* padrão da ferramenta, pois através de algumas configurações é possível reduzir esse tempo. Assim como o Nikto, o Arachni não aglutina as vulnerabilidades em uma única seção, o que torna difícil a contagem total do número de vulnerabilidades iguais. Apesar disso, o relatório é um dos mais detalhados, provendo uma ampla explicação do problema e formas de solucioná-lo. Também informa o nível de risco da vulnerabilidade encontrada.

O Arachni identificou vulnerabilidades tidas como de alto risco, entre elas duas que estão contidas nas listas do OWASP e CWE. É possível visualizar a lista de vulnerabilidades identificadas pela ferramenta através das tabelas 13, 14 e 15.

Tabela 13 - Vulnerabilidades encontradas pela ferramenta Arachni no site Simples nacional

Alerta	Ocorrências	Risco
Blind SQL Injection (differential analysis)	9	Alto
Cross-Site Request Forgery (CSRF)	95	Alto
Missing 'Strict-Transport-Security' header	1	Médio
Common directory	2	Médio
Unencrypted password form	4	Médio
Private IP address disclosure	44	Baixo
Missing 'X-Frame-Options' header	1	Baixo
Password field with auto-complete	2	Baixo
Interesting response	25	Informacional
Captcha protected form	4	Informacional
Insecure cookie	5	Informacional
Cookie set for parent domain	1	Informacional
HttpOnly cookie	1	Informacional
Allowed HTTP methods	1	Informacional

Fonte: Elaborado pelo autor, 2022.

Tabela 14 - Vulnerabilidades encontradas pela ferramenta Arachni no site Portal RegNUTES

Alerta	Ocorrências	Risco
Common directory	1	Médio
Insecure 'Access-Control-Allow-Origin' header	1	Baixo

Fonte: Elaborado pelo autor, 2022.

Tabela 15 – Vulnerabilidades encontradas pela ferramenta Arachni no site web RegNUTES

Alerta	Ocorrências	Risco
Common directory	1	Médio
Insecure 'Access-Control-Allow-Origin' header	1	Baixo
Password field with auto-complete	1	Baixo

Fonte: Elaborado pelo autor, 2022.

4.3 Ameaças a validade

Como foi discutido no tópico “escolha das ferramentas”, as ferramentas escolhidas são destinadas a encontrar vulnerabilidades em aplicações web e servidores. Assim, assume-se que são válidas para o comparativo.

Levando em conta o escopo do trabalho, no mundo do teste de software existe o fator de falsos positivos. Falso positivo é um termo utilizado para indiciar um elemento que foi elencado como potencialmente malicioso de forma errônea. Ou seja, este item não apresenta ameaça alguma. É um elemento bastante comum em análises de antivírus, por exemplo. A possibilidade de as ferramentas acusarem falsos positivos vai além do escopo de tratamento do trabalho, já que ele não engloba o teste das ameaças.

A validade da comparação se dará pelos resultados obtidos através do escaneamento das aplicações web, através do relatório gerado pelas ferramentas e identificação das vulnerabilidades de alto risco.

Através dos relatórios será possível verificar as vulnerabilidades encontradas, a forma como são apresentadas, a severidade da vulnerabilidade, e se a ferramenta apresenta alguma instrução de como solucioná-la.

5 CONCLUSÃO

Com tudo que foi exposto, fica nítida a necessidade do uso das mais diversas ferramentas para realizar o teste de software das formas mais variadas possíveis. O uso das ferramentas de scanners para a identificação de possíveis vulnerabilidades se prova útil como um reforço na segurança do software. A utilização de mais de uma ferramenta também se prova de extrema importância, pois existem implementações, abordagens e funções diferentes na produção desses softwares onde um scanner complementa o outro e podem apontar diferentes problemas no software no momento da análise como observado na avaliação das ferramentas na seção 4.

Se tratando de vulnerabilidades de risco baixo, médio e informacional todas as ferramentas obtiveram um desempenho satisfatório. Um destaque para o problema do uso de bibliotecas desatualizadas que foram identificadas durante a varredura, uma vez que softwares desatualizados podem carregar consigo vários problemas que podem ter sido solucionados em versões posteriores.

O tempo gasto pelas ferramentas também foi satisfatório. A análise manual destas aplicações levaria um tempo bem maior do que os observados nas análises, fazendo assim o uso das ferramentas de grande ajuda.

Arachni e OWASP Zed attack proxy se provaram as melhores ferramentas por conta dos seus relatórios detalhados, facilidade de uso e sugestões poderosas de como resolver as falhas identificadas. Foram as únicas ferramentas a identificar vulnerabilidades de alto risco dentre os softwares testados.

Outro ponto importante a se destacar é a diferença de vulnerabilidades encontradas em um software desenvolvido pelo NUTES e o software desenvolvido pelo governo. Porém, também é importante destacar a proporção e diferença de tamanho entre as aplicações. Onde o software produzido pela universidade foi desenvolvido seguindo boas práticas de programação e passou por vários tipos de testes que foram discutidos durante este trabalho.

REFERÊNCIAS

2021 CWE Top 25 Most Dangerous Software Weaknesses. *In:* **CWE**. Disponível em: https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html. Acesso em: 23 de fev. de 2022b.

2022 Security report: Software vendors saw 146% increase in cyber attacks in 2021, marking largest year-on-year growth. *In:* **CheckPoint**. Disponível em: https://research.checkpoint.com/2022/2022-security-report-software-vendors-saw-146-increase-in-cyber-attacks-in-2021-marking-largest-year-on-year-growth/. Acesso em: 11 de mar. de 2022.

8 Vulnerable Web Applications to Practice Hacking Legally. Ashlin Jenifa, 2021. *In:* **GeekFlare**. Disponível em: https://geekflare.com/practice-hacking-legally/. Acesso em: 17 de mar. de 2022.

About CWE. *In:* **CWE**. Disponível em: https://cwe.mitre.org/about/index.html. Acesso em: 12 de mar. de 2022a.

About the CVE program. *In:* **CVE**. Disponível em: https://www.cve.org/About/Overview. Acesso em: 12 de Mar. de 2022a.

Arachni web scanner. *In:* **GitHub/Arachni**. Disponível em: https://github.com/Arachni/arachni. Acesso em: 16 de março de 2022.

Automatic SQL injection and database take over tool. *In:* **SQLMap**. Disponível em: https://sqlmap.org/. Acesso em: 16 de março de 2022.

CVE. Glossary. *In:* **CVE**. Disponível em: https://www.cve.org/ResourcesSupport/Glossary#. Acesso em: 18 de fev. de 2022b.

CWE-125: Out-of-bounds Read. *In:* **CWE**. Disponível em: https://cwe.mitre.org/data/definitions/125.html. Acesso em: 23 de fev. de 2022e.

CWE-20: Improper Input Validation. *In:* **CWE**. Disponível em: https://cwe.mitre.org/data/definitions/20.html. Acesso em 23 de fev. de 2022f.

CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal'). *In:* **CWE**. Disponível em: https://cwe.mitre.org/data/definitions/22.html. Acesso em: 24 de fev. de 2022j.

CWE-352: Cross-Site Request Forgery (CSRF). *In:* **CWE**. Disponível em: https://cwe.mitre.org/data/definitions/352.html. Acesso em: 24 de fev. de 2022k.

CWE-416: Use After Free. *In:* **CWE**. Disponível em: https://cwe.mitre.org/data/definitions/416.html. Acesso em: 24 de fev. de 2022i.

CWE-434: Unrestricted Upload of File with Dangerous Type. *In:* **CWE**. Disponível em: https://cwe.mitre.org/data/definitions/434.html. Acesso em: 24 de fev. de 2022l.

CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection'). *In:* **CWE**. Disponível em: https://cwe.mitre.org/data/definitions/78.html. Acesso em: 23 de fev. de 2022g.

CWE-787: Out-of-bounds Write. *In:* **CWE**. Disponível em: https://cwe.mitre.org/data/definitions/787.html. Acesso em: 23 de fev. de 2022c.

CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting'). *In:* **CWE***.* Disponível em: https://cwe.mitre.org/data/definitions/79.html. Acesso em: 23 de fev. de 2022d.

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection'). *In:* **CWE**. Disponível em: https://cwe.mitre.org/data/definitions/89.html. Acesso em: 24 de fev. de 2022h.

DELAMARO, MARCIO EDUARDO; MALDONADO, JOSÉ CARLOS; JINO, MARIO. **Introdução ao teste de software**. 2.ed. Elsvier, 2016.

FELDERER, MICHAEL; BUCHLER, MATTHIAS; MARTIN, JOHNS; BRUCKER, ACHIM D.; BREU, RUTH; PRETSCHNER, ALEXANDER**. Security Testing: A Survey. In Advances in Computers,** 101, p. 1-51, 2016.

FindBugs™ - Find Bugs in Java Programs. *In:* **FindBugs**. Disponível em: https://findbugs.sourceforge.net/. Acesso em: 16 de março de 2022.

HABIB, Eduardo. Teste de segurança: Testes de segurança em aplicações web. **Engenharia de Software Magazine**. 24.ed. DevMedia, 2013.

IEEE. **IEEE Std 1044-2009, IEEE Standard Classification of Software Anomalies**. New York: IEEE, 2009.

MOLYNEAUX, Ian. **The Art of Application Performance Testing**. O'Reilly Media, 2014. p. 38-39.

Nikto Web Server Scanner. *In:* **Nikto**. Disponível em: https://cirt.net/nikto2. Acesso em: 16 de março de 2022.

No Go To Fail. *In:* **GitHub/NoGoToFail**. Disponível em: https://github.com/google/nogotofail. Acesso em: 16 de março de 2022.

OWASP Top 10:2021. *In:* **OWASP**. Disponível em: https://owasp.org/Top10/A00_2021_Introduction/. Acesso em: 11 de fev. de 2022b.

OWASP Vulnerable Web Application Directory. *In:* **OWASP**. Disponível em: https://owasp.org/www-project-vulnerable-web-applications-directory/#. Acesso em: 21 de mar. de 2022a.

OWASP Zed Attack Proxy (ZAP). *In:* **ZapProxy**. Disponível em: https://www.zaproxy.org/. Acesso em: 16 de março de 2022b.

SOMMERVILE, IAN. **Engenharia de Software**. 10. ed. Pearson Universidades, 2011.

Spider. *In:* **ZapProxy**. Disponível em: https://www.zaproxy.org/docs/desktop/start/features/spider/. Acesso em: 15 de abr. de 2022a.

w3af – Open Source Web Application Security Scanner. *In:* **w3af**. Disponível em: https://w3af.org/. Acesso em: 16 de março de 2022.

WEIDMAN, Georgia. **Penetration Testing: A Hands-On Introduction to Hacking**. No Starch Press, 2014.

Wfuzz: The Web fuzzer. *In:* **Wfuzz**. Disponível em: https://wfuzz.readthedocs.io/en/latest/. Acesso em: 16 de março de 2022.

What are the most secure programming languages? *In:* **WhiteSource**. Disponível em: https://www.whitesourcesoftware.com/most-secure-programming-languages/. Acesso em: 15 de mar. de 2022.

What is Kali Linux? *In:* **Kali**. Disponível em: https://www.kali.org/docs/introduction/what-is-kali-linux/. Acesso em 28 de fev. de 2022.