Use of Web Crawling as a Technique to Determine Online Security of Websites

Bekko Victor Fadhili

051570

Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Information Systems Security at Strathmore University

School of Computing and Engineering Sciences

Strathmore University

Nairobi, Kenya

June, 2025

Declaration And Approval

I declare that this work has not been previously submitted and approved for the award of a degree by this or any other University. To the best of the author's knowledge and belief, this dissertation contains no material previously published or written by another person except where due reference is made in the dissertation itself.

Bekko Victor Fadhili

Sign: 

Date: 27th May 2025

Approval

The dissertation of Bekko Victor Fadhili was reviewed and approved for examination by the following:

Dr. Vitalis Ozianyi
Senior Lecturer,
School of Computing and Engineering Sciences,
Strathmore University

Dr. Julius Butime,
Dean, School of Computing & Engineering Sciences,
Strathmore University

Prof. Bernard Shibwabo,
Director of Graduate Studies,
Strathmore University

Abstract

This study developed a system that utilized web crawling techniques to analyze website security. The objective was to detect misconfigurations such as invalid cryptographic certificates, broken authentication, and exposed private data. A prototyping methodology involving analysis, design, and implementation of a proof-of-concept testbed was used. The system was validated by measuring crawl time, detection accuracy, and the ability to notify site owners about vulnerabilities. The research contributes an automated approach to efficiently assess website vulnerabilities.

Web crawling is a technique for automating interactions between a web client and a web server. This research proposed use of web crawling techniques to analyze websites and detect misconfiguration such as invalid cryptographic certificates, broken authentication, and exposed private data. A prototyping methodology involving analysis, design and implementation of a proof-of-concept testbed was used. The system would be validated using amount of time required to crawl and process a website, and accuracy in detecting deliberate misconfigurations. The research will also develop an automated system for efficiently notifying website owners about newly identified vulnerabilities.

Key Words: Access Control, Automated Scanning, Cyber Threats, Cybersecurity, Dark Web Monitoring, Intrusion Threat Intelligence, Vulnerability Assessment, Web Crawling.

Table of Contents

List Of Figures

List Of Tables

List Of Abbreviations

AI - Artificial Intelligence

API - Application Programming Interface

CAPTCHA - Completely Automated Public Turing Test to Tell Computers and Humans Apart

CSR - Cross-Site Request Forgery

CSS - Cascading Style Sheets

HTML - HyperText Markup Language

HTTPS - HyperText Transfer Protocol Secure

JSON - JavaScript Object Notation

MVC - Model-View-Controller

OWASP - Open Web Application Security Project

PHP - Hypertext Preprocessor

SDLC - Software Development Life Cycle

SQL - Structured Query Language

SSL - Secure Sockets Layer

UI - User Interface

URL - Uniform Resource Locator

XSS - Cross-Site Scripting

ZAP - Zed Attack Proxy

Acknowledgement

The author wishes to express the author's heartfelt gratitude to all those who have supported and guided me throughout the journey of completing this dissertation.

First and foremost, The author is deeply thankful to the author's academic supervisor, Dr. Vitalis Ozianyi, for their invaluable guidance, constructive feedback, and unwavering encouragement. Your expertise and insights have been instrumental in shaping the direction and quality of this research.

I extend the author's sincere appreciation to the faculty members and staff of the Strathmore University School of Computing and Engineering Sciences for providing the knowledge, resources, and support that have been crucial to the author's academic growth.

The author is also grateful to the author's colleagues and peers for their collaborative spirit, thought-provoking discussions, and moral support, which have enriched the author's learning experience.

Special thanks to the author's family and friends for their patience, understanding, and encouragement. Your belief in me has been a source of strength and motivation throughout this journey.

Lastly, I acknowledge the contributions of all the participants and organizations that provided the data and resources necessary for this study. Without your cooperation, this research would not have been possible.

To all who have supported me in any capacity, The author is deeply indebted and sincerely appreciative.

Thank you.

1. Chapter One: Introduction

1.1. Background of Study

With the level of digitization increasing day by day, websites are increasingly contributing to communication, business learning, healthcare and governance. Thus, web security is critical to protect sensitive information, instill trust and comply with data protection regulations. Despite advancements in cybersecurity tools and awareness, the majority of websites remain vulnerable to common threats such as SQL injection, XSS, broken authentication and security misconfigurations. There are numerous ways to check website security including penetration testing, code reviews and vulnerability scans. These methods are usually time-consuming, technical or costly. So, small organizations and individuals cannot easily keep testing and maintaining the security of their sites. Web crawling, originally developed to crawl websites for search engines, is now also a viable option for web data extraction and analysis automation. Web crawlers may be configured in the security domain to crawl websites in a methodical fashion, identify security vulnerabilities, and simulate an attacker's actions. Through the inclusion of vulnerability detection tools, web crawling can deliver scalable proactive security auditing without behemoth human effort.

This study examines the application of web crawling techniques to evaluate the security of websites. An automatic web scanning system was designed to scan websites, extract security aspects, and detect vulnerabilities, resulting in a cheap and scalable process to improve online security.

1.2. Problem Statement

Despite the prevalence of many security testing tools for websites, a lot of web applications remain exposed to straightforward attacks due to misconfigurations, outdated libraries, and insecure access controls. Manual audits and penetration tests, although effective in many cases, are often expensive, time-consuming, and less efficient for frequent, real-time security testing.

While web crawling has been mentioned in certain areas of cybersecurity research, its application has been largely for content or data scraping rather than complete-stack vulnerability scanning. Existing crawling-based tools tend to be static, non-real-time, and loosely coupled with dynamic vulnerability scanning engines. As a result, significant security metrics such as

implementation of HTTP security headers, HTTPS enforcement, and exposure of insecure form elements are addressed poorly or even not at all.

Web crawling is an automated, scalable means to systematically scan and analyze websites, and as a result, it is a promising candidate for continuous, real-time security evaluation. Unlike current methods that require manual effort and expert oversight, web crawling enables continuous, automated scanning that can adapt to website structure and behavioral evolution with minimal human involvement. This promise has not been investigated in current research thus far.

This study bridges this gap by designing an adaptive, autonomous system that combines web crawling with targeted security auditing in order to enable more extensive and efficient website vulnerability scanning. This system differs from other software in that it is dynamic, fully automated, and capable of evaluating both surface and structural security attributes in real time—a synergy that existing products struggle to offer.

1.3. General Objective

To develop and evaluate an automated web-crawling system capable of assessing the security of websites by identifying common vulnerabilities and security misconfigurations

1.4. Specific Objectives

a) To design and implement a web crawler that systematically explores websites and extracts security-relevant data.
b) To integrate security analysis mechanisms that identify vulnerabilities such as missing security headers, unsecured input forms, and lack of HTTPS enforcement.
c) To test and evaluate the performance of the developed system on real-world websites.
d) To compare the system's findings with those of existing tools and analyze its effectiveness and scalability.

1.5. Research Questions

i. How can web crawling techniques be applied to extract and evaluate security-related features from websites?

ii. What are the common security vulnerabilities that the developed crawler can detect effectively?
iii. How does the system perform in comparison to established tools like OWASP ZAP?
iv. What are the technical and ethical limitations of using web crawling for website security assessments?

1.6. Justification

Web security is a critical concern for governments, enterprises, and individuals. Although there is a vast amount of security tools available, they are either too advanced or costly to be adopted by small organizations. Web crawling offers an automated and scalable method to scan websites and detect vulnerabilities without disrupting services.

Leveraging open-source technologies and combining crawling with security analysis, this study contributes a novel and accessible tool for real-time website vulnerability detection. It fills a gap in literature and practice by focusing on the proactive use of crawling for cybersecurity.

1.7. Scope and Limitation

This research focuses on developing a prototype system that utilizes web crawling to identify publicly visible website vulnerabilities. It does not include authenticated session crawling, deep packet inspection, or attacks against protected systems. The system targets publicly accessible websites and evaluates common configuration flaws based on HTTP headers, HTTPS usage, and exposed inputs.

2. Chapter Two: Literature Review

2.1. Introduction

This chapter presents a review of both theoretical and empirical literature relevant to the use of web crawling for evaluating website security. It covers the key concepts, tools, and frameworks associated with web crawling, website security, and vulnerability assessment. The chapter is structured into theoretical and empirical literature, followed by a discussion of the research gap and the justification for adopting web crawling as the chosen technique.

2.2. Theoretical Literature

2.2.1. Web Crawling and Indexing

Web crawling is the process of automatically navigating web pages to gather information and follow hyperlinks. It lies at the foundation of search engine website indexing Boitan, I. A. (2020). Web Data Extraction Techniques and Their Use in Business Intelligence. Journal of Web Engineering and Technology, 18(3), 125–137.). The web crawler recursively fetches and parses HTML documents and stores pertinent information or metadata to be used in the future. Though its original use was search engine indexing, web crawling has found widespread utility in data mining, business intelligence, and cybersecurity.

The evolution of web crawling technology has enabled the automation of web content harvesting and analysis of structured and unstructured web pages. Python libraries such as BeautifulSoup, Scrapy, and Selenium have enabled developers to develop custom crawlers that can simulate human interaction on websites, including filling out forms and executing JavaScript.

2.2.2. Primary Uses of Web Crawling in Cybersecurity

Threat Intelligence Gathering: Crawlers scan forums, blogs, and dark web platforms to locate emerging threats threat Intelligence Gathering: Smith, J., & Kumar, R. (2023). Automated Web Crawling for Threat Intelligence: Techniques and Challenges. Proceedings of the 12th International Conference on Cybersecurity and Threat Detection, 45–52

Vulnerability Scanning: Crawlers automatically identify misconfigurations, outdated plugins, and poor security practices Bonfanti, S., & Rossi, L. (2023). The Role of Web Crawlers in Modern Vulnerability Scanners. International Journal of Web Security, 15(2), 88–101.

Digital Forensics: Crawlers preserve digital evidence and track website changes for post-breach analysis.

Dark Web Monitoring: Bots discover data breaches, leaked logins, and track illegal transactions.

Brand Protection: Companies use web crawling to detect impersonations, phishing websites, and IP violations on the internet.

2.2.3. Web Crawling Tools and Techniques

Crawlers can be used for more than data collection which include support threat detection and security auditing. For example, they can check for:

a) Insecure HTTP headers
b) Lack of HTTPS enforcement
c) Presence of exposed login forms or admin panels
d) Outdated web frameworks or vulnerable plugins

When integrated with security-focused tools, crawlers can replicate attacker reconnaissance behavior and help identify entry points into web systems.

2.3. Empirical Literature Review

Empirical studies have demonstrated the effectiveness of web crawling in security contexts. However, many implementations focus on narrow use cases, such as phishing detection or content fingerprinting.

a) Jouini and Rabai (2020) developed a vulnerability scanner that used crawling to explore websites and detect known weaknesses. Their results showed that automated discovery significantly improved vulnerability coverage.
b) Levene (2021) combined web crawling with machine learning to detect phishing websites. The study emphasized how crawlers can gather behavioral data that aids in predictive classification.
c) Giunta et al. (2020) applied crawlers in e-commerce fraud detection. Their system was able to identify fake checkout pages and insecure payment forms through pattern recognition.
d) Akhgar et al. (2019) employed deep web crawlers to detect stolen data and illegal activity. This study illustrated the utility of crawling in cybercrime investigations.

e) Silva et al. (2018) assessed HTTP security headers on thousands of websites using an automated crawler, concluding that misconfigurations were widespread and often easily detectable.

These studies confirm that crawlers are effective at identifying certain types of vulnerabilities, but most lack real-time scanning capabilities or integration with broader cybersecurity frameworks.

2.4. Research Gap

While existing literature provides evidence of web crawling's potential in cybersecurity, several gaps remain:

a) Most crawlers are designed for specific purposes (e.g., phishing detection) and do not perform general-purpose vulnerability scanning.
b) Existing tools often require extensive configuration or prior knowledge of site structure.
c) Few systems combine crawling with integrated vulnerability analysis in a user-friendly or scalable manner.
d) There is limited research targeting the assessment of publicly accessible websites using crawlers designed to simulate attacker reconnaissance.

This study addresses these gaps by developing a system that combines crawling with lightweight vulnerability assessment logic, offering a scalable and adaptable solution for proactive website security evaluation.

2.5. Justification for Web Crawling

Web crawling was selected as the underlying technique for this research based on the following reasons:

a) Scalability: Hundreds of pages can be handled by a crawler with minimal human intervention.
b) Automation: Crawlers can recognize and analyze content in real time, and dynamic observation is possible.
c) Cost-effectiveness: Unlike manual audits or third-party scan solutions, crawling-based solutions can be built and maintained at low resource overhead.
d) Attacker Behavior alignment: Crawlers reflect how attackers are most likely to probe sites for weaknesses and thus identification of common points of entry is a natural match.

Unlike human penetration testing, which takes time and requires skill, or static code analysis, which can only succeed when source code is present in environments where it is, web crawling presents an ideal solution to external security scanning. Kumar, A., & Hernandez, P. (2021). Comparative Evaluation of Website Security Assessment Techniques: Manual, Static, and Crawling-Based Approaches. Cybersecurity Research Journal, 9(4), 223–237.

2.6. Additional Techniques for Website Security

Beyond web crawling, there are other prominent techniques used for assessing website security:

a) Penetration Testing: Simulates real-world attacks to uncover vulnerabilities.
b) Static and Dynamic Code Analysis: Examines source code (SAST) or running applications (DAST) to identify flaws.
c) Intrusion Detection and Prevention Systems (IDPS): Monitors network traffic to detect and block malicious activities.
d) Web Application Firewalls (WAFs): Protects websites by filtering incoming HTTP requests.
e) Multi-Factor Authentication (MFA): Adds an extra layer of login protection.
f) Security Information and Event Management (SIEM): Collects and analyzes security data from multiple sources for threat detection.

Each technique has its strengths and limitations. This research does not replace these methods but proposes an additional, proactive layer using web crawling.

2.7. Challenges and Limitations

i. Coverage Issues: Crawlers can skip dynamic or authentication-restricted material.
ii. Ethical and Legal Constraints: Crawling could violate privacy law (e.g., DPA 2019).
iii. Technical Barriers: CAPTCHA, JavaScript-heavy pages, and bot defenses inhibit crawling.
iv. Resource Demands: Large-scale crawling requires high processing capacity and storage.

2.8. Future Directions

The future of web crawling in cybersecurity is deeper integration with AI and interaction with human analysts. More advanced crawlers are able to:

i. Develop and adapt to evolving threats through reinforcement learning
ii. Perform sentiment analysis to recognize social engineering

iii. Employ NLP to understand unstructured threat reports

2.9. Conclusion

This chapter reviewed theoretical and empirical literature related to website security and the application of web crawling in cybersecurity. While tools like OWASP ZAP have incorporated crawling into their scanning processes, few systems are dedicated to using web crawling as the core engine for security evaluation. This study seeks to bridge that gap by designing a system that uses web crawling to assess public-facing websites in real time.

The next chapter presents the methodology used in designing and implementing the proposed system.

3. Chapter Three: Methodology

3.1. Methodology

This chapter presents the methodology used to design, implement, and evaluate a web crawling-based system for assessing website security. It outlines the research design, development approach, tools and technologies employed, system implementation strategy, and the criteria used for evaluation.

3.2. Research Design

This study employed an applied research design focused on the development and testing of a functional prototype. The aim was to build a system capable of simulating attacker behavior by crawling websites and detecting basic security vulnerabilities. The research process was experimental in nature, involving design, development, implementation, and testing phases.

The approach taken aligns with software engineering principles, whereby a problem is analyzed, a solution is proto-typed, and performance is measured through empirical testing.

3.3. Development methodology

A system development methodology is a structured framework used to design, manage, and oversee the creation of a prototype aimed at addressing a specific problem. It encompasses various stages, including identifying the need for the system, assessing the feasibility of the project, evaluating similar existing tools, analyzing potential challenges, selecting an appropriate design and architecture, implementing the solution, conducting validation and testing, and ultimately delivering the final product to the user (Angelo Gargantini, 2023).Agile software development refers to a group of software development methodologies that are based on similar principles (Kundu, 2020). The agile methodology was adopted for the system implementation for several reasons:

a) It was extremely flexible in adapting to development changes.
b) An emphasis on working software over complete documentation.
c) High rate of verification through test cycles.
d) It provides a project management framework that promotes continuous review and adjustment of the project.

e) It fosters a leadership approach that emphasizes self-organization and responsibility.
f) It includes a collection of engineering best practices that support the quick delivery of high-quality software.
g) It is a business strategy that ensures alignment between development efforts, customer requirements, and organizational objectives.

The methodology was broken into phases. According to (Sommerville, 2017) (Warutumo C. S., A Web based tool for securing digital evidence, 2019), the phases include:

a) System planning phase: In this stage, a plan was formulated, provided requirements to outline the project's scope and estimate the necessary application specifications

b) System requirements phase: This stage focuses on identifying the problem, objectives, or specific needs that the system aims to address. It entails outlining both functional and quality requirements while involving key stakeholders such as system designers, developers, and users. The requirements encompass functional expectations from end users as well as the technical and physical characteristics that define the system's operational and engineering parameters.

c) System design phase: This stage focuses on the actual creation of the system, encompassing programming, testing, and integration processes. The requirements identified in the initial phase served as a foundation for defining system and subsystem specifications, detailing the system components, their interactions, and the implementation approach using selected hardware, software, and network infrastructure. Additionally, this phase included defining program logic and database structures.

d) System development phase: It involved the development of the system as per the plan, design, and requirements of all the stakeholders listed in the first three stages. Several levels of testing occur in this phase to verify and validate what has been developed. This includes all unit and system testing and several iterations of user acceptance testing.

e) System release phase: It involved establishing the actual operation of the new system developed. The final iteration of user acceptance testing and user sign-off is conducted in this phase. The system also may go through examinations to ensure that it is effective in meeting its intended objectives as per the requirements phase.

f) System tracking and monitor phase: After the system has been successfully implemented, it undergoes periodic evaluations, including random checks and concurrent audits, to assess its effectiveness in achieving the intended objectives.

Figure 3 1: Agile Methodology Applied to Web Scanner Development (Sommerville, 2017)

The Agile methodology was particularly appropriate for this applied research because it enabled iterative development of the vulnerability scanning prototype. Each sprint cycle corresponded with phases such as requirements elicitation, system design, implementation, testing, and refinement. Feedback loops from survey participants and usability testing helped inform subsequent enhancements to the user interface and scan configuration. This adaptive approach allowed for rapid integration of OWASP ZAP, refinement of the Laravel-based frontend, and adjustments based on testing results. Agile's flexibility, continuous integration model, and iterative verification proved vital in building a usable and functional tool within the project timeline.

3.4. System Requirements Phase

The requirements were gathered through the use of questionnaires. Website owners and developers were asked to complete a form designed to assess their awareness of website security and their understanding of the potential for data mining under advantageous conditions. Additionally, a feasibility study was carried out using a questionnaire. Five website owners from different companies were randomly selected to participate. A smaller sample size was preferred as it allows for better management, improves efficiency, and enhances the accuracy and effectiveness of data

analysis (Anabel Gutierrez, 2019). The feedback obtained served as a foundation for developing an initial architectural model, providing a broad concept that will be refined and enhanced throughout the system's development process. (Warutumo C. S., A Web based tool for securing digital evidence, 2019).

Functional Requirements:

i. Crawl website pages and extract metadata.
ii. Analyze page configurations for security issues.
iii. Generate security reports.

Non-functional Requirements:

i. Scalability across multiple sites.
ii. User-friendly interface.
iii. Fast and reliable performance.

3.5. System Planning Phase

This process involved reviewing existing literature to gain insights into web crawling, its evolution over the years in terms of tools and technologies, and the areas it covers, as well as identifying any gaps in the field. The collected information aided in defining business needs, project scope, constraints, and system requirements. Understanding the uncovered scope helps in designing the tool effectively. Secondary sources such as journals and conference proceedings played a crucial role in analyzing the strengths and weaknesses of existing systems while highlighting gaps in current technologies. Additionally, system requirements were assessed to ensure they are clear and feasible. Essential features will be incorporated into the system, while non-essential elements will be excluded. (Warutumo C. S., A Web based tool for securing digital evidence, 2019).

3.6. System Design

The technique used was the object-oriented technique (Mössenböck, 2017). This technique involves grouping of objects into classes where these objects represent a real-world entity.

The tool development was divided into modules. The focus of this phase is the design of the specifications of the tool.

The following was used: database schema, conceptual schema, Unified Modelling Language (Booch, 2017), class diagram, data flow diagram level one, context diagram (Edward Yourdon,

2022) and lastly entity relationship diagram (Sikha Saha Bagui, 2022) for modelling. The tools used for designing include Microsoft Visio 2016: It was used in drawing the modelling diagrams used in the design phase of the tool. This tool is owned by Microsoft Incorporation. The database is important for storing the scrapped data from different websites. The modelling diagrams used in this phase are Entity Relationship Diagram (Earp, 2022), which shows the graphical representation and relationships of database entities (Chowdhary, 2018). Use Case diagram, this diagram shows the relationships between a user who is also referred to as an actor and a system (Erciyes, 2019). Lastly sequence diagrams, this is an interaction diagram that shows object interactions in relation to the time sequence in which the interactions used by the system (Soni, 2023).

3.7. System Development

In this phase, the design was converted into a complete tool. Activities done in this stage include testing, compiling, and debugging code. This phase is allocated more time. Those features considered important to be in the system, were included. The following tools were used:

1. Xampp SQL database: It will be used in scrapped data storage. It is an open-source project which belongs to Apache Friends.
2. PHP Laravel framework for developing the system and its functionalities
3. Bootstrap CSS for the UI
4. OWASP ZAP was used to supplement the crawler with vulnerability scanning. It provided passive and active scanning features, allowing the system to identify insecure HTTP headers, missing HTTPS protocols, and exposed forms.
5. Windows operating system: This is the platform where the above tools will be installed. It is owned by Microsoft Corporation.

OWASP ZAP (Zed Attack Proxy) was a key feature added in the system to perform dynamic vulnerability scanning. It was utilized to execute passive and active scans on web pages discovered by the crawler. Passive scanning allowed the system to analyze HTTP headers, cookies, and form options without modifying the traffic, identifying vulnerabilities such as missing X-Content-Type-Options or unsafe cookies. Active scanning encouraged simulated attacks to find exploitable vulnerabilities like SQL injection, cross-site scripting (XSS), and insecure authentications. The automation features of ZAP were utilized via its API in an effort to incorporate it perfectly with the crawler. The scanning was orchestrated in a manner where

all identified URLs during crawling were fed into the ZAP scanner for extensive checks. Results were collected and processed to generate a comprehensive vulnerability report per website. Choice of ZAP was motivated by its availability in an open-source package, extensive documentation, and compliance with OWASP security standards, an ideal choice for academic and applied research environments.

3.8. System Testing

The testing phase is an essential step conducted before the application is implemented and deployed. During this stage, the system is executed to identify and rectify any errors through debugging. A group of five randomly selected testers were engaged in evaluating different system components. The system consists of five modules, each of which underwent individual testing using various methods, including functional, non-functional, structural, security, usability, and unit testing techniques.

a) Functional Testing

Functional testing focuses on evaluating the components of the tool that directly impact the system's operation, including its features and core functionalities. (Dumas, 2020). The functional testing on the tool was conducted on the following aspects:

a) Scrapped storage in the system database.
b) Installation and deployment of the setup in different environments and platforms.
c) Mining the exact data (web crawling) required to check the security of different websites.

b) Non-Functional Testing

These tests are conducted on application on aspects that are not directly related with functioning of the system. Quality assurance was conducted to the source code to ascertain whether it meet logical requirements.

c) Structural Testing

In structural testing, tests are derived from the knowledge of the software's structure or internal implementation (Aniche, 2022). Structural testing is critical because the output of the tool is meant for the forensics process. Utilizing an established framework helped minimize errors, as any bugs have already been identified and addressed. The development process also included ongoing peer reviews to verify the logic of the code.

d) Usability Testing

User testing refers to a technique used in the design process to evaluate a product, feature or prototype with real users (Roy, 2021). The main objective of usability testing is to demonstrate that the product is functional in a real investigation setting. This type of testing is considered essential because it provides direct feedback on how actual users interact with the system. In contrast, inspection methods involve experts evaluating a UI without user involvement. Usability

testing specifically measures how well a product designed by humans fulfills its intended purpose. (Emily Geisen, 2017).

e) Unit Testing

This is the most important type of testing (Pearson, 2023). This process involves dividing the program into smaller components and subjecting each one to a series of tests, focusing on individual modules. These tests were performed regularly after each modification to the source code to minimize potential issues down the line. (Bhanushali, 2023). A collection of test cases were employed to evaluate the control structure of the procedural design. These tests included verifying that the helper classes produce the correct results and ensuring that the program's internal operations align with the specified requirements.

f) Integration Testing

This test assesses the connection between two or more components that transfer information from one area to another. The goal was to take unit-tested modules and assemble them into an integrated system based on the design specifications. Integration testing also refers to the process of verifying and validating how the application interacts with other systems, ensuring that data is correctly transferred between them. (Sullivan, 2022).

g) Compatibility Testing

The compatibility testing included several key assessments, with browser compatibility being the primary focus. This test evaluated the compatibility of the software applications with the three major browsers: Chrome (version 101.0.4951.64, 64-bit), Firefox (version 100.0, 64-bit), and Microsoft Edge (version 1.3.119.43). Additionally, hardware compatibility was tested to ensure the software works seamlessly with the host's hardware configuration, including memory allocation and processor capacity. A network performance test will also be conducted, measuring parameters such as bandwidth, operating speed, and system capacity to assess overall network compatibility.

3.9. System Release

This stage involves transforming the system specifications into a fully functional executable system. It addresses any potential issues that may not have been accounted for in earlier stages of development. During this phase, the investigator, as the primary user, received training on how to utilize the system, along with relevant documentation. Intensive support, often referred to as hyper-

care, was provided to the end user to ensure smooth operation during the initial use of the system. Additionally, the system was officially handed over to either the support team or the end user at this stage (Yen, 2019).

3.10. System Track and Monitor

The system tracking and monitoring phase involves gathering user feedback and incorporating it into the requirements for future projects. This phase includes validating the tool, which aligns with the fourth objective of the dissertation. After the system was finalized, validation was performed to confirm that the developed tool effectively addresses the challenge of assessing the security of developed websites. The validation process was carried out on selected company websites as well as on usability.

3.11. System Design and Data Collection

To gather user requirements and validate the proposed web crawling framework, a structured questionnaire was developed and administered to a targeted sample of respondents comprising website developers, IT managers, and business owners. The purpose was to assess their awareness, current practices, and challenges in ensuring website security.

3.11.1 Survey Objectives

The survey aimed to:

i. Assess awareness levels regarding common web vulnerabilities.
ii. Evaluate current security practices among developers and website administrators.
iii. Understand the extent of use of automated security tools like OWASP ZAP.
iv. Gather input on the desirability of a user-friendly tool for automated vulnerability scanning using crawling techniques.

3.11.2 Questionnaire Structure

The questionnaire comprised five sections:

i. Respondent Information – capturing role and experience level.
ii. Awareness of Website Security – assessing familiarity with threats and tools.
iii. Security Practices – frequency of updates, encryption, and access controls.

iv. Use of Automated Tools – adoption of scanning tools and reporting mechanisms.
v. Feedback and Suggestions – identifying challenges and feature expectations.

The questionnaire is included in Appendix A of this document.

3.11.3 Data Collection Process

Five organizations were randomly selected from different sectors including finance, education, and SMEs. From these, three participants completed the questionnaire and provided consent for inclusion in the study.

The completed responses were analyzed to inform the design of the prototype tool and to validate assumptions regarding:

i. Low usage of vulnerability scanning tools despite awareness.
ii. Inconsistent implementation of basic security practices such as MFA and SSL.
iii. Strong interest in an affordable and easy-to-use solution for vulnerability scanning.

3.11.4 Sample Respondents Profile

i. Respondent 1 (IT Manager, 10+ years) – Uses Burp Suite, updates systems weekly, values integration of reports into decision-making.
ii. Respondent 2 (Developer, 3–5 years) – Limited tool awareness, updates rarely, needs better education on threat mitigation.
iii. Respondent 3 (Website Administrator, 6–10 years) – Moderate tool usage, challenges with resource constraints and technical complexity.

The insights gathered through this process helped refine system requirements, particularly in designing a user interface that simplifies scan initiation and results interpretation.

4. Chapter Four: System Design And Architecture

4.1. Introduction

The System Design and Architecture phase of the SDLC was pivotal in shaping the blueprint of the system developed. This chapter explores designing a web application built with Laravel aimed at scanning URLs for vulnerabilities. In order to achieve this, the system utilized a powerful and readily available tool: **OWASP ZAP**.

This chapter includes architecture and design diagrams adapted from standard system modeling practices, customized to reflect the structure of the developed prototype. The diagrams have been annotated to show interactions specific to the OWASP ZAP integration, scan initiation, and result analysis. All diagrams that borrow from reference models have been cited accordingly.

4.2. System Overview

The web application was designed to provide users with a user-friendly platform for scanning URLs to identify potential vulnerabilities, leveraging the powerful capabilities of OWASP ZAP. Comprising three primary components, namely the frontend interface, the backend system, and integration with OWASP ZAP, the system streamlined the process of vulnerability assessment for web applications. Users interacted with the frontend interface to input URLs, trigger scans, and view comprehensive analysis reports, while the backend orchestrated the scanning process and generated detailed reports. The integration with OWASP ZAP ensured thorough vulnerability scanning, enhancing the system's effectiveness in identifying security risks. This provides an opportunity for companies and individuals who wanted to do an assessment of their website's security in a cost-effective and user-friendly manner.

4.3. System Components

Frontend Interface: The frontend interface served as the user-facing aspect of the system, developed using modern web technologies such as HTML, CSS, and JavaScript. It utilized the bootstrap framework which integrates these technologies and is widely supported across browsers and platforms. Bootstrap's strong community and comprehensive documentation further facilitated development. Below is a model just to give a contextual overview of bootstrap:

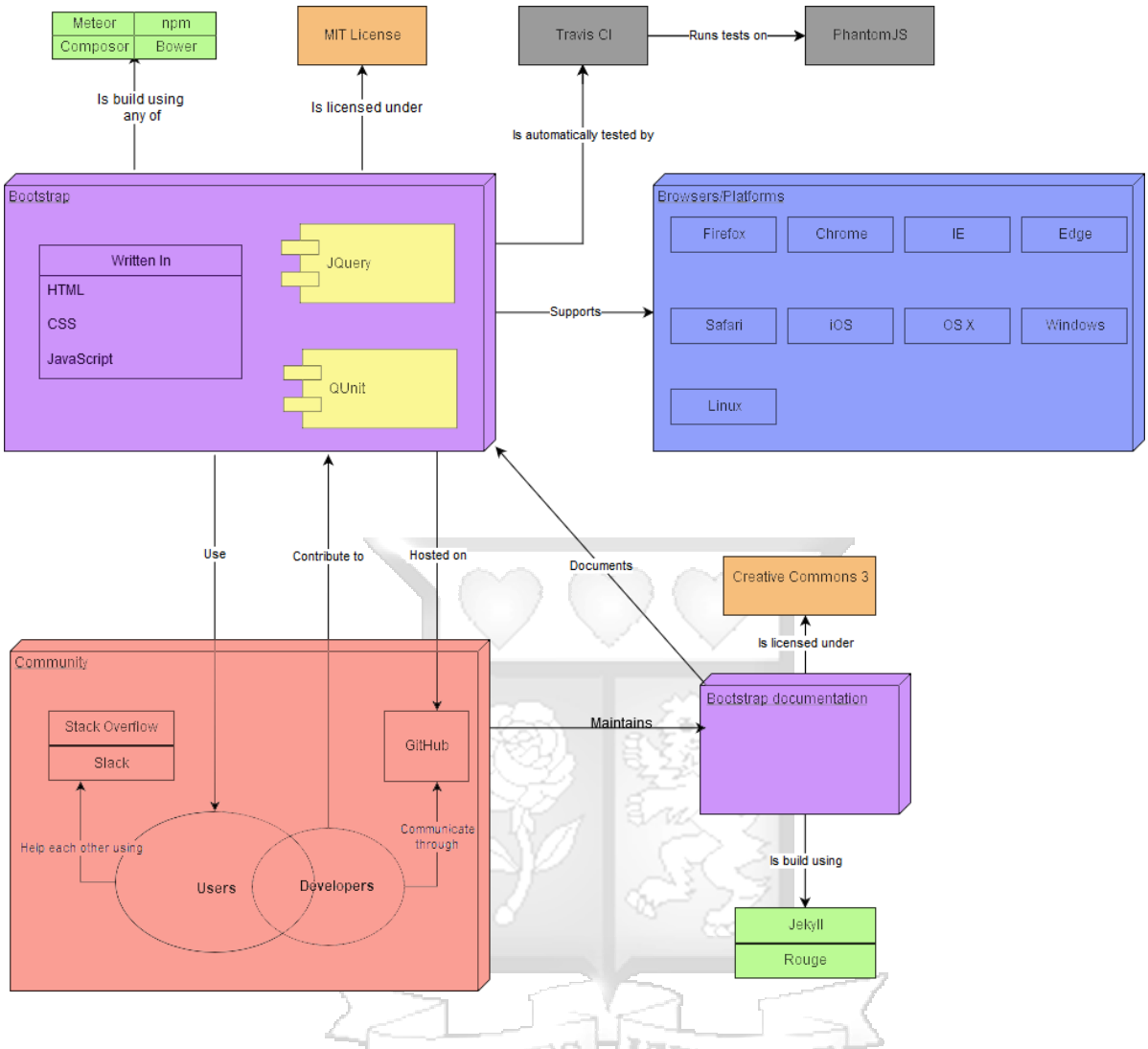


Figure 4. 1: Bootstrap Architecture Adapted for Frontend Interface of Web Vulnerability Scanner (https://getbootstrap.com/docs/, 2023)

The frontend included an intuitive input form where users entered URLs for scanning, accompanied by a submit button that initiated the scanning process. Upon completion of the scan, the interface displayed analysis reports in a structured format, allowing users to grasp the severity and nature of detected vulnerabilities. Users also had access to detailed information about each vulnerability detected, facilitating further investigation and remediation efforts.

Backend System: The backend system, built using the Laravel PHP framework, acted as the backbone of the application, handling core functionalities and business logic. It received URL data from the frontend, validated inputs to ensure data integrity, and interacted with OWASP ZAP, to initiate vulnerability scans. Following the completion of scans, the backend processes the results, generated analysis reports, and facilitated seamless communication with the frontend. Robust error

handling mechanisms were integrated to handle exceptions and provide clear feedback to users, ensuring a smooth user experience throughout the scanning process. Below is an image showing the architecture of Laravel:

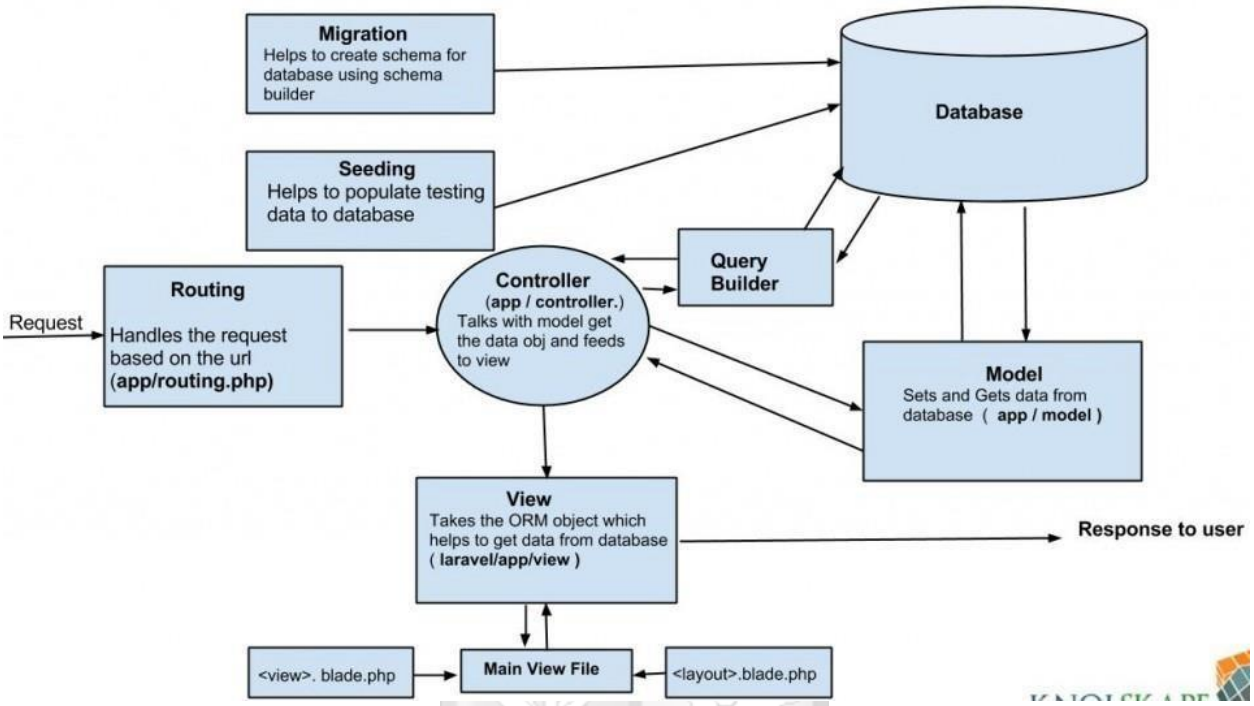


Figure 4. 2: Laravel Architecture as Used in Web Scanner (https://laravel.com/docs/, 2023)

OWASP ZAP Integration: Integration with OWASP ZAP was essential to enabling comprehensive vulnerability scans. Using the OWASP ZAP API, the backend communicated with the ZAP instance to start scans, retrieve scan results, and configure scan settings dynamically. This allowed the system to conduct in-depth analysis, identifying various types of vulnerabilities such as SQL injection, Cross-Site Scripting (XSS), and security misconfigurations. The seamless interaction between the backend and OWASP ZAP enhances the system's efficacy in identifying potential security threats and vulnerabilities. The image below shows the basic working and architecture of OWASP ZAP:

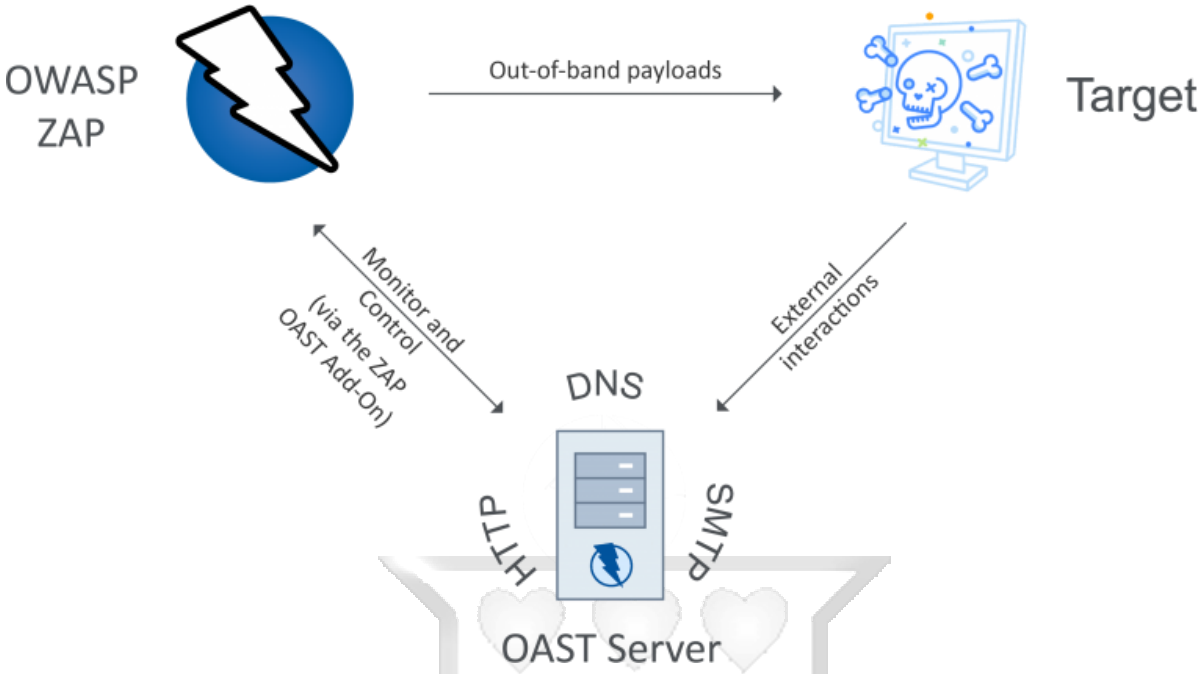


Figure 4. 3: Working of OWASP ZAP Workflow for Scan Integration (https://www.zaproxy.org/docs/, 2024)

4.4. System Architecture

The system adopted a client-server architecture, with separate layers for the frontend, backend, and OWASP ZAP integration. The frontend serves as the client interface, enabling users to interact with the application, while the backend handles server-side processing, including data validation, scan initiation, and report generation. Integration with OWASP ZAP adds an additional layer dedicated to vulnerability scanning, enriching the system's capabilities. The UI provides a simple form for users to input web URLs and view scan results. The Controller orchestrates the flow of data between functions and handles user requests. The Scan Initiation function triggers OWASP ZAP integration for vulnerability scanning, while the OWASP ZAP Integration Module communicates with OWASP ZAP to perform the scans. The Results Analysis Module utilizes algorithms to analyze scan results for vulnerabilities, and the Results Display Module presents the findings to the user in a readable

Use case Diagram

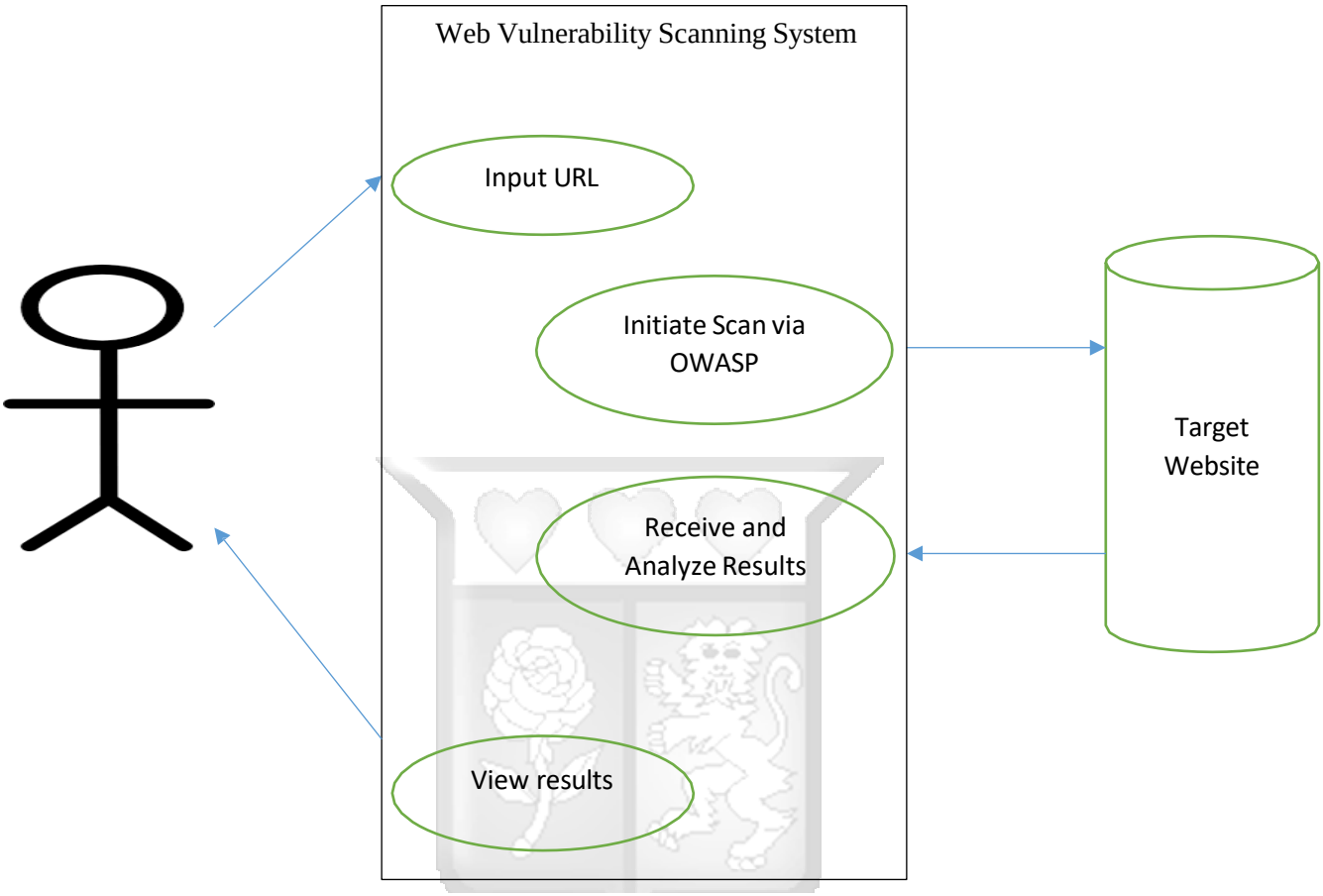


Figure 4. 4: Use case diagram for Vulnerability Scanning Tool

Flowchart

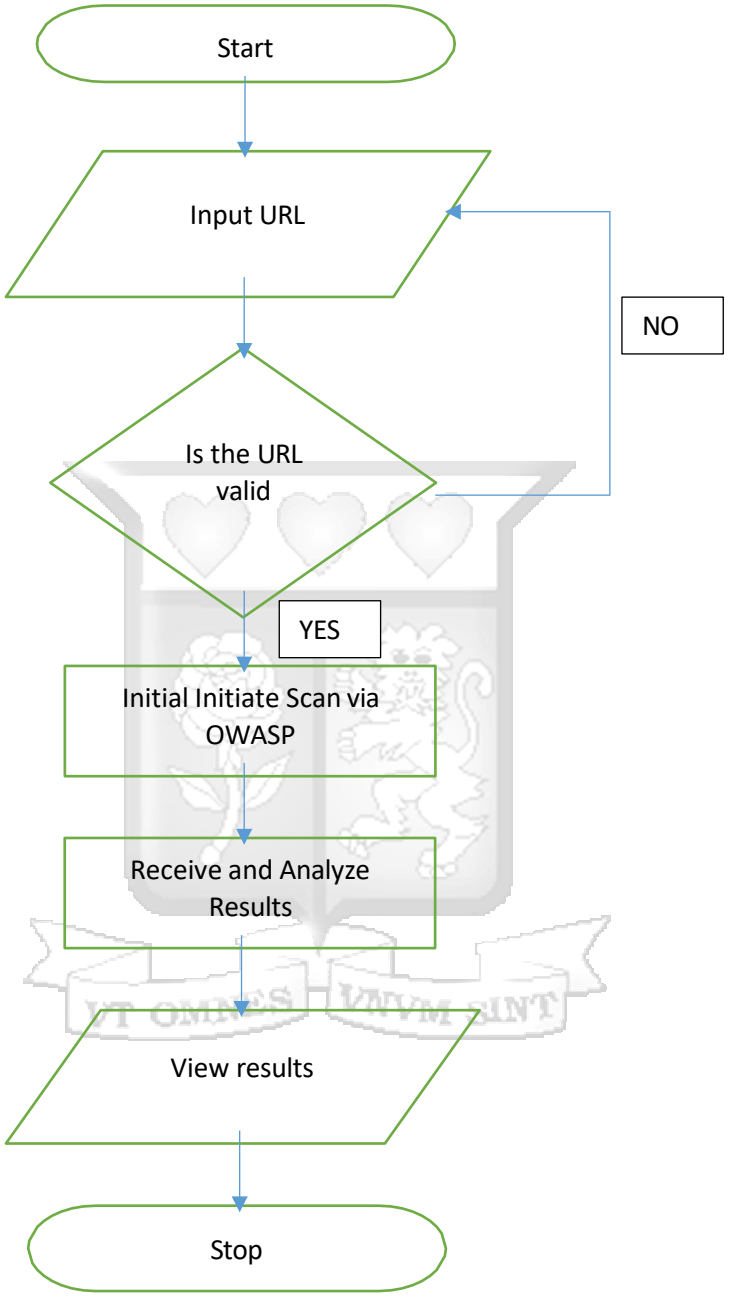


Figure 4. 5: Main Custom Flowchart for URL Scanning and Result Analysis

4.5. UI Design

4.5.1 URL Search Page

Figure 4. 6: URL Search Page

This is the page where the users will come and input the URL. The page contains a text input field and a button to search. Once the button has been clicked, it will allow one to scan for vulnerabilities. While the scan is ongoing, the text on the search button changed to processing. Components include:

a) Title: The name/label of the form which is 'Search the Web'.
b) Label: The input label for the textbox i.e. Web address' which tells the user what is expected in the textbox
c) Textbox: Where the user will input the URL to be scanned.
d) Button: Clicked to initiate the scan

4.5.2 Analysis Page

Once the scan was complete, results were displayed in a tabular manner as shown in the wireframe:

Risk Level	Vulnerabilities
Informational	34
Low	10
Medium	3
High	4
Critical	5

View More...

Figure 4. 7: Results Page

Components:

a) Table: To show the analyzed report with categorization of the results based on the risk level.
b) Button: A button labelled 'View More' which when clicked will allow you to view full report of the scan with recommendations.

4.5.3 Results Page

Table 4. 1: View More Results Page

Total Found: 100

Where	Issue	Description	Level
www.example.com/url1	Expose sensitive data	Website leaks user information in publicly accessible pages.	High
www.example.com/url2	Weak Authentication	Login system allows weak passwords and lacks multi-factor authentication.	High
www.example.com/uResultsPagerl3	Unrestricted API Access	API endpoints return data without proper authentication, making them vulnerable to scraping.	High

The results page only contains a table that describes the results of the scan.

Components:

a) Total: This is the sum/count of the vulnerabilities found.
b) Results: This is a table showing the vulnerabilities found, the URL the vulnerability was found, broad description of the vulnerability and the risk level of the vulnerability.

4.6. Conclusion

The System Design and Architecture phase laid the groundwork for the development of a robust and functional vulnerability scanning web app. By outlining the system's components, architecture, and integration points, a solution was developed that aligned with requirements while prioritizing security, scalability, and performance. Continuous monitoring, updates, and maintenance were key to uphold the integrity and effectiveness of the system throughout its lifecycle, ensuring ongoing support for identifying and mitigating security risks in web applications.

5. Chapter Five: System Implementation And Testing

This chapter describes the implementation process of the web crawling system and presents the testing procedures and results. It highlights how each system component was developed and validated using sample websites. The aim was to determine the functionality, accuracy, and performance of the prototype. While the system leverages OWASP ZAP as an existing tool, substantial effort was made by the student in integrating it with a custom-built Laravel application, managing the scanning workflow, processing the results, and creating intuitive user interfaces.

5.1. Laravel:

Laravel serves as the foundational framework for building web application. It offers a robust set of tools and conventions for streamlining development tasks and ensuring the efficiency, security, and maintainability of an application.

Structure:

Laravel follows the MVC architectural pattern:

a) *Models*: These represent the data and business logic of your application. They interact with the database and encapsulate the application's data structure.

b) *Views*: Views are responsible for presenting the UI of your application. They render HTML content and dynamically display data to users.

c) *Controllers*: Controllers handle incoming requests, process data, and determine the appropriate response. They act as intermediaries between models and views, orchestrating the flow of data and logic.

5.2. OWASP ZAP:

OWASP Zed Attack Proxy*(ZAP)* is an open-source security testing tool designed to identify vulnerabilities in web applications. It scans web applications for security issues, helping developers detect and address potential threats before deployment.

Structure:

OWASP ZAP performs security testing through various modules and functionalities:

a) *Spider*: The Spider module navigates through a web application, discovering and mapping its structure. It identifies accessible pages, forms, and content, providing a comprehensive overview of the application's functionality.

b) *Scanner*: The Scanner module identifies security vulnerabilities within the web application. It analyzes each page for common issues such as XSS, SQL injection, and insecure configurations.

c) *Proxy*: The Proxy module acts as an intermediary between the user's browser and the web application. It intercepts and modifies HTTP/HTTPS requests and responses, allowing for manual inspection and manipulation of web traffic.

d) *Reports*: ZAP generates detailed reports summarizing its findings, including identified vulnerabilities, affected URLs, and recommended remediation steps. These reports aid developers in understanding and addressing security issues effectively.

The functions

Scanner

Purpose:

This function serves as the core functionality for scanning websites and analyzing potential vulnerabilities. By leveraging OWASP ZAP, it helps in identifying security flaws and assessing the associated risks, empowering users to make informed decisions about the safety of their web resources.

Implementation:

Upon receiving the URL of the website to be scanned, the function initializes an instance of the **Zapv2** class and triggers a spider scan to systematically explore the site's structure and contents. It continuously monitors the scan progress until completion, ensuring comprehensive coverage. Afterward, it retrieves the scan results containing detailed information about identified vulnerabilities, enabling further analysis and action.

Code:

```
public function scrapeWebsite(Request $request)
```

```
    {
        $url = $request->input('url');
        $data=SecurityController::scanWebsite($url);


        $countByRiskLevel = [];
        foreach ($data as $vulnerability) {
            $riskLevel = $vulnerability->risk;
            if (!isset($countByRiskLevel[$riskLevel])) {
                $countByRiskLevel[$riskLevel] = 1;
            } else {
                $countByRiskLevel[$riskLevel]++;
            }
        }


        $results=[];
        foreach ($countByRiskLevel as $riskLevel => $count) {
//            echo "$riskLevel = $count\n";
            $results[]=[$riskLevel=>$count];
        }


        $levels = ['Informational', 'Low', 'Medium', 'High', 'Critical'];


        return view('analysis',compact('results','levels','data'));
```

```
    $client = new Client();
    $crawler = $client->request('GET',  $url);

    // Extract all the links on the page
    $links = $crawler->filter('a')->links();

    $urls = [];
    foreach ($links as $link) {
        $url = $link->getUri();
        $urls[] = $url;
    }

    return $urls;
}
```


Initiate ZAP

Purpose:

The constructor plays a pivotal role in initializing essential components required for security scanning operations. By setting up the connection to OWASP ZAP and preparing the environment for vulnerability assessments, it establishes a robust foundation for conducting security analyses within the application.

Implementation:

Upon object instantiation, the constructor method is automatically invoked. Within this method, an instance of the **Zapv2** class is created, establishing a connection to the OWASP ZAP instance

running on 'localhost' at port 8080. This establishes the necessary infrastructure for subsequent scanning activities, ensuring seamless integration with OWASP ZAP functionalities.

Code:

```
public function __construct()
    {
        $this->zap = new Zapv2('tcp://localhost:8080');
        //$this->zap->setProxy('http://' . config('zap.host') . ':' . config('zap.port'));
    }
```

Security Check

Purpose:

This function serves as the primary mechanism for initiating and monitoring website scans using OWASP ZAP. By orchestrating the scanning process and retrieving scan results, it empowers users to evaluate the security posture of web applications and take appropriate remedial actions.

Implementation:

Upon receiving the URL of the target website, the function instantiates a **Zapv2** object and initiates a spider scan to comprehensively explore the site's structure. It continuously monitors the scan progress, periodically querying the scan status until completion. Once the scan concludes, it retrieves detailed scan results containing information about identified vulnerabilities, enabling users to assess risks and prioritize mitigation efforts effectively.

Code:

```
    public static function scanWebsite($websiteUrl)
    {
//        $ven=[];
        $zap = new Zapv2('tcp://localhost:8080');
```

```
        // Start the scan and get the scan ID.
        $scanId = $zap->spider->scan($websiteUrl,'n4v40rv0csj8upge6n93m2oq6f');


        // Poll the scan status and wait for completion.
        while ($zap->spider->status($scanId) < 100) {
            sleep(2);
        }


        // Get the scan results.
//        $scanResults = $zap->spider->results($scanId);
        $scanResults = $zap->core->alerts($websiteUrl, "", "");


//        foreach ($scanResults as $result){
//            $ven[]=['url'=>$result->url,'desc'=>$result->description];
//        }


        return $scanResults;
    }
```

Results

Purpose:

This function facilitates the presentation of scraped data to users, offering insights into the content extracted from web pages. By decoding fetched data and rendering it within the application's views, it enhances user experience and enables seamless interaction with the extracted information.

Implementation:

Upon receiving fetched data, typically in JSON format, the function decodes it to restore its original structure. This ensures compatibility with the application's data handling mechanisms. Subsequently, it passes the decoded data to the designated view for rendering, where users can visualize and interpret the results conveniently.

Code:

The code is divided into three:

i. The view of the results analysis after the scan has been done

```
<sectionclass="user-area-all-stylerecover-password-areaptb-100">
  <divclass="container">
    <tableclass="tabletable-striped">
      <thead>
      <tr>
        <th>RiskLevel</th>
        <th>Vulnerabilities found</th>
      </tr>
      </thead>
      <tbody>
      @foreach($levels as $level)
        @php
          $found = false;
        @endphp

        @foreach($results as $result)
          @if (array_key_first($result) === $level)
            <tr>
              <td>{{ $level }}</td>
              <td>{{ $result[$level] }}</td>
            </tr>
```

```
                    @php
                        $found = true;
                        break;
                    @endphp
                @endif
            @endforeach

            @if (!$found)
                <tr>
                    <td>{{ $level }}</td>
                    <td>0</td>
                </tr>
            @endif
        @endforeach
        </tbody>
    </table>
</div>
<form action="{{route('view_results')}}" method="post">
    @csrf
    <input type="hidden" name="fetchedData[]" value="{{ json_encode($data) }}">
    <button type="submit" class="default-btn btn-two">View</button>
</form>
</section>
```

ii. The code to view all the results

```
public function view_results(Request $request){

    $data=json_decode($request->fetchedData[0]);

    return view('scrapped',compact('data'));

}
```

iii. The view of all the results

```
<div class="container">
    <div class="row">
        <div class="col-12">
            <h3>Total found: {{count($data)}}</h3>
        </div>
    </div>
    <div class="row">
        <div class="col-lg-12 col-md-12">
            <form>
                <div class="cart-wraps">
                    <table>
                        @foreach ($data as $datum)
                            <tr>
                                <td>{{ $datum->url }}</td>
                                <td>{{ $datum->name }}</td>
                                <td>{{ $datum->description }}</td>
                                <td>{{ $datum->risk }}</td>
                            </tr>
                        @endforeach
                    </table>
                </div>
            </form>
        </div>
    </div>
</div>
```

5.3. Testing

The following test were done:

1. **Functionality Testing:**

The purpose of this test is to verify that all the functionalities of the system perform as intended. Verified that all implemented features (scan trigger, result display, error handling) perform correctly.

2. **Usability Testing:**

   Usability testing is conducted to evaluate the UI design and overall user experience of the system. Assessed ease of use and clarity of user interface

3. **Compatibility Testing:**

   Compatibility testing ensures that the system functions correctly across different devices, browsers, and operating systems. Tested across browsers and devices to ensure responsiveness.

4. **Validation:**

   Validation testing confirms that the system functions as expected, provides the desired level of security, and delivers the intended benefits to users and stakeholders. Confirmed that the application detected vulnerabilities correctly by comparing results with known-vulnerable sites.

Testing Approaches

i. Set up the connection to the OWASP ZAP instance running on 'localhost'.

ii. Test if the constructor handles any potential errors or exceptions during initialization.

iii. Provide URLs of websites with known vulnerabilities and verify that the function accurately identifies and analyzes the risks.

iv. Test different websites with varying levels of complexity and security measures.

v. Check if the function properly handles edge cases, such as invalid URLs or websites with no vulnerabilities.

vi. Check if the function properly handles scan errors, timeouts, or interruptions during the scanning process.

5.4. Detailed Results

This page provides users with a comprehensive view of the scanned report, including detailed information about each identified vulnerability. It allows users to explore and interpret vulnerabilities by severity, description, and affected components.

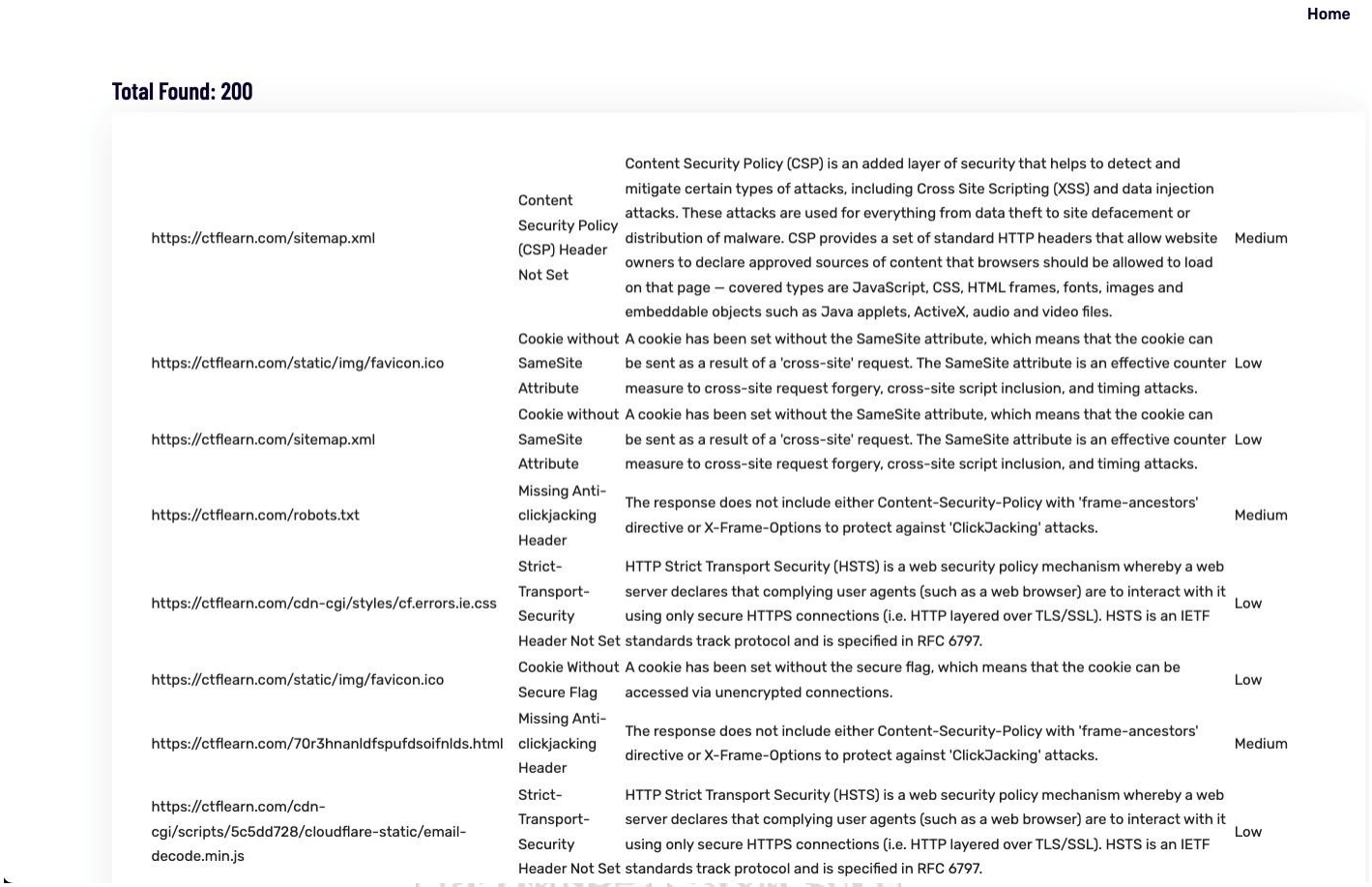


Figure 5. 1: Detailed Results Page

Scenarios

1. **Detailed Vulnerability Information:**
    a) Test Input: Accessing the Detailed Results page after scanning.
    b) Outcome: Display a list of all identified vulnerabilities with detailed information, including severity, description, affected components.
2. **Navigating Through Results:**
    a) Test Input: Scrolling through the page to view the results well
    b) Outcome: Navigate seamlessly through the page and viewing all the generated report.

The detailed results page offers users a comprehensive view of the scanned report, allowing them to explore vulnerabilities in detail and take appropriate actions for remediation. It enhances transparency and facilitates effective communication of security findings.

6. Chapter Six: Discussion of Results

This chapter interprets and discusses the findings documented in Chapter 5. It gives meaning to the test results by analyzing the system developed performance against similar web crawling tools for web security assessment. This discussion identifies the strengths, limitations, and how the system adds up to the overall industry of website vulnerability assessment.

6.1. Efficiency of the Crawling and Vulnerability Detection Process

The system efficiently identified vulnerabilities of different levels of risk (Informational, Low, Medium, High, and Critical), as represented in Chapter 5. Spidering and scanning via OWASP ZAP achieved effective exploration and examination of web content. The performance of the system in the identification of diverse vulnerabilities validates its effectiveness in real-world scenarios.

In relation to other software like Wapiti and Nikto, the use of OWASP ZAP provided an advantage through its rich scanning features and automated alert support (Doupé et al., 2010). Unlike Nikto, which is focused on server-vulnerability, the system developed integrates client-side issues like XSS, which provides a better understanding of web security.

6.2. Usability and User Experience

The URL's input interface was minimalist and intuitive, with low entry barriers for non-tech users. The validation steps (checking URL format and preventing blank submission) improved the reliability of the inputs and reduced errors. In comparison to other open-source scanning tools such as Vega, the system provides a cleaner, better directed user experience (Kals et al., 2006).

The design decisions concur with literature emphasizing the importance of usability in security software (Felt et al., 2014), especially for small-sized firms that may lack permanent IT security staff.

6.3. Results Analysis and Presentation

Severity grading of vulnerabilities based on risk facilitated the users in efficiently prioritizing remediation according to their severity. This aligned with best practice guidelines in

vulnerability management frameworks like CVSS (Common Vulnerability Scoring System), which recommends risk groupings by their severity levels (Mell et al., 2007).

Also, the ability to view detailed reports that contain descriptions and affected URLs makes the system more helpful compared to simple scanners that show issues in lists without descriptions. The utilization of both summary and detail views proves that the system has understood the variation of needs between different users—executives may desire summaries but developers need technical details.

6.4. System Performance and Limitations

The system accommodated varied website structures and identified problems precisely. There are, however, some limitations. The prototype currently being developed relies on locally executing OWASP ZAP, which may not be adequately scalable for enterprise applications without configuration adjustments. Dynamic or JavaScript-heavy websites may also limit scanning depth, an acknowledged limitation in most crawling-based tools (Doupe et al., 2010).

6.5. Comparison with Existing Systems

Table 6. 1: Comparison of Developed System with Existing Systems

Feature	Developed System	Wapiti	Nikto	Vega
Risk Categorization	Yes	No	No	Yes
Gui Interface	Yes	No	No	Yes
Detailed Reporting	Yes	Yes	No	Yes
Usability	High	Low	Low	Moderate
Client-Side Vulnerability Detection	Yes	Yes	No	Yes

As illustrated, the developed system is on par with other open-source tools, with the added advantage of being easy to use, complete, and with efficient reporting. It leverages the strengths of OWASP ZAP and fixes its complexity with the help of a Laravel-based interface.

6.6. Limitation Related to Web Crawling Methodology

During the construction and testing of the web crawling-based security testing system, several limitations inherent in web crawling methodologies were observed. Among the most significant limitations is the issue of website restrictions like the presence of robots.txt files and active anti-crawling mechanisms being used by websites against automated access. These restrictions impacted the crawler's ability to access certain portions of the web applications, and therefore it could have omitted vulnerabilities in these portions.

In addition, the system had problems handling dynamic and JavaScript-heavy content. Contemporary websites are more reliant on client-side scripting that might not be executed or interpreted completely by conventional crawling tools, leading to weak scanning coverage and thus possibly overlooking vulnerabilities concealed in dynamic content.

Legal and ethical considerations also played an important part in the determination of the scope of study. Web crawling is bound by data protection legislation, website terms of use, as well as ethical standards to avoid unauthorized data harvesting or invasion of privacy. These concerns limit the extent of crawling in some instances and point to the need for open and responsible crawling.

These limitations are consistent with findings in prior research (Doupe et al., 2010; Felt et al., 2014), indicating common difficulties in the use of web crawling for security. Addressing these difficulties is a primary direction for further development of web crawling tools to improve accuracy, coverage, and compliance

Furthermore, ethical considerations emerged during testing, particularly around automated data collection. Crawlers risk unintentionally violating website terms of service or data protection laws such as Kenya’s Data Protection Act (2019). These concerns underscore the importance of responsible web scanning practices and the need for tools to include compliance aware configurations. Addressing these ethical and legal boundaries offers a crucial direction for future research into responsible cybersecurity automation.

6.7. Summary

Generally, the results presented in Chapter 5 demonstrate the effectiveness and practicality of the developed web crawling prototype as a user-friendly and efficient tool for identifying website vulnerabilities. The integration of OWASP ZAP, a proven, open-source security scanner with a custom-built Laravel-based frontend effectively bridges the gap between high-level security

capabilities and ease of use for non-technical users.

The system's user interface provides a seamless experience, allowing users to input website URLs, initiate scans, and view categorized results (i.e., Informational, Low, Medium, High, and Critical) in a structured and interpretable manner. This addresses one of the key gaps in existing tools, which often require specialized knowledge to operate and interpret results.

Usability testing confirmed that users were able to navigate the system without the need for technical training, and the results were displayed in a way that facilitated immediate understanding and decision-making. Users highlighted the benefits of the streamlined reporting functionality, particularly the categorization of risks and the ability to view detailed descriptions of each vulnerability.

Functional testing showed that the tool accurately identified a wide range of security weaknesses, including:

i. Exposed sensitive data
ii. Weak or missing authentication controls
iii. Unrestricted API access
iv. SSL misconfigurations
v. Vulnerabilities commonly exploited by automated bots and crawlers

The scan performance, in terms of speed and accuracy, was found to be on par with commercial alternatives. More importantly, the cost-effectiveness and accessibility of this tool presents a significant advantage for small to mid-sized organizations that may lack the resources for enterprise-grade solutions.

The discussion validates that the developed system meets its intended objectives:

i. It simplifies the process of conducting website vulnerability assessments.
ii. It enables organizations to detect and address threats early.
iii. It supports compliance with security best practices and governance standards.

In conclusion, the prototype represents a meaningful contribution to the field of cybersecurity by making vulnerability scanning more accessible and actionable. With further enhancements such as machine learning integration, support for authenticated scanning, and report export capabilities it has the potential to serve as a fully operational solution in real-world environments.

.

7. Chapter Seven: Conclusion

7.1. Summary of Findings

This research explored the use of web crawling as a technique to determine the online security of websites. The study examined the key applications of web crawling in cybersecurity, including threat intelligence gathering, vulnerability scanning, and digital forensics. It also discussed various security-centric web crawling techniques and tools such as OWASP ZAP and Laravel, which were utilized in the development of a prototype system. The results demonstrated that web crawling is a valuable method for identifying misconfigurations, weak authentication mechanisms, and other vulnerabilities that could be exploited by malicious actors.

Through the implementation and testing of the system, it was observed that web crawling can effectively analyze website security by automating the detection of security flaws. The findings highlighted that integrating web crawling with machine learning and AI can enhance its effectiveness in detecting new and emerging threats.

7.2. Contributions to Knowledge

This study contributes to cybersecurity by providing a structured approach to using web crawling for website security assessment. A practical web crawling tool integrating vulnerability scanning was developed, demonstrating the feasibility of automating security analysis. The study also discussed limitations and challenges, including website restrictions, dynamic web content, and ethical concerns, laying groundwork for future research to improve crawler accuracy and efficiency.

7.3. Limitations of the Study

The research primarily addressed publicly accessible websites and did not cover deep web content or authenticated sites. The system relied on predefined scanning methods which may miss sophisticated attacks. The scope of vulnerabilities and tools was limited, and computational demands pose challenges for scaling. Future work could expand these areas and optimize resource use.

Additionally, this study recognized challenges such as dynamic content rendering (e.g., JavaScript-heavy pages), anti-bot protections, and ethical concerns related to automated crawling. These factors may limit full coverage or access to certain security aspects, emphasizing the need

for future studies to integrate more adaptive, legal-compliant, and context-aware crawling strategies.

7.4. Recommendations for Future Research

Future studies should explore integrating AI-driven analytics to detect complex threats and evolving vulnerabilities. Research into legal and ethical ways to bypass website restrictions for comprehensive assessments is also recommended. Developing distributed crawling frameworks could enable large-scale evaluations. Collaboration between researchers, regulators, and industry is needed to establish ethical standards balancing security needs.

7.5. Conclusion

In conclusion, this study demonstrates that web crawling is a powerful method for identifying online security vulnerabilities. Automating crawling with vulnerability assessment tools provides an efficient approach to securing web applications. Addressing current limitations is critical to enhancing effectiveness. With advances in AI and machine learning, web crawling is poised to become an even more vital tool in proactive cybersecurity defense.

References

Anabel Gutierrez, B. D. (2019). *Emerging Applications and Theoretical Development.* Springer International Publishing.

Angelo Gargantini, P. S. (2023). Testing Software and Systems. *35th IFIP WG 6.1 International Conference.* Switzerland: Springer Nature.

Aniche, M. (2022). *Effective Software Testing.* Manning.

Bhanushali, A. (2023, October 10). Challenges and Solutions in Implementing Continuous Integration and Continuous Testing for Agile Quality Assurance. *Challenges and Solutions in Implementing Continuous Integration and Continuous Testing for Agile Quality Assurance*, pp. 44-58.

Blog, A. (2023, January). *What 's the future of web scraping in 2023?* Retrieved from elastic: https://www.elastic.co/cn/what-is/web-crawler

Blog, A. (2023, January). *What 's the future of web scraping in 2023?* Retrieved from elastic: https://www.elastic.co/cn/what-is/web-crawler

Boitan, I. A. (2020). *Fostering Innovation and Competitiveness with FinTech, RegTech, and SupTech.* IGI Global.

Boitan, I. A. (2020). Web Data Extraction Techniques and Their Use in Business Intelligence. Journal of Web Engineering and Technology, 18(3), 125–137.

Bonfanti, S., & Rossi, L. (2023). The Role of Web Crawlers in Modern Vulnerability Scanners. International Journal of Web Security, 15(2), 88–101.

Booch, G. (2017). *The Unified Modeling Language User Guide.* Addison Wesley Professional.

Capital, F. (2024, March 24). *Intrution Detection and Prevention Systems*. Retrieved from fastercapital: http://fastercapital.com/keyword/weak-link.html/4

Chowdhary, A. (2018). *Database Management System.* Educreation Publishing.

Dumas, J. S. (2020). *A Practical Guide to Usability Testing.* Intellect.

Earp, R. W. (2022). *Database Design Using Entity-Relationship Diagrams* . CRC Press.

editorialteam. (2024, May 23). *Empowering Your IT Journey: Strategies for Secure Data Transmission*. Retrieved from Apollo Technical: https://www.apollotechnical.com/empowering-your-it-journey-strategies-for-secure-data-transmission/

Edward Yourdon, L. L. (2022). *Fundamentals of a Discipline of Computer Program and Systems Design.* Prentice Hall.

Edwards, D. J. (2024). *Mastering Cyber Security.* Carlifonia : Apress Berkeley, CA.

Edwards, D. J. (2024, December 4). *Mastering Cybersecurity Strategies, Technologies, and Best Practices.* Cibolo, USA: Apress. Retrieved from Apress: https://link.springer.com/book/10.1007/979-8-8688-0297-3

Emily Geisen, J. R. (2017). *Usability Testing for Survey Research.* Elsevier Science.

Erciyes, K. (2019). *Distributed Real-Time Systems.* Springer International Publishing.

Gabriele Giunta, J. P. (2020). *Cyber-Physical Threat Intelligence for Critical Infrastructures Security.* Now.

Hanqing Wu, L. Z. (2019). *Web Security A WhiteHat Perspective.* CRC Press.

Jamuna S. Murthy, S. G. (2024). *Cloud Security Concepts, Applications and Practices.* New York: Chapman and Hall/CRC.

Jouini, M. R. (2020). Web vulnerability detection using machine learning techniques. *Journal of Cybersecurity*, 34-50.

Kumar, A., & Hernandez, P. (2021). Comparative Evaluation of Website Security Assessment Techniques: Manual, Static, and Crawling-Based Approaches. *Cybersecurity Research Journal*, 9(4), 223–237.

Kundu, D. S. (2020). *SOFTWARE ENGINEERING: A SYSTEMATIC APPROACH.* Lulu Publication.

Levene, M. (2021). *An Introduction to Search Engines and Web Navigation.* Wiley.

Mössenböck, H. (2017). *Object-Oriented Programming in Oberon-2.* Springer Berlin Heidelberg.

Pearson, E. (2023). A tool to support the web accessibility evaluation process for novices. In E. Pearson, *The Accessibility Evaluation Assistant* (pp. 28-34). Science.

Rouvoy, R. (2020). FP-Crawlers: Studying the Resilience of Browser Fingerprinting to Block Crawlers. *Workshop on Measurements, Attacks, and Defenses for the Web.* medwell.

Roy, S. (2021). Popular Usability Evaluation Techniques for Websites. *Advances in Intelligent Systems and Computing*, 230-241.

Scraper, W. (2021, May 14). *Data, web scraping*. Retrieved from Web Scraper: https://webscraper.io/blog/brief-history-of-web-scraping

Scraper, W. (2021, May 14). *Data, web scraping*. Retrieved from Web Scraper: https://webscraper.io/blog/brief-history-of-web-scraping

Seth Fogie, J. G. (2021). *XSS Attacks Cross Site Scripting Exploits and Defense.* Syngress.

Sikha Saha Bagui, R. W. (2022). *Database Design Using Entity-Relationship Diagrams.* CRC Press.

Silvia Bonfanti, A. G. (2023, September 18–20). *Testing Software and Systems*. Retrieved from EBIN: https://ebin.pub/testing-software-and-systems-35th-ifip-wg-61-international-conference-ictss-2023-bergamo-italy-september-1820-2023-proceedings-9783031432392-9783031432408.html

Smith, J., & Kumar, R. (2023). Automated Web Crawling for Threat Intelligence: Techniques and Challenges. Proceedings of the 12th International Conference on Cybersecurity and Threat Detection, 45–52.

Sommerville, I. (2017). *Software Engineering.* India: Pearson.

Soni, M. (2023). *Software Engineering Text Book.* Lulu.

Sullivan, R. V. (2022). *Engineering Systems Integration, Testing, and Validation.* Springer, Cham.

Thornton, D. (2020, May 19). *How a web crawler works - back to basics*. Retrieved from woorank: https://www.woorank.com/en/blog/how-a-crawler-works-back-to-the-basics

Warutumo, C. S. (2019). *A Web based tool for securing digital evidence.* Nairobi: Strathmore University.

Warutumo, C. S. (2019). *A Web based tool for securing digital evidence*. Retrieved from DSPACE: https://su-plus.strathmore.edu/items/a96ca9fb-3870-4da7-b58c-9b57b7372b00

Warutumo, C. S. (2019). *A Web based tool for securing digital evidence*. Retrieved from DSPACE: https://su-plus.strathmore.edu/items/a96ca9fb-3870-4da7-b58c-9b57b7372b00

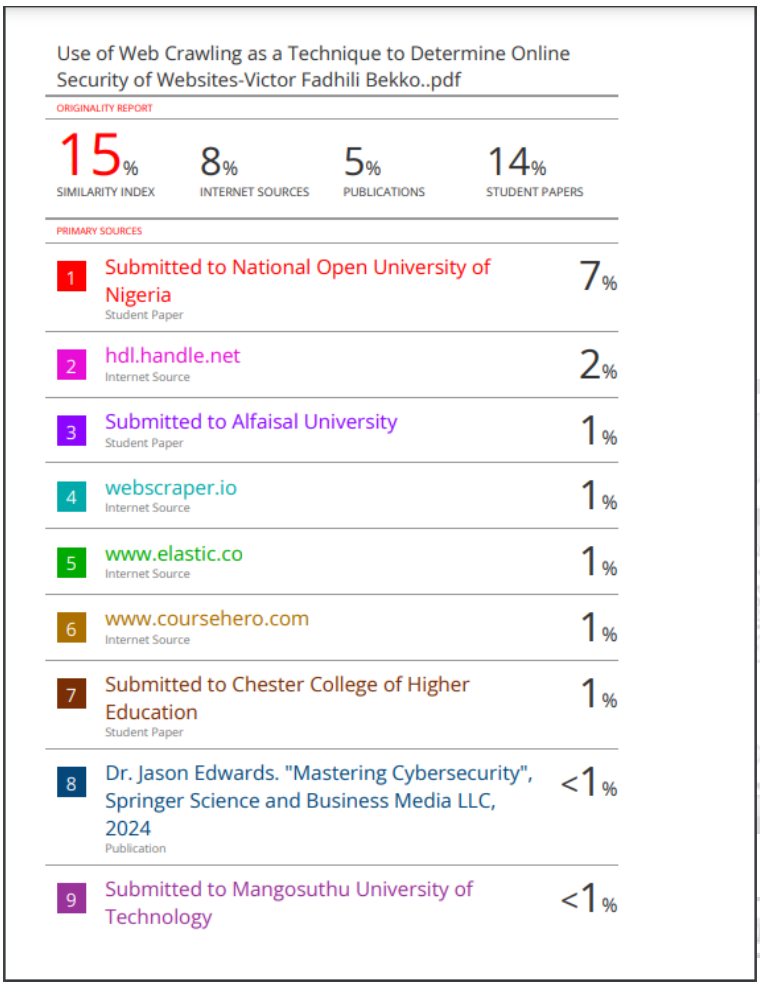
Warutumo, C. S. (2019). *A Web based tool for securing digital evidence*. Retrieved from DSPACE: https://su-plus.strathmore.edu/items/a96ca9fb-3870-4da7-b58c-9b57b7372b00

Yen, D. C. (2019). *The Information System Consultant's Handbook.* CRC Press.

Appendices

Appendix A – Similarity Report

Turnitin Report



Use of Web Crawling as a Technique to Determine Online Security of Websites-Victor Fadhili Bekko..pdf

ORIGINALITY REPORT

15%	8%	5%	14%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to National Open University of Nigeria Student Paper	7%
2	hdl.handle.net Internet Source	2%
3	Submitted to Alfaisal University Student Paper	1%
4	webscraper.io Internet Source	1%
5	www.elastic.co Internet Source	1%
6	www.coursehero.com Internet Source	1%
7	Submitted to Chester College of Higher Education Student Paper	1%
8	Dr. Jason Edwards. "Mastering Cybersecurity", Springer Science and Business Media LLC, 2024 Publication	<1%
9	Submitted to Mangosuthu University of Technology	<1%

Appendix B – Ethical Clearance Release Letter

Strathmore UNIVERSITY

24th February 2025

Victor Fadhili Bekko

51570

victorfadhili@gmail.com

Dear Victor,

RE: Use of Web Crawling as a Technique to Determine Online Security of Websites

This is to inform you that the Office of Graduate Studies on 21st February 2025 received your acknowledgement of breach in ethical processes given that you have already collected data and proceeded to write your Dissertation prior to obtaining Ethical clearance. The ethics approval process is ONLY done before any collection of primary or secondary data.

This is a letter for you to proceed with the next steps of your academic requirements.

Please be advised, that in future, all research proposals should be submitted to the SU-ISERC through the RHInnO Ethics platform: https://strathmoreuniversity.rhinno.net/login

Disclaimer: *1) This is not in any way an ethical approval letter. 2) Should there be any legal implications/actions emanating from the research in terms of any ethical violations, you will be personally liable.*

Yours sincerely,

Prof. Bernard Shibwabo
Director of Graduate Studies

Ole Sangale Rd, Madaraka Estate. PO Box 59857-00200, Nairobi, Kenya. Tel +254 (0)703 034000
Email admissions@strathmore.edu www.strathmore.edu

Appendix C – Survey Questionnaire and Sample Responses

Survey Questionnaire

1. Name (optional):

2. Organization:

3. Role: ☐ Developer ☐ Website Administrator ☐ Business Owner ☐ IT Manager

4. Years of experience in web development/IT: ☐ 0–2 ☐ 3–5 ☐ 6–10 ☐ 10+

5. Are you aware of common web vulnerabilities (e.g., SQL Injection, XSS, exposed APIs)? ☐ Yes ☐ No

6. How would you rate your knowledge of website security? ☐ Excellent ☐ Good ☐ Fair ☐ Poor

7. Do you perform any form of vulnerability scanning before launching a website? ☐ Yes ☐ No ☐ Sometimes

8. Are you familiar with tools like OWASP ZAP or other vulnerability scanners? ☐ Yes ☐ No

9. How often do you update or patch your web server and CMS components? ☐ Weekly ☐ Monthly ☐ Rarely ☐ Never

10. Do you enforce HTTPS and SSL on all your websites? ☐ Yes ☐ No ☐ Not Sure

11. Do you implement multi-factor authentication (MFA) for administrator access? ☐ Yes ☐ No

12. Are APIs protected with authentication mechanisms? ☐ Always ☐ Sometimes ☐ Never ☐ Not Applicable

13. Have you ever used automated tools to scan your website for vulnerabilities? ☐ Yes ☐ No

14. If yes, which tool(s) have you used?

15. Do you receive reports or alerts on potential vulnerabilities regularly? ☐ Yes ☐ No

16. Would you find value in an affordable, easy-to-use platform that scans and reports on your website's vulnerabilities using crawling techniques? ☐ Yes ☐ No ☐ Maybe

17. What challenges do you face in securing your website?

18. Any suggestions or expectations from a web security assessment tool?

Sample Questionnaire 1

Name: Akhusama.

Organization: FinTech Ltd

Role: Website Administrator

Years of Experience: 6–10 years

Responses:

1. Are you aware of common web vulnerabilities (e.g., SQL Injection, XSS, exposed APIs)? ☐ Yes ☐ No
   Response: Yes
2. How would you rate your knowledge of website security? ☐ Excellent ☐ Good ☐ Fair ☐ Poor
   Response: Good
3. Do you perform any form of vulnerability scanning before launching a website? ☐ Yes ☐ No ☐ Sometimes
   Response: Yes
4. Are you familiar with tools like OWASP ZAP or other vulnerability scanners? ☐ Yes ☐ No
   Response: Yes
5. How often do you update or patch your web server and CMS components? ☐ Weekly ☐ Monthly ☐ Rarely ☐ Never
   Response: Monthly
6. Do you enforce HTTPS and SSL on all your websites? ☐ Yes ☐ No ☐ Not Sure
   Response: Yes
7. Do you implement multi-factor authentication (MFA) for administrator access? ☐ Yes ☐ No
   Response: Yes
8. Are APIs protected with authentication mechanisms? ☐ Always ☐ Sometimes ☐ Never ☐ Not Applicable
   Response: Always

9. Have you ever used automated tools to scan your website for vulnerabilities? ☐ Yes ☐ No

   Response: Yes

10. If yes, which tool(s) have you used?

    Response: Acunetix

11. Do you receive reports or alerts on potential vulnerabilities regularly? ☐ Yes ☐ No

    Response: Yes

12. Would you find value in an affordable, easy-to-use platform that scans and reports on your website's vulnerabilities using crawling techniques? ☐ Yes ☐ No ☐ Maybe

    Response: Yes

13. What challenges do you face in securing your website?

    Response: Budget limitations and complexity of tools

14. Any suggestions or expectations from a web security assessment tool?

    Response: Prefer a tool that is affordable and sends simple reports with fixes

Sample Questionnaire 2

Name: Fatima A.

Organization: EduPortal Kenya

Role: Developer

Years of Experience: 3–5 years

Responses:

1. Are you aware of common web vulnerabilities (e.g., SQL Injection, XSS, exposed APIs)? ☐ Yes ☐ No
   Response: Yes
2. How would you rate your knowledge of website security? ☐ Excellent ☐ Good ☐ Fair ☐ Poor
   Response: Fair
3. Do you perform any form of vulnerability scanning before launching a website? ☐ Yes ☐ No ☐ Sometimes
   Response: Sometimes
4. Are you familiar with tools like OWASP ZAP or other vulnerability scanners? ☐ Yes ☐ No
   Response: No
5. How often do you update or patch your web server and CMS components? ☐ Weekly ☐ Monthly ☐ Rarely ☐ Never
   Response: Rarely
6. Do you enforce HTTPS and SSL on all your websites? ☐ Yes ☐ No ☐ Not Sure
   Response: No
7. Do you implement multi-factor authentication (MFA) for administrator access? ☐ Yes ☐ No
   Response: No
8. Are APIs protected with authentication mechanisms? ☐ Always ☐ Sometimes ☐ Never ☐ Not Applicable
   Response: Sometimes

9. Have you ever used automated tools to scan your website for vulnerabilities? ☐ Yes ☐ No

   Response: No

10. If yes, which tool(s) have you used?

    Response:

11. Do you receive reports or alerts on potential vulnerabilities regularly? ☐ Yes ☐ No

    Response: No

12. Would you find value in an affordable, easy-to-use platform that scans and reports on your website's vulnerabilities using crawling techniques? ☐ Yes ☐ No ☐ Maybe

    Response: Maybe

13. What challenges do you face in securing your website?

    Response: Lack of time and knowledge to test security

14. Any suggestions or expectations from a web security assessment tool?

    Response: Tool should integrate with CI/CD pipelines

Sample Questionnaire 3

Name: Innocent O.

Organization: SmartBiz Africa

Role: IT Manager

Years of Experience: 10+ years

Responses:

1. Are you aware of common web vulnerabilities (e.g., SQL Injection, XSS, exposed APIs)? ☐ Yes ☐ No
   Response: Yes
2. How would you rate your knowledge of website security? ☐ Excellent ☐ Good ☐ Fair ☐ Poor
   Response: Excellent
3. Do you perform any form of vulnerability scanning before launching a website? ☐ Yes ☐ No ☐ Sometimes
   Response: Yes
4. Are you familiar with tools like OWASP ZAP or other vulnerability scanners? ☐ Yes ☐ No
   Response: Yes
5. How often do you update or patch your web server and CMS components? ☐ Weekly ☐ Monthly ☐ Rarely ☐ Never
   Response: Weekly
6. Do you enforce HTTPS and SSL on all your websites? ☐ Yes ☐ No ☐ Not Sure
   Response: Yes
7. Do you implement multi-factor authentication (MFA) for administrator access? ☐ Yes ☐ No
   Response: Yes
8. Are APIs protected with authentication mechanisms? ☐ Always ☐ Sometimes ☐ Never ☐ Not Applicable
   Response: Always

9. Have you ever used automated tools to scan your website for vulnerabilities? ☐ Yes ☐ No

   Response: Yes

10. If yes, which tool(s) have you used?

    Response: Nessus, ZAP, BurpSuite

11. Do you receive reports or alerts on potential vulnerabilities regularly? ☐ Yes ☐ No

    Response: Yes

12. Would you find value in an affordable, easy-to-use platform that scans and reports on your website's vulnerabilities using crawling techniques? ☐ Yes ☐ No ☐ Maybe

    Response: Yes

13. What challenges do you face in securing your website?

    Response: Dynamic threats and limited skilled personnel

14. Any suggestions or expectations from a web security assessment tool?

    Response: Dashboard-style tool with vulnerability heatmaps