ScanSentinel - Análise de vulnerabilidades em aplicações web, com geração automática de relatórios

Claython da Silva
Curso de Ciência da Computação
Universidade Franciscana
CEP 97010-032 – Santa Maria, RS, Brasil
claython.silva@ufn.edu.br

Sylvio André Garcia Vieira
Curso de Sistemas de Informação
Universidade Franciscana
CEP 97010-032 – Santa Maria, RS, Brasil
sylvio@ufn.edu.br

***Resumo*—Os servidores Web contém dados sensíveis que, caso expostos, poderiam comprometer tanto o anonimato quanto a segurança dos sistemas. As aplicações hospedadas neles permitiam a ocorrência de ataques cibernéticos capazes de revelar essas informações. Este trabalho teve como objetivo avaliar possíveis vulnerabilidades em sistemas web e gerar automaticamente relatórios destinados aos seus administradores. Para isso, foram utilizadas a linguagem Python integrada às ferramentas e bibliotecas ZAP, Requests, SQLMap, Mutillidae e Flask. Ao final do projeto, os relatórios gerados demonstraram-se eficazes ao fornecer informações claras e estruturadas, permitindo ações mais assertivas por parte dos administradores dos sistemas avaliados.**

***Palavras-chave*-Segurança da Informação; Vulnerabilidades Web; Automação de Testes; OWASP ZAP; SQLMap**

I. Introdução

As aplicações web são alvos frequentes de ataques cibernéticos, expondo dados sensíveis e comprometendo a integridade dos sistemas. Estudos revelam um aumento significativo desses ataques, no Brasil, ocorreram 357.422 ataques no segundo semestre de 2023, um aumento de 8,8% em relação ao primeiro semestre e que vem aumentando com o passar dos anos [1].

Diante desse cenário, a segurança digital tornou-se uma preocupação central para empresas que desenvolvem e mantêm aplicações web [2]. A implementação de medidas de proteção é essencial para preservar a confidencialidade e integridade, garantindo a confiança e segurança dos usuários.

Criminosos virtuais realizam varreduras de centenas de milhares de computadores na internet em busca de vulnerabilidades e aplicam, quando encontram, ataques relacionados às falhas encontradas, como Injeção de SQL, que burla sistemas de logins para obter usuários e senhas, ou *Cross-site scripting*, que faz a injeção de código malicioso no site, ou até mesmo DDoS (*Distributed Deny of Service*) que é a queda total do servidor web.

A análise de vulnerabilidades em aplicações web é uma prática fundamental para identificar e corrigir possíveis falhas de segurança. A automatização desse processo, aliada à geração de relatórios detalhados, permite uma resposta mais ágil e eficaz às ameaças, contribuindo para a robustez das aplicações e a mitigação de riscos cibernéticos.

A. Objetivo Geral

Realizar análise de vulnerabilidades web com geração automática de relatórios por meio de uma ferramenta computacional.

B. Objetivos Específicos

- Identificar Vulnerabilidades em aplicações WEB - Analisar falhas que ocorrem com frequência, como injeção SQL, XSS e ataques DDoS.
- Automatizar a detecção de vulnerabilidades - Desenvolver um sistema para realizar os testes segurança.
- Gerar relatórios detalhados das vulnerabilidades encontradas - Estruturar um relatório claro com base nas vulnerabilidades encontradas para visualização dos dados.
- Avaliar o impacto das vulnerabilidades na segurança da aplicação – Mensurar o risco de cada falha identificada para o funcionamento da aplicação e a proteção dos dados.
- Propor soluções para mitigar as vulnerabilidades encontradas – Fornecer diretrizes para melhorar a segurança das aplicações analisadas.

II. Justificativa

A segurança em aplicações web é um fator crítico, especialmente com o crescimento do número de ataques cibernéticos [2]. Segundo o relatório de segurança publicado pela empresa Cloudflare, em 2024 [2], muitas empresas e desenvolvedores não realizam testes periódicos de segurança, deixando sistemas expostos a invasões. Este trabalho se torna relevante pois permite a identificação de falhas que comprometem a integridade de sistemas, reduzindo os riscos e contribuindo para a manutenção e aprimoramento desses sistemas.

III. Referencial Teórico

Nesta seção serão definidos os conceitos necessários para a compreensão deste trabalho.

A. Falha de segurança

Uma falha de segurança é uma vulnerabilidade em um sistema, aplicativo, rede ou dispositivo que pode ser explorada por um atacante para comprometer a confidencialidade, integridade ou disponibilidade dos dados e recursos.

Alazmi e Leon [3] advertem que as aplicações web tornaram-se mais suscetíveis a riscos de segurança. Com isso, a avaliação de vulnerabilidade de aplicações web é fundamental para segurança e proteção contra ameaças e ataques cibernéticos.

Servidores Web são servidores responsáveis por aceitar requisições HTTP e HTTPS de clientes web e entregar respostas desses protocolos, geralmente em forma de páginas web que tenham conteúdos estáticos (como textos e imagens) e/ou dinâmicos. O conteúdo disponibilizado pelo servidor é visto pelo cliente por meio de navegadores tais como Microsoft Edge, Firefox ou Google Chrome [4].

De acordo com Silva et al. [5], a maioria das técnicas utilizadas para atacar as páginas de sites são as mesmas desde tempos passados até os atuais. Isso ocorre, na maioria das vezes, pelo fato de que muitos desenvolvedores não possuem o conhecimento sobre diversos tipos de ataques. Além disso, ainda de acordo com Silva et al. [5], outro motivo é o fato de que, devido a alta demanda de projetos, o site é disponibilizado rapidamente para o usuário, sem a busca e a análise de possíveis falhas ou vulnerabilidades.

No contexto da segurança de aplicações web, métricas são indicadores quantitativos utilizados para medir, avaliar e acompanhar aspectos relacionados à detecção de vulnerabilidades, desempenho de ferramentas, impacto de ataques e efetividade de medidas de proteção [4].

Dentre as métricas mais habituais, de acordo com Assunção [4] tem-se que dar atenção especial para:

- Precisão - A precisão refere-se à proporção de vulnerabilidades corretamente identificadas em relação ao total de alertas gerados pela ferramenta.
- Falso positivo - Os falsos positivos ocorrem quando a ferramenta indica uma vulnerabilidade que, na verdade, não existe. Ou seja, a ferramenta gera um alerta falso para algo que não é uma falha real.
- Falso negativo - Os falsos negativos acontecem quando a ferramenta deixa de identificar uma falha real, ou seja, não detecta uma vulnerabilidade que realmente existe.
- Tempo de execução - O tempo de execução mede quanto tempo a ferramenta leva para concluir uma análise e detectar as vulnerabilidades no sistema. Essa métrica é importante, especialmente em ambientes de grande escala, onde a rapidez da análise pode ser um fator crítico.
- Desempenho geral - O desempenho geral avalia a combinação de todas as métricas gerais citadas, levando em consideração tanto a precisão quanto a eficiência da ferramenta.

B. OWASP ZAP (Zed Attack Proxy)

É uma ferramenta gratuita e de código aberto desenvolvida pela comunidade OWASP para testes de segurança em aplicações web [3].

C. SQLMap

O SQLMap é uma ferramenta automatizada de testes de segurança focada na identificação e exploração de vulnerabilidades de injeção SQL em aplicações web. Ela realiza varreduras capazes de detectar diferentes tipos de falhas em parâmetros, formulários e URLs, além de possibilitar a extração de dados, identificação de bancos de dados e execução de comandos avançados quando a vulnerabilidade é confirmada.

D. Unified Modeling Language(UML)

A UML é uma forma gráfica de demonstrar alguns parâmetros do sistema, que neste projeto foram utilizados:

1) Diagrama de Caso de uso: O diagrama de caso de uso UML é uma ferramenta visual utilizada para representar as interações entre um sistema e os seus usuários (atores)

2) Diagrama de Atividades: O diagrama de atividades UML é uma ferramenta visual que representa o fluxo de atividades dentro de um sistema, processo ou software.

E. Linguagem Python

De acordo com Palharini e Zamberlan [6] Python e uma linguagem de programação de alto nível, gratuita e multiplataforma. É também uma das linguagens mais demandada por empresas e organizações do mundo inteiro por ser simples de aprender e aplicar, tem uma grande comunidade engajada e conta com muitas bibliotecas úteis para desenvolvimento de muitas ferramentas.

F. Mutillidae

O framework Mutillidae permite simular as vulnerabilidades em ambientes controlados para fins de teste. O Mutillidae é um projeto da OWASP, disponível para download e deve ser instalado no ambiente local juntamente com o servidor Apache e o banco de dados MySqL. Ele fornece um ambiente web completo com diversas páginas que possuem falhas propositalmente inseridas, para mostrar o risco dessas vulnerabilidades em um ambiente real [4].

G. Análise estática

Análise estática é um tipo de análise de código que é feita sem executar o programa. Ou seja, ela examina o código-fonte, bytecode ou binário de um software em busca de erros, vulnerabilidades ou padrões suspeitos, tudo isso apenas lendo o código, sem rodá-lo.

H. Requests

Conforme Plotze et al. [7], Requests é uma biblioteca Python que permite enviar solicitações HTTP/1.1 com facilidade. A Requests simplifica o processo de fazer solicitações web e oferece uma série de recursos úteis, como gerenciamento de sessões, cookies e autenticação básica. Com a biblioteca Requests, os desenvolvedores podem facilmente recuperar dados de APIs e sites da web sem precisar lidar com a complexidade do protocolo HTTP. Além disso, a biblioteca possui uma documentação clara e bem organizada, o que torna fácil sua compreensão.

I. Flask

Flask é um framework para desenvolvimento de aplicações web em Python. Ele foi projetado para ser flexível e eficiente de expandir. Diferentemente de frameworks mais robustos, como Django, o Flask oferece especialmente os componentes essenciais, como roteamento, tratamento de requisições e servidor de desenvolvimento. Por sua modularidade, é amplamente utilizado para criar APIs, sistemas pequenos e médios e serviços web de forma eficiente.

J. Scrum

De acordo com Palharini [6], Dentre as metodologias ágeis que são aplicadas atualmente, uma das principais é a Scrum. Essa metodologia tem por prioridade o desenvolvimento de um produto/serviço de modo menos complexo, mais eficiente e focado em resultados, dividido em sprints (período de tempo fixo e curto), normalmente de 7 a 15 dias, onde, no final de cada sprint é realizada uma retrospectiva para analisar e discutir o que pode ser melhorado no proximo.

K. Github

Segundo Palharini [6] GitHub é uma plataforma online de hospedagem de código-fonte. O GitHub permite que desenvolvedores e equipes de desenvolvimento compartilhem, versionem e colaborem em projetos de maneira eficiente.

L. SQL Injection

SQL Injection é um tipo de ataque que permite a alteração de forma maliciosa dos comandos SQL que são enviados ao banco de dados através de uma URL. O mesmo pode ser realizado por alguma entrada de informação em alguma página que tenha comunicação com o banco de dados, podendo inserir, alterar, consultar e excluir informações armazenadas [5].

M. Cross-Site Scripting

O ataque Cross-site Scripting (XSS) permite inserir scripts em uma parte da página sem permissão em uma zona privilegiada para ser executado quando acessado [5].

N. DDoS

DDoS (Distributed Denial of Service) é um tipo de ataque em que múltiplos dispositivos, geralmente computadores comprometidos, enviam um grande volume de requisições a um servidor, aplicação ou rede, com o objetivo de sobrecarregar seus recursos e torná-lo indisponível para usuários legítimos.

IV. Trabalhos Correlatos

A. Um Web Crawler para Projeções e Análise de Vulnerabilidades de Segurança e Consistência Estrutural de Páginas Web

O trabalho de Silva et al. [5] tem por objetivo desenvolver um *Crawler*, que é uma ferramenta automatizada usada para explorar e analisar páginas web de maneira sistemática. O foco estava em projeções (previsões) e análise de vulnerabilidades de segurança. O estudo descreve o uso de um crawler automatizado, que é um software que navega por sites coletando informações e verificando sua segurança. Esses crawlers podem realizar uma varredura automática para detectar vulnerabilidades, como injeção de SQL, falhas em autenticação e criptografia, que são as mais comuns de serem encontradas. O desenvolvimento foi realizado em Python, linguagem escolhida por sua simplicidade, legibilidade e ampla disponibilidade de bibliotecas voltadas à manipulação de dados web. Essa escolha também facilita a prototipação rápida e a integração com outras ferramentas de análise, contribuindo para uma implementação mais ágil e eficiente do sistema que foi proposto.

O resultado do estudo foi a implementação de uma ferramenta útil para detectar falhas de segurança e inconsistências estruturais nas páginas web. A ferramenta aumentou a confiabilidade das páginas verificadas ao identificar potenciais riscos de segurança. A pesquisa ajudou a mostrar que a análise automatizada pode ser uma forma eficaz e eficiente de validar grandes volumes de sites sem depender exclusivamente de testes manuais.

B. Análise de Eficiência na Detecção de Vulnerabilidades em Ambientes WEB com o uso de Ferramentas de Código Aberto

- **Objetivo:** Realizar uma análise comparativa entre cinco ferramentas de código aberto para testes de penetração, com foco na identificação de vulnerabilidades em um ambiente controlado. A pesquisa se baseou nas dez categorias de riscos mais frequentes em aplicações web, conforme o Top Ten da OWASP. O estudo teve como objetivo avaliar a capacidade das ferramentas em detectar falhas de segurança, comparando aspectos como precisão, taxa de falsos positivos e negativos, tempo de execução e desempenho geral.
- **Método:** Cinco ferramentas foram configuradas e utilizadas para realizar uma varredura em um ambiente controlado, baseado no framework Mutillidae, que contém

83 falhas divididas nas categorias de risco da OWASP. As ferramentas utilizadas foram:

- OWASP ZAP: Ferramenta automatizada para detecção de vulnerabilidades em aplicativos web, focada em segurança e testes de penetração.
- SQLMAP: Especializada em detecção e exploração de vulnerabilidades de injeção de SQL.
- W3AF: Framework para testes de segurança e auditoria, capaz de detectar uma ampla gama de vulnerabilidades.
- SKIPFISH: Ferramenta de varredura e análise de falhas de segurança em aplicações web.
- NIKTO: Scanner de vulnerabilidades em servidores web, focado em falhas de configuração.

As ferramentas foram executadas com configurações padrão, realizando *Crawling* e análise de vulnerabilidades no ambiente de testes. Em alguns casos, foi realizada análise manual para confirmar falhas e testar páginas não detectadas automaticamente.

- **Resultados:**
  - **W3AF:** Melhor taxa de precisão, detectando 61 das 83 falhas (73%).
  - **NIKTO:** Segunda colocada, com 50 falhas detectadas corretamente.
  - **OWASP ZAP:** Maior número de falsos positivos (72), mas identificou corretamente 47 falhas.
  - **SKIPFISH:** Baixa precisão, detectando apenas 31% das falhas.
  - **SQLMAP:** Excelente desempenho na detecção de falhas da categoria A1 (Injeção de Código), mas falhou em outras categorias.

  Os autores concluíram que o W3AF foi a melhor ferramenta em termos gerais. O SQLMAP foi eficaz em detectar injeções de SQL, mas limitado a essa categoria. O OWASP ZAP e NIKTO foram bons em algumas áreas, mas apresentaram falsos positivos. O SKIPFISH, embora eficaz em algumas categorias, teve desempenho geral abaixo das expectativas. A pesquisa sugere melhorias nas ferramentas de código aberto para reduzir falsos positivos e melhorar a precisão na detecção de vulnerabilidades.

C. Segurança Cibernética em Roteadores Wi-Fi: abordagem automatizada para Coleta e Análise de Firmware

O objetivo do estudo foi desenvolver uma ferramenta automatizada para a coleta e análise estática de firmwares de roteadores Wi-Fi, com o intuito de identificar vulnerabilidades que possam comprometer a segurança desses dispositivos. A proposta visa otimizar o processo de detecção de falhas, evitando inspeções manuais demoradas, e proporcionando uma análise mais rápida, eficiente e escalável. Além disso, a metodologia foi projetada para ser integrada ao framework SCREEN, um sistema voltado à coleta, armazenamento e análise de grandes volumes de dados relacionados à segurança de dispositivos embarcados, o que permite uma abordagem em larga escala para a identificação e mitigação de riscos.

Para a análise de segurança, foram utilizadas duas ferramentas de análise estática:

Semgrep: Ferramenta de código aberto que permite encontrar padrões de código perigosos ou não conformes com boas práticas de segurança. É leve, rápida e permite a criação de regras personalizadas.

CodeQL: Ferramenta da GitHub que transforma o código-fonte em um banco de dados consultável, permitindo realizar buscas por vulnerabilidades com base em consultas lógicas. É usada por grandes projetos de segurança por sua precisão na identificação de falhas complexas.

O trabalho foi realizado com o uso da linguagem Python, escolhida por sua simplicidade e grande variedade de bibliotecas voltadas à automação de tarefas e análise de dados.Para automatizar a coleta de dados nos sites dos fabricantes, foi utilizada a biblioteca Scrapy, por sua eficiência na construção de web crawlers. Foram coletadas 262 imagens de firmware(códigos-fontes) de 10 fabricantes diferentes, sendo que 141 dessas imagens tiveram os sistemas de arquivos extraídos com sucesso.

Os resultados mostraram que a ferramenta foi eficaz na identificação automatizada de informações relevantes à segurança, evidenciando falhas como arquivos expostos, credenciais embutidas e diretórios acessíveis, que poderiam ser explorados por atacantes, destacando-se que: O Semgrep identificou 7.257 possíveis vulnerabilidades nos firmwares analisados. O CodeQL detectou 3.892 potenciais falhas de segurança. Esses dados evidenciam que a ferramenta foi capaz de realizar uma análise profunda, contribuindo diretamente para a detecção de problemas críticos em dispositivos de rede.

D. Conclusões dos trabalhos correlatos

O trabalho de Silva et al. [5] contribui para este projeto ao demonstrar a aplicação de ferramentas automatizadas para análise de vulnerabilidades em páginas web. A abordagem apresentada reforça a viabilidade e a eficácia da automatização na detecção de falhas de segurança, como injeção de SQL e falhas de autenticação, alinhando-se ao objetivo deste trabalho de realizar testes automatizados e gerar relatórios coerentes. Além disso, serve como base metodológica para a estruturação do sistema proposto, evidenciando os benefícios da análise sistemática de grandes volumes de páginas sem a necessidade de testes manuais extensivos.

O trabalho de Assunção [4] contribui para o projeto ao apresentar uma avaliação comparativa entre cinco ferramentas de código aberto voltadas à detecção de vulnerabilidades em aplicações web. A pesquisa reforça a importância do uso de soluções automatizadas no processo de identificação de falhas como injeção de SQL, XSS e CSRF, utilizando

ferramentas desenvolvidas majoritariamente em linguagens como Python, C e Perl. Além disso, o estudo fornece métricas relevantes como precisão, tempo de execução e taxa de falsos positivos, que podem ser adotadas neste projeto como base para validação e comparação de resultados. A aplicação prática dessas ferramentas em um ambiente controlado (Mutillidae) também contribui metodologicamente para a estruturação dos testes propostos.

O trabalho de Bertolino et al. [8] contribuiu para uma melhor compreensão dos conceitos de automação, coleta de dados e análise de vulnerabilidades, auxiliando no desenvolvimento deste projeto. Embora não tenham sido utilizadas diretamente as ferramentas Semgrep, CodeQL e Scrapy, o estudo de suas funcionalidades auxiliou na aplicação prática das ferramentas utilizadas no projeto para automatizar os testes de segurança.

V. Metodologia

Este trabalho foi desenvolvido seguindo a metodologia Scrum, que determinou, não somente a frequência das reuniões, como também os temas que foram abordados nelas.

Para a implementação do sistema, utilizou-se a linguagem Python, devido à sua simplicidade e familiaridade, e pela grande quantidade de bibliotecas voltadas para análise e segurança. Essa escolha permitiu maior produtividade e utilização eficiente das ferramentas e bibliotecas.

A biblioteca *Requests* foi utilizada para realizar requisições HTTP e inspecionar respostas do servidor. Ela foi escolhida por sua estabilidade e praticidade em lidar com diferentes métodos de requisição, o que tornou o processo de validação de comportamento do servidor rápido e confiável.

A geração do relatório (Figura 4) foi feita com a biblioteca *ReportLab*, que permite a configuração e estruturação do arquivo relatório em PDF, selecionada por permitir a criação de PDFs personalizados de forma organizada. Sua flexibilidade possibilitou a estruturação de relatórios detalhados, incluindo textos, tabelas, seções e destaque visual para diferentes níveis de vulnerabilidades em cores.

A interface web, que pode ser observada na Figura 3, foi desenvolvida com o framework *Flask*, que foi escolhido por permitir construir uma aplicação web funcional, com baixo esforço computacional para funcionamento da página, facilitando a integração entre a camada de apresentação e as configurações internas da análise de segurança.

No processo de testes, utilizou-se o ambiente vulnerável OWASP Mutillidae, que foi escolhido por oferecer um conjunto variado de falhas propositalmente inseridas, permitindo validar a eficácia do sistema em condições controladas. Sua ampla utilização na área de segurança reforça sua confiabilidade como ambiente de teste.

Para a detecção automatizada de vulnerabilidades, foram utilizadas as ferramentas OWASP ZAP e SQLMap. O ZAP foi escolhido por ser uma das ferramentas mais completas e reconhecidas para análise de segurança em aplicações web, permitindo identificar falhas como XSS e problemas de configuração. Já o SQLMap foi selecionado por sua especialização em detecção e exploração de injeções SQL, possibilitando a avaliação precisa desse tipo de vulnerabilidade.

O sistema foi desenvolvido como uma aplicação web e seu projeto/código disponibilizado no GitHub (https://github.com/ClaythonSilva/ScanSentinel_TCC2), e testado localmente para garantir que todos os recursos, funcionalidades e relatórios funcionassem corretamente em um ambiente controlado. Dessa forma foi possível validar o comportamento da aplicação antes de sua possível utilização em cenários reais.

Para organização do sistema, foram definidos os requisitos Funcionais (RF) e Não Funcionais (RNF):

A. Requisitos Funcionais

RF1: **Realizar Varreduras em Aplicações Web** — O sistema é capaz de realizar varreduras em aplicações web, permitindo a análise de possíveis vulnerabilidades. Essa funcionalidade pode ser visualizada no diagrama de caso de uso (Figura 1) "Executar análise de vulnerabilidades"(Figura 1 Item C), nos quais o usuário pode solicitar análises automáticas sobre uma aplicação informada.

RF2: **Detecção de Vulnerabilidades** — O sistema é capaz de detectar vulnerabilidades como Cross-Site Scripting (XSS) e SQL Injection. Este requisito foi aplicado de forma indireta no diagrama, por meio da interação do caso de uso "Executar análise de vulnerabilidades"(Figura 1 Item C) com a "Ferramenta de Scanner", que representa o módulo responsável pela detecção das falhas.

RF3: **Geração de Relatórios Automáticos** — O software gera relatórios contendo as informações obtidas nas varreduras de forma automática. Essa funcionalidade será visualizada no caso de uso "Gerar relatório automático"(Figura 1 Item E), o qual realiza essa operação sem necessidade de intervenção técnica do usuário, de forma que realizará a comunicação com o sistema e gerenciará o relatório e criação do arquivo.

RF4: **Análise por URL** — O sistema aceita como entrada uma URL de uma aplicação web, que é o alvo da análise, e posteriormente o usuário deve informar qual ferramenta de scanner deseja que seja utilizada (Figura 1 Item B). Essa ação será visualizada no caso de uso "Inserir URL da aplicação"(Figura 1 Item A), onde o usuário informa o alvo a ser analisado, dando início ao processo de análise.Esta ação também pode ser visualizada no primeiro passo no Diagrama de Atividades (Figura 2 Item A).

RF5: **Registrar Logs** — O sistema registra logs do processo de varredura, incluindo data, hora e status das requisições realizadas. Essa funcionalidade é exibida no

caso de uso "Visualizar resultados da análise"(Figura 1.D), onde os eventos gerados são automaticamente registrados pelo sistema.

RF6: **Exibir Resultados** — O sistema exibe os resultados da análise de forma clara e compreensível para o usuário, incluindo as vulnerabilidades encontradas e sugestões automáticas de melhorias por meio de um PDF (Figura 1.E).Essa funcionalidade é visualizada por meio do caso de uso "Visualizar resultados da análise"(Figura 1 Item D), no qual o usuário pode ver de forma organizada os resultados que foram obtidos e sugestões de melhoria caso haja vulnerabilidades.

RF7: **Exportar Informações** — O sistema permite a exportação dos relatórios gerados em formato PDF. Essa funcionalidade é visualizada no caso de uso "Baixar relatório"(Figura 1 Item F), onde o usuário realiza o download do relatório gerado pelo sistema.

B. Requisitos Não Funcionais

RNF1: **Desempenho:** O sistema realiza a análise em um tempo adequado a complexidade apresentada pelo site a ser analisado, bem como a geração do relatório.

RNF2: **Usabilidade:** A interface do sistema é intuitiva, permitindo que usuários comuns consigam utilizá-lo com facilidade.

RNF3: **Segurança:** O sistema não armazena senhas ou informações sensíveis da aplicação analisada. A comunicação com ferramentas externas é realizada via HTTPS.

RNF4: **Confiabilidade:** O sistema é capaz de lidar com falhas de conexão ou erros internos durante a análise, registrando essas falhas tanto nos relatórios quanto nos logs sem interromper bruscamente o funcionamento.

RNF5: **Manutenibilidade:** O código-fonte do sistema está devidamente documentado, de forma a facilitar a manutenção e futuras atualizações.

No fluxo de atividades que podemos visualizar na Figura 2, inicia-se com o usuário informando a URL da aplicação que deseja que seja realizada a análise, e o sistema verifica se a URL informada está acessível e válida, para evitar erros durante a varredura, caso não seja válida, o sistema informará ao usuário por meio de uma notificação.

Posteriormente o usuário escolhe a ferramenta de scanner que será utilizada, após a escolha, inicia-se a análise de vulnerabilidades que podem comprometer a aplicação-alvo, onde são simulados diferentes tipos de ataques, para identificar possíveis vulnerabilidades, e suas informações são registradas no sistema, como data, hora e status das requisições para um acompanhamento mais preciso.

Após a conclusão da análise, são exibidos os resultados das vulnerabilidades encontradas, informando o quão prejudiciais elas podem ser para a aplicação e como geralmente são evitadas/tratadas por meio de um relatório, que pode ser visualizado no PDF gerado (Figura 2 Item F). Ainda é possível que o usuário deseje fazer o download deste

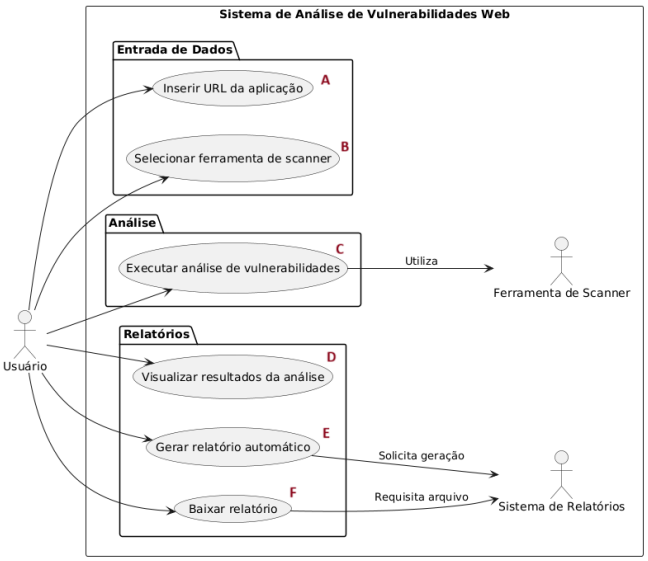

FIGURA 1: Diagrama de Caso de Uso
Fonte: Dos autores [9]

relatório (Figura 2 Item G) se preferir salvá-lo. Caso contrário o relatório ficará disponível para visualização na página gerada(Figura 2 Itens H e I).

VI. RESULTADOS

Como resultado, obteve-se um sistema funcional para análise de vulnerabilidades, cuja interface é apresentada na Figura 3. A aplicação foi organizada em três seções principais: a primeira, uma inserção da URL da aplicação a ser analisada (Figura 3 Item A). Em seguida, foi disponibilizada a seleção da ferramenta de varredura que o usuário desejava utilizar (Figura 3 Item B), permitindo escolher entre os scanners integrados. Por fim, incluiu-se um botão para iniciar o processo de análise de vulnerabilidades (Figura 3 Item C), que aciona o scanner escolhido e gera automaticamente o relatório que pode ser observado na Figura 4.

A geração do relatório foi realizada com a biblioteca *ReportLab*, escolhida pela sua flexibilidade e praticidade na criação de documentos PDF. Foram organizadas as informações coletadas durante as análises, como alertas do OWASP ZAP, resultados do SQLMap, respostas do servidor e dados de pré-testes e foram estruturados em um formato padronizado, incluindo títulos, seções, tabelas, diferenciação visual por níveis de risco e indicadores gráficos, como cores de severidade assim definidas:

- Cinza claro (informacional): para dados informacionais, sendo um alerta que não representa risco de segurança, mas fornece informações úteis sobre o comportamento da aplicação, sua configuração ou possíveis pontos de atenção.
- Azul claro (baixo): Indica um risco baixo, algo que não compromete diretamente a segurança, mas pode

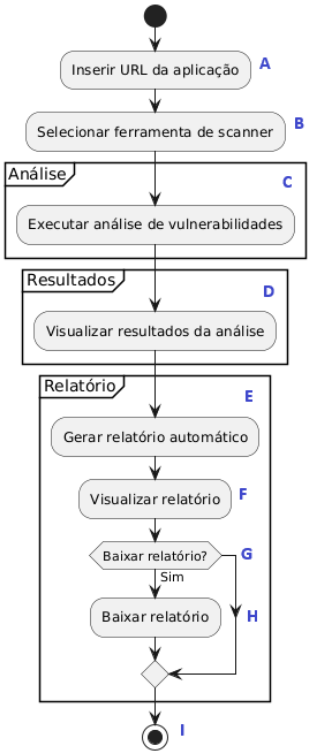

FIGURA 2: Diagrama de Atividades
Fonte: Dos autores [9]

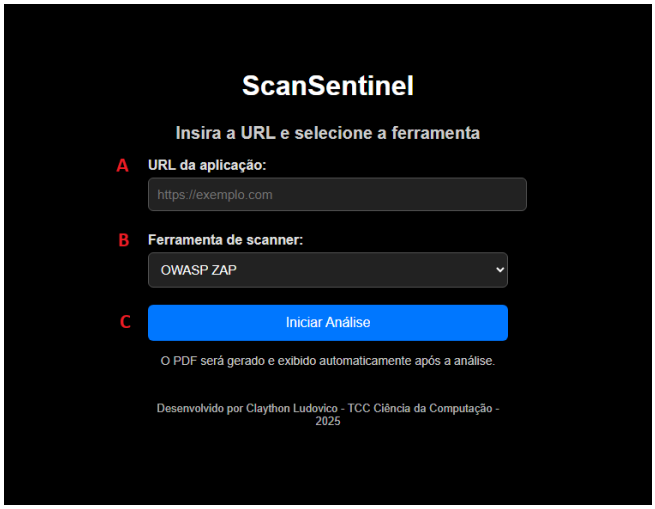

FIGURA 3: Implementação do Site
Fonte: Dos autores [9]

ser combinado com outras falhas para causar um risco maior, como por exemplo versões do servidor expostas, cabeçalhos de segurança ausentes e informações adicionais na resposta HTTP.

- Laranja (médio): Representa vulnerabilidades com risco moderado, capazes de causar impactos reais, mas que normalmente exigem condições específicas ou algum nível de acesso.
- Vermelho (alto): São vulnerabilidades graves, que permitem ataques severos, muitas vezes levando a comprometimento total da aplicação, vazamento de dados ou até mesmo execução remota, tendo um impacto crítico no sistema.

FIGURA 4: Relatorio gerado
Fonte: Dos autores [9]

Na página do relatório (Figura 4) gerada pelo sistema é possível visualizar informações importantes coletadas na análise, como o tempo de resposta, sendo o tempo que foi preciso para ser realizada a análise na URL informada pelo usuário, o tamanho em kilobytes da resposta gerada, as tecnologias utilizadas na URL alvo, e os cookies que foram detectados. Abaixo destas informações, também é possível visualizar no relatório a data e hora que foi realizada a determinada análise.

Além disso, é realizado um pré-teste de SQL injection com a URL informada (Figura 5), em que o sistema realiza a tentativa de inserir o código "' OR '1'='1"(exemplo de injeção SQL) na URL recebida, como uma tentativa de achar uma vulnerabilidade antecipada no endereço antes de realizar a varredura completa, caso seja encontrada alguma alteração no pré-teste, uma mensagem de aviso será infor-

```python
logging.info("Executando pré-teste de SQL Injection.")
try:
    injected_url = target + "' OR '1'='1"
    r2 = requests.get(injected_url, timeout=8)

    if r.status_code != r2.status_code or len(r.content) != len(r2.content):
        logging.warning("Possível SQL Injection detectado no pré-teste.")
        pdf.drawString(2*cm, y, "Pré-teste SQL Injection: Resposta ALTERADA (possível vulnerabilidade)")
    else:
        logging.info("Pré-teste SQL Injection indica que não parece vulnerável.")
        pdf.drawString(2*cm, y, "Pré-teste SQL Injection: Não parece vulnerável")
    y -= 20

except Exception as e:
    logging.error(f"Erro no pré-teste SQL Injection: {str(e)}")
    pdf.drawString(2*cm, y, "Pré-teste SQL Injection: Falha ao testar")
    y -= 20
```

FIGURA 5: Pré-teste de injeção SQL
Fonte: Dos autores [9]

mada no relatório para o usuário.

As vulnerabilidades detectadas pelas ferramentas são informadas tanto em quantidade, quanto pelo nome, descrição e solução correspondente. São originadas pela base de dados acerca das vulnerabilidades. Observa-se que, ao utilizar o OWASP ZAP, algumas destas vulnerabilidades aparecem apenas em inglês. Isso ocorre porque o ZAP mantém a base de dados de alertas e descrições originalmente em inglês, e nem todos os textos foram traduzidos para outros idiomas, logo, quando determinada vulnerabilidade tem sua versão em português, é automaticamente disponibilizada no relatório, caso contrário, sua versão original em inglês é exibida.

VII. CONCLUSÃO

Ao retomar os objetivos específicos deste trabalho, podemos observar que cada um deles desempenhou um papel essencial para o resultado final da pesquisa. Identificar vulnerabilidades em aplicações web mostrou-se fundamental, pois somente a partir da compreensão clara das falhas é possível contextualizar a importância da segurança em ambientes reais e evidenciar a recorrência dessas ameaças no cenário atual.

A automatização da detecção de vulnerabilidades demonstrou sua relevância ao permitir que o processo fosse executado de forma padronizada, eficiente e reproduzível. A automação reduziu a dependência de análise manual, minimizou erros humanos e contribuiu para criar uma ferramenta prática, capaz de ser integrada em diferentes cenários.

A parte da geração de relatórios detalhados foi crucial para transformar resultados técnicos em informações compreensíveis e úteis. Essa estruturação das vulnerabilidades encontradas permitiu visualizar o estado da aplicação de maneira clara, facilitando a interpretação dos riscos tanto por desenvolvedores quanto por gestores, além de reforçar a importância da transparência na comunicação de falhas.

Além disso, avaliar o impacto de cada vulnerabilidade identificada foi relevante para compreender não apenas a existência das falhas, mas também suas consequências reais no funcionamento da aplicação e na proteção dos dados. Essa avaliação possibilitou estabelecer prioridades, contextualizar riscos e demonstrar como cada vulnerabilidade pode comprometer diferentes camadas da segurança do sistema.

Por fim, propor soluções para mitigar as vulnerabilidades encontradas reforçou o objetivo do trabalho, contribuindo para que a pesquisa não se limitasse à identificação dos problemas, mas também apresentasse formas eficientes para minimizá-los. Esse objetivo foi essencial para conectar a teoria e a prática, oferecendo diretrizes que podem ser adotadas por desenvolvedores, equipes de segurança e empresas que buscam fortalecer a proteção de seus sistemas.

Durante o desenvolvimento, foram integradas bibliotecas e ferramentas como Requests, OWASP ZAP e SQLMap, possibilitando a identificação de falhas comuns, como SQL Injection e problemas de configuração em páginas HTML.

O sistema foi testado em ambiente local, validando seu funcionamento e garantindo a geração automática de relatórios que auxiliam na mitigação dos riscos de segurança.

Como trabalhos futuros pode-se considerar a implemetação e utilização de novas tecnologias para análise de diferentes vulnerabilidades, e até mesmo das vulnerabilidades mais comuns com diferentes testes, para obtenção de mais informações relevantes acerca das mesmas. Além disso, pode também ser realizada a tradução das vulnerabilidades que não estão disponíveis no português, visando uma maior compreensão para diferentes usuários.

Com isso, o projeto demonstrou que é possível automatizar o processo de análise de vulnerabilidades, oferecendo uma ferramenta prática e confiável para administradores e desenvolvedores, contribuindo para a melhoria da segurança de sistemas web.

REFERÊNCIAS

[1] CNN Brasil. *Ataques hackers aumentam 8,8% no Brasil, e país segue como 2º mais atacado do mundo*. 2023. URL: https://www.cnnbrasil.com.br/nacional/ataques-hackers-aumentam-88-no-brasil-e-pais-segue-como-2o-mais-atacado-do-mundo/ (acesso em 09/03/2025).

[2] Cloudflare. *Relatório de ameaças DDoS para o 2º trimestre de 2024*. 2024. URL: https://blog.cloudflare.com/pt-br/ddos-threat-report-for-2024-q2/ (acesso em 09/03/2025).

[3] A. Mohammed et al. *Customizing OWASP ZAP: A Proven Method for Detecting SQL Injection Vulnerabilities*. 2023. DOI: 10.1109/ICCCR56703.2023.10132146. URL: https://ieeexplore.ieee.org/document/10132146 (acesso em 02/05/2025).

[4] Marcos Flávio Araújo de Assunção. *Análise de Eficiência na Detecção de Vulnerabilidades em Ambientes WEB com o uso de Ferramentas de Código Aberto*. Trabalho de Conclusão de Curso. 2015. URL: https://core.ac.uk/download/pdf/51455489.pdf (acesso em 02/04/2025).

[5] M. P. Soares A. P. Silva F. P. S. Lima. *Um Web Crawler para Projeções e Análise de Vulnerabilidades de Segurança e Consistência Estrutural de Páginas Web*. 2023. URL: https://seer.atitus.edu.br/index.php/revistasi/article/view/869/782 (acesso em 02/04/2025).

[6] Eduardo Palharini e Alexandre Zamberlan. *Web Crawler para portais da area do Direito*. Trabalho de Conclusão de Curso. 2023. URL: https://tfgonline.lapinf.ufn.edu.br/media/midias/TFG_EduardoFinal.pdf.

[7] Oliveira Plotze e Anna Patricia Zakem China. *ANÁLISE DE TENDÊNCIAS DE VAGAS DE TI NO LINKEDIN UTILIZANDO WEB SCRAPING*. 2023. URL: http://www.fatecrp.edu.br/WorkTec/edicoes/2023-1/trabalhos/ADS/artigo2.pdf.

[8] Lourenço Alves Pereira Junior Guilherme Bertolino Françoa Taffarel. *Metodologia Automatizada para Aquisição e Análise Estática de Firmwares de Roteadores Wi-Fi*. 2024. URL: https://sol.sbc.org.br/index.php/sbseg_estendido/article/view/30161/29969 (acesso em 22/04/2025).

[9] Claython Da Silva e Sylvio André Garcia Vieira. *Dos Autores*. 2025.