

UNIVERSITI POLY-TECH MALAYSIA

WEB VULNERABILITY SCANNER

NUR IZZAH AKMAL BINTI MOHD RESTU

**BACHELOR OF INFORMATION
TECHNOLOGY (HONS) IN
CYBERSECURITY**

UNIVERSITI POLY-TECH MALAYSIA

Faculty of Computing & Multimedia

WEB VULNERABILITY SCANNER

(WITZCORE)

**NUR IZZAH AKMAL BINTI MOHD RESTU
A M2311015271**

FYP4085

MAY 2025

Declaration of Originality

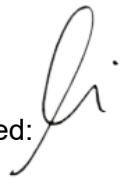
This project is all my own work and has not been copied in part or in whole from any other source except where duly acknowledged. As such, all use of previously published work (from books, journals, magazines, internet, etc.) has been acknowledged within the main report to an item in the References or Bibliography lists.

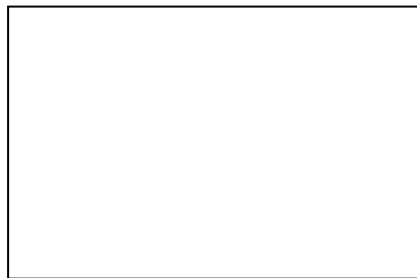
I also agree that an electronic copy of this project may be stored and used for the purposes of plagiarism prevention and detection.

Copyright Acknowledgement

I acknowledge that the copyright of this project and report belongs to Universiti Poly-Tech

Malaysia.

Signed: 



Date: 19/11/2025

Office Stamp

Abstract

The goal of this project is to develop a Web Vulnerability Scanner to help users automatically and easily detect security vulnerabilities in websites. SQL Injection, XSS, poor server configuration and missing security headers are some of the attacks that constantly threaten web applications. Manually checking for these issues is not possible for many users, especially small companies, who do not have the tools or skills to do so. To address this, the system will allow users to enter a URL, and the scanner will scan the entire system with trusted tools such as OWASP ZAP and Nikto that scan for both application level and server level vulnerabilities.

The system is built on the Flask web framework with the help of Celery and Redis to implement background scanning functionality to ensure that the system does not slow down. The scan is processed in the background when the user initiates the scan, so that the interface can be responsive. All scan findings are stored in a central database (SQLite), and users have the option to view them in a clean dashboard summary of Critical, High, Medium, and Low risks. Other important features included in the platform are the ability to log in with OTP (One Time Password), user and administrator role-based access control, deep scan requests, scan history, and PDF report generation which makes the system more secure and professional.

A User Acceptance Testing (UAT) session was conducted to make it usable and effective. The interface was reviewed by users, the scanning process was tested, and the clarity of the results was assessed. Most users reviewed the system was user-friendly, and the features were up to standard in terms of cybersecurity. According to the feedback provided by users, it was found that the tool needs to be improved by adding better names to vulnerability, more clearly displayed scan information, and perhaps future features that integrate with search engines. Overall, the final product has successfully demonstrated the effectiveness of automated scanning as an instrument to provide users with the opportunity to identify risks in a timely manner, improve website security, and foster learning and awareness about cybersecurity.

Table Of Contents

1. INTRODUCTION.....	15
1.1. Introduction.....	15
1.2. Project Background.....	15
1.3. Problem Statement	16
1.3.1. Lack of Accessible Tools for Basic Vulnerability Detection.....	16
1.3.2. Lack of Role-Based Access for New System Users	16
1.3.3. Scan Results Are Hard to Understand	16
1.4. Project Objectives.....	17
1.4.1. To develop a web-based vulnerability scanner for Netwitz clients.....	17
1.4.2. To implement a secure login system with role-based access control.....	17
1.4.3. To display scan results in a clear and understandable format	17
1.5. Scope and Target User	18
1.5.1. Project Scope	18
1.5.2. Product Scope.....	19
1.5.3. Target User	19
1.6. Overview of This Report.....	20
2. LITERATURE REVIEW.....	21
2.1. Introduction.....	21
2.2. Investigation.....	21
2.2.1. Definition SQL Injection.....	23
2.2.2. SQL Injection Prevention	23
2.2.3. Type of SQL Injection.....	24
2.2.4. Definition of Cross-Site Scripting	25
2.2.5. Cross-Site Scripting Prevention.....	25
2.2.6. Type of Cross-Site Scripting	26
2.3. Related Works	26
2.4. Comparison.....	30
2.5. Discussion.....	31
2.6. Conclusion	31
3. METHODOLOGY.....	32
3.1. Introduction.....	32
3.2. Agile Methodology	32
3.3. Phases in Agile Methodology	33
3.3.1. Requirement	33
3.3.2. Design.....	33
3.3.3. Development	34
3.3.4. Testing	34

3.3.5.	Deployment	34
3.4.	Conclusion	35
4.	REQUIREMENTS	36
4.1.	Introduction.....	36
4.2.	Data Gathering Techniques.....	36
4.2.1.	Questionnaire	36
4.3.	Functional Requirement.....	37
4.4.	Non-Function Requirement.....	38
4.5.	System Requirement	38
4.5.2.	Python.....	39
4.5.3.	Flask.....	40
4.5.5.	SQLite	41
4.5.6.	Laptop.....	41
4.6.	Conclusion	42
5.	ANALYSIS	43
5.1.	Introduction.....	43
5.2.	Data Gathering Analysis.....	43
5.2.1.	Questionnaire Analysis.....	43
5.2.2.	Interview Analysis	51
5.3.	Use Case Model	53
5.4.	Flowchart.....	55
5.4.1.	Flowchart Admin user.....	55
5.4.2.	Flowchart User	57
5.5.	Conclusion	58
6.	DESIGN	59
6.1.	Introduction.....	59
6.2.	Interface Design.....	59
6.2.1.	Wireframe Web Vulnerability Scanner	60
6.3.	Database Design	66
6.3.1.	Data Dictionary	67
6.4.	Data Flow Diagram (DFD)	73
6.4.1.	Entity Relational Diagram (ERD)	74
6.5.	Security System Framework	75
6.6.	Docker Framework.....	76
6.7.	Flow of the system.....	77
6.8.	Conclusion	78
7.	IMPLEMENTATION	79
7.1.	Introduction.....	79
7.2.	Execution Platform	79

7.2.1.	Development Platform	80
7.3.	Implementation Tools	80
7.3.1.	Software.....	81
7.3.3.	Hardware.....	89
7.4.	System Interface	89
7.4.1.	User Interface	90
7.4.2.	User-Only Interface: Dashboard and Deep Scan Request Page	96
7.4.3.	Admin-Only Interfaces	98
7.5.	Significant Functions	101
7.5.1.	Web Application Security Scanning Using OWASP ZAP.....	101
7.5.2.	Server Vulnerability Detection Using Nikto	103
7.5.3.	OTP Verification	105
7.5.4.	Role-Based Access Control (RBAC)	105
7.5.5.	CSRF Exemption Decorator	106
7.5.6.	Activity Logging	107
7.6.	Conclusion	108
8.	TESTING	109
8.1.	Introduction.....	109
8.2.	Functional Testing	109
8.2.1.	Unit Testing.....	110
8.2.2.	Integration Testing	112
8.2.3.	System Testing.....	113
8.2.4.	User Acceptance Testing.....	114
8.2.5.	User Acceptance Testing Test Case Table.....	115
8.2.6.	Questionnaire Feedback.....	117
8.3.	Non-Functional Testing	126
8.3.1.	Performance Testing.....	126
8.3.2.	Security Testing	127
8.3.3.	Usability Testing.....	128
8.3.4.	Compatibility Testing	128
8.4.	Conclusion	129
9.	PROJECT MANAGEMENT	130
9.1.	Introduction.....	130
9.2.	Project Schedule.....	130
9.2.1.	Work Breakdown Structure	130
9.2.2.	Gantt Chart	131
9.3.	Risk Management	134
9.4.	Conclusion	136

10. CONCLUSION	137
10.1. Introduction.....	137
10.2. Achievement	137
10.2.1. Achievement of Objective 1: To develop a web-based vulnerability scanner for Netwitz clients	137
10.2.2. Achievement of Objective 2: To implement a secure login system with role-based access control	138
10.2.3. Achievement of Objective 3: To display scan results in a clear and understandable format 138	
10.3. Constraint and Limitation.....	139
10.4. Future Work and Recommendation.....	140
10.5. Conclusion	141
Appendix A – Requirements Specification Document.....	142
Appendix B – User Manual	155
Appendix C – Turnitin Result.....	192
Appendix D – Logbook	202
References.....	206

List of Figures

Figure 2.1 OWASP ZAP	27
Figure 2.2 Wapiti.....	28
Figure 2.3 BurpSuite.....	29
Figure 3.1 Agile Methodology Diagram (Redirect Notice, n.d).....	32
Figure 4.1 Visual Studio Code (Canonical, 2019)	39
Figure 4.2 Python (File:Python Logo and wordmark.svg - Wikimedia Commons, 2008)	39
Figure 4.3 Flask (Iqbal, 2025)	40
Figure 4.4 OWASP ZAP API OWASP Zed Attack Proxy Scan - Visual Studio Marketplace, n.d.).....	40
Figure 4.5 SQLite (SQLite Tutorial - an Easy Way to Master SQLite Fast, 2024)	41
Figure 4.6 Hardware Specification	41
Figure 5.1 Question 1	44
Figure 5.2 Question 2	44
Figure 5.3 Question 3	45
Figure 5.4 Question 4	45
Figure 5.5 Question 5	46
Figure 5.6 Question 6	46
Figure 5.7 Question 7	47
Figure 5.8 Question 8	47
Figure 5.9 Question 9	48
Figure 5.10 Question 10	48
Figure 5.11 Question 11	49
Figure 5.12 Question 12	49
Figure 5.13 Question 13	50
Figure 5.14 Question 14	50
Figure 5.15 Question 15	51
Figure 5.16 Conduct Meeting with client by Google Meet	51
Figure 5.17 Use Case Diagram	53
Figure 5.18 Flowchart Admin User.....	55
Figure 5.19 Flowchart User.....	57
Figure 6.1 Login Page	60
Figure 6.2 OTP Verification Page.....	60
Figure 6.3 Forgot Credentials Page	61
Figure 6.4 Dashboard Page.....	61
Figure 6.5 Scan Page	62
Figure 6.6 Scanning Report Page.....	63
Figure 6.7 Notification Page.....	64
Figure 6.8 Scanning History Page	64
Figure 6.9 Activity Log Page	65
Figure 6.10 User Management Page	65
Figure 6.11 Log Out Page.....	66
Figure 6.12 Data Flow Diagram	73
Figure 6.13 Entity Relational Diagram	74
Figure 6.14 Security System Framework	75
Figure 6.15 Docker Framework.....	76
Figure 6.16 Flow of the System	77
Figure 7.1 Kali Linux (Abrams, 2025).....	80
Figure 7.2 Visual Studio (Canonical, 2019).....	81
Figure 7.3 Docker (Okoli, 2023).....	82
Figure 7.4 OWASP ZAP (K, 2025).....	83
Figure 7.5 Nikto (Amal Tom Parakkaden, 2023).....	84

Figure 7.6 Flask (Iqbal, 2025)	85
Figure 7.7 SQLite (SQLite Tutorial - an Easy Way to Master SQLite Fast, 2024b)	86
Figure 7.8 HTML (Wikipedia contributors, 2025)	87
Figure 7.9 CSS (File:CSS3 Logo and wordmark.svg - Wikimedia Commons, 2016)	88
Figure 7.10 JavaScript (Lebbos, 2023)	88
Figure 7.11 Hardware Specification	89
Figure 7.12 Login Page	90
Figure 7.13 OTP Verification Page	90
Figure 7.14 Administrator Contact Form	91
Figure 7.15 Scan Page	91
Figure 7.16 Scan Result Page	92
Figure 7.17 Web Vulnerability Result	92
Figure 7.18 Server Information Result	92
Figure 7.19 Server Information Result	92
Figure 7.20 Notification Page	94
Figure 7.21 Scan History Page	94
Figure 7.22 Profile Page	95
Figure 7.23 Activity Log Page	95
Figure 7.24 User Dashboard Page	96
Figure 7.25 User Dashboard Page	96
Figure 7.26 Deep Scan Request	97
Figure 7.27 Admin Dashboard Page	98
Figure 7.28 User Management Page	98
Figure 7.29 Add New User Page	99
Figure 7.30 User Details Page	99
Figure 7.31 Edit User Page	100
Figure 7.32 Manage Deep Scan Request	100
Figure 7.33 ZAP API Key Authentication	101
Figure 7.34 Scan Time and Rule Limit	101
Figure 7.35 Vulnerability Grouping	102
Figure 7.36 Security Header Checking	103
Figure 7.37 Categorizing Server Security	103
Figure 7.38 Saving server Vulnerabilities	104
Figure 7.39 OTP Verification	105
Figure 7.40 Role-Based Access Control	105
Figure 7.41 CSRF Exemption Decorator	106
Figure 7.42 Activity Logging	107
Figure 7.43 Activity Logging Message	107
Figure 7.44 Activity Logging Message	107
Figure 8.1 Beta Testing with Client	115
Figure 8.2 Result Questionnaire 1	117
Figure 8.3 Result Questionnaire 2	117
Figure 8.4 Result Questionnaire 3	118
Figure 8.5 Result Questionnaire 4	118
Figure 8.6 Result Questionnaire 5	119
Figure 8.7 Result Questionnaire 6	119
Figure 8.8 Result Questionnaire 7	120
Figure 8.9 Result Questionnaire 8	120
Figure 8.10 Result Questionnaire 9	121
Figure 8.11 Result Questionnaire 10	121
Figure 8.12 Result Questionnaire 11	122
Figure 8.13 Result Questionnaire 12	122
Figure 8.14 Result Questionnaire 13	123

Figure 8.15 Result Questionnaire 14	123
Figure 8.16 Result Questionnaire 15	124
Figure 8.17 Result Questionnaire 16	124
Figure 8.18 Result Questionnaire 17	125
Figure 9.1 Work Breakdown Structure	131
Figure 9.2 Gantt Chart	132

List of Tables

Table 2.1 Distribution of Common Web Application Vulnerabilites (Iterasec,2024) 22

Table 2.2 Comparison of Existing Project 30

Table 4.1Functional Requirement for User 37

Table 4.2 Functional Requirement for Admin..... 37

Table 4.3 Non-Function Requirement Table 38

Table 5.1 Interviewee Information 51

Table 5.2 Interview Questions Table 52

Table 6.1 Data Dictionary for Company Table 67

Table 6.2 Data Dictionary for the User Table 67

Table 6.3 Data Dictionary for the Scan table 68

Table 6.4 Data Dictionary for Vulnerability Table 69

Table 6.5 ServerVulnerability Table 70

Table 6.6 DeepScanRequest Table 71

Table 6.7 Data Dictionary for Notification Table 72

Table 6.8 Data Dictionary for UserLog Table 72

Table 8.1 Unit Testing Table 111

Table 8.2 Integration Testing Table 112

Table 8.3 System Testing Table 113

Table 8.4 User UAT Table..... 115

Table 8.5 Admin UAT Table 116

Table 8.6 Performance Testing Table 126

Table 8.7 System Security Testing Table 127

Table 8.8 Vulnerability Scanning Security Testing Table 127

Table 8.9 Usability Testing Table 128

Table 8.10 Compatibility Testing Table 128

Table 9.1 Gantt Chart Table 133

Table 9.2 Risk Management Table 134

Acknowledgements

Alhamdulillah, all praise and gratitude to Allah SWT for giving me the strength, patience and determination to successfully complete this Final Year Project. This journey was not easy, but it was achieved through the endless support and encouragement from many important individuals. My deepest appreciation goes to my beloved parents, especially my father, who always provided support, advice and sacrifices that motivated me to move forward. Their prayers, love and encouragement have been the greatest source of strength throughout this project. I would also like to express my deepest appreciation to my supervisor, Mr. Mohd Akmal bin Mohd Azmer, for his guidance, patience, constructive feedback and continuous support from the beginning until the completion of this project. My appreciation also goes to my examiner, Mrs. Noornajwa binti Md Amin, for her valuable comments, suggestions and evaluations that helped improve the quality of this project.

I would also like to extend a huge thank you to the Final Year Project Coordinator, Mrs. Norfazlina binti Johar, for her dedication in managing the FYP process smoothly and providing continuous assistance to all students. I would also like to express my deepest gratitude to Nurul Batrisyia binti Mohd Khir, Security Analyst at Netwitz Sdn. Bhd., who served as my project client and provided valuable insights, guidance and trust that helped shape this system into a more practical and valuable solution. Special thanks to my friends and classmates for their support, cooperation and assistance, especially during testing, debugging and presentation preparation. Finally, I would like to appreciate the facilities, resources and learning environment provided by the university, which enabled me to successfully complete this project. May all contributions be rewarded, and may this project benefit others in the future, especially in the field of cybersecurity.

1. INTRODUCTION

1.1. Introduction

This chapter introduces a project that focuses on improving website security by developing a user-friendly web vulnerability scanner. Most small companies face challenges in identifying common website vulnerabilities due to limited technical knowledge and the complexity of existing scanning tools. The project was developed for Netwitz Sdn. Bhd., an ICT solutions provider in Malaysia that offers cybersecurity services to its clients, most of whom are non-IT companies. Although Netwitz is implementing manual vulnerability scanning services for their clients, there is no system that allows clients to conduct basic scans themselves. To improve this process, the proposed solution is a web-based system that allows Netwitz clients to scan their own websites in a more automated and user-friendly manner. The system will facilitate early identification of vulnerabilities, make their clients more aware of potential web security threats, and notify Netwitz to perform deeper scans when needed. It also includes role-based access controls to separate permissions between administrators and clients. Overall, the project benefits both Netwitz and clients by increasing security awareness, empowering clients to take action, and improving the efficiency of Netwitz's service.

1.2. Project Background

Website security is a growing issue among businesses of all sizes, and the threat of cyber-attacks is increasing day by day. In Malaysia, cybersecurity breaches continue to increase, and many businesses remain vulnerable due to a lack of preparation and limited security practices (Cyber Security Asean, 2024). Small and medium business often lack technical knowledge and appropriate tools to identify common website vulnerabilities such as SQL Injection, Cross-Site Scripting, and Scripting as mentioned in all OWASP Top 10. While popular tools, such as Burp Suite, are widely used in the modern cybersecurity industry, these tools are quite complex and may not be suitable for users with limited knowledge and no technical background.

This project was developed for Netwitz Sdn. Bhd. is a Malaysian ICT solutions provider that offers cybersecurity and infrastructure services to a variety of clients, mostly non-IT companies. Currently, Netwitz performs all vulnerability scanning on behalf of its clients. However, there is no system in place to allow its clients to perform their own scans. To address this limitation, the proposed solution is to provide a web-based vulnerability scanner designed to allow Netwitz customers to scan their websites in a simple and automated manner. The system will include role-based access control (RBAC) features for customers and administrators and will display scan results in a clear and understandable format. However, this tool only offers to scan for common external vulnerabilities and not for comprehensive penetration testing or code-level analysis. The project aims to redetection and awareness, accelerate vulnerability detection, and improve Netwitz service delivery by empowering customers with a more accessible and user-friendly security tool.

1.3. Problem Statement

This section outlines the main problems that led to the development of a web vulnerability scanning system for Netwitz Sdn. Bhd. Although Netwitz provides vulnerability scanning services to its clients, currently no system is in place that allows clients to perform basic scans on their websites. In addition, there were no role-based access controls to manage different user permissions, and the scan results were too technical for the users to understand. The goal was to identify gaps in the current process and explain why a new system was needed. Each problem was discussed to show how it affected the current situation and how the proposed solution could help resolve these issues.

1.3.1. Lack of Accessible Tools for Basic Vulnerability Detection

Many companies, particularly those that do not have a special IT team, face challenges in detecting and fixing common website vulnerabilities. Existing tools like Burp Suite are commonly used by security experts and are highly technical and complicated for users from non-IT companies to operate. Because of this, clients fully depend on Netwitz Sdn. Bhd staff to manually perform scans their website and then send the report back to them. This slows down the response time and reduces the clients' ability to take fast action against threats. According to (Advisory, 2025) cyberattacks in Malaysia are increasing, and businesses need easier and more accessible ways to detect vulnerabilities before they are exploited.

1.3.2. Lack of Role-Based Access for New System Users

Additionally, Netwitz currently does not have a system that allows its clients to log in, view their scan results, or perform basic scans themselves. There is also no role-based access control (RBAC) to differentiate between what Netwitz staff or customers can access in the system. Without RBAC, it becomes difficult to grant the right permissions, ensure secure access, and monitor what each user is allowed to do. According to Neumetric (2024), RBAC is a critical part of cybersecurity because it ensures that users only have access to the data and features, they need for their role, reducing the risk of data misuse or disclosure. Without these controls, the system lacks structure and accountability.

1.3.3. Scan Results Are Hard to Understand

Most existing vulnerability scanners generate result that are too technical and difficult for users to understand. Although Netwitz's clients are from non-IT companies that may have IT departments, their teams often do not specialize in cybersecurity or web application security. Because of this, they may struggle to read and interpret the scan results correctly. According to CyberSecurity Malaysia (2023), a lack of awareness and understanding among users is one of the main reasons why many security threats are not detected or reported early.

1.4. Project Objectives

In this section, the main objectives of the proposed system are outlined. The project's mission is to develop a web vulnerability scanning system to support Netwitz Sdn. Bhd. and its clients. The system aims to improve website security management by allowing Netwitz clients to scan their websites, while still being able to notify Netwitz for deeper scans when necessary. It will also provide role-based access features and display scan results in a clear and understandable format for users to interpret and take appropriate action. The following objectives guide development of this project:

1.4.1. To develop a web-based vulnerability scanner for Netwitz clients

This objective focuses on developing a web-based system that Netwitz clients can use to perform basic scans on their websites. The system will detect common security issues such as SQL Injection, Cross-Site Scripting (XSS), and insecure headers, all of which are part of the OWASP Top 10 web application vulnerabilities (Itema, n.d.). The system also includes a feature that allows its clients to notify Netwitz staff directly through the platform if they would like to perform a more in-depth scan. By automating the scanning process, the system helps reduce the burden on Netwitz staff, enables threat detection faster, and empowers their clients to take the first step in protecting their websites while still engaging Netwitz for further analysis.

1.4.2. To implement a secure login system with role-based access control

This objective aims to create different levels of access for users in the system. Administrator users who are employees of Netwitz will be allowed to manage users, view all scan activities and system settings, while client users will only be able to perform scans and view their results. This limits each user can see what they are allowed to access, and it also reduces the risk of unauthorized access. This will make the use of the system more organized and secure.

1.4.3. To display scan results in a clear and understandable format

Many existing vulnerability scanning tools display technical reports that are difficult for users to understand. The goal is to create a visual dashboard that presents scan results using plain language, clear risk labels (e.g., High, Medium, Low), and helpful recommendations. The goal is to make the results understandable to Netwitz clients, so they can quickly take the right action and can request a more in-depth scan from Netwitz if necessary.

1.5. Scope and Target User

This project is designed to develop a user-friendly web vulnerability scanning system for Netwitz Sdn. Bhd. and its clients. The system aims to improve website security management by allowing Netwitz clients, most of whom are not IT companies, to scan their websites for vulnerabilities on their own. While most of these companies may have IT departments, their staff may lack specialized cybersecurity knowledge. By providing a simple tool, the project enables Netwitz clients to be more involved in identifying and learning about website vulnerabilities. Netwitz staff will continue to conduct more in-depth manual scans when requested by their clients, making the entire process more efficient and responsive. This dual-tier strategy makes the process more efficient as the initial scan will be handled by the client, while Netwitz focuses on advanced assessments. The system also features role-based access controls to manage and separate access between Netwitz administrators and client users.

1.5.1. Project Scope

This project focuses on the development of a web-based vulnerability scanner that allows clients of Netwitz Sdn. Bhd. to independently perform basic scans on their websites. The system will target common web vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), misconfigured headers, and others. A user-friendly interface will be developed to guide Netwitz clients through the scanning process and display simplified results that are easy to understand, especially for users without deep technical knowledge. In addition, the system will also have a notification feature for their clients to inform Netwitz when a more in-depth manual scan is required. Role-Based Access Control (RBAC) will be implemented to divide permissions between Netwitz administrators and client users, making the system easier to manage and more secure. However, the scope does not include advanced functionality such as source code scanning, internal network scanning, or integration with enterprise-level security systems. The main goal is to provide a lightweight and easily accessible tool for external vulnerability scanning.

1.5.2. Product Scope

The product developed in this project will be a web-based system that serves as a support tool for Netwitz and its clients. The web-based system will be online and accessible through a current web browser, without the need for complex installation. Their clients will be able to log in to their account, perform a vulnerability scan of their website, view the results, and notify Netwitz staff if further action is required. The focus is on automation, ease of use and readability of the output display, and functionality. While this product will allow customers to detect common web vulnerabilities, it will not offer advanced security testing capabilities such as detailed penetration test reports, automatic patch recommendations, or integration with other third-party security tools such as Nessus or Burp Suite. The goal is to bring awareness and early detection of website security vulnerabilities to enable non-technical users to participate in securing their digital assets while still relying on Netwitz for expert-level analysis when needed.

1.5.3. Target User

The primary target users of this system are Netwitz Sdn. Bhd. Netwitz employees and their clients. These users will have different responsibilities and different levels of access based on their roles.

1. Administrator (Netwitz Staff)

This user role is assigned to internal Netwitz staff who are responsible for managing the entire system. Administrators have full access to all platform features, including managing clients' accounts, assigning roles, viewing scan logs, and monitoring overall system usage. They also perform more in-depth manual scans if the customer requests further investigation. The primary responsibility of the administrator is to ensure that the system is being used safely, efficiently, and in compliance with company security protocols.

2. Client (Netwitz Client)

These users are representatives from Netwitz customer companies, typically from non-IT companies but have IT department. They are given access to perform basic website vulnerability scans on their websites, view their scan results, and receive simplified reports. They do not have administrative permissions and are unable to manage other users. If the scan detects vulnerability, they can notify the Netwitz staff through the system for further action. This role is designed to empower clients to take early steps in web security without requiring advanced technical knowledge.

1.6. Overview of This Report

This chapter provides a full introduction to the Web Vulnerability Scanner project. It begins by giving the reasons why the project was necessary and why it was needed, especially to Netwitz Sdn. Bhd. and its customers. The chapter then provides a history of website security issues and how businesses struggle to identify the most common vulnerabilities on their websites. The problem statement highlights the main problems of the existing manual scanning process such as the inaccessibility of available tools, no role-based access controls and the inability to interpret technical scan results. The project objectives are well defined to show what the system aims to achieve including the creation of a user-friendly scanner, provision of secure access through role-based access and provision of easy-to-understand scan results. Finally, the scope and target users are determined to state the boundaries of the system, the intended features and who will benefit from using the system. Overall, this chapter provides a prerequisite for the entire project, as it introduces the motivation, purpose and journey of the solution being developed.

2. LITERATURE REVIEW

2.1. Introduction

This chapter presents an overview of existing research and technology related to the development of a web vulnerability scanner system for Netwitz Sdn. Ltd. and its clients. As cyberthreats like SQL Injection (SQLi), Cross-Site Scripting (XSS), and other vulnerabilities continue to increase among many businesses, especially small and medium enterprises (SMEs) they are struggling to detect and manage these issues due to a lack of technical expertise and limited access to simplified security tools. Netwitz currently performs all scans manually for its clients, who are mostly their clients from non-IT companies, highlighting the need for a more accessible and efficient solution. This literature review aims to explore related systems, security tools, and role-based access control (RBAC) models that support scanning, reporting, and user management. The review also identified gaps in existing solutions and highlighted the methods and technologies that can guide the development of systems tailored to the needs of both Netwitz's staff and their clients.

2.2. Investigation

Web vulnerability scanners are tools used to detect vulnerabilities in websites that can be exploited by attackers. These scanners automate the process of identifying threats such as SQL Injection (SQLi), Cross-Site Scripting (XSS), insecure HTTP headers, and vulnerable server configurations. However, existing tools such as Burp Suite are either too complex or too expensive for small and medium enterprises (SMEs) (PortSwigger, 2025; Arc Journals, 2023). As a result, many small businesses are unable to perform routine scans, leaving them vulnerable to common threats. This project aims to develop a web-based vulnerability scanner that provides essential scanning functionality with simple reporting specifically designed for non-technical users and smaller organizations.

This project focuses on developing a simplified web-based vulnerability scanner tailored for SMEs in Malaysia, specifically Netwitz Sdn. Bhd's customers, who are mostly non-IT companies. The system proposed in this project allows these customers to perform basic scans on their own websites through an easy-to-use interface and scan results in a clear and understandable format. As reported by the National CyberSecurity Center (NCSC, 2024), scanning for common vulnerabilities is one of the most effective ways to improve cybersecurity, especially for organizations with limited knowledge. The tool is designed to be browser-compatible and can be used across devices, making it practical for businesses to adopt without major changes to their workflows. By focusing on minimal technical requirements, the tool can become a practical part of daily operations for Netwitz clients.

The technical development of the system uses an affordable and accessible technology stack. The front-end will be developed using HTML, CSS, and JavaScript, while Python with Flask will serve as the back-end framework. The scan results will be stored in a SQLite database and the scanning engine will be built using open-source tools such as ZAP API, BeautifulSoup, and Requests. The design follows OWASP Top 10 guidelines to ensure relevance to real-world threats. The system will also feature role-based access control (RBAC) to manage permissions between Netwitz staff and clients, ensuring proper access and system security. Based on insights from NCSC (2024), PortSwigger (2025), and Saleem et al. (2024), the project ensures that the solution is cost-effective, scalable, and relevant to the security needs of SMEs in Malaysia.

Common Threats in Web Application

In addition to understanding the structure and implementation of vulnerability scanners, it is critical to know the most common web application threats that are supposed to be identified by these scanners. Iterasec (2024) states that the most common vulnerabilities discovered in web applications are SQL Injection, Cross-Site Scripting (XSS), Security Misconfiguration, Cross-Site Request Forgery (CSRF), Broken Authentication, and Exposure of Sensitive Data. Such vulnerabilities usually arise from insecure coding methods, neglect of regular updates, or poor security policies during web application development. To provide some indication of the prevalence and applicability of these vulnerabilities, Table 2.1 provides a summary of the occurrence of widely detected vulnerabilities in web applications from recent industry reports:

Table 2.1 Distribution of Common Web Application Vulnerabilites (Iterasec,2024)

Vulnerability Type	Frequency (%)
SQL Injection (SQLi)	18%
Cross-Site Scripting (XSS)	21%
Security Misconfiguration	16%
Cross-Site Request Forgery	10%
Broken Authentication	14%
Sensitive Data Exposure	12%
Others	9%

2.2.1. Definition SQL Injection

SQL Injection (SQLi) is a well-known and perilous web security threat that occurs when an attacker executes a malicious Structured Query Language (SQL) query by inserting contaminated input in a web form field or a URL parameter. On the server, the input is run by the server-side application, which enables the attacker to gain unauthorized access, as well as control, to the database. Imperva (2024) says that SQLi may enable the attacker to access sensitive information, edit or delete information, as well as gain full administrative access to the system in some cases. This weakness normally arises when the user input is not validated and sanitized appropriately prior to its application in SQL queries.

SQL Injection attacks take a special place as they target the data layer of the application core. According to Acunetix (2024), these attacks are mostly directed at web forms such as login pages, search boxes, feedback forms, among others, where the data entered by the users is transferred to a database without much care. The attacker can insert segments of SQL code that can alter the format of the initial query. As an example, such input as `admin' OR '1'='1` may be provided to bypass the authentication to gain access to restricted parts of the site. This implies that robust input validation and parameterized queries should be considered as the basic protection mechanisms.

2.2.2. SQL Injection Prevention

SQL injection (SQLi) is a serious security risk where the attacker is able to exploit the flaws in the web application by injecting malicious code into some fields like login forms or search boxes. They explore the target system and identify the locations where user input is linked to the database and test the locations through special custom-designed code to determine the response of the system. In the case of success, attackers can steal sensitive data, change the records, or take complete control of the server. The prevention strategies must incorporate the use of either parameterized queries or prepared statements, which separate the user input with the direct database commands (OWASP, 2024). The validation and sanitization of input should also be applied to prevent the use of malicious characters like quotes or different symbols, and the least privilege principle should be utilized when assigning minimum access privileges to database users. Web Application Firewalls (WAFs) can be used to spot and prevent suspicious operations, and companies are recommended to perform scans as well as penetration tests to discover weak points in time (Acunetix, 2024). All these measures work to prevent the easy compromise or system destruction by attackers.

2.2.3. Type of SQL Injection

SQL Injection (SQLi) attacks come in many forms, and each has its own techniques and goals. This category cannot be fully understood without successfully implementing a detection and mitigation strategy. Differentiations are created depending on how the data is retrieved and the level of interaction required with the application. As stated by Acunetix (2024) and SentinelOne (2024), the three most frequently mentioned classification categories include in-band SQLi, inferential SQLi, and out-of-band SQLi.

- i. In-Band SQLi: This is the most common and simplest form of SQL injection. It occurs when the same communication channel is used to launch and retrieve the results of the attack. There are two main subcategories
 - a. Error-Based SQLi: In this type of attack, the attacker makes the database generate error messages to obtain information such as the database version or table name.
 - b. Union-Based SQLi: This can be done through the SQL UNION keyword which allows the results of multiple queries to be combined, thus allowing the attacker to retrieve information stored in heterogeneous tables.
- ii. Inferential SQLi (Blind SQLi): Unlike in-band SQL injection which sends data directly to the attacker, inferential attacks rely on observing the application's response to external inputs to obtain information.
 - a. Boolean-Based Blind SQLi: The attacker injects a query that returns true or false and then monitors the application's response.
 - b. Time-Based Blind SQLi: This process takes advantage of the delay time taken by the server to respond and set the truth of the statement by intentionally causing a delay using SQL functions such as SLEEP() to create a delay.
- iii. Out-of-Band SQLi: It is used when the attacker cannot use the same channel to initiate and access the attack data. It exploits the server's ability to send DNS or HTTP requests to a machine under the attacker's control. It is rare, but easy in situations where the usual attack channel is disabled, but outbound connections are possible to the database server.

In short, various styles of SQL injection require multi-layered defense mechanisms. By gaining knowledge about the nuances of each type of variation, organizations will be able to adjust their security measures more precisely and detect malicious activity with a higher accuracy rate. In such a situation, web vulnerability scanners should be coded in such a way to identify all variants to provide complete protection against SQLi attacks.

2.2.4. Definition of Cross-Site Scripting

A cross-site Scripting (XSS) vulnerability in web security allows attackers to slip harmful code into pages sent to clients. Most often, the payloads are JavaScript code and run within the user's browser by passing the barrier of the same-origin policy. Simply put, an application is vulnerable when it contains untrusted data in a web page that is not validated or escaped on the application, which means the attacker can execute scripts, causing hijacking of the user session, redirecting the user to some evil website, or stealing confidential information.

PortSwigger (2024) indicates that XSS occurs anywhere on the user side, rather than on a server, whereas when it comes to web applications dealing with user-entered content like in comments or search box, it is particularly harmful. Hackers develop payloads that, after being executed, allow conducting a broad spectrum of illicit activities, such as mounting phony log-in forms, to stealing characters. According to Acunetix (2024), the danger of XSS is grave since one injection can impact the thousands of users, particularly in applications that are visited by many people.

2.2.5. Cross-Site Scripting Prevention

The XSS attack process begins with an attacker discovering a susceptible web input that does not sanitize or encode output. An attacker then injects an injected script, commonly JavaScript, into a user input field which shows user input. When another user loads the page, the script executes within their browser and behaves as if it were coming from a trusted website.

- a. Injection: Hand-crafted script injected into one of the vulnerable fields of entry.
- b. Storage or Reflection: Script cached (stored XSS) or returned straight back to the victim (reflected XSS).
- c. Implementation: The script has the same privileges as the browser when directing it to be executed upon loading the page by the user.
- d. Exploitation: Targeted thefts of cookies, capture of inputs, and user redirection.

To prevent this, OWASP (2024) and Veracode (2024) recommend a combination of output encoding, input validation, and modern security headers. Output encoding converts the input from users to plain text rather than executable code. Content Security Policy (CSP) comes to the rescue and prevents malicious scripts. It is also important that developers abandon risky functions such as `eval()` and properly filter HTML with reputable libraries

2.2.6. Type of Cross-Site Scripting

Cross-site scripting (XSS) is a form of code theft wherein the trust the user gives is hijacked and replaced by some evil snippets of code being inserted into otherwise legit snippets of code in a web application. There are also 3 main flavors that should be taken into consideration, namely Stored XSS (Persistent XSS), Reflected XSS (Non-Persistent XSS), and DOM-Based XSS. The commonality behind them all is the discovery of a vulnerability in the manner that web applications handle input and output by the user, and all aimed at the same ultimate objective, to exploit the end-user.

- i. **Stored XSS (Persistent XSS):** The malicious script is kept by the server permanently, and most frequently in the database. Whenever one visits the infected location, the server sends back the script. This version may be applied to cover an extremely broad audience as every visitor speaks the same script.
- ii. **Reflected XSS(Non-Persistent XSS):** In this, the payload is not permanent, this is merely reflected to the server nearly immediately after the payload has been received. You can think of it as an unintentional benefit in an error message or a search outcome that includes some of the initial requests. To use one, the user is usually expected to click on a specially made link.
- iii. **DOM-Based XSS:** In this XSS all is on the client. Any information is not sent to the browser, but the attacker manipulates the Document Object Model (DOM). The script can also be run when it is within the browser although it can be filtered in the server.

All these types of XSS are different risk levels as it depends on the way the web application receives and portrays data. The countermeasure, which is to be put up to prevent XSS, should consider client-side protection and server-side protection.

2.3. Related Works

In this section, some of the popular web vulnerability scanning tools with similar functionality to the proposed system are described, which are OWASP ZAP, Wapiti and BurpSuite. They are widely applied to identify threats like SQL Injection and Cross-Site Scripting according to OWASP Top 10. They are effective but usually need a high level of technical expertise and therefore may not be recommended to non-technical users or small companies that do not have an IT security team. As an example, OWASP ZAP has powerful functions but is complex to set up and Wapiti is simple but uses a command-line interface and BurpSuite is a sophisticated tool and displays results mainly in a technical format. Through the analysis of these tools, the project found that this was a need for a simpler web-based scanner that would support automated scanning, clear reporting, and role-based access control according to the needs of Netwitz clients.

2.3.1. OWASP ZAP

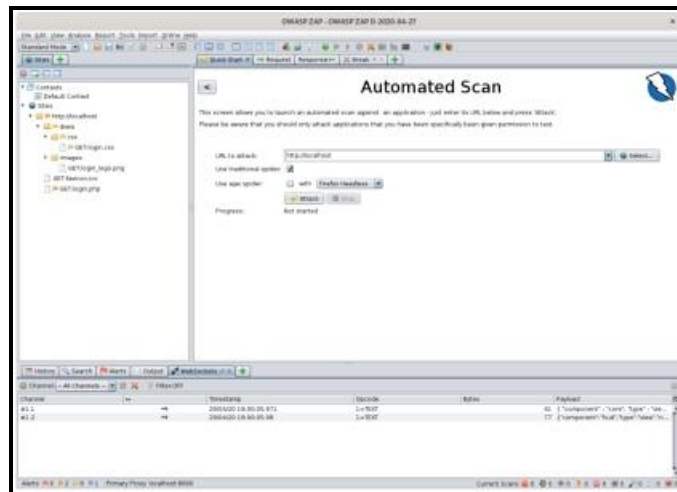


Figure 2.1 OWASP ZAP

OWASP ZAP (Zed Attack Proxy) is a top-rated, open-source web application vulnerability scanner project of the Open Worldwide Application Security Project (OWASP). ZAP is aimed at both experts and beginners, and it is used to identify vulnerabilities in web applications during development and testing phases (Wallarm,2024). When working, ZAP acts as an intercepting proxy between the browser of the user and the target web application. It allows exploring the behavior of the application in detail, modifying the requests and responses by examining HTTP and HTTPS traffic. The application offers passive scanning, which inspects requests without altering traffic, and active scanning, which actively tests the application for known vulnerabilities (OWASP Dorset, 2020).

ZAP features such as spider crawling of web applications, fuzzing, WebSocket scanning, session management, and REST APIs for integration into CI/CD pipelines. ZAP also includes scripting support so that users can write custom scan rules and automate workflows (StackHawk, 2024). On the security control front, ZAP utilizes OWASP Top 10 vulnerability attack vectors with coverage of leading threat categories such as SQL Injection and Cross-Site Scripting (XSS).

Though powerful, ZAP is not always easy to set up and might be too much for individuals without a background in penetration testing paradigms. Furthermore, as extensive as its functionality is, the interface can be less intuitive for users without technical backgrounds. But the project under discussion sidesteps such usability issues by providing both a simplified graphical user interface and an in-built user role management system so that even companies without extensive cybersecurity knowledge can perform scans.

2.3.2. Wapiti



Figure 2.2 Wapiti

Wapiti is an open-source, command-line scanner designed to identify web vulnerabilities by conducting black-box testing aimed at ascertaining the security of web applications. As opposed to those analyzing source code, Wapiti operates by simulating the activity of an external user on the application, injecting payloads to test potential vulnerabilities like SQL Injection, Cross-Site Scripting, file disclosure, and command execution (Wapiti Project, 2024). It first crawls the target website to collect links, forms, and parameters, and then launches automatic attacks with pre-defined payloads to detect vulnerabilities.

Some of the features of Wapiti are fuzzing of URLs, analysis of attack parameters, user-definable payload sets, and extensive HTML reports. It can handle SSL sites as well as cookies and basic authentication, increasing its compatibility with contemporary web applications. Its power lies in how it is extremely simple and light-weighted, being a valuable addition to automated security processes (LinuxSecurity, 2024). Nevertheless, Wapiti lacks a graphical user interface or a sophisticated dashboard for the visualization of scanning results, which could restrict its usage among individuals lacking expertise.

Regarding security scanning, Wapiti is favoured of a modular, attack-based design and has ongoing list of the known vulnerabilities. However, its major flaw is in the fact that it does not provide real-time monitoring of threats and cannot use a session-based scan or anything beyond unauthenticated scan. This project aims to overcome these gaps by including session management, real-time reporting and by providing a simpler, web-based interface to simplify the use and make the tool more convenient and viable among small and mid-sized organisations.

2.3.3. Burp Suite



Figure 2.3 BurpSuite

Burp Suite is an effective web vulnerability scanning tool mainly employed by security experts and ethical hackers to find and exploit web vulnerabilities. It is characterized by a set of built-in tools, including the proxy, scanner, intruder, and repeater, that enable users to intercept and alter HTTP traffic, automate scans, and manually test vulnerabilities, including SQL Injection, Cross-Site Scripting (XSS), and authentication vulnerabilities (Vaadata, 2023). GeeksforGeeks (2024) states that Burp Suite is efficient when it comes to simulating real-life attacks and it is widely utilized in penetration testing and, therefore, is a tool of choice among the professionals.

Nevertheless, regardless of its potential, Burp Suite cannot be considered the right tool to use in case of non-technical users or small companies because of its complicated interface and high learning curve. Its setup, manual procedures, and technical reporting demands a fair grasp of cybersecurity concepts, which most SMEs do not have. In addition, even though a free version exists, the advanced features like the automated scanner can only be found in the paid professional version which can prove costly to smaller organizations. These shortcomings indicate the necessity of the more user-friendly and less expensive tool, like the one suggested in this project.

2.4. Comparison

This section will provide a comparative analysis of 3 web vulnerability scanners, OWASP ZAP, Wapiti and Burp Suite. The tools find extensive applications in cybersecurity to find and evaluate the flaws in web applications. All tools have certain advantages, such as OWASP ZAP is associated with automated scanning and the flexibility of open-source tools, Wapiti is distinguished by a simple and lightweight command-line interface, and Burp Suite is characterized by the strong capabilities of manual and semi-automated testing. These tools however have some limitations. As an example, Burp Suite might be too technical and expensive to be used in smaller organizations, Wapiti does not have a graphical interface that makes it less usable by non-technical users, and OWASP ZAP, despite being easy to use, might still be too intimidating to new users because of the number of features available. Table 2.2 below outlines the comparison among these tools based on key evaluation criteria, highlighting the gaps that the proposed system aims to address particularly simplicity, accessibility, and suitability for small to medium enterprises (SMEs)

Table 2.2 Comparison of Existing Project

Criteria Project	OWASP ZAP	Wapiti	Burp Suite
Features	Automated scanning, active & passive scans, reporting, API support, spidering, scripting	Automated scans, command-line, supports various attacks, simple reporting	Advanced scanning, manual and automated testing, detailed reports, extensible with plugins
Users	Good for people with some IT knowledge, but also used by security experts	Best for people comfortable with command-line and technical tasks.	Mainly for security professionals and experts.
Navigation	GUI and API, menu-driven, some complexity for new users	Command-line, minimal interface, not user-friendly	Powerful GUI, many options, steep learning curve for beginners
Security Element	Covers OWASP Top 10, customizable scans, frequent updates	Covers many vulnerabilities, focuses on core web attacks	Covers almost all known web vulnerabilities, deep analysis, customizable
Result	Gives reports that show risk levels and suggestions, can export in different formats.	Gives simple reports in text or basic HTML.	Gives full, detailed reports with advice for fixing problems.

2.5. Discussion

After analyzing the available tools like OWASP ZAP, Wapiti, and Burp Suite, the current project was mostly influenced by the combination of automated and manual capabilities of OWASP ZAP. However, it is aimed to simplify such functions to small and medium enterprises (SMEs) and, in particular, to the clients of Netwitz Sdn. Bhd., many of whom do not have strong technical skills. One of the main design properties is the ability to perform automatic and manual scans. This two-fold architecture allows the user to leave the scanning part to the system or to examine certain aspects manually. This dual approach is useful for both beginners and more experienced users.

The system will also present scan results in simplified format. Though scanners like OWASP ZAP and Burp Suite normally produce highly technical reports, which are difficult to understand by a non-technical user, the proposed scanner will convert these reports into understandable and clear language. Such functionality enables users not only to determine the problems but also to recognize the risks levels of each vulnerability. In addition, the system will give a step-by-step fix guide to each vulnerability identified. This aspect is aimed at empowering the users because they can take direct informed action to solve the problems for users to understand. These improvements make the system more viable and approachable, particularly to SMEs that need to improve their web security stance with limited resources available.

2.6. Conclusion

In conclusion, this chapter has reviewed the leading research, tools and approaches specific to web vulnerability scanning, with a particular emphasis on small and medium-sized enterprises (SMEs). The investigation highlights how web-based threats such as SQL Injection (SQLi) and Cross-Site Scripting (XSS) continue to pose serious risks, especially for organizations without advanced cybersecurity systems. Analysis of tools such as OWASP ZAP, Wapiti and Burp Suite demonstrates that, while these tools are powerful, they may not be suitable for non-technical users due to their complexity and technical reporting formats.

A detailed comparison of the tools reveals various strengths and limitations, from the advanced scanning capabilities of OWASP ZAP to the lightweight design of Wapiti. However, there remains a significant usability and accessibility gap for SMEs. Inspired by OWASP ZAP, the proposed system will combine core scanning functionality while simplifying the user interface and presentation of scan results. The most important features including automatic and manual scanning, scan result explanations, and actionable fix guides have been selected to make the vulnerability detection process more accessible and helpful to Netwitz Sdn. Bhd. clients. Furthermore, the literature review has provided a good foundation for the design and development of a user-friendly, cost-effective vulnerability scanner to meet the needs of SMEs in Malaysia.

3. METHODOLOGY

3.1. Introduction

This chapter explains the methodology used by Web Vulnerability Scanner system. The system will help the Netwitz clients to detect the security threat on their websites. An agile development methodology was chosen, considering the goal of this project, because it allows making iterative adjustments based on the results of test and feedback of the user. Agile Methodology allows the project to be developed in a manageable small stage which is requirement gathering, analysis, design, development, test, deployment and review process and the specific intent of these stages is to ensure that each stage of development meets the needs of the user along with the project objective.

3.2. Agile Methodology



Figure 3.1 Agile Methodology Diagram (Redirect Notice, n.d)

Agile approach is selected to this project as it is iterative and flexible, it can be used in the creation of the Web Vulnerability Scanner. Agile operates on the principle of breaking the development process into small parts that can be developed, tested and refined individually. This makes it easier to fix problems early, adjust to user feedback, and add new features when needed. Unlike the traditional methods which require a strict fixed-plan-based schedule to follow, Agile recognizes that projects need to be easily adjustable. It is much more beneficial when compared to the conventional working process as it allows the development team to concentrate on the needs of the users, especially the needs of the small business owners and individuals with a low level of technical expertise. Agile development also encourages teamwork, regular updates, and an urgent delivery of the results to enable users to test the system and give feedback. This approach helps ensure that the system is useful, easy to use, and effective in identifying website security issues (Atlassian, 2024).

3.3. Phases in Agile Methodology

The Agile methodology applied in this project is structured around six main phases which are Requirement, Design, Development, Testing, Deployment, and Review. These phases are cycled repetitively so that ongoing improvement is possible in development. Each phase is designed to produce a functioning portion of the system or its improvement derived from feedback. This renders the system user-focused, responsive to evolving needs, and increasingly improved with each cycle.

3.3.1. Requirement

In the Requirements phase, the focus was on gathering and identifying the required features that a Web Vulnerability Scanner system must provide to meet the needs of Netwitz Sdn. Bhd. and its clients. The main goal was to provide a user-friendly, automated and easy-to-understand scanning tool for clients from non-IT companies who often lack the resources or technical skills to use complex vulnerability scanners. Informal interviews and an online questionnaire were conducted to make it more relevant. The main requirements proposed were to have secure user login, role-based access control, a scanning engine that can detect common vulnerabilities such as SQL Injection (SQLi) and Cross-Site Scripting (XSS) and easy-to-understand scan results. In addition, the system should allow users to notify Netwitz if a deeper scan is required.

3.3.2. Design

The Design Phase outlines how the system will function and appear to the end user. This involves the development of a clear and user-friendly user interface, with separate pages for login, dashboard, scanning module, scanning results and role-based management. The design also includes the development of a system flowchart, use case model, data flow diagram and Entity Relationship Diagram (ERD) to illustrate both data relationships and workflow. Special attention was paid to ensure that the system was simple enough for users with limited knowledge and administrators. The modules will be arranged in such a way that they ensure secure access, easy navigation and correct data representation. The final deliverables include interface mock-ups, flowcharts and technical blueprints that will be used in the next development phases.

3.3.3. Development

The implementation of the system during the development phase is done using a complete stack layout. HTML, CSS, Bootstrap, and JavaScript are used to create the frontend to provide a responsive and easy-to-use interface. The backend is developed with Python and Flask framework to handle logic processing, user authentication, and routing. The scanning engine has libraries including requests, re, BeautifulSoup, and urllib.parse. Besides, OWASP ZAP API is incorporated to enhance vulnerability discovery. The user data and scan records are stored with the help of SQLite as the database. The development process is an evolution one where each functionality is tested and refined before moving to the next one.

3.3.4. Testing

The Testing stage helps to check each component of the system and ensure that it operates properly and meets user expectations. Functional testing is used over the scan-report modules, scanning engine, and the login system. Usability testing is used to test whether the interface is simple, fast and can be used by people who do not have technical knowledge. System integration testing ensures an effective flow between the backend, frontend, and database. All bugs and errors are correctly recorded, resolved and retested. The responses of Netwitz staff and its clients are taken to make adjustments to enhance the user-friendliness of the system and its general reliability.

3.3.5. Deployment

The Deployment phase involves actual use of the system in a hosting or local server solution. It tests all features, including user login, scanning, and dashboards in a live environment. The system is accessible via a standard web browser and supports multiple devices. It includes a user manual to guide using the program to log in, perform scans, and interpret results. Deployment also involves setting up server configurations and ensuring secure database connections for system operation

3.3.6. Review

The Review Phase consists of collecting and analyzing user feedback after the system is deployed. Netwitz staff and their clients can share their experiences using this system through informal surveys or reviews. The data were collected during this process can allows for the identification of usability issues or unmet needs that were not recognized during previous testing. The review also assesses how well the system meets its goal of providing an accessible scanning tool for SMEs. This phase is used to plan further updates and enhancements, ensuring that the system will continue to improve in line with user needs and new challenges in the cybersecurity field.

3.4. Conclusion

In conclusion, agile methodology adopted in the creation of the Web Vulnerability Scanner was a good and appropriate move in this project. It was iterative and flexible that made it possible to develop the system in manageable phases and to continuously improve the system, depending on the results of the tests and the feedback of the users. This flexibility was particularly useful in the application of the features of security like the SQL Injection and Cross-Site Scripting (XSS) detection which require accuracy and stability. Requirements gathering was a stage that was well planned and implemented to make sure that the project did not exceed the scope and meet the expectations of the user. The consideration of end-user needs, the choice of suitable technologies, such as Html, CSS, Bootstrap, JavaScript, Python with Flask, SQLite and the OWASP ZAP API, helped the project to find the compromise between technical functionality and accessibility by its users. In general, Agile gave a framework and freedom needed to develop a feasible, secure, and user-friendly web vulnerability scanner.

4. REQUIREMENTS

4.1. Introduction

This chapter explains the process that was needed to identify and collect system requirements in developing a Web Vulnerability Scanner. The requirements have been grouped into two which is functional and non-functional. Functional requirements explain what the system should be capable of doing such as scanning websites against vulnerabilities whereas non-functional requirements relate to how the system should perform in terms of usability, reliability and efficiency (Beaton, 2024). To create the system based on real user expectations and business needs, there are 2 data gathering techniques that were applied in this chapter, a structured questionnaire and interview. The techniques provided helpful feedback to guide the characteristics, behaviour, and limits of the system to make it usable and safe for the end users.

4.2. Data Gathering Techniques

Data collection is an important process of gathering information used in the design of a system and this makes the system consistent with the real needs and expectations of the users. It ensures the system's functionality is effective in operation, user focus and the ability to solve material problems by using systematic practice of practical techniques that can generate feedback and insights that must be provided by the user community (Chu, 2024).

4.2.1. Questionnaire

An online questionnaire was distributed by using Google Forms to gather feedback users. The questionnaire consisted of 15 multiple choice questions to determine the level of user awareness about site security, experience working with vulnerability scanners and desired system features. The questionnaire was created in the form of closed-ended statements to allow an easier and consistent analysis of responses. To ensure the reliability and meaningfulness of the results, the questionnaire targeted a response number of at least 50 users. This approach was especially productive in reaching many users in a short and efficient way.

4.2.2. Interview

To facilitate the process, an interview session was held with the client together with the administration of a questionnaire. It was an important meeting in terms of the validation of the problem statement of the project, clarification of the objectives, and identification of the specific business needs of the client. The interview involved asking specific questions to determine the characteristics that the client expected, the client concerns about web security, and how preferably the vulnerability reports were to be given. The following key system constituents were defined and verified by the interview process, the possibility to log in to the system, capabilities of scanning, and the delivery of vulnerability reports in a format that is easily understood for user.

4.3. Functional Requirement

The main functional requirements of the Web Vulnerability Scanner system are described in the following section, and the key features and activities that the system should support to meet the requirements of Netwitz Sdn. Bhd. and its clients. The system aims to allow the clients to carry out basic scans, read the results with ease and notify Netwitz when there is need to perform higher level scanning. It also provides multi-level access controls to regulate the scope of the privileges of every user in the system. According to GeeksforGeeks (2025), functional requirements are essential in defining what the system must do in terms of operations, user interactions, and data handling. The table below summarizes the key functional requirements of the system:

Table 4.1 Functional Requirement for User

Function	Requirement
Website Vulnerability Scan	Users must be able to enter a URL and perform a basic vulnerability scan for common issues such as SQL Injection (SQLi), XSS, and insecure headers.
View Scan Results	Scan results must be displayed in a clear and simplified format with severity levels (High, Medium, Low).
Notify Netwitz for Deep Scan	User can request a detailed manual scan from Netwitz through the system.
Scan History	Clients can view only their own scan history and previous results.
Dashboard Interface	User can view a dashboard that showing total scans, completed scans, and company information.
Profile	User can update their personal details and upload a profile picture.
Download Report	User can download their own scan reports in supported formats

Table 4.2 Functional Requirement for Admin

Function	Requirement
User Management	Admin can create, update, deactivate, and delete user accounts, including assigning roles and organization.
View All Scan Activities	Admin can view all scans performed by any user under their organization.
Dashboard Interface	Admin sees a dashboard with total organization scans, total staff, company statistics, and system-wide summaries.
Monitor Deep Scan Requests	Admin can view user deep scan requests, attach manual scan reports, and send responses.
Profile	User can update their personal details and upload a profile picture.
Download Report	User can download their own scan reports in supported formats

4.4. Non-Function Requirement

The non-functional requirements are used to outline how the Web Vulnerability Scanner would operate to ensure that it is practical, secure and reliable in the real-world implementation. The functional requirements include the tasks the system can perform, and the non-functional ones determine the effectiveness of the task performance. Since the system would be implemented in Netwitz Sdn. Bhd. and among its non-technical clients, it should be user-friendly, return results quickly, and protect user information. These requirements ensure the system is not only working but also convenient, secure, and dependable for daily use.

Table 4.3 *Non-Function Requirement Table*

Category	Requirement Description
Performance	The system should produce scan results within a reasonable time to avoid user frustration and ensure efficiency.
Usability	The interface must be simple, clean, and easy to navigate, especially for users with limited cybersecurity or technical background.
Reliability	The system must perform scans consistently without crashing, missing data, or producing false results.
Security	All user data, including login credentials, scan history, and reports, must be encrypted and securely stored to prevent unauthorized access or data leaks.

4.5. System Requirement

This section provides the hardware and software requirements of developing and executing the Web Vulnerability Scanner effectively. The system is a web-based system, which needs a stable development environment, appropriate backend and frontend technologies, and a lightweight database to store users and scan information. The system will be developed, tested and hosted in a local development environment.

4.5.1. Visual Studio Code



Figure 4.1 Visual Studio Code (Canonical, 2019)

The integrated development environment (IDE) that has been selected to be used in this project is Visual Studio Code (VS Code) because it is lightweight and has a large variety of features and supports a wide variety of programming languages and extensions. It has strong capabilities like syntax highlighting, IntelliSense, built-in terminal, Git support, and debugging, which is necessary in the effective development of web and Python. It is compatible with HTML, CSS, JavaScript, and Python, and therefore it is the best choice when it comes to the development of Web Vulnerability Scanner, which combines the front-end and the back-end components. Besides, VS Code has a vast extension library, which makes it easy to integrate with other tools, including Flask and SQLite, to have a more efficient development process (Staff.C, 2025).

4.5.2. Python



Figure 4.2 Python (File:Python Logo and wordmark.svg - Wikimedia Commons, 2008)

Python is a popular high-level programming language that is simple, readable, and flexible, which is why it is suitable both for beginners and to professionals. In this project, Python is the main backend language that drives the engine of web vulnerability scanning. Due to its rich ecosystem of libraries such as requests, re, BeautifulSoup, and urllib.parse, the system can process and analyze the content of websites efficiently to detect vulnerabilities. It is also selected because Python has good community support and integrates with security testing tools like Wapiti and OWASP ZAP API. As it is emphasized by Coursera (2023), Python particularly fits web development and automation practices, which are consistent with the technical requirements of this system.

4.5.3. Flask



Figure 4.3 Flask (Iqbal, 2025)

Flask is a small and lightweight Python web framework and is a perfect framework to create web-based applications efficiently. The main backend framework applied in this project is Flask, which is used to control the routing, handle HTTP requests and bind the user interface to the scanning engine and the database. It is a modular code organization and can be integrated with other Python libraries, which is a reason to use it to develop a system such as the Web Vulnerability Scanner. It has a minimalistic structure and is simple, allowing the developers to concentrate on the fundamental functionality without being complicated. GeeksforGeeks (GeeksforGeeks, 2025a) listed the tools and features that Flask offers to create scalable and secure web applications, which is why it is a reasonable option to use in this project.

4.5.4. OWASP ZAP API

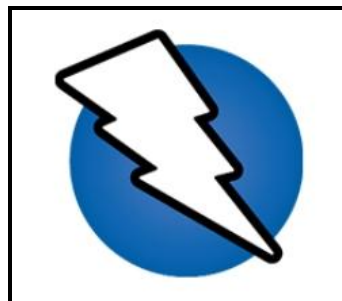


Figure 4.4 OWASP ZAP API OWASP Zed Attack Proxy Scan - Visual Studio Marketplace, n.d.)

OWASP ZAP (Zed Attack Proxy) API is a free-of-charge security tool that is created to identify web application vulnerabilities. As an optional third-party security scanner that can be used in this project, OWASP ZAP API can be employed to augment the identification of the threats, such as SQL Injection (SQLi) and Cross-Site Scripting (XSS). When the API is combined with Python, the system will be able to conduct automated security tests and to produce reports regarding possible vulnerabilities of a particular web page. The API is an easy means of performing security scans programmatically, which is in line with the general objective of developing a trustworthy and easy-to-use Web Vulnerability Scanner. As noted by Plain English Kalkman (2023) the ZAP API enables programmers to automate pentesting tasks, so it can be particularly helpful when continuously locating and mitigating security weaknesses throughout the development process.



Figure 4.5 SQLite (SQLite Tutorial - an Easy Way to Master SQLite Fast, 2024)

SQLite is an in-process library that contains a self-contained, serverless, zero-configuration, transactional SQL database engine. It stores data in a single file on disk, making it easy to set up and manage without requiring a separate server process. In the current project, SQLite will be used as the storage of user authentication information and scan history, log activity and vulnerability reports. Its lightweight installation requirements and ease of integration with Python make the engine a good choice when the project is looking to have an easy-to-use database that is nevertheless dependable. According to (Klie, 2025) discusses, SQLite has become widely accepted due to its strong performance and the fact that it does not require a lot of resources, the traits that make it especially applicable to small and medium-sized projects.

4.5.6. Laptop



Figure 4.6 Hardware Specification

Hardware Requirement	Specification
Computer Brand	Asus Tuf Gaming A15
Processor	AMD Ryzen 7 7435HS
Memory (RAM)	32 GB
System	64-bit operating system, x64-based processor
Operating System	Windows 11 Home Single Language
Storage	512GB

4.6. Conclusion

In conclusion this chapter has presented the major requirements that are required to make sure that the Web Vulnerability Scanner system is developed successfully. Through the use questionnaires and interview, functional and non-functional requirements were determined to make the design of the system correspond to the needs and expectations of users. The functional requirements center on the main behavioral components like scanning of vulnerabilities, user authentication, and results reporting whereas the non-functional requirements involve usability, performance, and security. Also, the required system tools and technologies, such as Visual Studio Code, Python, Flask, and the use of OWASP ZAP API, were mentioned in this chapter. The gathered information and the tools chosen are a basis to create a safe, productive and easy to use application.

5. ANALYSIS

5.1. Introduction

This chapter describes the analysis stage of the Web Vulnerability Scanner project, which is devoted to the investigation of the systems requirements and the transformation of them into visual models. This section starts with the presentation of the data gathering using questionnaires in the form of pie charts to get a better idea of user expectations and preferences. To explain the system structure and user interaction further, there are two major models applied which are a flowchart and a use case diagram. The flowchart describes the process of operations in the system, and the use case diagram presents the primary functions that users will use. These diagrams help in keeping the system design on track with the functional objectives as well as the user requirements. Every diagram is accompanied by an explanation to give a better insight of how the system behaves both technically and on the user side.

5.2. Data Gathering Analysis

Data gathering was carried out as an important process during the initial project phases to make sure that the system developed would meet the expectations and the practical needs of the users. Structured questionnaire to those users who might own or manage websites was employed as the main method. The technique was selected because it can gather a large variation of responses quickly and it is also applicable when the responses of many users are required in a very short time. The gathered information assisted in designing the characteristics of the system, user interface, and scanning capabilities, making them relevant and easy to use.

5.2.1. Questionnaire Analysis

The questionnaire contained 15 close-ended questions that were sent out using Google Forms and 50 responses were required as the minimum. The questionnaire was designed to determine the level of knowledge of the users, their experience in the management of the websites, security issues, and preferences in the vulnerability scanning tools. The answers were used to inform major decisions about the functionality, usability, and prioritization of the features of the system

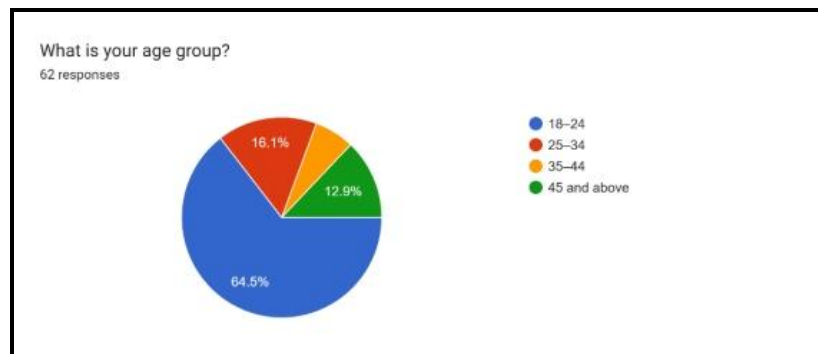


Figure 5.1 Question 1

Most of the respondents surveyed were aged between 18 and 24 (64.5 %), and the second-largest group consisted of 25- to 34-year-old respondents (16.1 %). The number of participants aged 45 and more was 12.9 % and the number of those aged 35-44 was 6.5 %. These results indicate that the system will most likely be well received by younger customers who spend much time online or otherwise occupied by digital activities. By determining this overwhelming age group will help the design team to focus on aspects of functionality that will directly meet the demands of this group.

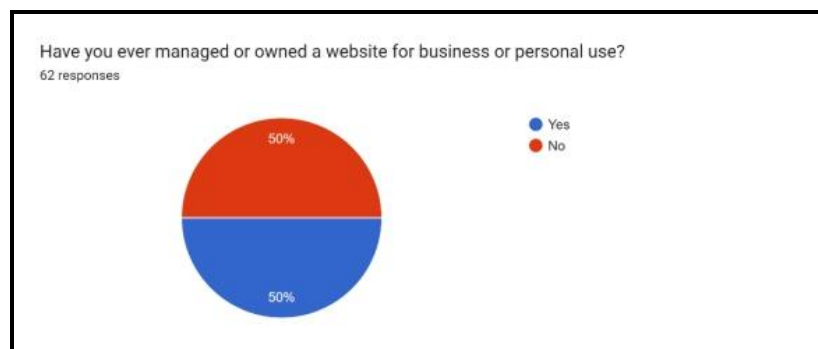


Figure 5.2 Question 2

The survey showed that, precisely, 50 % of the respondents have experience of managing or owning a site and the other 50 % do not. The results show the user includes experienced users and beginners. It is on this premise that the system being developed should meet two competing goals, the system should be simple enough to be used by new users and at the same time functional enough to be used by experienced ones.

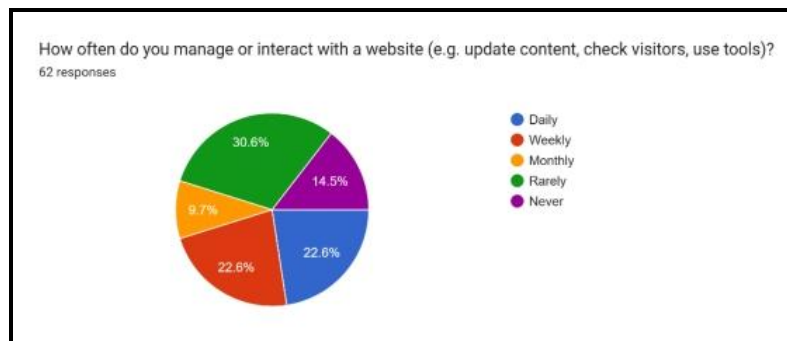


Figure 5.3 Question 3

The survey showed an inconsistent trend of the monitoring of websites, 22.6 % of them interact with their websites daily or weekly, 9.7 % interact once a month, 30.6 % rarely, 14.5 % never. These statistics highlight the fact that a good number of users do not check their sites regularly and this makes it clear that there is a need for an automated scanner that will provide updates without the user having to check the sites regularly.

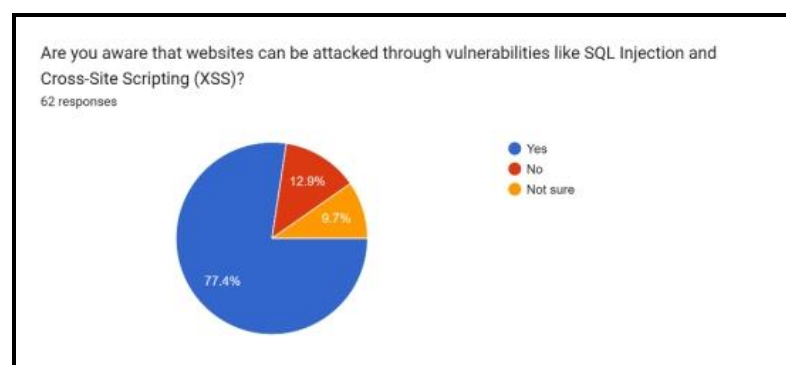


Figure 5.4 Question 4

The large majority (77.4%) knows of such attacks, 12.9% do not know and 9.7% are unsure. These statistics prove that most users understand the importance of web security, but there is still a need to breach a gap in the knowledge of a few users, that is why the presence of educational elements and clear explanations are necessary across the system.

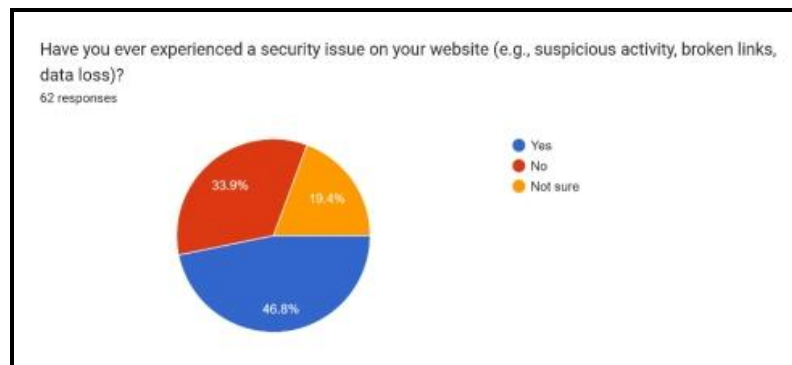


Figure 5.5 Question 5

46.8% of the respondents have experienced some security problems and 33.9% have not, and 19.4% are not certain. This shows how serious the threat that users are exposed to and prove the need of a reliable vulnerability scanner that would prevent such incidents.

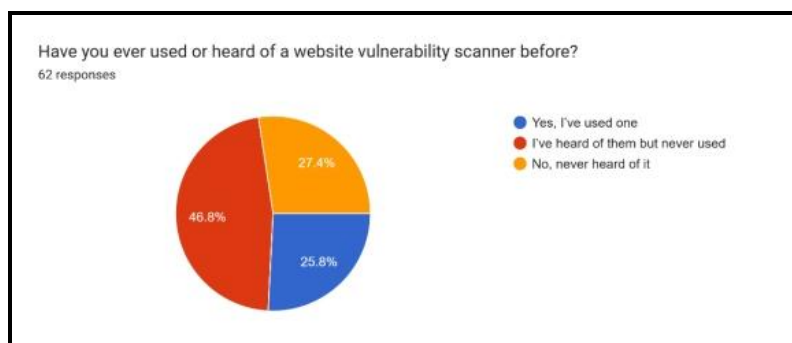


Figure 5.6 Question 6

Based on the chart above, only 25.8% have used one, while 46.8% have heard of it but have not used it and 27.4 percent have never heard of it at all. This shows the unfamiliarity or unavailability of such tools and explains why it is necessary to create user-friendly and accessible solutions that are targeted to a wider audience.

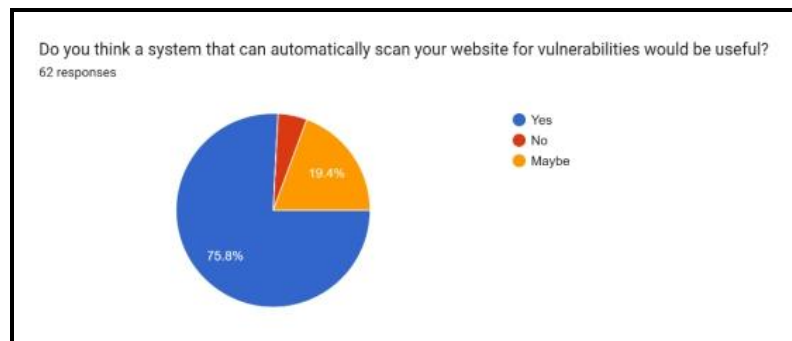


Figure 5.7 Question 7

Based on the questionnaire, when asked whether such a system that could automatically scan websites for vulnerabilities was a good idea, 75.8% of those surveyed chose the option to say Yes, 19.4% chose Maybe and 4.8% chose No. The results show the validity of the main idea of the project since the development of the expected solution is coordinated with the automated scanning. a useful tool by people without a technical background.

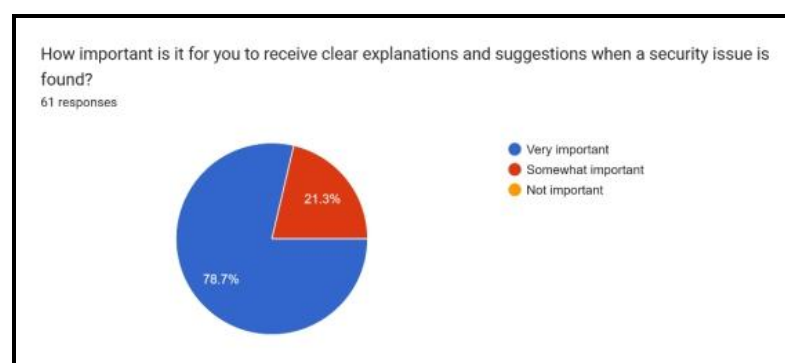


Figure 5.8 Question 8

Based on chart, 78.7% of the respondents stated that receiving clear explanations and suggestions when a vulnerability is found, and the rest of the respondents which is 21.3% considered it to be of the importance of “Somewhat Important.” None of the respondents categorized this practice as unimportant and this highlights the fact that security tools must produce reports that are technically accurate but are easy to comprehend and reasonable to the user.

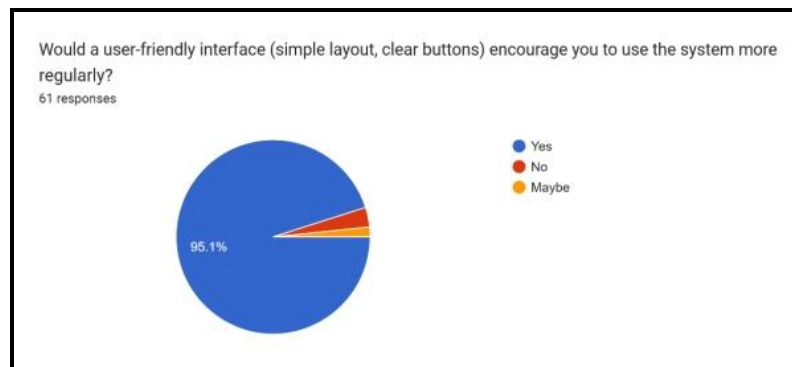


Figure 5.9 Question 9

According to result, 95.1% of the respondents felt that the interface should be user friendly to encourage more frequent use of the system and only 3.3% felt otherwise. Another 1.6% were undecided. These figures prove that there is a need to prioritize a clean and intuitive UI/UX design in building a system, as the usability of the interface is a critical factor that determines the engagement of the user and the long-term adoption.

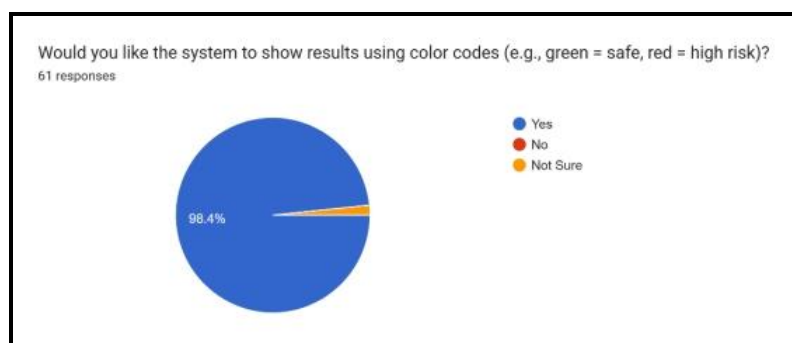


Figure 5.10 Question 10

Almost all the 98.4% respondents responded that they would like to see the results of scans presented in color codes (e.g. green safe, red high risk). Only 1.6% were not sure, and none of the participants disagreed. This proves that the use of color-coded visual clues is an efficient method of displaying the levels of vulnerability, and users, particularly non-technical users can easily interpret the results of the scan within a short period of time.

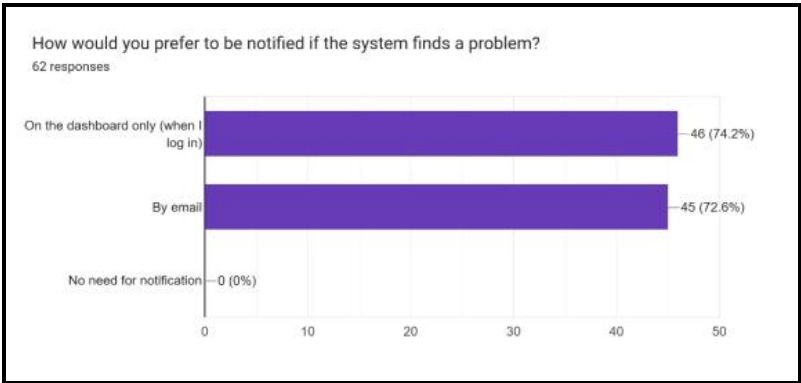


Figure 5.11 Question 11

The result shows a significant tendency of the respondents to get the notifications of a systemic security issue via various communication channels. 74.2% preferred dashboard alert, and 72.6 percent supported email notification. These findings mean that a good security architecture should be able to push alerts through various modalities to ensure that there is timely dissemination of information and immediate response to incidents.

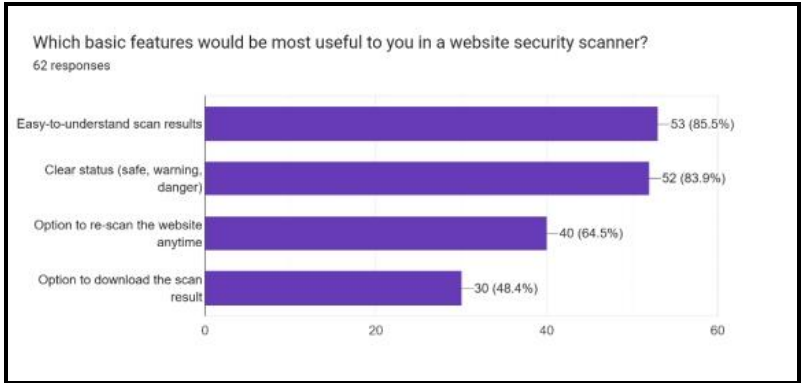


Figure 5.12 Question 12

The participants have listed some of the main characteristics that they would prefer on a web site scanner. The most favoured features were easily readable scan result (85.5%), the clarity of status like safe/warning/danger (83.9%), possibility to re-scan the web site (64.5%) and the possibility to download scan results (48.5%). Such responses will be used directly to prioritize features in the system so it can satisfy the expectations of the users.

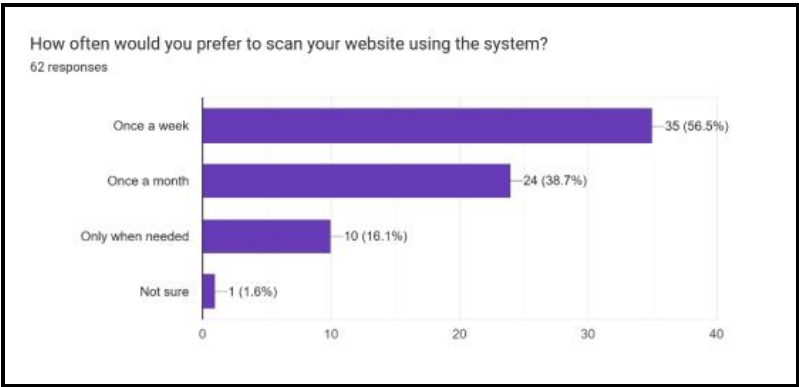


Figure 5.13 Question 13

Regarding the preferred scanning frequency of the websites, 56.5 % of the respondents chose the option once a week, 38.7 % of the respondents chose once a month and 16.1 % of the respondents preferred to scan the sites only when the need arises. An insignificant percentage (1.6 %) was undecided. This data indicates that the system should enable a user to select and customize scan frequency to match individual or organizational needs.

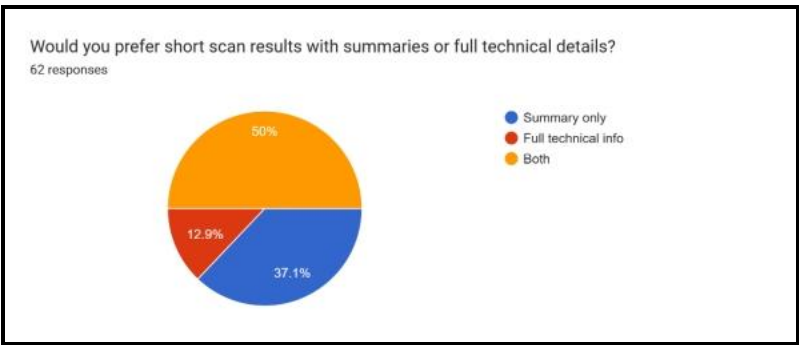


Figure 5.14 Question 14

50 % of the respondents preferred reports that contain both brief summaries and scans, 37.1 % wanted only the summary reports, and 12.9 % wanted the full technical explanation. The results indicate that there is a need to offer a variety of reporting formats to suit the diverse technical skills levels that are witnessed among the users.

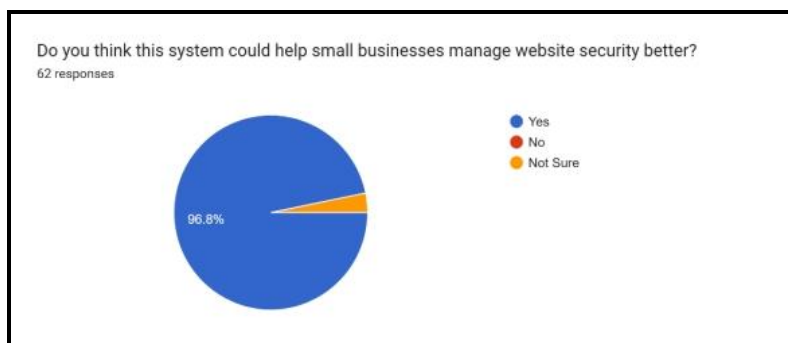


Figure 5.15 Question 15

96.8% of the respondents agreed that the system will assist small businesses to take better control of security of their websites. 3.2% were undecided, and no one disagreed. This greatly affirms the ability of the system to bring actual value, particularly to small businesses which do not have specific cybersecurity resources or expertise.

5.2.2. Interview Analysis

An interview session to validate the system requirements by confirming that the Web Vulnerability Scanner meets real-world business needs was conducted with representatives from Netwitz Sdn. Bhd. The purpose of the interview was to assess the system features, usability, and security expected by users. Responding to the interview were Netwitz employees involved in managing cybersecurity operations and customer support. The data related to the interview participants is as follows:

Table 5.1 Interviewee Information

Items	Details
Name	Nurul Batrisyia binti Mohd Khir
Position	Security Analyst
Company	Netwitz Sdn Bhd
Interview Method	Online Meeting
Date of Interview	23 May 2025



Figure 5.16 Conduct Meeting with client by Google Meet

Table 5.2 Interview Questions Table

Questionnaire	Client Response	Analysis
What features do you want in the web vulnerability scanner?	The system must allow users to scan websites by typing in a URL and get a clear vulnerability report. The client also stated that OTP login and easy role separation are required.	Review essential scanning and security capabilities. Confirm that the system should be able to identify basic vulnerabilities and secure access.
Do you require a login system with user roles?	Yes, there is a need to separate administrator and client roles.	Confirm the need for RBAC to provide proper security and access rights.
Should users be able to notify Netwitz for further action if needed?	Yes, the customers must have options to order more detailed, manual scans when they are attracted by the initial results.	Confirm the need for a Deep Scan Request feature to facilitate communication with Netwitz staff
Do you prefer a dashboard to display summary information?	Yes, a dashboard allows users to have a quick overview of scan history and summary information.	Confirms that a dashboard is a needed usability feature to improve user experience.
Is it necessary for users to download their scan reports?	Yes, the user requires downloadable reports in the documentation.	Confirm the need to have a PDF export function.
What types of vulnerabilities should the scanner detect?	SQLi, cross-site scripting, and insecure headers, among other common web vulnerabilities.	Confirms that the scope of the system is limited to simple external vulnerabilities, as aligned with the project direction.
What information should be included in the scan results?	The client is very clear that the scan results should contain a description of the problem, recommendations or solutions and a report format suitable for the client to understand and act upon.	This confirms the need for three essential elements in any scan output: (1) a clear description of the vulnerability, (2) recommended solutions/mitigation measures and (3) reporting elements. This allows the system to achieve Objective 3: Display scan results in a clear and understandable format.

Overall, the interview helped confirm that the system must prioritize ease of use, clarity of scan output, and a lightweight security structure that aligns with the technical ability of typical Netwitz clients.

5.3. Use Case Model

Use case model is a graphical and functional model to describe the usage of the Web Vulnerability Scanner system by the users. It determines the main tasks users can complete and describes the system's reaction to the tasks. This model is important in the process of making the system meet the expectations of the user due to the emphasis on real life usage cases. It also facilitates interaction between the admin and user because the system functionality is placed in a comprehensible structure. Visual Paradigm (Protected, 2025) suggests that use case diagrams could be used to define the scope of the system and be aligned with the requirements of the user since it describes the role of actors and system behaviour.

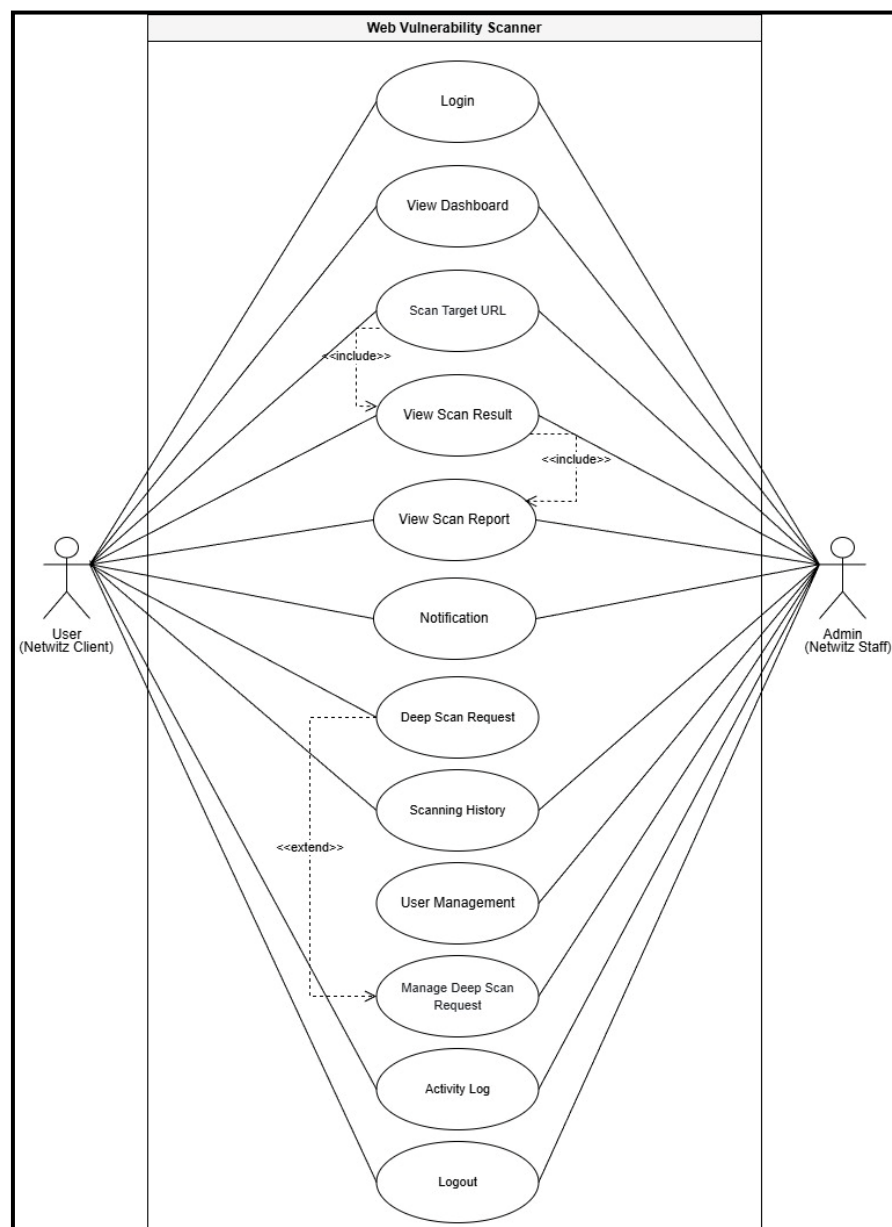


Figure 5.17 Use Case Diagram

The flow begins when either a user or an admin log into the Web Vulnerability Scanner system and accesses the dashboard. Both can submit a website URL for scanning, which always includes viewing the scan result, and from there, both can view detailed scan reports. If a user needs further analysis, they can send a notification to the Netwiz admin for a deeper scan. Admins have additional privileges which is they can manage users, view and reply to notifications, review the scanning history, and access the log activity. Finally, both roles can log out of the system. This use case flow mirrors the corresponding user and admin flowcharts, clearly delineating shared and admin-only actions, as well as the inclusion and extension relationships between core functionalities.

5.4. Flowchart

5.4.1. Flowchart Admin user

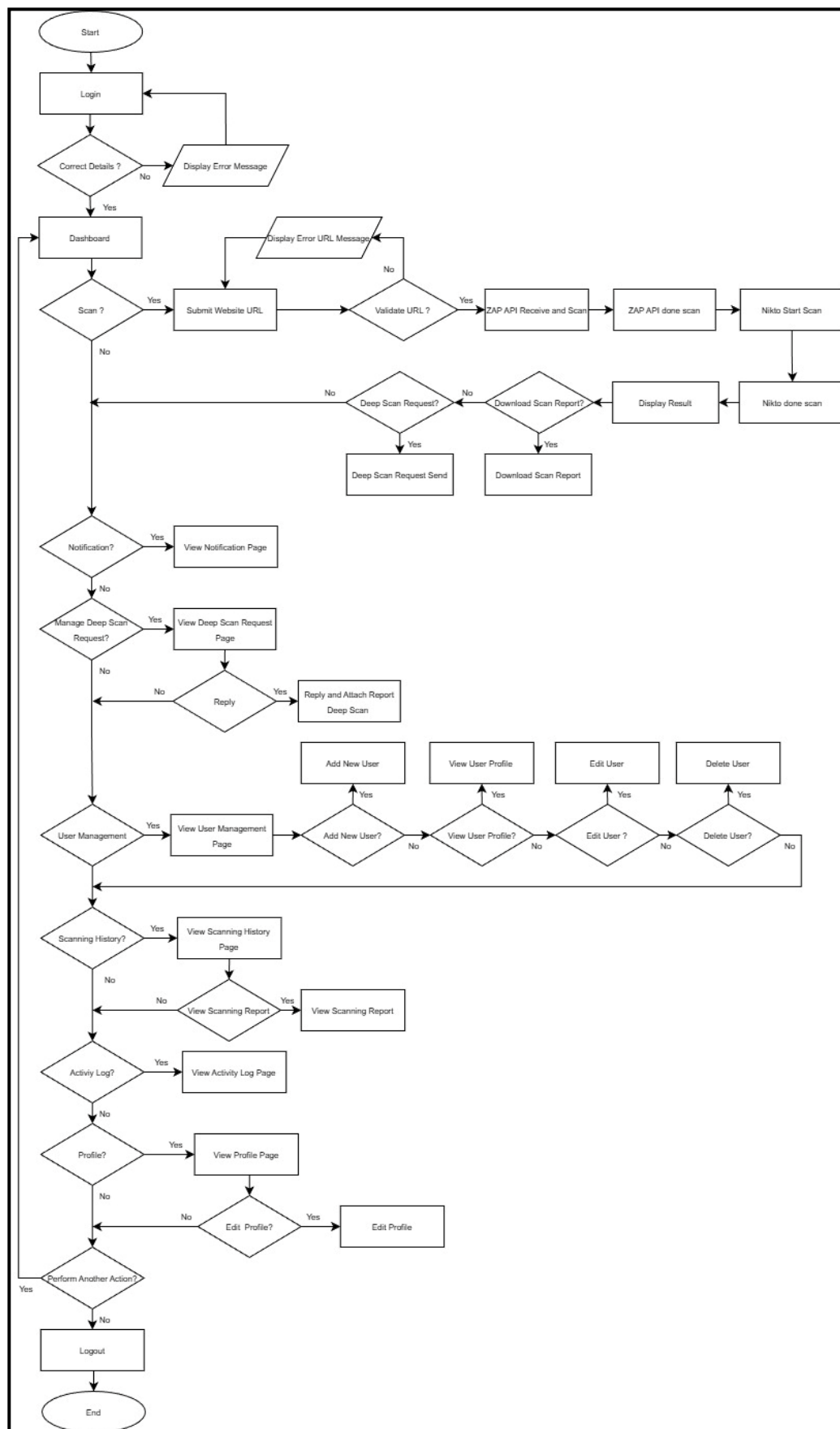
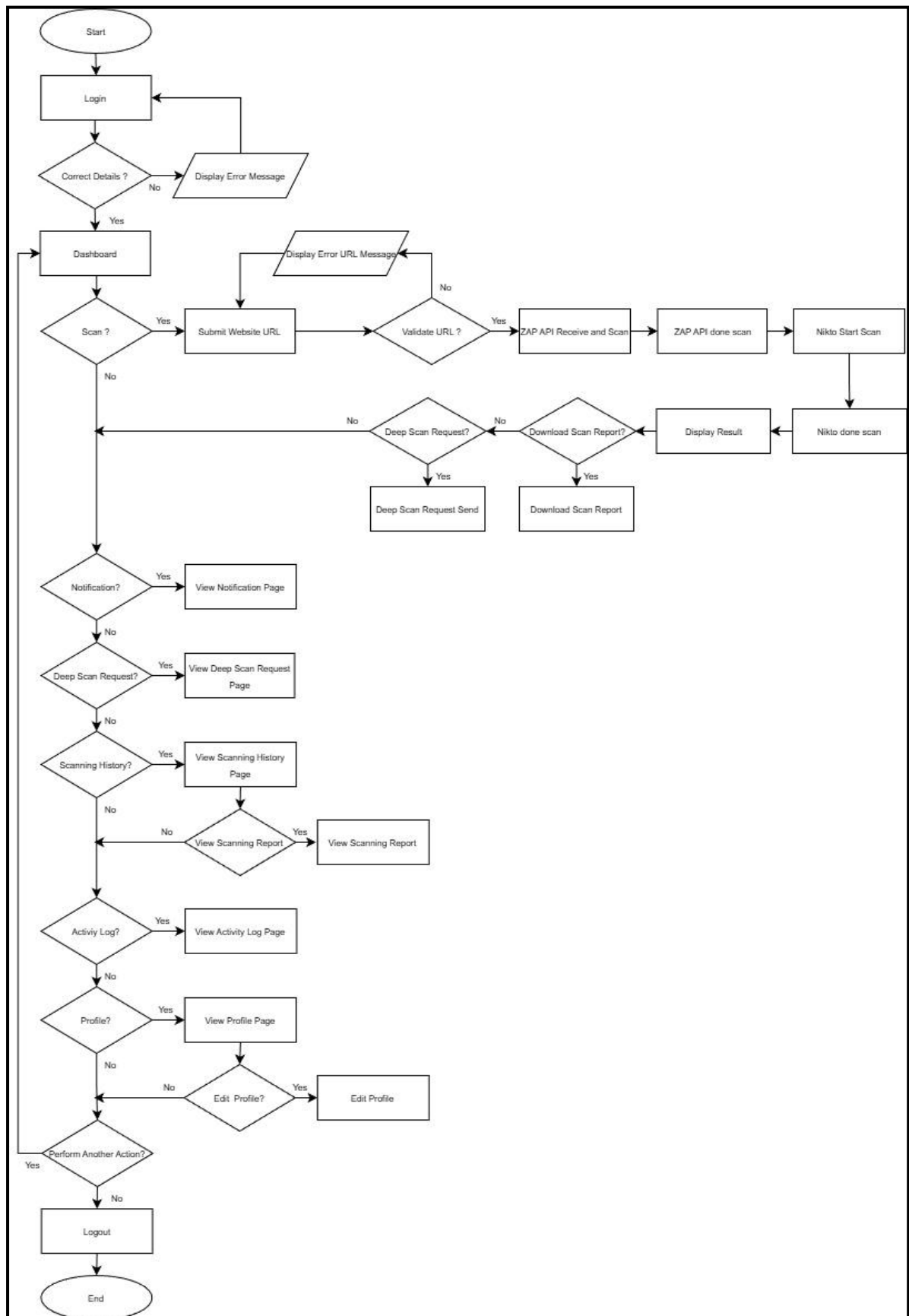


Figure 5.18 Flowchart Admin User

The system begins with a user logging in. If user inserts correct details, it will go through dashboard. Otherwise, an error message will emerge. From there, users can scan a website by submitting the website URL, which the system verifies. If the URL is valid, the system sends it to the ZAP API and Nikto to scan and display scan results. The user can view the scan report, and they can access it directly. Administrators can access the user management dashboard, and they also have additional options, which is adding, editing, deleting, or assigning roles to the users. The admin users can view and reply to Deep scan requests that they receive by user. This flow ensures that everyone can securely log in, scan websites, manage users and roles, and view important information or activities as needed.

5.4.2. Flowchart User**Figure 5.19** Flowchart User

The system starts with the user logging in. If user inserts correct details, it will go through dashboard. Otherwise, an error message will appear. From there, users can scan a website by submitting the website URL, which is verified by the system. If the URL is valid, the system sends it to the ZAP API and Nikto to scan and display the scan results. Users can view the scan report, and they can access it directly. In case of any error with the scan results, the user can notify the Netwiz Administrator to run a deeper scan. Once the action has been performed, the user has the option to perform other actions or log out to end the session. This flow allows the user to securely log in, scan the website, analyse the results and issue a deeper scan when necessary.

5.5. Conclusion

In conclusion, these chapters have given a detailed explanation of the information collected and the system requirements based on it. The results of the questionnaire and the interview have assisted in proving the right path of the project and determining the needs of the users like ease of use, visual output, and timely vulnerability alerts. To describe the key functions of the system and interactions with the user, use case diagrams, flowcharts and data flow diagram were created, which demonstrate the importance of the work of the users and system. These visual models explain the way the system will function in the real world. Overall, the analysis provided in this chapter will provide a solid basis of design and development stage, which will make the Web Vulnerability Scanner user-friendly and technically competent.

6. DESIGN

6.1. Introduction

This chapter shows the design phase of the Web Vulnerability Scanner system which is the roadmap of the development process. The design will ensure that the system addresses the functional and non-functional requirements identified in the previous phase, including secure role-based access, automated scanning and an interface that is easy to use by non-technical users. It addresses several important aspects such as interface design, database structure and system workflow to ensure that all components will be successfully integrated. Wireframes are drawn to sketch a simple and clear user interface design to enable users to navigate through the system. Database designs, such as Entity Relationship Diagrams (ERD), Data Flow Diagrams (DFD) and data dictionaries, help data be stored and retrieved in an efficient and secure manner. This design phase provides a clear foundation for building a scalable, secure and user-friendly system that will resonate with the operational needs of Netwitz Sdn. Bhd. and its clients and guide the developer to structure each module in a consistent and effective manner.

6.2. Interface Design

The interface design of the Web Vulnerability Scanner system aims to offer a simple, consistent and user-friendly interface to accommodate the interests of both Netwitz staff and customers. During this step, wireframes were created to show the layout and structure of the system without focusing on the visual design. This wireframe can be used to design the layout of the main features of each page including input fields, navigation menus, buttons and content areas of each page, including the login page, dashboard, scan page, scan results page, notification page and activity log. According to GeeksforGeeks (2024) an effective user interface design must be clear, usable and easy to navigate, so that users can navigate the system effectively. By adhering to these principles, wireframes allow for the creation of a simple workflow - logging in, performing a scan and reviewing results, sending notifications and storing a history of previous actions. This initial design also acts as a guide between the requirements analysis and development phases, ensuring that the system implementation remains aligned with the project objectives of simplicity, security and accessibility (GeeksforGeeks,2024).

6.2.1. Wireframe Web Vulnerability Scanner

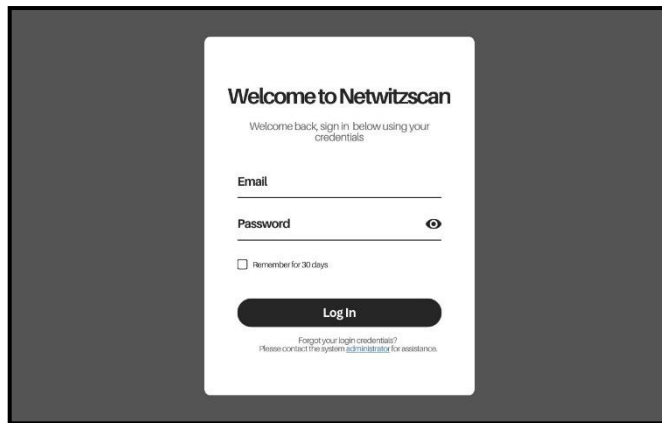


Figure 6.1 Login Page

The login page seen in Figure 6.1 is designed for Netwitz staff and users for easy, clean, and secure interaction. There is an email entry box and a password box for users to complete standard login authentication. There is also a prompt for users who have forgotten to enter their credentials and is labeled, “Please contact the administrator.” After the user enters their email and password and logs into the account, a new step is initiated, requiring the user to enter an OTP sent via a secure channel to the user’s personal email account. This is an additional layer of security for cases where the account credentials have been compromised.

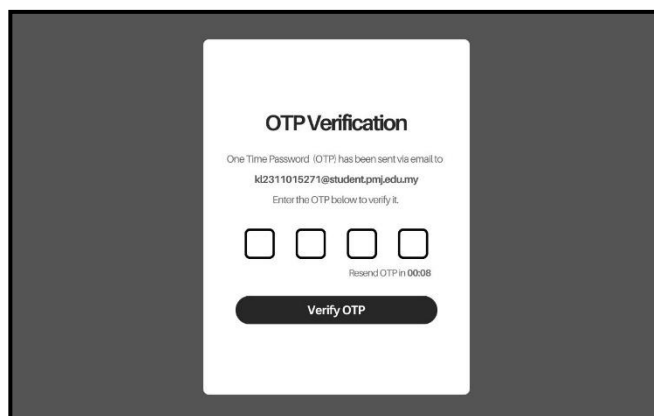


Figure 6.2 OTP Verification Page

Figure 6.2 illustrates the construction of the OTP Verification Page which will further add more security features after the correct credentials have been entered. This page contains an OTP field where users can enter a six-digit code, which will be sent to their contact information on file. A Resend OTP button is available in case the code is not received or has expired for the user’s convenience. After entering the code, users can complete the verification process to securely access the dashboard by pressing the Verify OTP button. This page also offers two-step verification, reducing the risk of unauthorized access and is designed to be user-friendly for both staff and user.

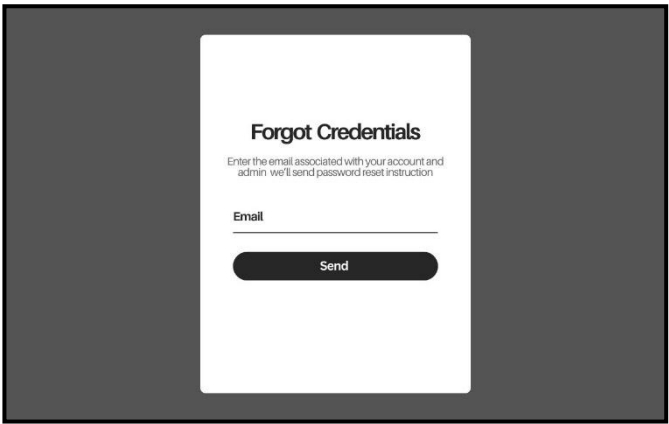


Figure 6.3 Forgot Credentials Page

According to Figure 6.3, the Forgotten Credentials Page helps users who have lost their credentials regain access to their accounts. In this case, the user only needs to fill in an email field where the user has to submit the request by clicking the button titled `Submit.` After this user action, the system notifies the administrator who is responsible for helping the user reset the newly lost credentials or restore the existing credentials. The design is simple to allow less technically sophisticated users to get help without experiencing any problems, while the system security is not compromised.

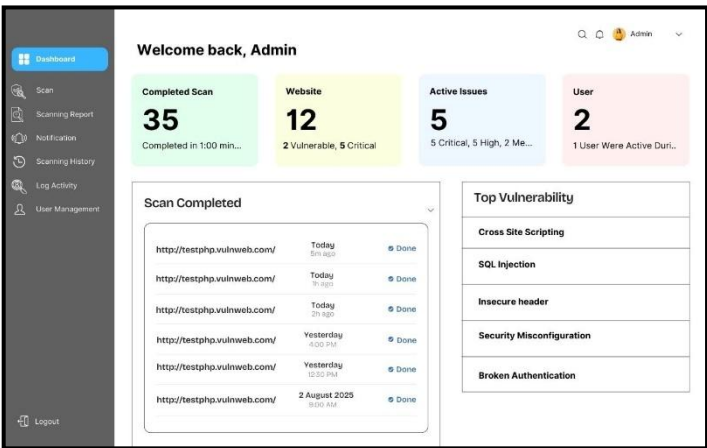


Figure 6.4 Dashboard Page

Figure 6.4 presents the Dashboard page, which is the dashboard of the Web Vulnerability Scanner and displays an overview of system activity and other related metrics. The left panel contains menus that allow users to navigate to other parts of the site, including the scanning, reporting, notifications, and account management sections. In the center of the page, there is a summary card that shows key metrics including the total number of scans completed and the number of issues being actively worked on, as well as the number of unique websites that have been scanned and are in the issue review queue. Located in the summary section is a list of the most recent scans, the top of which displays the most critical issues to provide users with a quick reference to assist in active reviews and monitoring of important issues. In the upper right corner, an alarm bell icon is included to notify users of notifications that may be important to them, and next to that, a personal icon used to access the user's personal

dashboard and account configuration settings. This arrangement allows the dashboard to operate in an efficient manner, allowing user activity on the system to be monitored, with the activity being able to be rotated to other system functions.

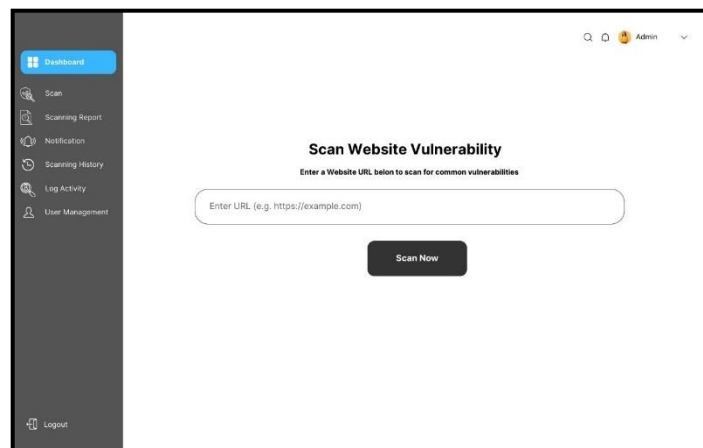


Figure 6.5 Scan Page

Figure 6.5 shows how all users, even those who are not technically inclined, can easily perform scanning operations on the Scan Page. The Scan Page has a URL input that the user can use to indicate the web page that he or she wants to scan for vulnerabilities. The Start Scan button, which the user can click to start the scanning process, will be at the bottom of the input field. The scan will check the site for common vulnerabilities such as SQL injection (SQLi), cross-site scripting (XSS), and other misconfigured settings. While the scan is running, the user will be able to see the progress of the scanning operation with a progress bar that updates in real time. The simplicity of the Scan Page structure allows the user to perform scans in an organized and fast manner.

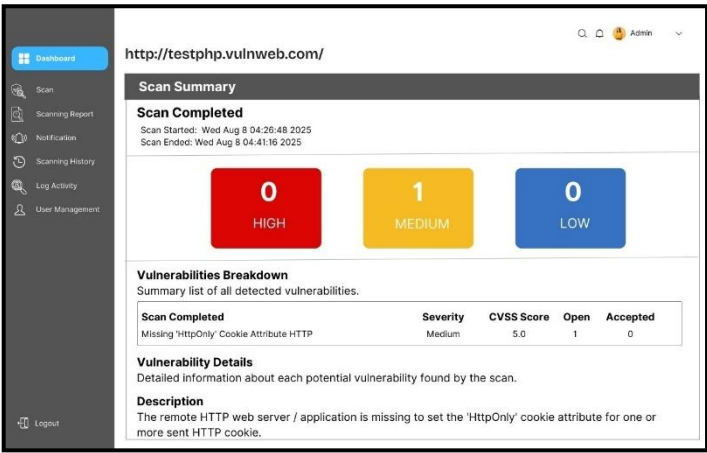


Figure 6.6 Scanning Report Page

The Scan Report page, as shown in Figure 6.6, captures the detailed and organized results of the website scan. For quick context of the scanning session, the top of the report shows an overview of the scan, which includes the target URL, scan start time, and scan end time. This allows the user to get the context of the scanning session. In the results summary table, the total number of vulnerabilities detected, and the total number of vulnerabilities detected are sorted by severity level High, Medium, and Low, showing the user the overall risk level of the website. Further, below the summary table, there is another table that lists the vulnerabilities, their severity level, CVSS score, status, and other attributes to assist the user in continuing remediation efforts and to allow the user to set appropriate priorities. Each identified vulnerability contains a detailed description that explains the severity level and risk of the vulnerability in plain language so that non-technical users can understand and take appropriate action regarding the vulnerability and the priorities needed to mitigate the risk. This makes the report accessible to users so that they can take steps to improve the security of their website.

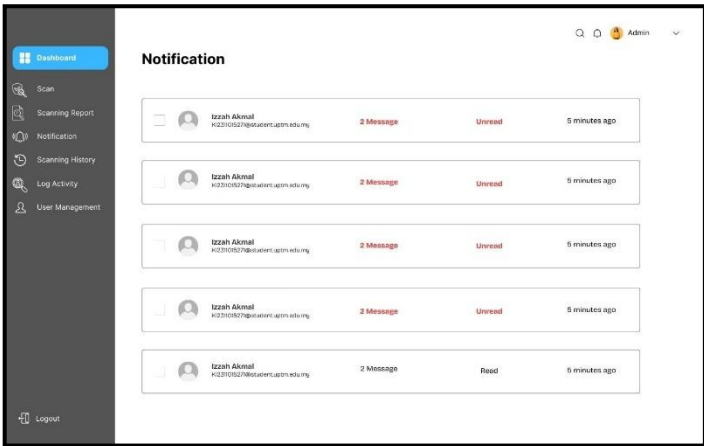


Figure 6.7 Notification Page

Figure 6.7 shows that the System Notifications section provides an overview of the System Updates section. There is a notification section that communicates the latest activity in the system, such as when a scan is complete, a vulnerability is detected, or a message is delivered. The notifications are organized in a concise list format to help users keep track of the latest system updates. The system updates users about the status of their scans and other important system alerts that require their attention. This is to ensure that users are always informed and do not miss out on any important system updates.

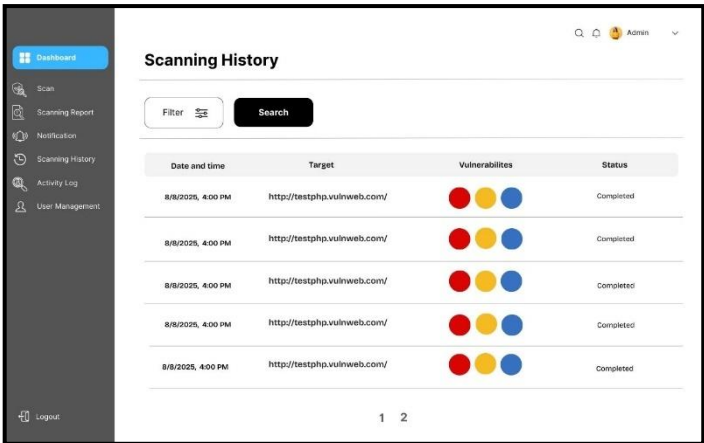


Figure 6.8 Scanning History Page

Figure 6.8 illustrates how the Scan History Page allows users to view the history of all previous scans. Additional features such as filter boxes and search buttons help users find specific scans by website, date, or status in an efficient manner. History entries include the date and time of the scan, the target URL and vulnerabilities found, and the final status of the scan. This page provides the user with the status of previous activities and an overview of the scan so that users can access the results.

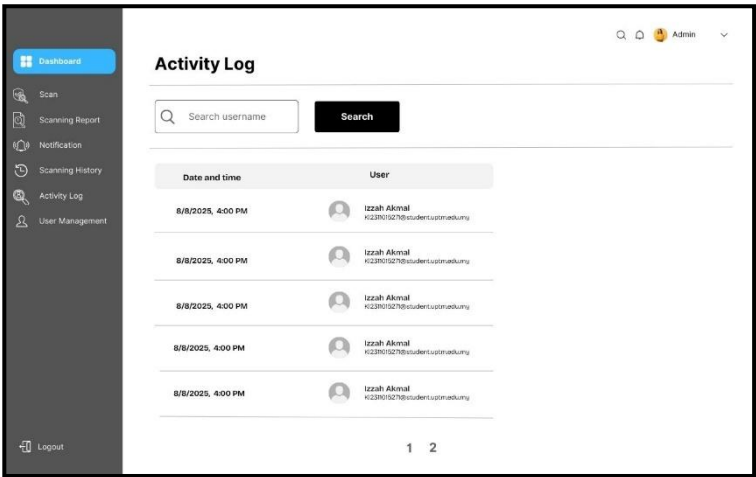


Figure 6.9 Activity Log Page

As illustrated in Figure 6.6, there is an Activity Log page that outlines all the user actions that are important in each system use case. Columns have been added for users to quickly narrow down and filter specific activities, while containing the activity totals to enhance search functionality. Individual records show when and which users performed the activity. This page promotes transparency by allowing higher administrative users to systematize and structure their usage and activities in the system.

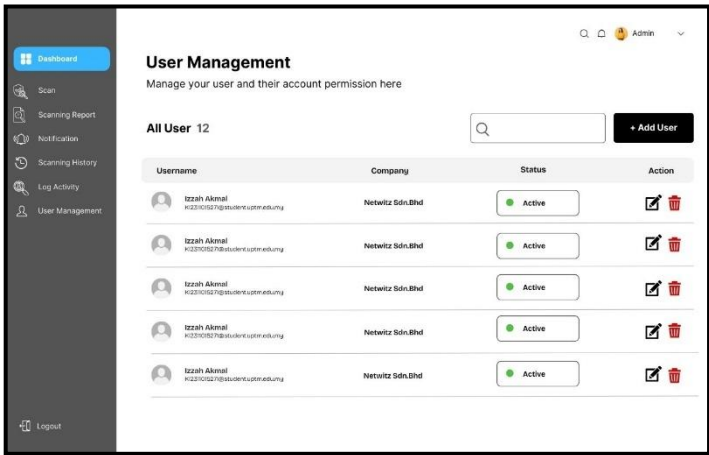


Figure 6.10 User Management Page

As shown in 6.10, the User Management Page is an administrative-only tool that enables Netwitz administrators to effectively control system users and their access privileges. The page has a search field to find a particular user easily and an Add User field to create new accounts. The user entries include their account status that is active or inactive to assist administrators to monitor access in one glance. Further, there are action buttons to edit user details or delete users in case they are needed. This page enables the proper organization, accessibility, and management of all user accounts in a secure and structured manner that supports the role-based access control (RBAC) mechanism that is in place in the system and promotes overall operational stability.

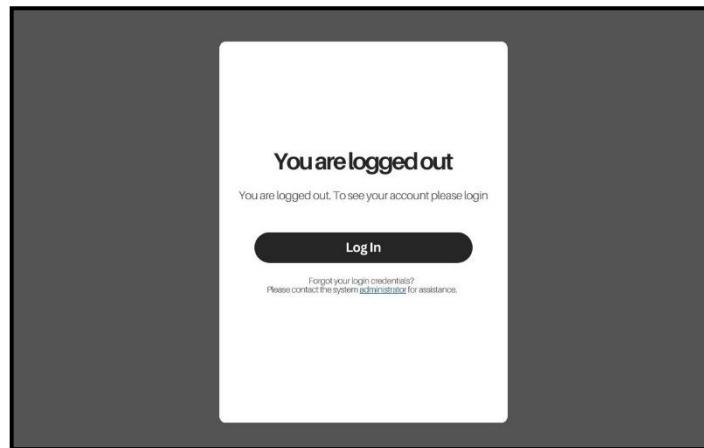


Figure 6.11 Log Out Page

The Logout Page is the final page in the system workflow, confirming the successful logout of the application user, as illustrated in Figure 6.11. This page has a Log In button that allows the user to navigate back to the login page for the user to log in again. The design provides a smooth and secure transition flow between the end of a session and the start of a new session, and the ease of navigation is simple and easy. By clearly indicating that the session has been successfully terminated, this page also enhances system security by preventing people from logging out when the user logs out.

6.3. Database Design

The Web Vulnerability Scanner database design focuses on secure, organized, and operationally efficient information management that prioritizes the basic usability of the application. Including the integrity of an efficient data management system including a valuable information system to include user data, scans, vulnerabilities, and notifications, and activity data to create and improve logs. A system of adherence to relational database rules and principles ensures the foundation of reliability and trustworthiness. The design structure further enhances the system's on-premises operations to incorporate a role-based access control (RBAC) system, and administrators and customer users will gain controlled and differentiated levels of accessibility. Furthermore, the database structure strengthens the operating system infrastructure, thus accommodating rapid integration of functions, modules or upgrades without re-engineering the main operating system. The true measure of system design sophistication is the extent to which the system design provides seamless communication between the system component interfaces and the system's operational database (GeeksforGeeks, 2024). The result is operational effectiveness for Web Vulnerability Scanner without compromising user data confidentiality or user operational inconvenience.

6.3.1. Data Dictionary

Table 6.1 Data Dictionary for Company Table

Field Name	Data Type (Size)	Description
id	Integer (Primary Key)	Unique identifier for each company record.
name	String (255)	Company name.
created_at	DateTime	Timestamp when the company record was created.
users	Relationships	List of users under this company

The Company data table in the database module contains all information system users for the organization. It contains four fields. The ID field contains the company's integer primary key and its unique identification number. The name field contains the company name, encoded as a string of up to 255 characters. The create_at field records the exact date and time when the firm record was created in the system and is of type DateTime. A user field is a relationship field that associates a company with all its users, allowing for easy collection and administration of users within the same company. Overall, this table contains all the company data in the form of a structured table.

Table 6.2 Data Dictionary for the User Table

Field Name	Data Type (Size)	Description
id	integer (primary key)	Unique identifier for each user in the system.
username	string (150)	Unique username used by the user to log in.
email	string (150)	Unique email address for the user, used for login and notifications.
password_hash	string (150)	Encrypted password for secure authentication.
role	string (100)	Defines the user role, such as admin or client, to support role-based access control (RBAC).
created_at	datetime	The date and time of the user account was created.
company_id	integer (foreign key)	References the Company table to associate the user with a specific company
otp_secret	String(64)	Secret key used to generate OTP codes.
trusted_device	text	Stores trusted device tokens in JSON format.
profile_picture	String (255)	File path for user profile picture.
is_active	Boolean	Indicates if the user account is active.
scans	Relationship	All scans initiated by this user.
notifications	Relationship	Notifications sent by this user.
logs	Relationship	Activity logs related to this user.
shared_notifications	Relationship	Notifications shared with this user.

Table 6.2 details the Users table in the database, which contains information related to system users. Each record in the table starts with an id, which serves as the primary key to ensure each user is unique. The username and email fields capture login information, with constraints enforced to ensure the uniqueness of each. password_hash contains the hash of the user's password. The role column indicates the user's status in the system, either as an administrator or a user, for the purposes of permission rules. created_at captures the time the record was instantiated. company_id is a foreign key that binds a user to a specific entity in the Companies table. otp_secret contains the secret used for one-time password generation. trusted_device contains details of devices that the user has declared as trusted. The profile_picture column contains the user's avatar. The is_active column is a Boolean value, which describes the active status of the user record. The table also contains several relationship fields, scans, notifications, logs, and shared_notifications, which easily associate a user with the scans they have performed, notifications they have received, activities they have logged, and notifications they have shared. In conclusion, this specific table organizes all user-related information in an organized manner and stores it securely.

Table 6.3 Data Dictionary for the Scan table

Field Name	Data Type (Size)	Description
id	integer (primary key)	Unique identifier for each scan record.
user_id	integer (foreign key)	References the User table to identify which user initiated the scan.
company_id	integer (foreign key)	Company associated with the scan.
target_url	string (255)	Stores the URL of the website that is being scanned.
domain	String (255)	Stores the domain extracted from the target URL for reporting purposes.
scan_type	string (20)	Specifies the type of scan performed
status	string (20)	Indicates the status of the scan, such as pending, in progress, or completed
result_summary	text	Provides a summary of the scan results.
started_at	datetime	Records the date and time when the scan started.
completed_at	datetime	Records the date and time when the scan finished.
created_at	datetime	Automatically records the date and time the scan entry was created.
is_shared	boolean	Indicates if the scan is shared.
vulnerabilities	relationship	vulnerabilities found in this scan.
deep_scan_requests	relationship	Deep scan requests linked to this scan.

In Table 6.3, the structure of the scan table is provided. Each row contains a unique scan id, the primary key of the table. Each record must be associated with a system user, hence the inclusion of the foreign key attributes user_id and company_id. The target_url field is included to store the URL of the website being scanned and the domain field is for reporting purposes to store the domain name being scanned. For reporting purposes, the scan_type field captures the selected scan record and the status field, which can be pending, processing or completed, describes the status of the scan. The result_summary field contains a summary of the results found from the scan. The system records important timestamps associated with the scan in the form of the started_on and finished_on and created_on datetime fields. Whether the scan results are shared with other users is indicated by the is_shared field. Finally, the vulnerability and deep_scan_requests fields are the relationships that each scan records to the corresponding vulnerability and to the record of any requests for deep scans. The table serves to organize and store data to store and track the system's scanning activity.

Table 6.4 Data Dictionary for Vulnerability Table

Field Name	Data Type (Size)	Description
id	integer (primary key)	Unique identifier for each vulnerability record.
scan_id	integer (foreign key)	Links the vulnerability to a specific scan in the Scan table.
title	string (255)	Short title or name of the vulnerability found.
vuln_type	string (100)	Type of vulnerability, such as SQLi or XSS.
risk_level	string (10)	Indicates the severity of the vulnerability (e.g., High, Medium, Low).
description	text	Detailed explanation of the vulnerability, including its nature and potential impact.
recommendation	text	Suggested steps or actions to fix or mitigate the vulnerability.
evidence	text	Supporting details or proof
url	string (512)	Affected URL.
method	string (10)	HTTP method (GET/POST)
param	string (255)	Parameter affected.
attack	text	Attack payload used.
otherinfo	text	Additional information.
confidence	string (50)	Confidence level of detection.
references	text	External references.
cweid	string (50)	CWE ID.
wascid	string (50)	WASC ID
response_time	float	server response time.
created_at	datetime	The date and time when the vulnerability record was created.

Table 6.4 illustrates the structure of the Vulnerabilities table, which contains details of vulnerabilities collected from each scan. The id field is the primary key of the table. The scan_id field links each vulnerability to the scan in which it was identified. The title and vuln_type fields describe the name and category of the vulnerability, which may include SQL Injection or Cross-Site Scripting. The risk_level field specifies the threat of greater concern (High, Medium, or Low), and the description field describes vulnerability and why it is important. The recommendations field contains actions that can be taken to mitigate the risk. The evidence field contains evidence or information that supports the existence of the problem. The remaining fields store additional details about vulnerability, which include the url, method, param, attack, and other information fields, indicating where the vulnerability occurred, the HTTP method used, the parameters targeted, and the payload responsible for the vulnerability. The confidence field describes the scanner's certainty about the validity of the detection, while the reference, cweid, and wascid fields offer details of the vulnerability and the corresponding standards. The response_time field describes the time it took the server to respond, and the created_at field describes the time when the vulnerability count was created. The table provides a comprehensive overview of each security issue encountered during the exam.

Table 6.5 ServerVulnerability Table

Field Name	Data Type (Size)	Description
id	integer (primary key)	Unique identifier for each vulnerability record.
scan_id	integer (foreign key)	Links the vulnerability to a specific scan in the Scan table.
category	String (100)	Type of server weaknesses
title	string (255)	Short title or name of the vulnerability found.
risk_level	string (210)	Severity level
description	text	Detailed explanation of the vulnerability, including its nature and potential impact.
recommendation	text	Suggested steps or actions to fix or mitigate the vulnerability.
url	string (512)	Affected URL.
method	string (10)	HTTP method (GET/POST)
banner	string (255)	Server banner information
X_powered_by	string (255)	Technology info disclosed by server
references	text	External references.
created_at	datetime	The date and time when the vulnerability record was created.

The Server Vulnerability Table, which contains details about server-related vulnerabilities found during a scan, is described in Table 6.5. While the `scan_id` field links the issue to the specific scan in which it was found, the `id` field serves as the primary key to uniquely identify each server vulnerability record. The title gives the vulnerability a short name, and the category field indicates the type of server vulnerability. The severity of the problem is indicated by the `risk_level`, and the impact of the vulnerability on the system is explained in detail in the description. The steps or actions required to fix or mitigate the risk are listed in the recommendations field. Details such as `url`, which displays the affected page; `method`, which records the HTTP method used banner, which indicates the server banner and `x_powered_by`, which displays the technology information exposed by the server, are used to store additional technical information. The `created_at` field documents the date and time the entry was created, and the references field includes useful external links related to the vulnerability. All things considered, this table is helpful in monitoring server-related problems that could expose sensitive data or compromise security.

Table 6.6 *DeepScanRequest Table*

Field Name	Data Type (Size)	Description
<code>id</code>	integer (primary key)	Unique deep scan request id
<code>scan_id</code>	integer (foreign key)	Related scan report
<code>user_id</code>	integer (foreign key)	User who requested deep scan
<code>status</code>	string (20)	Request status (Pending/Complete)
<code>admin_note</code>	text	Admin note
<code>result_file</code>	string (255)	Path to uploaded file
<code>requested_at</code>	DateTime	When the request was made
<code>responded_at</code>	DateTime	When admin responded

Table 6.6 discusses the `DeepScanRequest` table, which includes all user requests for advanced administrator-level scans. Each request is assigned a unique `id`, which is the primary key for the table. Each user request is associated with a specific scan report via the `scan_id` field, while the `user_id` field contains the identifier of the user who made the request. The `status` field captures whether a particular request is still open, or if it has been completed. The `admin_note` field is for a comment entry by the administrator, and the `result_file` is a field that contains the file path to the report for the completed deep scan. Each user request is assigned a timestamp contained in the `requested_at` field and the administrator's timestamp is stored in the `responded_at` field. Such information makes it easier to organize deep scan requests in a more efficient way and manage them more easily.

Table 6.7 Data Dictionary for Notification Table

Field Name	Data Type (Size)	Description
id	integer (primary key)	Unique identifier for each notification.
sender_id	integer (foreign key)	Users who send the notification
recipient_id	integer (foreign key)	User who receives the notification
message	text	Notification content shown to the recipient.
type	string (20)	Category of notification
is_read	boolean	Read status flag; False by default, True once the user views it.
created_at	datetime	Timestamp when the notification was created.

Table 6.5 describes the notification table which includes notifications sent to users from the system. `id` is the primary key of the table. `sender_id` represents users who send notifications, and `recipient_id` is the user who receives the notification. The `message` field includes the actual message and the `type` field describes the category of notification (e.g. alert, update). The `is_read` field is a flag field that indicates whether the notification has been opened (False) and subsequently becomes True when the user has accessed the notification. `created_at` is the field that contains the timestamp of when the notification is created. In summary, the table is designed to handle system notifications and important messages are sent to users in the system.

Table 6.8 Data Dictionary for UserLog Table

Field Name	Data Type (Size)	Description
id	integer (primary key)	Unique identifier for each log record.
user_id	integer (foreign key)	References the User table to indicate which user performed the action
action	string (50)	Describes the action performed, such as login, logout, or scan initiation.
details	text	Provides additional information or context about the action.
created_at	datetime	Records the date and time when the log entry was created.

As detailed in Table 6.6, the UserLog table records every important action executed by every user in the system. The `id` attribute is a log primary key. The `user_id` attribute associates the log with a particular user that performed the action. The `action` attribute contains brief descriptors of the actions performed by a user, like logins, logouts, or scan initiations. The `details` attribute contains additional information and context that perhaps the action alone may not sufficiently convey. The `created_at` attribute contains the timestamp of the moment the log entry was created. The table is useful for tracking user activity for security, surveillance, and auditing.

6.4. Data Flow Diagram (DFD)

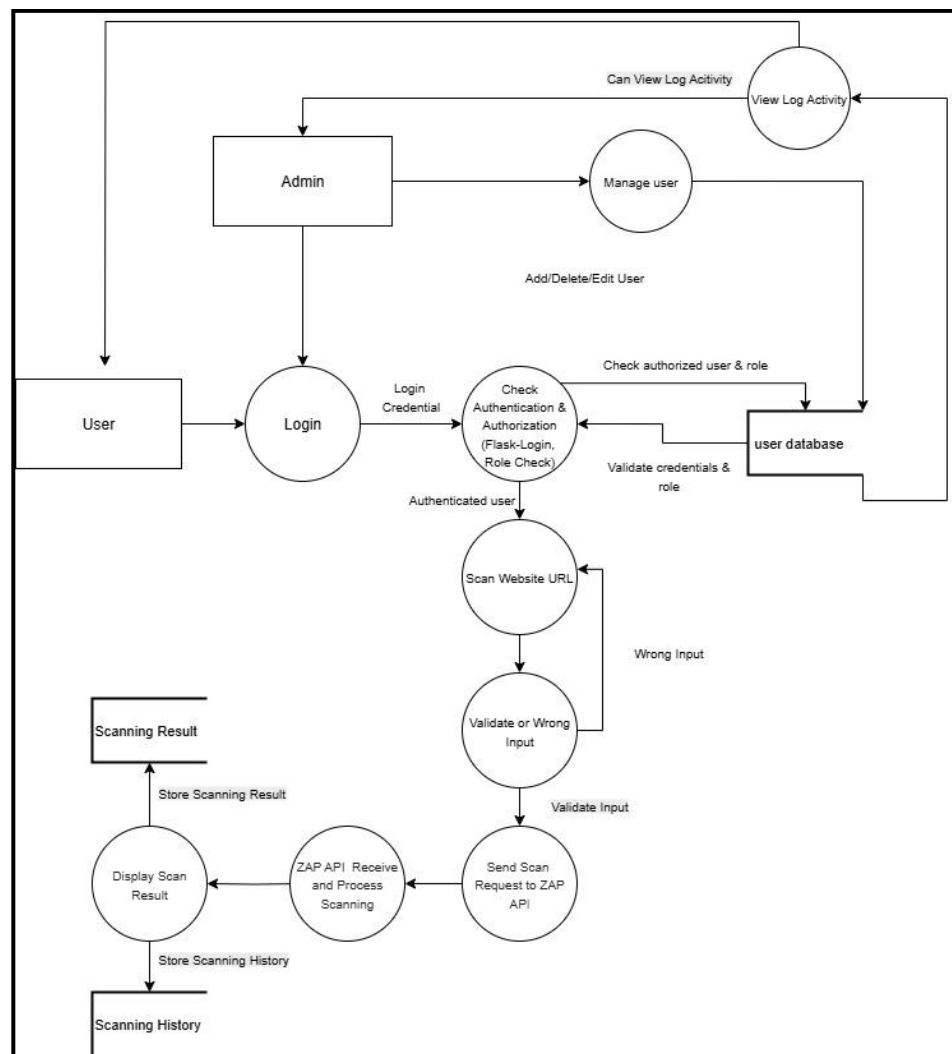


Figure 6.12 Data Flow Diagram

The DFD shows the process that comprises a Web Vulnerability Scanner system. The DFD depicts the User authentication process system to display the scan results. First, the system authenticates the user with the login function and verifies the user role via the flask login admin module. The admin overrides the user account and has an audit function to track the user actions. Once the user is fully authenticated, the user submits the URL that they want to scan. The system scans it and runs the syntax and sends an error message back to the user if it is broken. The system submits the valid URL to the scanning engine where the scanning task is queued. In this case, a request is made to the ZAP API where an automated pen test is performed and then submitted to Nikto for scan evaluation. The results of both are combined to report to the user through the system interface. The data is stored in a database and the results for the scan are filed in an easily retrievable scan history repository. The architecture of this system captures the entire data flow, including user authentication, data validation, data scanning, persistent storage and final output.

6.4.1. Entity Relational Diagram (ERD)

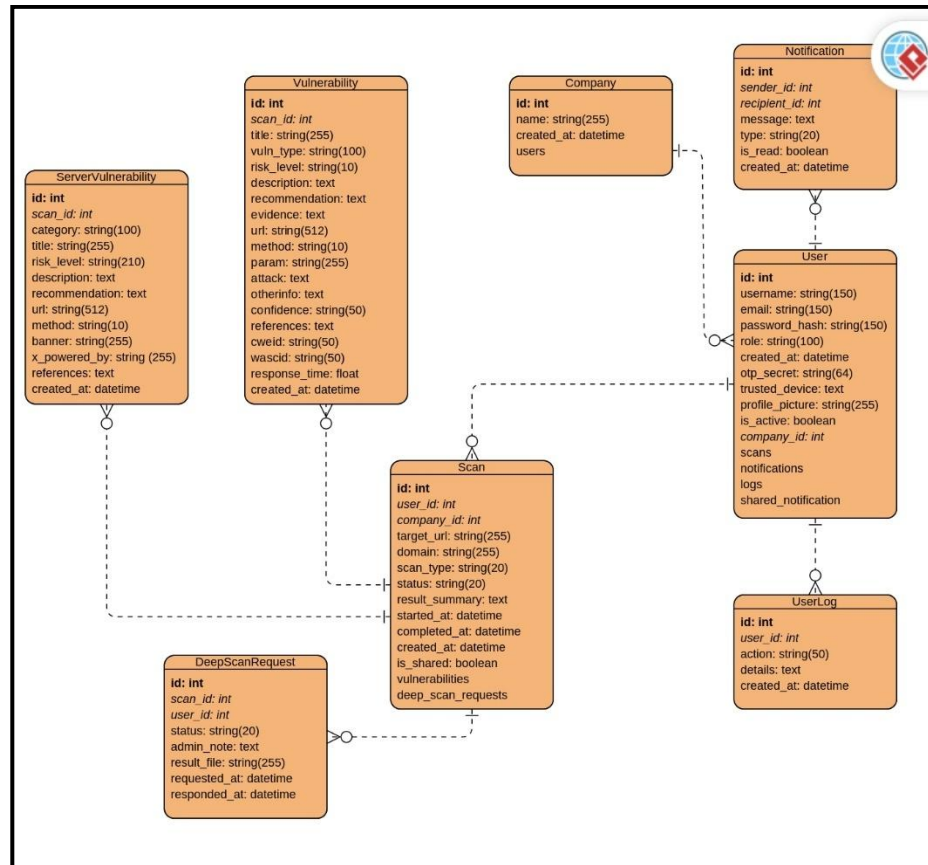


Figure 6.13 Entity Relational Diagram

The Web Vulnerability Scanner system database is structured in such a way that all the relevant information about users, companies, scans, vulnerabilities, notifications and system activities are organized and stored in a clear and organized manner. The purpose of each table has a purpose, and the tables are related by primary keys and foreign keys to ensure proper and meaningful relationships. The Company table is used to store company information and connects to the User table where each user is part of a single company and can perform multiple functions. All scans made by the user are registered in the Scans table and all the vulnerabilities found during each scan are stored in the Vulnerabilities and Server Vulnerabilities tables. Deep scans can also be requested by the user and stored in the DeepScanRequest table. The Notifications table is used to process communication within the system, and this is where the user can send and receive multiple notifications. Monitoring and auditing of all user activities are stored in the UserLog table. Overall, the design of this database has a clear one-to-many relationship which ensures that the data is properly structured, easy to manage and efficiently used by the system.

6.5. Security System Framework

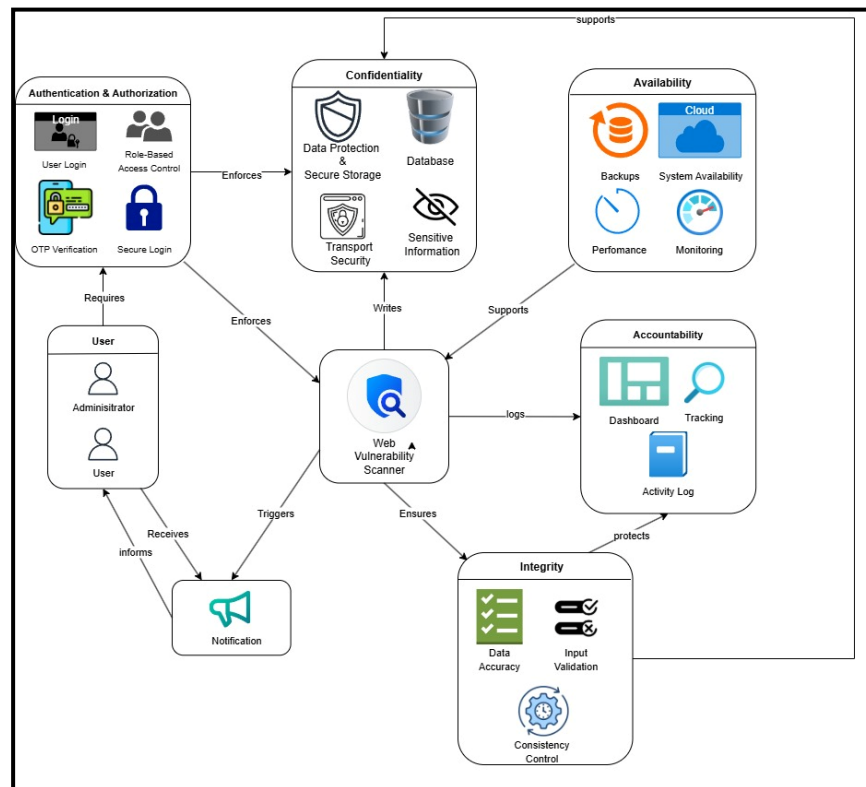


Figure 6.14 Security System Framework

This illustration shows how the basic Security concepts are integrated into the Web Vulnerability Scanner system infrastructure. The system implements authentication and authorization security components through user login, one-time passwords, and role access restrictions so that only specific users can use the system. Secure data storage and communication lines ensure the confidentiality of sensitive information and scan results through processes that protect the information securely. The system architecture also ensures that system users can always access the scanner, as System Availability is a key principle of the architecture. This is ensured through system performance and Availability monitoring, balanced system performance, and periodic system backups. Dashboards, activity tracking, and user activity logging combine to increase operational transparency, which is also a missing principle of Accountability. Finally, the system ensures Data Integrity through a rigorous process of user input validation, and through a data validation process for accuracy, and has a controlled process for consistency across the system. Therefore, the system is Secure, Reliable, and Trustworthy to users and administrators.

6.6. Docker Framework

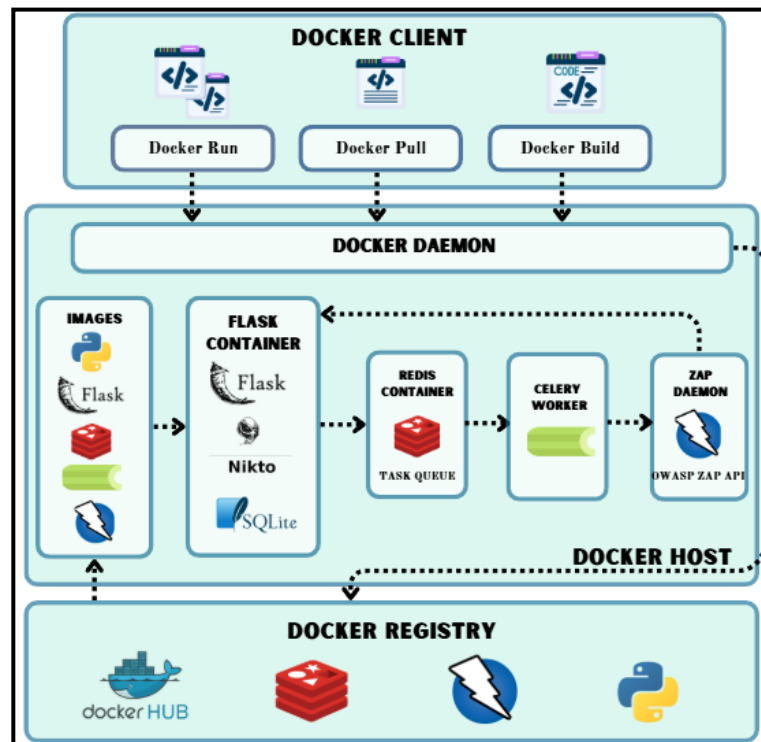


Figure 6.15 Docker Framework

This diagram shows how Docker is used to run my Web Vulnerability Scanner system in a clean and organized way. The Docker client is where commands like `docker build`, `docker pull`, and `docker run` are used to manage the application. These commands communicate with the Docker daemon, which handles all container operations in the background. On the Docker host, the system is separated into multiple containers: a Flask container runs the main web application along with SQLite and Nikto, a Redis container acts as a task queue, a Celery worker container processes background scanning tasks and a ZAP daemon container runs the OWASP ZAP API to perform automated vulnerability scanning. These containers work seamlessly, making the system more stable, more manageable, and consistent across different machines. Docker registries, such as Docker Hub, store images like Flask, Redis, and ZAP so that they can be downloaded and used at any time. Overall, Docker helps keep the system running efficiently, is easy to use, and avoids problems caused by different environments.

6.7. Flow of the system

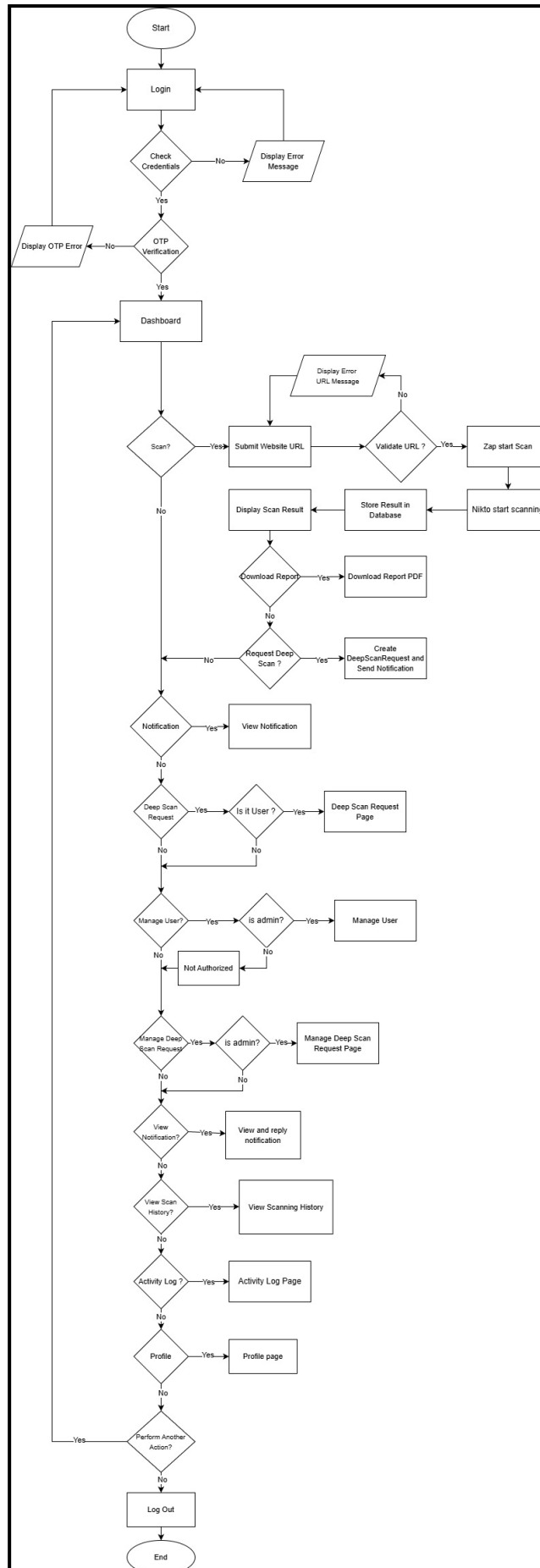


Figure 6.16 Flow of the System

Figure 6.15 shows the entire process flow followed by my Web Vulnerability Scanner system, starting with the user logging into the system until the time the user terminates the use of the system. The user first fills in his login details, and the system verifies the accuracy of the credentials. If they are incorrect, an error is shown. If the credentials are correct, the user needs to go through OTP verification to provide an additional level of security. Once the user is successfully authenticated, they are redirected to the dashboard, where they have the option of various actions including scanning the website, notifications, user management (Administrators only), deep scan requests, scan history, activity logs. After the user enters the URL to be scanned, the system verifies the URL followed by the activation of ZAP scan and Nikto scan. Once the scan is complete, these results are stored in the database and shown to the user who can also download the report or deep scan. The administrator can scan through deep scan and respond to the user. In the process, the system ensures that only the appropriate role (user or administrator) is granted access. Once the user is done, he is logged out and the session is terminated safely. This flowchart helps in getting a clear picture of how the user navigates through the system and how the system connects its various features.

6.8. Conclusion

Chapter 6 design provides a comprehensive plan on how to build a Web Vulnerability Scanner system to meet the functional and non-functional requirements that have been observed in Chapter 5. It starts with the interface design where wireframes are provided for each module such as login, OTP authentication, dashboard, scan page, results page, notifications and user management where simplicity and ease of use are prioritized. The database design section describes how tables are organized, the relationships between information and drawings such as Data Flow Diagrams (DFD) and Entity Relationship Diagrams (ERD) to ensure that information is processed in a secure and effective manner. The security framework recognizes important steps such as authentication, role-based access control, data confidentiality, integrity and availability in protecting users and system data. Docker offers a simple and effective structure that allows my web vulnerability scanner to be implemented in a consistent, reliable and easy way across different environments. The system flow is a process that describes how the system will be used by users, starting points being the login, scan, reporting and logout processes. Overall, this chapter is a good roadmap for the development phase as it lays out all the details of the system's structure and security and how users will interact with it to offer a reliable, scalable, and user-friendly solution.

7. IMPLEMENTATION

7.1. Introduction

This chapter outlines the implementation phase of the Web Vulnerability Scanner project, describing the process of transforming the system design into a fully functional web-based application. The process focuses on building the following functionalities outlined during the design stage: secure login via One Time Password (OTP) authentication, role-based access control (RBAC), which should be achieved by scanning URLs with the ZAP API in OWASP, and a clear results dashboard for users. Implementation also involves selecting suitable software and hardware platforms to support development, integrating front-end and back-end components, and configuring tools to ensure efficient and secure system performance. Next, this chapter further describes the practical steps taken during development, the actual system interface, and the key security functionalities implemented to meet the project goals and objectives. By presenting these details, this chapter shows how the system evolved from a planned design to a working solution that meets both functional and non-functional requirements.

7.2. Execution Platform

The execution platform refers to the environment where a system is built, tested, and run to ensure that all the components work together as they should. It includes software and hardware that support the development and implementation of the project. Well-defined implementation platform ensures consistency, compatibility, and stability throughout the development cycle. The implementation platform for this project was designed to provide a trusted platform that hosts the web-based vulnerability scanner and its core features, such as secure authentication, scanning, and reporting. By using the right platform, the system can be developed efficiently and tested to ensure that all modules of the system, including back-end processes and the user interface itself, work the way they should. The development and hosting environment used for this project is described in the subsections below.

7.2.1. Development Platform

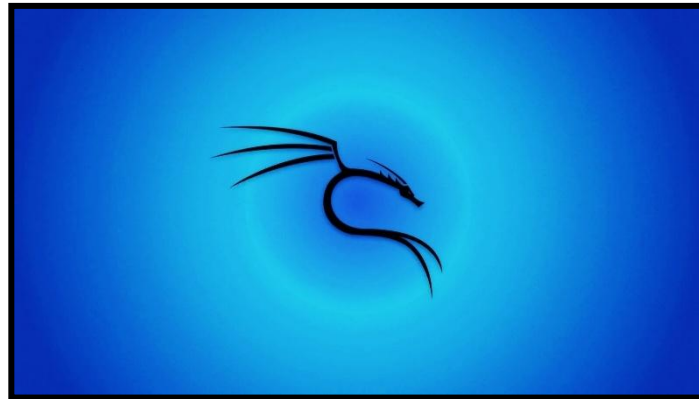


Figure 7.1 Kali Linux (Abrams, 2025)

As seen in Figure 7.1, Kali Linux, one of the Debian-based distributions, is used in the fields of Cybersecurity research, penetration testing, and proprietary software development. Kali Linux is the primary development platform for this project because it provides a stable environment that is biased towards security, is optimal for the creation of Web Vulnerability Scanners, and can be used to run tests for applications. Since Kali Linux is lightweight and has excellent support for Python and Python libraries, it can be completely reliable for performing backend and database operations of applications. Additionally, Kali Linux is security-focused, providing an operating system with a minimal number of features and easy access to the tools needed for testing and validation, thus creating a seamless environment for development. For this reason, Kali Linux is the most optimal choice for Web Vulnerability Scanner development.

7.3. Implementation Tools

Implementation tools include the set of software, programming languages, and hardware used to build and run a web vulnerability scanning system. These tools provide the environment and resources needed to code, test, and deploy the application. Software tools such as operating systems, development environments, and support utilities are also used in this project to ensure a smooth procedure for both development and scanning. Frameworks and programming languages play a key role in outlining the system logic, user interface, and its connection to the scanning engine. At the same time, hardware also provides the physical resources to run the development platform and test environment. Together, these tools form the foundation for the successful implementation and integration of all system modules.

7.3.1. Software

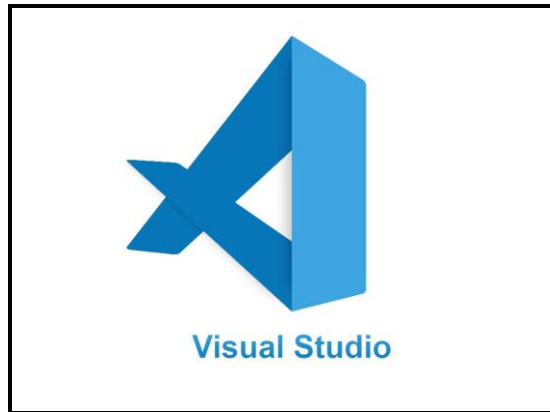


Figure 7.2 Visual Studio (Canonical, 2019)

The main integrated development environment (IDE) used to compile and maintain the source code of a project is Visual Studio Code (VS). VSCode is a simple yet powerful code editor that covers a large range of programming languages and extensions, which makes it quite suitable for developing web applications. It provides all the necessary features such as syntax highlighting, debugging, a built-in terminal and integrated version control features; thus, simplifying the coding and testing process. The extension marketplace makes it possible to include valuable extensions such as Python and Docker extensions, thus making it seamless to develop with Flask and with container environments. Using VSCode makes the development process more streamlined and organized, ensuring that the coding patterns are consistent and facilitating maintenance throughout the full project lifecycle.



Figure 7.3 Docker (Okoli, 2023)

Visual Studio Code (VS) is the most widely used integrated development environment (IDE). VS Code works flexibly with various programming languages and various forms of plug-ins which make it very easy to carry out web application development. It provides all the necessary tools for programming and test execution. It also provides very useful features for programming such as syntax highlighting, debugging, embedded terminal and integrated remote repositories. A marketplace offering several useful plugs is also available, for example, those designed for Python and Docker, which are very helpful in development and work activities involving Flask and Dockerized environments. With the features and functions provided by VS Code, the development process becomes more organized and easier, also, throughout the project it is easier to implement coding patterns which allow maintenance to be carried out more smoothly.



Figure 7.4 OWASP ZAP (K, 2025)

The project has integrated the OWASP ZAP (Zed Attack Proxy) API as the primary scanning engine for identifying vulnerabilities in web applications. In the implementation step, the API has been implemented to connect to a Flask-based web application in a Docker container. After the user enters the target site URL on the system's scanning interface, the backend initiates the scanning process via the ZAP API. The API performs automated security testing to detect common vulnerabilities and provides formatted results that are stored in a database. These findings are then plotted on a dashboard in an easy-to-understand format. The scanning workflow has been streamlined using a programmatic API, eliminating the need for manual testing and allowing the system to output vulnerability reports in near real-time. This integration helps in achieving its primary goal of providing an affordable yet powerful security assessment tool, the Web Vulnerability Scanner.



Figure 7.5 Nikto (Amal Tom Parakkaden, 2023)

Website misconfigurations and security vulnerabilities are problems that can be identified using Nikto, an open-source web server scanner. The scanner evaluates the server using a black box and matches its configuration against a database containing many known security issues, outdated software versions, and misconfigurations. The result of Nikto's scan is a list of possible vulnerabilities that the server administrator may have. Servers with outdated versions of Apache or Nginx software, exposed directories, or insecure server header configurations can all be considered vulnerable. Nikto does not harm the server. It is powerful, however, because it can list many vulnerabilities by simply considering the configuration. Widely used and easy to implement, Nikto is often added to other web vulnerability scanners as a basic web security vulnerability assessment and assessment solution. Nikto is a powerful, high-speed, and easy-to-implement web server scanner.



Figure 7.6 Flask (Iqbal, 2025)

Flask was selected as the main backend framework to create and maintain the core functionality of the Web Vulnerability Scanner. Flask is used for user authentication, page routing, and transferring data between the user interface, scanning engine (ZAP API), and the database. It implemented role-based access control and OTP authentication for increased security. It also provides integration with the ZAP API by providing an endpoint that you can use to cause the target URL to be scanned when the user submits the form. Flask is a lightweight microframework that emphasizes the separation of concerns through its modular design. Decoupled routes, models, and templates lead to improved maintainability and scalability. A built-in HTTP server and integration with other Python libraries make it easy to interact with the database and present the results of a scan. This interoperability is vital for efficient and secure vulnerability assessments.



Figure 7.7 SQLite (*SQLite Tutorial - an Easy Way to Master SQLite Fast, 2024b*)

SQLite was chosen to store all the required data created by the Web Vulnerability Scanner. In this project, the SQLite database is used to store user authentication information such as username, email address, hashed password, user type, OTP authentication, etc., which provides secure access control. It also maintains the scan history, including target URL, scan type, and timestamp, as well as user activity logs and a repository of vulnerability reports generated after each scan. The fact that it can be easily integrated with Flask and that it can communicate directly with the database without a dedicated server makes it possible due to its single file storage mechanism. Such a clean layout makes it easier to operate and offers a guarantee in the uniformity of information storage even in local development environments. SQLite transaction support provides data integrity used in the logging process, as well as maintaining a scan history, which can be used to provide traceability and accountability in the system. Its lightweight and serverless features make it an ideal solution to use in this project as it is fast, low maintenance and efficient in managing application data without other infrastructure.

7.3.2. Programming Language



Figure 7.8 Python (*File:Python Logo and wordmark.svg - Wikimedia Commons, 2008b*)

Web Vulnerability Scanner is written using Python as its primary development language due to its simplicity, readability, and rich library ecosystem. In this project, Python is largely used to develop the backend logic of the application, with the Flask web framework used in conjunction with the OWASP ZAP API, Nikto and SQLAlchemy Object-Relational Mapping to ensure smooth database interactions. The ability to interact with the ZAP scanner using Python with a RESTful API allows applications to submit scan requests and retrieve vulnerability scan results in real-time. In addition, Python scripts handle critical operations that include user authentication, input URL validation, scan scheduling, and vulnerability result processing before presentation on the dashboard. The flexibility of the language ensures that libraries that manage security features are easily integrated, thus making it easy to script the scanning process without having to maintain code. Its cross-platform nature also ensures that it can operate well in a Docker environment and there is consistency during the development and testing process. The project achieves a powerful and scalable backend, which is a key factor in secure scanning data management, by leveraging Python's simplicity and its extensive library support.



Figure 7.8 HTML (*Wikipedia contributors, 2025*)

The Web Vulnerability Scanner user interface uses HTML (Hypertext Markup Language) as the backbone of the interface. It defines the page layout for the entire page, which includes a login page, a form with fields for email and password, an OTP authentication page, a dashboard layout with menus and navigation cards, and a table that will be used to present scan results and logs. All buttons, input fields, and site areas are initially created with HTML, so that interface items are grouped semantically and are easy to read by developers and browsers.



Figure 7.9 CSS (File:CSS3 Logo and wordmark.svg - Wikimedia Commons, 2016)

CSS is used to format and organize HTML, making the system attractive and easy to use. Style sheets define the color scheme, typography, spacing, and general layout to create a clean and professional look. In addition, CSS supports responsiveness, and the interface will fit perfectly on different screen sizes. Examples include styling buttons with labels like Scan, Verify OTP, and Log Out to improve visibility and usability, and making tables with scan and log reports look like tables with similar fonts and borders to make them easier to read.



Figure 7.10 JavaScript (Lebbos, 2023)

Web Vulnerability Scanner has used JavaScript to enhance interactivity and user experience. It allows for real-time validation of form inputs, including login credentials and URL entry scans, allowing users to get immediate feedback before data is entered. JavaScript is also used to provide dynamic features with the dashboard, such as updating a real-time listing of the latest scans without having to reload the page. In addition, it handles basic front-end functionality, including enabling or disabling controls, notifications, and providing seamless navigation between parts of the system.

7.3.3. Hardware



Figure 7.11 Hardware Specification

Hardware Requirement	Specification
Computer Brand	Asus Tuf Gaming A15
Processor	AMD Ryzen 7 7435HS
Memory (RAM)	32 GB
System	64-bit operating system, x64-based processor
Operating System	Windows 11 Home Single Language
Storage	512GB

7.4. System Interface

The System Interface section contains a description of the graphical and interactive elements created for the Web Vulnerability Scanner system. It explains how users engage with the system step by step through a series of well-organized and user-friendly pages to gain the ability to navigate and access all features securely. Each interface is presented in a simple and easy-to-understand manner, so that both technical and non-technical users can log in, authenticate, start scanning, and view results without getting lost easily. The front-end is developed using HTML, CSS, Bootstrap, and JavaScript, while the back end is developed using Flask for authentication, role access, and data communication. In addition to being more usable and visually appealing, this approach is in line with the project's vision to offer a secure, accessible, and efficient platform for detecting and managing vulnerabilities in websites.

7.4.1. User Interface

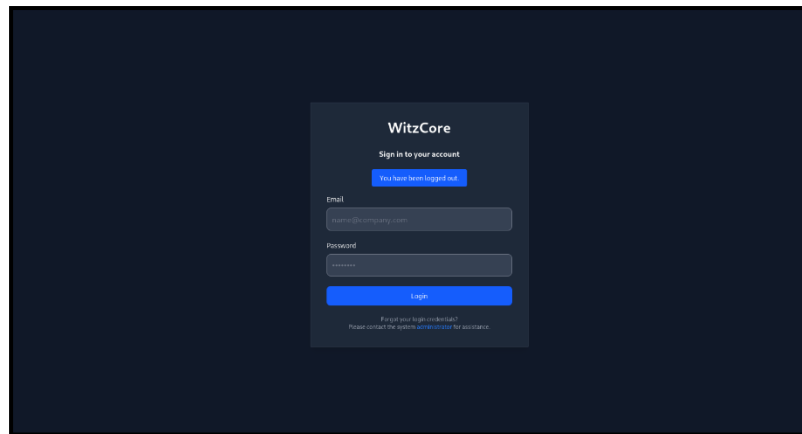


Figure 7.12 Login Page

The Web Vulnerability Scanner system offers users an interface where they can authenticate themselves. Only users such as administrators and registered user can authenticate themselves by entering a valid organizational email address along with a password. The page is developed using the Flask-Login authentication system which provides a standard way to handle session management and implement a secure authentication system. The design users see when they access the page is clean, simple, and intuitive. The simplicity of the design encourages a fast and seamless login, allowing the system to maintain high standards of security and reliability.

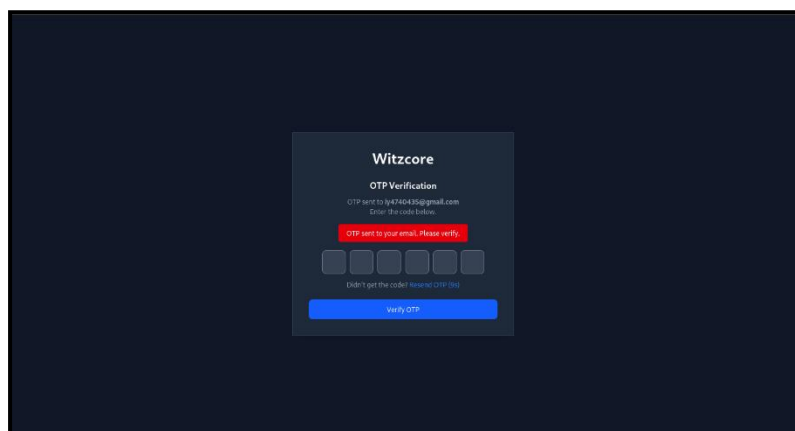


Figure 7.13 OTP Verification Page

The OTP Verification Page is a second security step to enhance the authentication process. After successful verification of login page credentials, the system automatically generates a One-Time Password (OTP) and sends it to the user's registered email address. The user must enter the OTP within the given time frame to complete the authentication process and have access to the system. This additional step greatly helps in reducing the risk of unauthorized access and compromising credentials. The integration of OTP verification is in line with the project's goal of creating a secure and trusted environment, and in doing so ensures that only authenticated users can proceed to the system dashboard and perform vulnerability scanning operations.

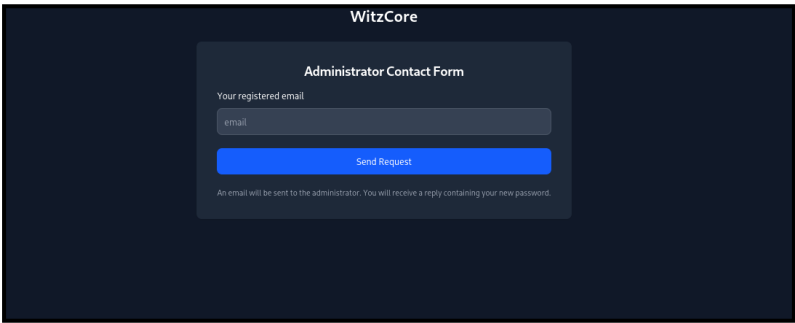


Figure 7.14 Administrator Contact Form

The Administrator Contact Form is a user interface-only form that allows users to contact the system administrator when they encounter a problem or need assistance. On this page, users simply enter their registered email address and make a request. The system automatically notifies the administrator once it is submitted and they can act on the notification. This will facilitate communication between the user and the administrator in the event of account-related issues such as password resets or access difficulties. It also helps in maintaining security as unauthorized users cannot access the administrator.

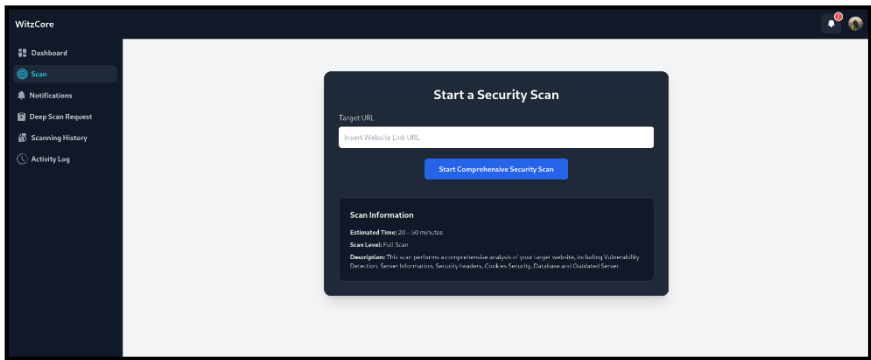


Figure 7.15 Scan Page

The Scan Page is the main working interface in the Web Vulnerability Scanner system, where users launch new security scans. This interface allows input of target URLs and configuration of scan parameters: scan type, attack strength and alert threshold according to the user's needs. Optional checkboxes allow users to select special scanning modes such as AJAX Spider and Validation Scan and thus enable more specific testing. The interface is tightly integrated with the OWASP ZAP API, allowing dynamic analysis of web applications to be automatically performed and checking for potential vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS) and misconfigurations. Designed with a clear and intuitive layout, the Scan Page ensures that users can easily initiate and manage scanning activities.

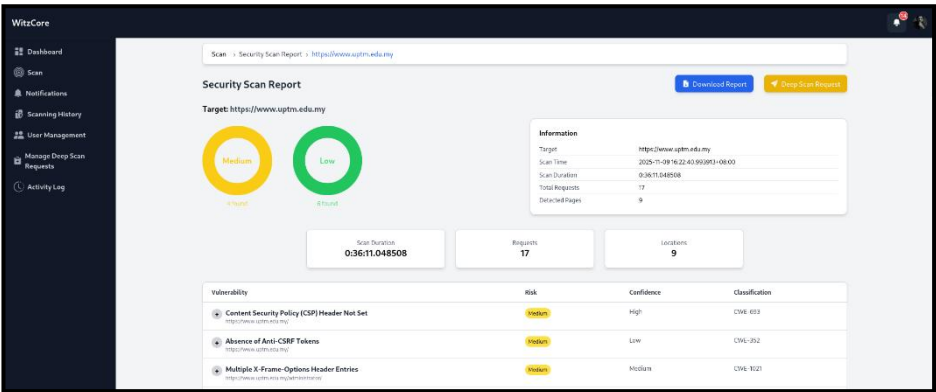


Figure 7.16 Scan Result Page

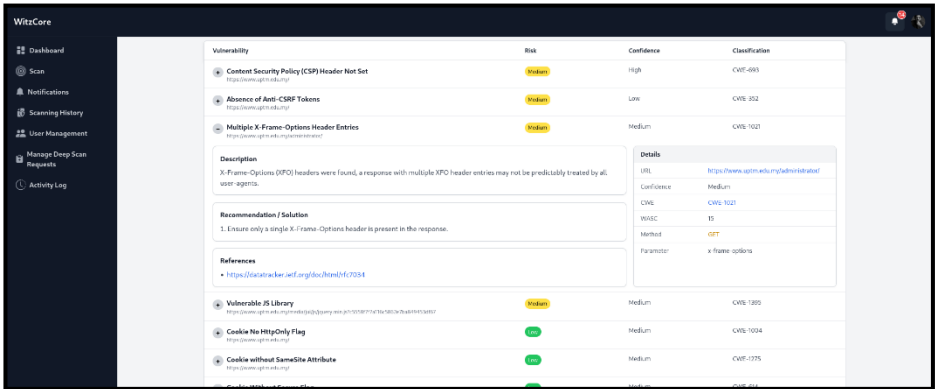


Figure 7.17 Web Vulnerability Result

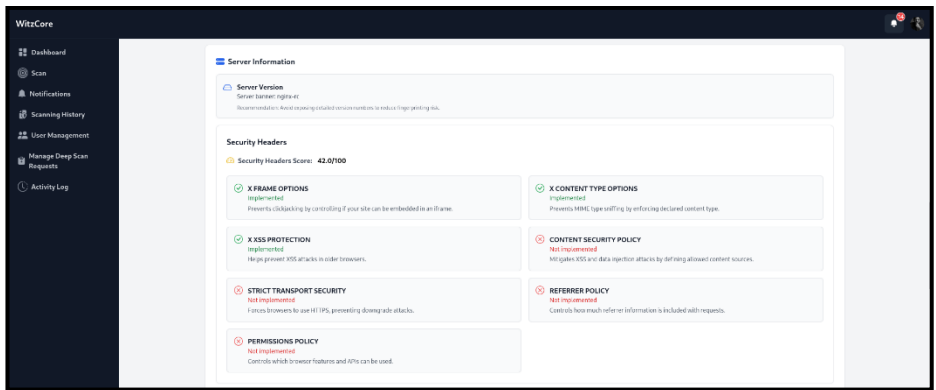


Figure 7.18 Server Information Result

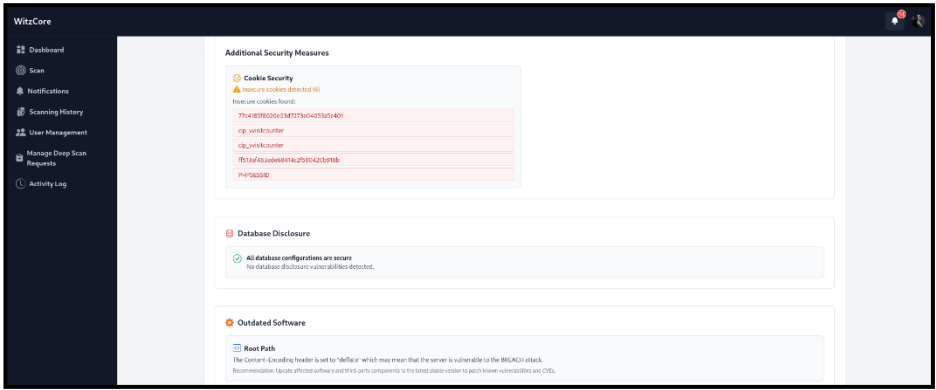


Figure 7.19 Server Information Result

This function provides details about the security scan results for the websites entered by the user. While both administrators and users have the same interface design, the system ensures that each user only has access to the scan results that are relevant to their organization. The system presents the scan results on the website in a clean and easy-to-understand manner. The most prominent section is a summary that lists the number of vulnerabilities that have been found and places them in their respective severity collections, namely Medium and Low. Additionally, it outlines critical scan data such as the portal scanned, timestamp, scan duration, number of pages detected, and number of requests sent. The following section of the summary is a table where each vulnerability is listed along with its corresponding risk, confidence, and classification (CWE/WASC) level. Users have the ability to collapse or expand the table to view the vulnerability details as well as detailed disclaimers, recommendations, external references, and technical data that includes the attacked URL, attack method, parameters, and payload.

Additional features for server-side application results provide server version information and the status of several important security headers, including X-Frame-Options, X-Content-Type-Options, HSTS, CSP, and Referrer-Policy. It tells the user if the server is implementing basic security measures. Either the administrator or the user can use the "Download Report" button to get a full report or can use the "Request Deep Scan" menu to initiate a more extensive scan. This interface has the advantage of systematically detailing and organizing the presentation of various vulnerabilities in a way that is easy for the user to understand to facilitate efficient identification of vulnerable security gaps.

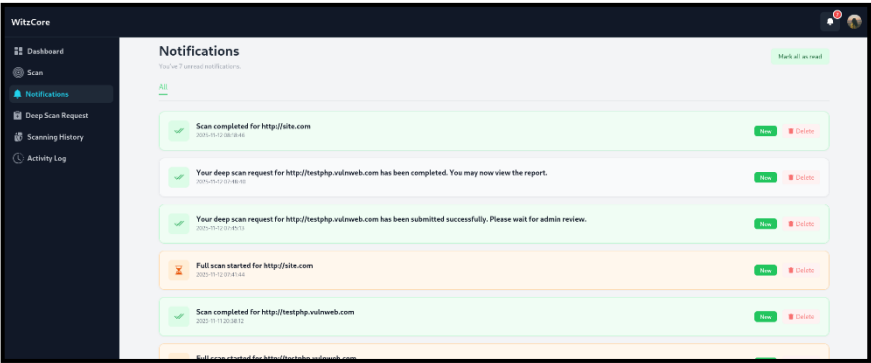


Figure 7.20 Notification Page

The Notifications page for admin and default users is designed the same. Notifications are presented in a simple, unified layout, presenting a list of system updates in order such as, scan completed, status of your deep scan request, and account-related notifications. Despite the similarity of the interface, each user is sent purpose-specific notifications related to their activities and organization. This process is designed to keep users informed while controlling privacy and access limits.

The screenshot shows the 'Scan History' page in the WitzCore interface. The left sidebar contains navigation links: Dashboard, Scan, Notifications, Scanning History (selected), User Management, Manage Deep Scan Requests, and Activity Log. The main content area is titled 'Scan History' and shows a table of scan records. The table has six columns: Target, Status, Started, Duration, Result, and Action. The table lists ten completed scans for various targets, including http://testphp.vulnweb.com and https://www.opm.gov. Each row has a 'View Details' link and a red icon.

TARGET	STATUS	STARTED	DURATION	RESULT	ACTION
http://testphp.vulnweb.com	Completed	Nov 15, 2025 10:11	0:11:57.603175	View Details	
http://testphp.vulnweb.com	Completed	Nov 12, 2025 00:12	0:11:58.135149	View Details	
https://www.opm.gov	Completed	Nov 09, 2025 16:22	0:36:11.648568	View Details	
https://www.opm.gov	Completed	Nov 09, 2025 12:11	2:04:00.371895	View Details	
http://testphp.vulnweb.com	Completed	Oct 13, 2025 01:36	0:00:17.085319	View Details	
http://testphp.vulnweb.com	Completed	Oct 13, 2025 01:19	0:00:24.971019	View Details	
http://testphp.vulnweb.com	Completed	Oct 12, 2025 18:07	0:12:05.851245	View Details	
http://testphp.vulnweb.com	Completed	Oct 12, 2025 03:22	0:02:40.827267	View Details	
http://testphp.vulnweb.com	Completed	Oct 12, 2025 02:44	0:00:08.890722	View Details	
http://testphp.vulnweb.com	Completed	Oct 11, 2025 18:07	0:47:09.303893	View Details	

Figure 7.21 Scan History Page

Administrators and users alike have access to the Scan History interface, and the software shows a record of all scans that have ever been run. This section contains the following columns, target URL, status, scan date and time, duration of the scan, as well as reports that users can click on to view more details. The interface is the same, however, users can only view completed scans using their own credentials, while system administrators have access to more data based on their linked organization. This method of using a parallel design increases efficiency while data access is limited.

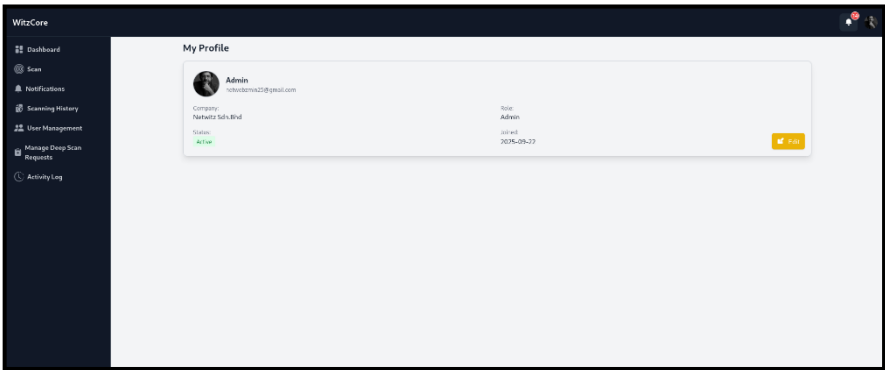


Figure 7.22 Profile Page

The My Profile page shows the same layout for all users, allowing them to view and update their personal details, profile picture, company, role, and account status. The design ensures a consistent user experience across roles, but each user can only access and edit their own information.

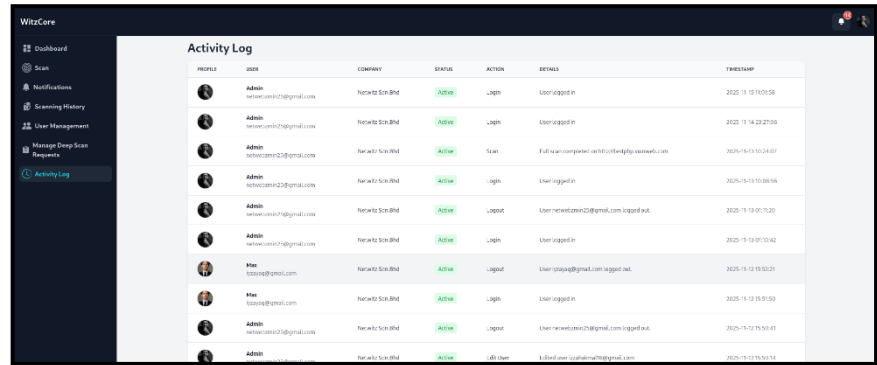


Figure 7.23 Activity Log Page

Both administrators and users interact with the same panel in the Activity log. It shows the history of each person's account activity in an audit trail, including login history, scan executions, profile modifications, system commands, and timestamps. However, standard users are limited to viewing only the log history that corresponds to their account. Administrators are allowed to monitor the activity logs of their users within their organization. While the depth of information varies, the system design is intended to be consistent, allowing for uniformity.

7.4.2. User-Only Interface: Dashboard and Deep Scan Request Page

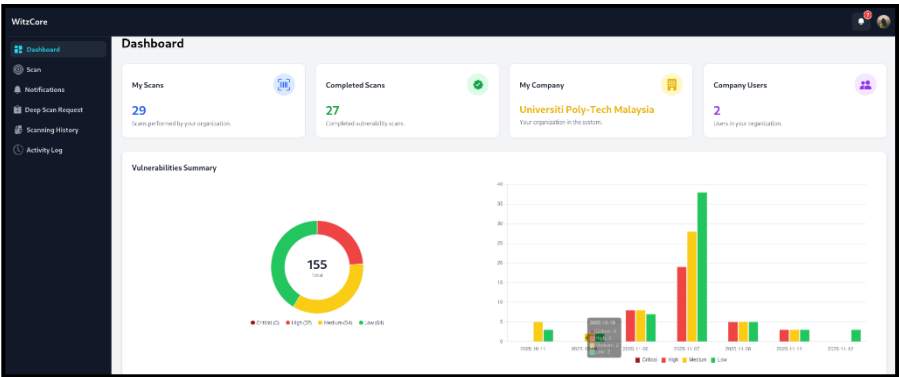


Figure 7.24 User Dashboard Page

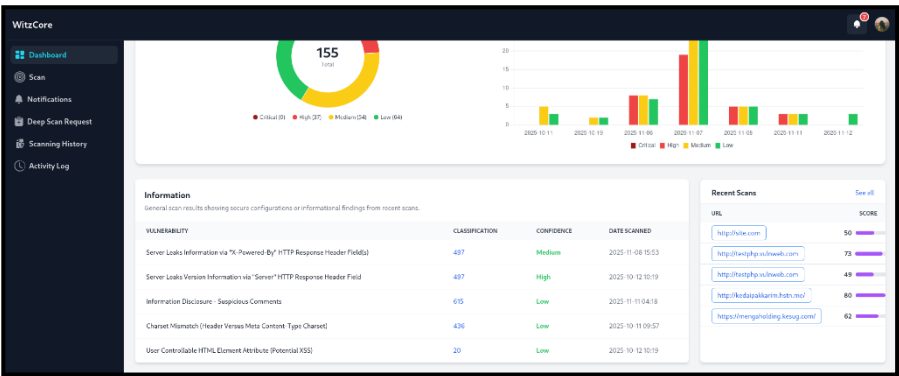


Figure 7.25 User Dashboard Page

The user dashboard shows a user-specific summary of all security scanning activity for the user's organization. The user dashboard shows summary details by organization, including the total number of scans by user, the total number of completed scans, the user's organization name, and the number of users in the same organization. The summary includes charts such as a chart for vulnerability summary and a trend chart that shows the user's security status for the scanned website. The dashboard contains a detailed record of the latest information findings showing the vulnerability title, vulnerability type, confidence level, and scan date. On the right side of the dashboard is a "Recent Scans" panel where users can find their most recently completed scan. Overall, the user dashboard is intended to allow users to track their web security activity and understand their organization's vulnerabilities.

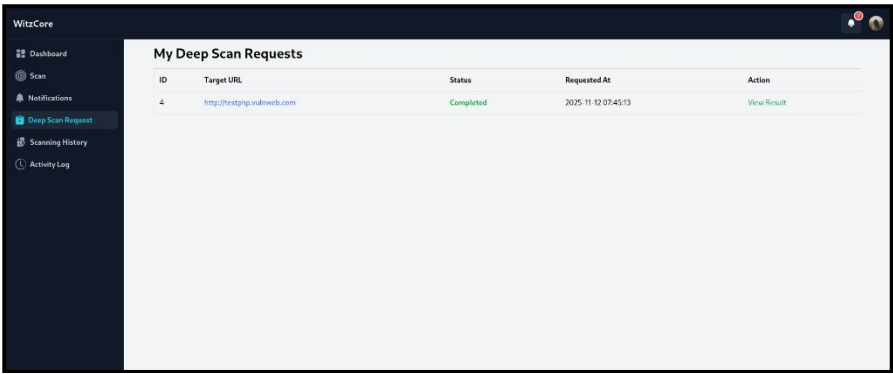


Figure 7.26 Deep Scan Request

The My Deep Scan Requests page is accessible only to regular users. This page outlines all deep scan requests submitted by users. This includes the target URL, submission date, and status. Once an administrator has completed and approved a deep scan, users have access to the results via the "View Results" action button. While both administrators and users have access to most of the system interface, this feature is user-only, as only users have the authority to submit deep scan requests. This division facilitates a streamlined workflow, where users request secondary scans, and administrators are tasked with evaluating and executing these requests.

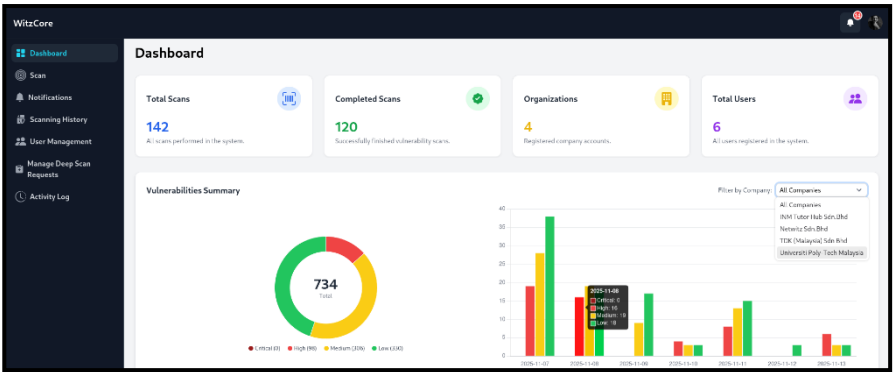


Figure 7.27 Admin Dashboard Page

The overall admin dashboard system, across each registered organization, provides the system. In contrast to the user dashboard, which provides information for a single company, the admin dashboard provides aggregate system statistics including total scans, total scans completed, total organizations, and total users. This information aids the administrator's ability to assess the system's usage and effectiveness. The dashboard displays an overview of the system and presents a summary of system vulnerabilities and trend charts over time. This can be filtered by organization so that the administrator can analyze security patterns over time for different organizations. The admin dashboard, while structured similarly to the user interface, encompasses the system rather than a single user. The purpose of the system is to assist the administrator in controlling and defending the organization, maintaining system functionality, and ensuring that security scans are comprehensive.

PROFILE	USER	COMPANY	ROLE	STATUS	CREATED	ACTIONS
	Izzah izzahakmal10@gmail.com	IKM User Hub Sdn Bhd	User	InActive	2025-11-12	
	Muhamad muhamad1027@gmail.com	TDK Malaysia Sdn Bhd	User	Active	2025-10-19	
	JOJO jojo1025@gmail.com	Unikawal Negeri- web Malaysia	User	Active	2025-09-14	
	Admin admin102@gmail.com	Nurafid Sdn Bhd	Admin	Active	2025-09-12	
	Marta marta101027@student.upm.edu.my	Unikawal Negeri- web Malaysia	User	Active	2025-09-11	
	Mua mua102@gmail.com	Nurafid Sdn Bhd	Admin	Active	2025-09-12	

Figure 7.28 User Management Page

This interface is open to administrators and gives them full control over all users registered in the system. Administrators can see a full list of users, their names, emails, companies, roles, statuses, and dates they were created. This page also has filters that can be used to view users by company. With the help of this interface, administrators can add new user accounts, update current details, or remove user accounts if required. This role is necessary to ensure user access is maintained and secured within the organization's needs.

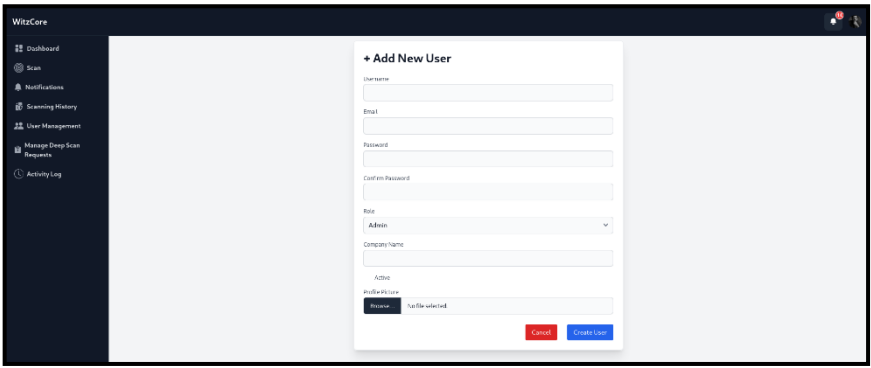


Figure 7.29 Add New User Page

The Add New User interface provides the functionality to register new users within the system. Admins must enter the necessary information, including the username, email, password, role (Admin or User), company name, and may upload a profile picture. This page provides the capability to create new accounts in an organised and safe manner, thereby facilitating the secure onboarding of new users to the system.

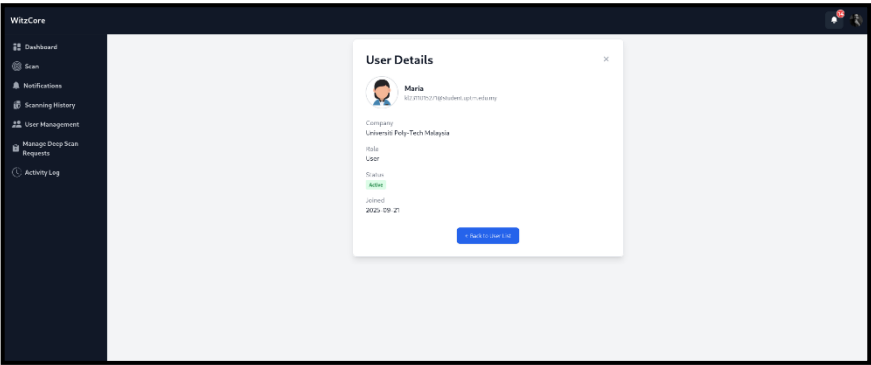


Figure 7.30 User Details Page

The User Details interface outlines the complete profile corresponding to a particular user. It presents essential information, such as the user’s name, email, organization, role, status, and date of entry into the system. This page is useful for administrators to get a summary of the user information and perform actions such as updating or deleting the user’s account.

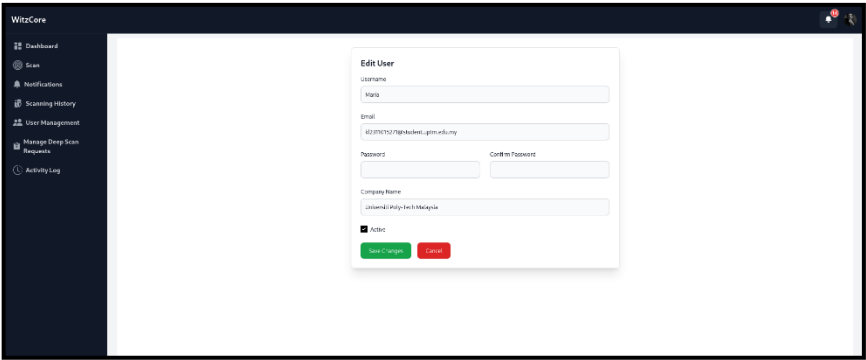


Figure 7.31 Edit User Page

The Edit User page allows administrators to change user information, should updates be required. Administrators can adjust user attributes, including username, email, password, company name, account status, and account state. This interface ensures the integrity and currency of user information, which is essential for system security and effective system user administration.

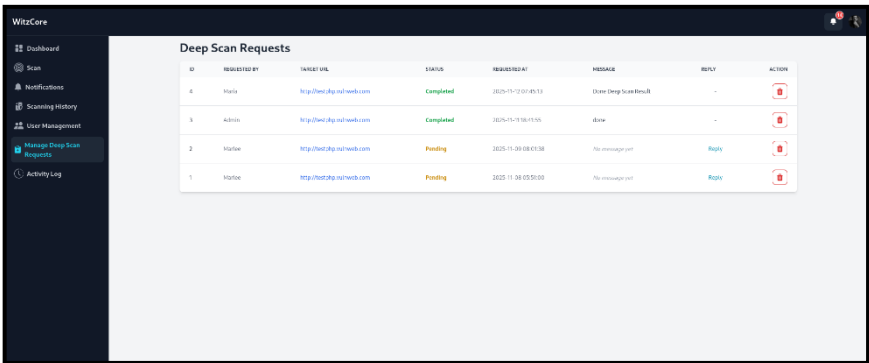


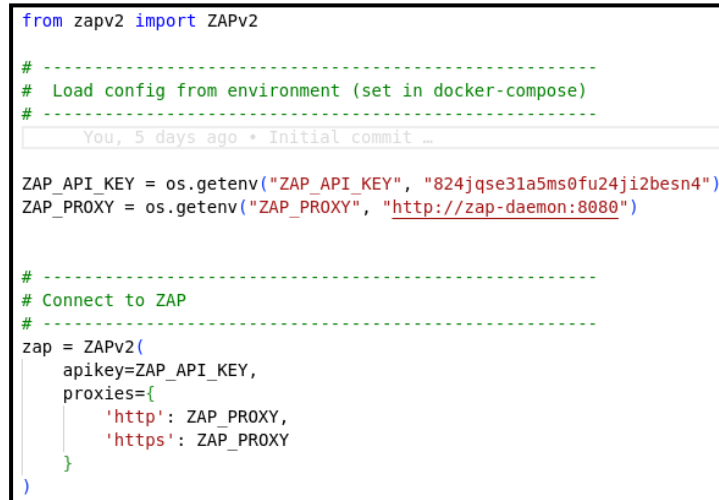
Figure 7.32 Manage Deep Scan Request

The Manage Deep Scan Requests interface is only available to administrators, and it is used to process all deep scan requests made by users. Here the entire list of requests is shown, with the requester name, target URL, request status (Pending or Completed), request date, and any messages provided by the administrator. Administrators can open each request and perform a deep scan manually and then update the request by including the scan result link and adding a reply. This interface has provided sufficient coordination between users and administrators where users can access verified and detailed scan results and administrators can view and act on all requests effectively.

7.5. Significant Functions

7.5.1. Web Application Security Scanning Using OWASP ZAP

1. ZAP API Key Authentication



```

from zapv2 import ZAPv2

# -----
# Load config from environment (set in docker-compose)
# -----
You, 5 days ago • Initial commit ...

ZAP_API_KEY = os.getenv("ZAP_API_KEY", "824jqse3la5ms0fu24ji2besn4")
ZAP_PROXY = os.getenv("ZAP_PROXY", "http://zap-daemon:8080")

# -----
# Connect to ZAP
# -----

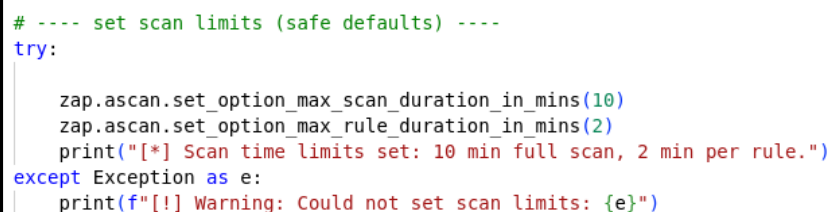
zap = ZAPv2(
    apikey=ZAP_API_KEY,
    proxies={
        'http': ZAP_PROXY,
        'https': ZAP_PROXY
    }
)

```

Figure 7.33 ZAP API Key Authentication

In this section of code, the system links to OWASP ZAP with the help of an API key, which serves as a secure password. API keys are not written in the code itself, instead they are stored in environment variables, which can be kept secret and secure. This is to ensure that only authorized programs or users can communicate with ZAP and run security scans. This key cannot be duplicated, and without it, no one can initiate or control the scan, which eliminates the possibility of misuse of the scanner by outsiders. The importance of this coding part is that it protects the security tool itself from unauthorized access and only trusted users can perform vulnerability testing.

2. Scan Time and Rule Limit to prevents abuse and overload



```

# ---- set scan limits (safe defaults) ----
try:
    zap.ascan.set_option_max_scan_duration_in_mins(10)
    zap.ascan.set_option_max_rule_duration_in_mins(2)
    print("[*] Scan time limits set: 10 min full scan, 2 min per rule.")
except Exception as e:
    print(f"[!] Warning: Could not set scan limits: {e}")

```

Figure 7.34 Scan Time and Rule Limit

This part of the coding sets a time constraint on the period during which ZAP can operate with its active scan. By doing so, the system will avoid time-consuming or infinite scans that can affect the server or use excessive CPU and memory. Scanning time limits can stabilize the system and prevent it from accidental Denial of Service caused by excessive scanning or frequent scanning. This is a security factor because it will guarantee that the scanning process is controlled, predictable and will not accidentally damage the functionality of the web application or server.

3. Vulnerability Grouping and Security Data Processing

```

grouped = {}
for alert in raw_alerts:
    plugin_id = str(alert.get("pluginId") or alert.get("id") or "").strip()
    key = {
        plugin_id,
        alert.get('alert'),
        alert.get('risk'),
        alert.get('description'),
        alert.get('solution')
    }

    if key not in grouped:
        grouped[key] = {
            'pluginId': plugin_id,
            'alert': alert.get('alert'),
            'risk': alert.get('risk'),
            'description': (
                alert.get('description')
                or alert.get('desc')
                or alert.get('otherinfo')
                or "No description available"
            ),
            'recommendation': (
                alert.get('solution')
                or alert.get('recommendation')
                or "No recommendation provided"
            ),
            'evidence': alert.get('evidence'),
            'instances': [],
            'uri': alert.get('url') or alert.get('uri'),
            'method': alert.get('method'),
            'param': alert.get('param'),
            'attack': alert.get('attack'),
            'response_time': alert.get('responseTimeInMs') or 0,
            'confidence': alert.get('confidence'),
            'cweid': alert.get('cweid'),
            'wascid': alert.get('wascid'),
            'reference': alert.get('reference'),
            'otherinfo': alert.get('otherinfo')
        }

```

Figure 7.35 Vulnerability Grouping

ZAP produces many results after a scan, and some results may show the same result in multiple areas. This code arrives at the results and merges them so that the program will show only duplicate and significant results. This extracts important areas such as risk indicators and described advice and removes duplicates. This is to prevent false results and ensure that the results shown to users are accurate and understandable. This is a security factor because it ensures that the most vulnerable data available will be handled in a secure manner without many errors that could result in complex, unprocessed, or repetitive data.

7.5.2. Server Vulnerability Detection Using Nikto

1. Security Header Checking

```
def fetch_security_headers(target_url):
    try:
        response = requests.get(target_url, timeout=10, verify=False)
        headers = {k.lower(): v for k, v in response.headers.items()}

        result = {}
        for h in HEADER_LIST:
            result[h] = "Implemented" if h in headers else "Missing"

        return result
    except Exception as e:
        print(f"[WARN] Could not fetch live headers: {e}")
        return {h: "Missing" for h in HEADER_LIST}
```

Figure 7.36 Security Header Checking

During this part of the code, the system makes a request to the target site and determines the presence of key security headers. Security headers are configurations that help secure web pages against attacks such as clickjacking, cross-site scripting, or content injection. The code places the actual headers of the site against a list of suggested headers including X-Frame-Options, Content-Security-Policy, Referrer-Policy, and others. If a header is missing, the system will mark it as missing and if it is identified, it will mark it as implemented. This is a security aspect as it helps in discovering weak server settings that might be exploited by malicious individuals and the site users are in a better position to understand the security measures that are not implemented to protect their site.

2. Categorizing Server Security Findings

```
def categorize_finding(desc: str) -> str:
    desc_lower = desc.lower()

    if "server:" in desc_lower:
        return "Web Server Type"
    elif "x-powered-by" in desc_lower:
        return "X-Powered-By Header"
    elif (
        "header is not set" in desc_lower
        or "header not set" in desc_lower
        or "missing header" in desc_lower
        or "suggested security header missing" in desc_lower
    ):
        return "Security Headers Missing"
    elif "phpmyadmin" in desc_lower or "pgadmin" in desc_lower or "mysql" in desc_lower:
        return "Database Disclosure"
    elif "cookie" in desc_lower:
        return "Cookies Security"
    elif "outdated" in desc_lower or "vulnerable" in desc_lower or "cve-" in desc_lower:
        return "Outdated Software "
    else:
        return "General Finding"
```

Figure 7.37 Categorizing Server Security

This section of code accesses Nikto's output and automatically sorts the results into various security categories. For example, when the output mentions missing headers, it is stored under Missing Security Headers, where it indicates outdated server software or version it is stored under Outdated Software, pages that have been exposed to a database are stored under Database Exposures and cookie issues are stored under Cookie Security. The code accomplishes this by searching through the text and identifying several keywords and mapping them to the appropriate categories. This is a security feature as it helps the system understand the raw security information and presents the information in a simple and systematic way that is easier for users with less technical expertise to understand the risks present on their server.

3. Saving Server Vulnerabilities to Database

```
for header, status in live_headers.items():
    db.session.add(ServerVulnerability(
        scan_id=scan_id,
        category="Security Header",
        title=header.title(),
        description=f"{header.title()} - {status}",
        recommendation="Implement this security header to enhance protection." if status == "Missing" else "Header properly implemented."
    ))
db.session.commit()
```

Figure 7.38 Saving server Vulnerabilities

Once the server vulnerabilities have been identified and classified, this section of the code stores the detected vulnerabilities in the system database. Each entry contains information such as the nature of the problem, a brief statement and a solution. As an illustration, when the security header is missing, the system records that information and a suggestion to turn it on. The ability to save findings will ensure that the user can look up their server security issues in the future, create a report or get the entire PDF scan report. This is a security factor as the system maintains a permanent record of vulnerabilities which is easy to detect, report and make decisions to improve the security of the site.

7.5.3. OTP Verification

```
import pyotp
import secrets

def generate_otp_secret():
    return pyotp.random_base32()

def generate_otp_code(secret):
    totp = pyotp.TOTP(secret, interval=300) # code valid for 5 minutes
    return totp.now()

def verify_otp(secret, code):
    totp = pyotp.TOTP(secret, interval=300)
    return totp.verify(code)
```

Figure 7.39 OTP Verification

Another security feature of this system is the OTP authentication which will provide an additional layer of protection when the user logs in. OTP (One Time Password) is created by a time-based algorithm, which is a code that is different every few minutes and can be used only once. When the user enters the email and password, a unique OTP code can be sent to the email. The system verifies the OTP in the code by matching the OTP entered by the user with the OTP generated with the help of a secret key, which is stored against the user. When a match is made, the user is allowed to log in, otherwise the user is not allowed to log in. This can be used to prevent illegal access even if the person who knows the user's password is the same. The encryption also removes the session OTP once it is used to prevent reuse. This security component secures the user account so that only the user who has access to the registered email can go through the login process and it is very difficult to hack into the system.

7.5.4. Role-Based Access Control (RBAC)

```
from flask import flash, redirect, url_for
from flask_login import current_user
from functools import wraps

def admin_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if current_user.role != 'admin':
            flash('You are not authorized to access this page.', 'danger')
            return redirect(url_for('main.dashboard'))
        return f(*args, **kwargs)
    return decorated_function
```

Figure 7.40 Role-Based Access Control

Role-Based Access Control (RBAC) policies have been implemented to protect confidential pages and actions within the system. This code uses the Administrator role decorator to verify that the user is an Administrator before being granted access to certain paths in the system. The system automatically blocks users without administrator privileges from attempting to visit the admin-only page, displaying an error and redirecting the user's dashboard. The importance of this role access control function is that the administrator is the only user equipped to manage users, approve deep scan requests, clean up records, and perform

actions with very high privileges. This role access control function will protect the system from unauthorized modification of sensitive data and ensure that only trusted individuals have control over critical system functions. This embedded control fosters positive system oversight and greatly reduces the risk of malicious activity in the system.

7.5.5. CSRF Exemption Decorator

```
def csrf_exempt(view_func):  
    """Manually exempt a route from CSRF validation."""  
    @wraps(view_func)  
    def wrapped_view(*args, **kwargs):  
        return view_func(*args, **kwargs)  
    wrapped_view._csrf_exempt = True  
    return wrapped_view
```

Figure 7.41 CSRF Exemption Decorator

The CSRF exemption decorator system configures how each route handles CSRF. Typically, CSRF protection prevents attackers from asking users to perform actions with the request by validating tokens on the request. But there are some other routes like API or AJAX endpoints that may not include CSRF tokens because they are triggered differently or handled by a completely different system. These routes are marked with the `csrf_exempt` decorator to indicate that the route can bypass CSRF checks, so as not to compromise the system. The decorator in the code wraps the original function and leaves a marker that tells Flask to ignore CSRF checks for that route. This is a safe security outcome because it gives developers a lot of control over where CSRF protection is needed and that CSRF protection can be intentionally turned off so that the system can continue to function, all without completely removing CSRF security.

7.5.6. Activity Logging

```
from app import db
from app.models import UserLog
from datetime import datetime

def log_user_action(user_id, action, details=None):
    log = UserLog(
        user_id=user_id,
        action=action,
        details=details,
        created_at=datetime.utcnow()
    )
    db.session.add(log)
    db.session.commit()
```

Figure 7.42 Activity Logging

```
if verify_otp(otp_secret, otp_input):
    login_user(user)
    log_user_action(current_user.id, "Login", "User logged in")
    session.pop('otp_user_email', None)
```

Figure 7.43 Activity Logging Message

```
log_user_action(current_user.id, "Delete Scan", f"Deleted scan for {scan.target_url}")
```

Figure 7.44 Activity Logging Message

Activity logging is used to document important user actions in the system, including system logins, file uploads, delete scans, or deep scan requests. The user action log records in the code include information such as the user ID, the action taken, the time it was taken, and any additional information in the UserLog table. The importance of this security element is that it leaves an audit trail that can guide administrators to understand what has happened in the system, especially when there is suspicious or unexpected behavior. Monitoring and recording such activities increase accountability and can assist in investigations in the event of a security breach. It also provides transparency as all significant changes or activities can be reviewed in the future to ensure that the system remains intact and secure.

7.6. Conclusion

In conclusion, this chapter introduces the implementation stages of the entire Web Vulnerability Scanner system, including both technical and security factors that ensure the project will function effectively and securely. This chapter discusses the tools used in software and hardware development such as Flask as the main backend framework, SQLite as the database, and OWASP ZAP API as the built-in scanning engine in a user-friendly web application. The system interfaces such as the Login Page, OTP Authentication Page, Scan Page, Scan Result Page, User Management Page, and Activity Log Page are designed in a way that provides secure, effective, and user-friendly user interaction. In addition, an outline of several potential interfaces, namely, the Dashboard, Notifications, Scan History, and Settings pages, has been outlined to demonstrate future system enhancements. The key functionality section identifies the security controls that have been integrated into the system, including CSRF exemption detector, two-factor authentication role-based access controls, background vulnerability scanning, and activity logging. All of these implementations demonstrate the system's ability to perform good web vulnerability scanning and maintain high levels of authentication, authorization, and data protection - thus achieving the project's goal of creating a secure, efficient, and user-friendly vulnerability assessment system.

8. TESTING

8.1. Introduction

This chapter describes the testing activities aimed at understanding the performance and quality of the Web Vulnerability Scanner system. The goal of the testing effort is to verify that the system is functioning properly and presents accurate results about vulnerabilities and to ensure that the system is secure and easy to use for all stakeholders. In this phase, several aspects were examined including the performance of the system to deliver results, system reliability, system responsiveness and system ease of use. The testing also ensured whether some basic security functions such as OTP Verification, role-based access control, secure scanning, session management and logging worked as expected under varying conditions. A systematic approach to testing was used, where each module was tested separately, and then retested as the system was to come for proper integration. The testing was done from both the developer and user perspectives, while the clients was asked to provide feedback on the functionality and user-friendliness of the system. All the information obtained from the testing activities was compiled to understand how well the system met its requirements and whether it was ready for real-world use.

8.2. Functional Testing

Functional testing was conducted to check whether all the features in the Web Vulnerability Scanner system are working as required. These tests check the functionality of the system from the user perspective and include functions such as authentication, scanning, user administration, reporting and activity logging. The system was tested to ensure that, for each function, it accepts valid input, rejects invalid input and no discrepancies exist between the user actions and the system responses. Some of the functions tested include logging into the system, OTP authentication, initiating basic and deep scans, generating and downloading vulnerability reports, user account management and retrieving scan history. For user functions, the tests showed that users can submit deep scans, monitor their scan status asynchronously and get notifications about their scans without being intrusive. For administrator functions, we tested that administrators could add users, edit user details, filter the user list and delete users with appropriate access level controls. For all meaningful user actions, we aimed to use truly realistic test scenarios, and the results concluded that each system module is functional and the system is fully functional without any hotspots. The key features of the system are proven to be functional, making the system reliable for users.

8.2.1. Unit Testing

To evaluate the functionality of each web application of the Web Vulnerability Scanner system, unit tests were completed to ensure that each module was functioning properly before the application was embedded. Such tests are important because bug detection is done early in the development cycle, which helps eliminate future maintenance expenses, as well as prevent larger bugs during subsequent testing phases. They also help developers gain a deeper understanding of the system by clearly showing how each function should behave, thus, aiding in more secure code modifications. In this case, unit tests focus on the main functions of the system, such as user authentication, OTP generation and OTP validation, vulnerability scan initiation, protected file transfer, and access control policy enforcement. In addition, unit tests in this case serve as a form of documentation that describes the desired results of various functions. Unit tests also facilitate code duplication as unit tests can be reused in future system migrations. In general, unit tests are important in this system during its development to ensure that the system is stable and reliable.

Table 8.1 Unit Testing Table

Component Tested	Test Step	Expected Output	Status
Celery Initialization	Load Celery app with Redis URL	Celery app initializes without errors	Pass
test_task()	Call test_task("Izzah")	Returns "Hello, Izzah! Task executed successfully."	Pass
run_full_scan_task	Simulate task start with valid parameters	App context loads and scan record retrieved	Pass
ZAP Scan Trigger	Provide valid target URL to run_zap_scan() through Celery	Returns list of ZAP alerts	Pass
ZAP Scan Failure Handling	Simulate ZAP connection failure	Task continues without crashing; alerts = []	Pass
Response Time Measurement	Send requests.get() to target	Returns time in milliseconds or None	Pass
Nikto Scan Trigger	Call run_nikto_scan(target_url)	Returns dict with findings, banner, headers	Pass
Nikto Scan Timeout Handling	Force long-running Nikto process	Returns { "error": "timeout" }	Pass
Parsing Nikto Results	Pass raw Nikto text into parse_nikto_output()	Returns categorized findings and header list	Pass
Server Vulnerability Storage	Insert ServerVulnerability entries	Records saved with correct fields	Pass
ZAP Vulnerability Storage	Insert multiple ZAP alerts into DB	All fields saved: title, URL, method, evidence	Pass
Custom Alert Overrides	Provide alert plugin ID found in CUSTOM_ALERTS	Description & recommendation replaced	Pass
Scan Summary Generation	After all findings inserted	Summary includes ZAP + Nikto results	Pass
Mark Scan Completed	End of task → finalize block	Scan status = "completed" and timestamp updated	Pass
Notification Creation	Call create_notification(user_id, msg)	Notification stored in DB	Pass
Activity Logging	Call log_user_action(user, action)	Log entry saved successfully	Pass
JSON Output Path	Provide scan_id in Nikto	JSON path generated correctly	Pass

8.2.2. Integration Testing

Integration testing ensures that separate system modules function properly when integrated. These tests emphasize the interaction of system constituents such as authentication, scanning engines, databases, PDF generation, notifications, and asynchronous background tasks managed by Celery.

Table 8.2 Integration Testing Table

Component Tested	Test Step	Expected Output	Status
Login Module + OTP Module	Enter valid login credentials and generate OTP, then verify OTP	System authenticates user and redirects to dashboard	Pass
Login Module + Notification System	Login successfully	Login action recorded and notification triggered	Pass
Scan Page + ZAP Engine + Database	Enter URL, Start Scan, ZAP triggered, Scan stored	Scan runs, data saved into Scan and Vulnerability tables	Pass
Scan Page + Celery Worker + Nikto Scanner	Start full scan, Task sent to Celery. Nikto scan executed	Background scan completes and results stored	Pass
Scan Result Page + Vulnerability Model	View completed scan result	Page displays vulnerabilities grouped by severity	Pass
Scan Result + PDF Export	Click "Download Report"	System generates PDF with ZAP + Nikto results	Pass
Deep Scan Request + Email Service	User submits deep scan request	Email sent to admin with PDF attachment + request logged	Pass
Admin Deep Scan Reply + File Upload	Upload Admin uploads result file and Submit reply	User receives notification and can download result	Pass
User Management + Database	Admin adds user and saves	New user stored in database and appeared in user list	Pass
User Management + Profile Picture Upload	User uploads profile picture and save	Image stored and displayed correctly	Pass
Activity Log + User Actions	Perform actions (login, scanning, delete scan)	Logs recorded in UserLog table	Pass
Notifications + CRUD Operations	View, mark as read, delete notifications	Notification state updates correctly	Pass
Scanning History + Database Retrieval	Open scanning history	History displayed with date, duration, scan ID	Pass
Access Control (RBAC) + Route Protection	User tries admin URL	Access denied / redirect to login	Pass

8.2.3. System Testing

System testing is performed to analyze the integrated application to check if all the parts are working as required. At this point, all the components of the web vulnerability scanner are tested to check if the expected results are met and the application is working properly from the end user's perspective. Additionally, this step involves testing the functionality between the modules and the system's response to different inputs, if all the outputs match the inputs, and if there are any variations to the outputs across various end-to-end scenarios.

Table 8.3 System Testing Table

Scenario	Test Step	Expected Output	Status
Full Login Process	Enter email and password, Submit, Enter OTP and Verify	User successfully logs in and is redirected to dashboard	Pass
Full Scan Execution	Go to Scan Page and insert URL, Start Scan and wait for completion	Scan runs successfully using ZAP & Nikto and stored in database	Pass
Scan Result Display	Open completed scan result page	All vulnerabilities, severity levels, and server headers displayed correctly	Pass
Download Scan Report	Click "Download Report"	System generates PDF containing ZAP and Nikto findings	Pass
Deep Scan Request Workflow	Open scan result and Click "Deep Scan Request"	Email sent to admin, request recorded, notification generated	Pass
Admin Deep Scan Reply	Admin opens request and uploads file. Submit reply	Reply stored, user notified, result downloadable	Pass
Deep Scan Request + Email Service	User submits deep scan request	Email sent to admin with PDF attachment + request logged	Pass
Admin Deep Scan Reply + File Upload	Upload Admin uploads result file and Submit reply	User receives notification and can download result	Pass
User Management (Admin)	Add/Edit/Delete user	User records updated correctly in database	Pass
Activity Log Tracking	User performs actions (login, scan, delete)	Activity logs recorded with timestamp and user ID	Pass
Notifications Page	User opens notifications. Mark as read or delete	Notification state updates and list refreshes correctly	Pass
Scanning History	Run scan and open scanning history	History displays accurate list with timestamps and statuses	Pass
Dashboard Visualization	Dashboard loads charts and summary	Statistics correctly calculated and charts displayed properly	Pass
Access Control (RBAC)	Normal user tries entering admin URL manually	System prevents access and redirects to login/403 page	Pass
Profile Update	User edits profile details or uploads image	Profile updates saved and reflected immediately	Pass
API Scan Endpoint	Access /api/scan and submit URL via JSON	API responds with scan ID and vulnerability list	Pass
System Stability and Performance	Run multiple scans and open multiple pages	System remains stable without crashing	Pass

8.2.4. User Acceptance Testing

User Acceptance Testing (UAT) is the final step in the testing process. In this step, users, or in this case, customers, test the software to determine whether their needs and expectations will be met once the software is released in a production environment. UAT is about validating the entire workflow, end-to-end. For this purpose, testing is performed in a simulated production environment using realistic data. Testing, at this stage, focuses on both functional and non-functional aspects so that users can determine the usability, accuracy, reliability, and functionality of the system. For this project, UAT is performed in two phases. The first phase involves the project supervisor, an internal evaluator who runs the UAT as if in an alpha testing phase. In the second phase, the actual customers, who are the end users, take the system to the beta testing stage of UAT. This process is designed to verify that the system meets the user expectations to facilitate the use of web vulnerability scanners and to ensure that the system is fully aligned with the user expectations.

1. Alpha Testing

Alpha testing is the acceptance testing phase, where the system is tested internally again before being brought to real users. Generally, it is conducted by developers and supervisors to determine whether the system's key functions are operating smoothly and whether the system is stable. For the current project, alpha testing involved a live demonstration of the web vulnerability scanner, which includes login, scanning, user management, and report generation functions, in front of my supervisor. My supervisor used the demonstration to verify the functionality of the system, state further improvements needed, and provide recommendations for preparing the system for client testing.

2. Beta Testing

Beta testing is the second component of acceptance testing. Here, the system is evaluated by real users in a genuine context. This step serves to verify the relevance, accessibility, and alignment of the system with genuine business demands. This project involves beta testing in collaboration with the client after the necessary modifications from the alpha testing phase are completed. The client applies their workflow to the web vulnerability scanner by performing scans, reviewing and downloading results, and submitting requests for in-scans. The feedback provided confirms that the system is working effectively for the users as intended and is ready to be deployed in their organization.



Figure 8.1 Beta Testing with Client

8.2.5. User Acceptance Testing Test Case Table

a. User UAT Test Case Table

Table 8.4 User UAT Table

Component	Test Step	Expected Output	Status
Login	Enter valid email & password, click login button	Redirected to OTP page	Pass
Login	Enter wrong password	Error message displayed	Pass
OTP Verification	Enter correct OTP	User logged in successfully	Pass
OTP Verification	Enter wrong or expired OTP	System rejects OTP	Pass
Perform Scan	Enter URL and start scan	Scan begins successfully	Pass
View Scan Result	Open completed scan	Vulnerability list and severity summary displayed	Pass
Download Report	Click "Download Report"	PDF downloaded successfully	Pass
Deep Scan Request	Submit request and confirmation appears	Request submitted successfully	Pass
View Deep Scan Requests	Open "My Deep Scan Requests"	Deep scan list displayed	Pass
View Deep Scan Result	Click "View Result"	Attached result opens/downloads	Pass
Scanning History	Open "Scanning History"	All past scans displayed	Pass
Profile	Update name or photo and save	Profile updated successfully	Pass
Dashboard	Open Dashboard	Shows user's total scan, completed scan, company name, and total users under same company	Pass
Activity Log	Open "Activity Log"	Records shown in correct order	Pass

b. Admin UAT Test Case**Table 8.5 Admin UAT Table**

Component	Test Step	Expected Output	Status
Login	Enter valid email & password, click login button	Redirected to OTP page	Pass
Login	Enter wrong password	Error message displayed	Pass
OTP Verification	Enter correct OTP	User logged in successfully	Pass
OTP Verification	Enter wrong or expired OTP	System rejects OTP	Pass
Perform Scan	Enter URL and start scan	Scan begins successfully	Pass
View Scan Result	Open completed scan	Vulnerability list and severity summary displayed	Pass
Download Report	Click "Download Report"	User created successfully	Pass
Add User	Fill form and submit	Request submitted successfully	Pass
Edit User	Update details and save	Changes saved successfully	Pass
Delete User	Click "Delete"	User removed successfully	Pass
View User List	Open User Management	All users listed correctly	Pass
Search & Filter Company	Apply filter or search	List updates correctly	Pass
Manage Deep Scan Requests	Open request and attach result	User receives result	Pass
Dashboard	Open Dashboard	Shows total scans, completed scans, total organizations, total user, and charts	Pass
Activity Log	Open "Activity Log"	Records shown in correct order	Pass
Profile	Update profile details/photo	Profile updated successfully	Pass

8.2.6. Questionnaire Feedback

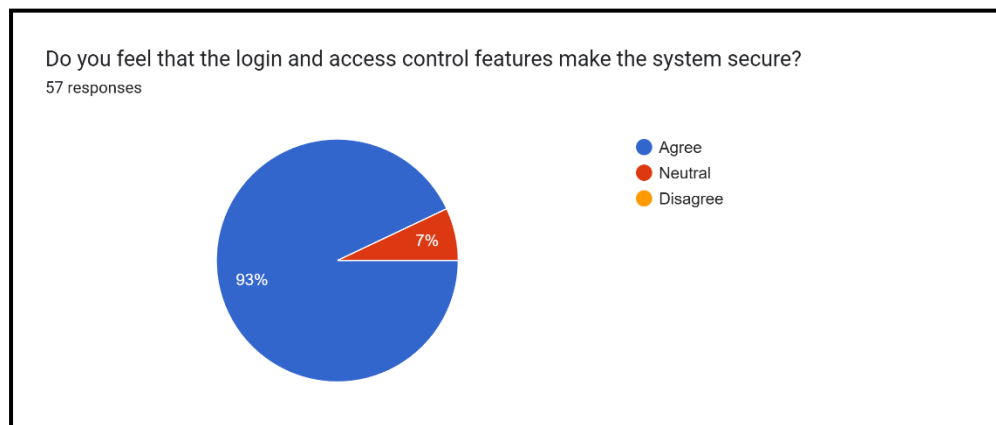


Figure 8.2 Result Questionnaire 1

Most respondent strongly believed that the system login and access control mechanisms offered a high level of security as their system environment. A total of 93%% of participants responded with 'Agree', thus indicating that they had a high level of confidence in implementing OTP authentication, password protection and role-based access control. A small number of respondents, 7%, chose 'Neutral', indicating that they either had no issues or did not have enough involvement to establish a strong position. Most notably, no one chose 'Disagree', indicating that no participant considered the system to be secure. These exemplary results indicate that participants felt that the system authentication and access control features exceeded the expected level of system security.

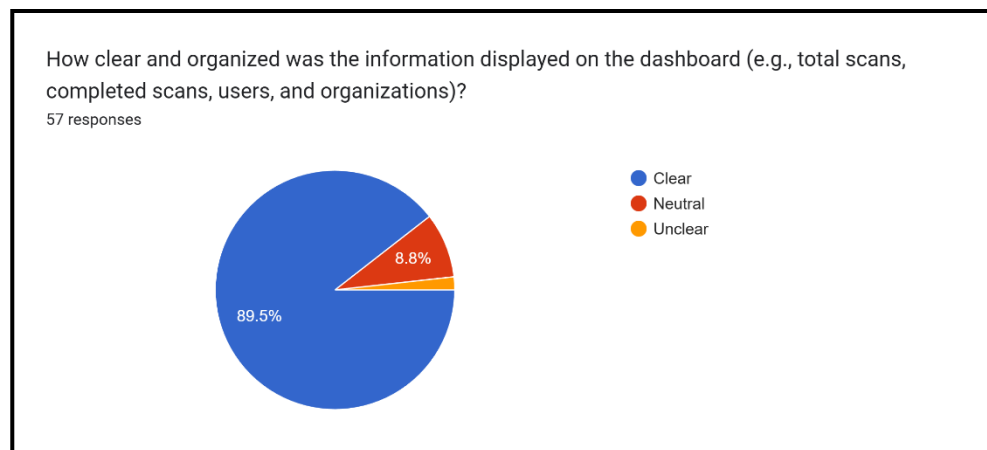


Figure 8.3 Result Questionnaire 2

Most respondents, 89.5%, indicated that they found the value documentation for the dashboard to be clear and structured. This shows that the dashboard design has been appropriately organized for easy navigation, and is able to prioritize and display important information such as scan statistics and company information. In contrast, 8.8% of respondents chose 'Neutral', possibly because they had to use the dashboard for a longer period of time, or they did not cover all sections. There were only 1.8% who felt that the dashboard was unclear, thus only a small number of users had problems understanding the information.

Overall, the results showed a major bias towards dashboard design as a factor contributing to greater user understanding and enjoyment.

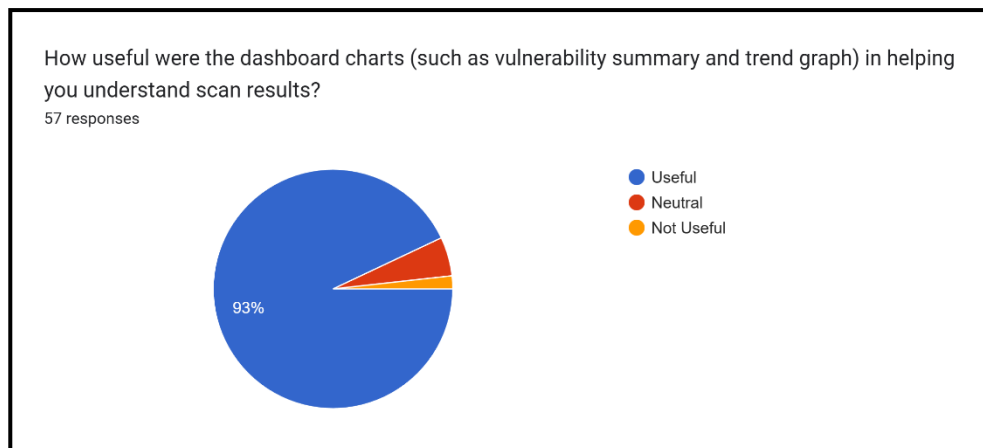


Figure 8.4 Result Questionnaire 3

Almost all respondents, at 93%, found dashboard charts helpful in understanding scan results. This illustrates the value of charts, especially in presenting summary weaknesses and trend figures, in effectively and more clearly communicating complex, technical information. Meanwhile, 5% of users selected 'Neutral' and 2% indicated that charts were not helpful, indicating that a very small segment of respondents may need further explanation or detail. In short, the data suggests that charts help users interpret scan results efficiently and more easily.

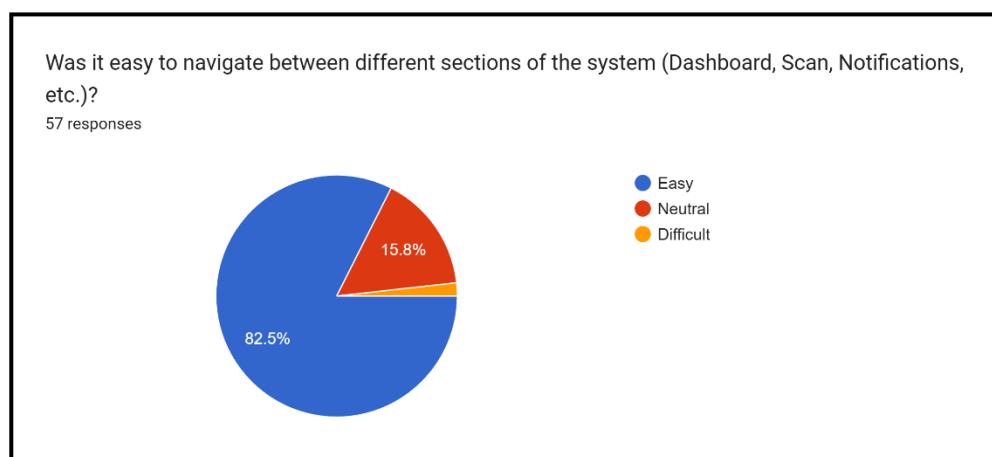


Figure 8.5 Result Questionnaire 4

Most respondents, 82.5% who claimed that they experienced low usability of the system, indicated that the system was easy to use, and thus, the system was somewhat well developed and organized. Approximately, 15.8% indicated neutrality which could be a sign that they had a low level of involvement and therefore did not experience any problems. Only 1.8% of the users claimed that navigation was problematic, which meant that the system did not show usability problems to a greater extent. In short, respondents left positive feedback about the system that it provided a smooth and highly usable system.

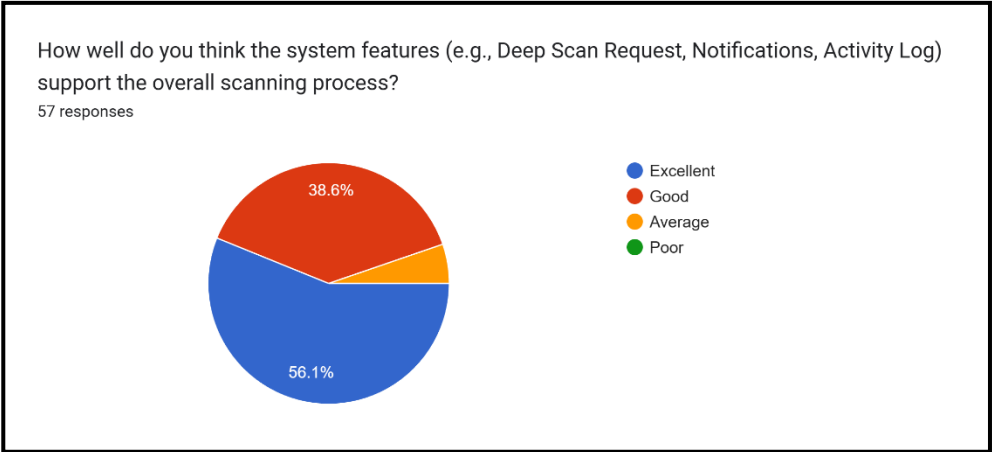


Figure 8.6 Result Questionnaire 5

Respondents rated the system's features on an overall excellence level, with over half of them, at 56.1%, rating the features as 'Excellent', which attests to the effective functionality of the Deep Scan Request, Notifications and Activity Log features in the system. On the other hand, 38.6%, rated them 'Good', indicating that the features are useful and function at a level considered satisfactory by the majority of users. Only a small minority, at 5.3%, rated the system features as Average, while Weak was not rated by any, indicating that the level of usability of the system features was rated positively, with a high level of acceptability. Therefore, the overall response indicated a high level of feature utility to assist users in facilitating the completion of workflows related to the scanning process.

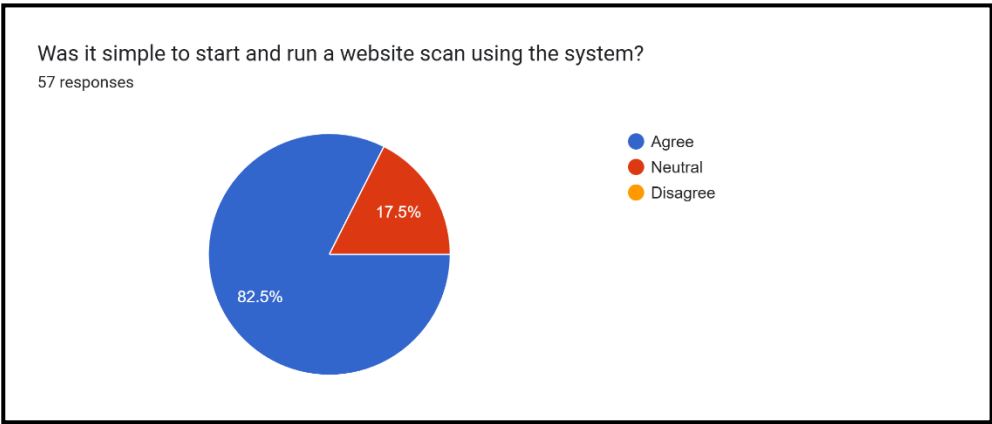


Figure 8.7 Result Questionnaire 6

Most respondents (82.5%) agreed that initiating and performing a website scan was an uncomplicated procedure. This shows that this process is easy to master even for users with little technical expertise. In contrast, 17.5% gave a neutral response, perhaps indicating that their interaction with the scanning interface was limited, even though the interface was easy to master. Critically, no respondents chose "Disagree." These results indicate that there is no frustration when starting the scan. These results also illustrate that the system is effective and users do not experience frustration while using the system for their scans.

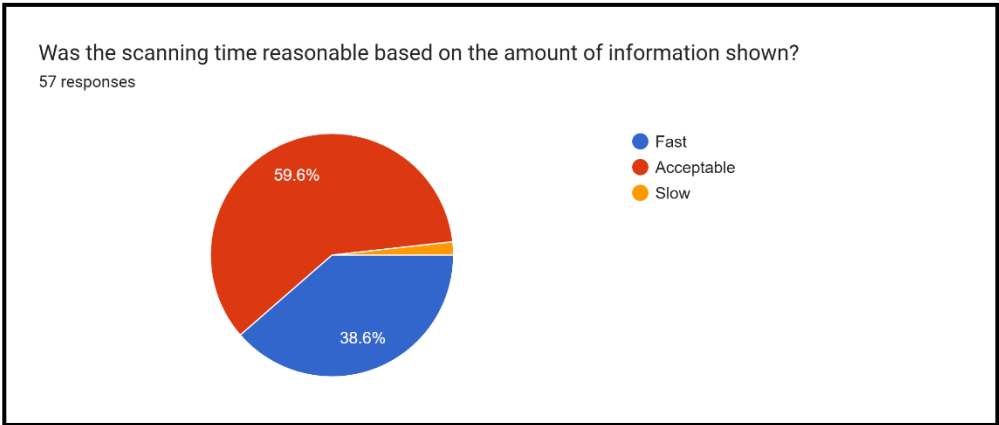


Figure 8.8 Result Questionnaire 7

Most respondents (59.6 %) rated the time taken to run a scan as Acceptable meaning they felt that the system provided results at a rate that was commensurate with the amount of data and results produced. A further 38.6% percent rated the scan time as Fast indicating that a large number of users were satisfied with the overall system performance and efficiency. Only 1.8% percent of respondents found the scan time to be Slow, indicating that a small number of respondents had to deal with significant delays. Overall, the ratings suggested that the system was able to generate scan results within a timeframe that could be considered reasonable and suitable for real-world use.

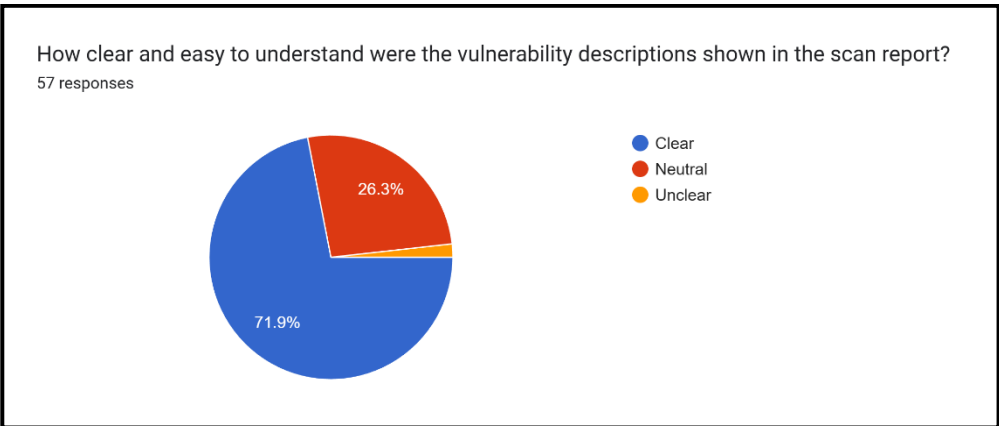


Figure 8.9 Result Questionnaire 8

The majority of respondents (71.9%) expressed their agreement on the issue of clarity and comprehensibility of the vulnerability description. Therefore, the report can demonstrate the articulation of the report's fundamental and related issues. An additional 26.3% chose a neutral position, indicating that the written material may be understandable but that certain audience segments may be less proficient or may require additional documentation to analyze the material at a sufficiently comprehensive level. 1.8% of respondents chose the description as "Unclear", indicating a very slight level of misunderstanding. This illustrates that the specific format of the report, as well as the style of explanation, contributes to the audience's increased

ability to assess the level of threat posed by the vulnerabilities identified during the assessment.

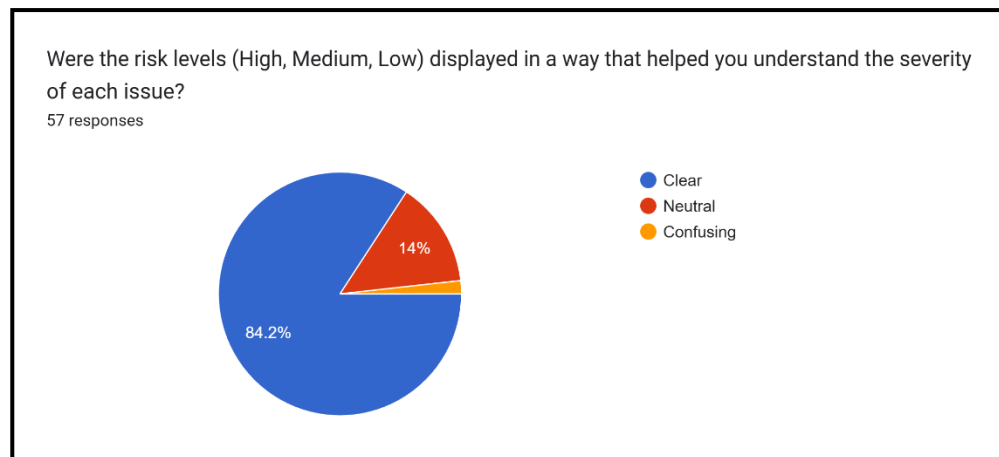


Figure 8.10 Result Questionnaire 9

Almost all participants (84.2%) agreed that the risk level was clearly displayed and users could differentiate scan results according to severity of High, Medium and Low. Meanwhile, 14% chose neutral, indicating that sufficient information was provided, although some users may need additional clarification. Only 1.8% found the risk level confusing. The overall results show that very few users misunderstood the risk level. Overall, the results show that users of scan reports can easily determine the severity of vulnerabilities and the report features, especially color coding and visual warnings, help achieve this.

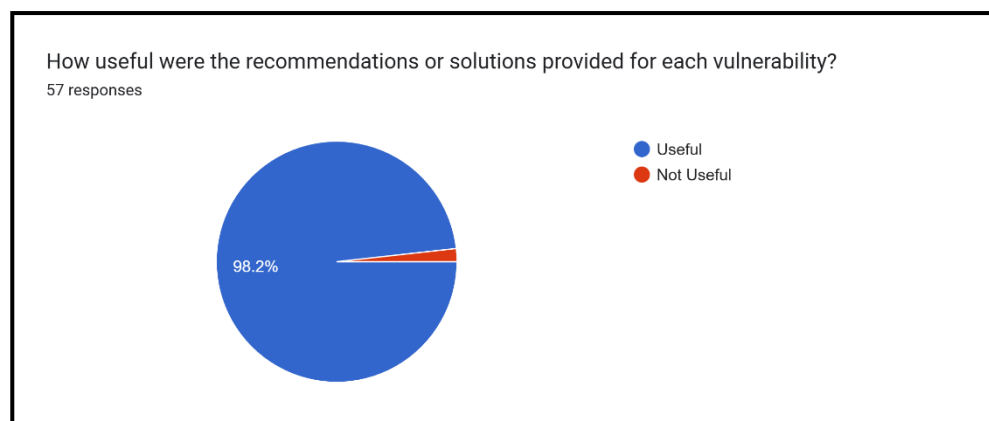


Figure 8.11 Result Questionnaire 10

A large majority of respondents, 98.2%, stated that the recommendations provided to them were useful, also indicating that the recommendations contained in the scan report assisted users in learning how to fix or mitigate the security concerns detailed. There were also 1.8% of respondents who said the suggestions given were useless, as that was a small margin. This clearly demonstrates that the instructions have been tailored and appropriate for specific users, and that the educational and technical guidelines involved are appropriate and relevant to all classes of individuals, each with varying levels of technology-related experience.

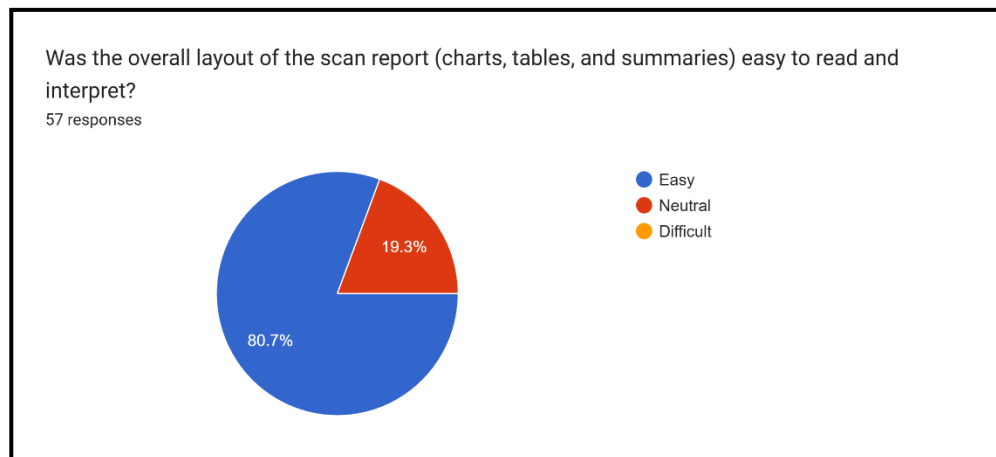


Figure 8.12 Result Questionnaire 11

The majority of respondent (80.7%) agreed that the scan report was easy to read and understand. Users appreciate the combination of charts, tables, and summaries designed to convey scan results. About 19.3% of respondents remained neutral, implying that while the layout was usable, they might need to familiarize themselves with some security vernacular to understand the details. Notably, zero users viewed the layout in detail with the aim of providing constructive feedback, illustrating that the layout is indeed user-centered.

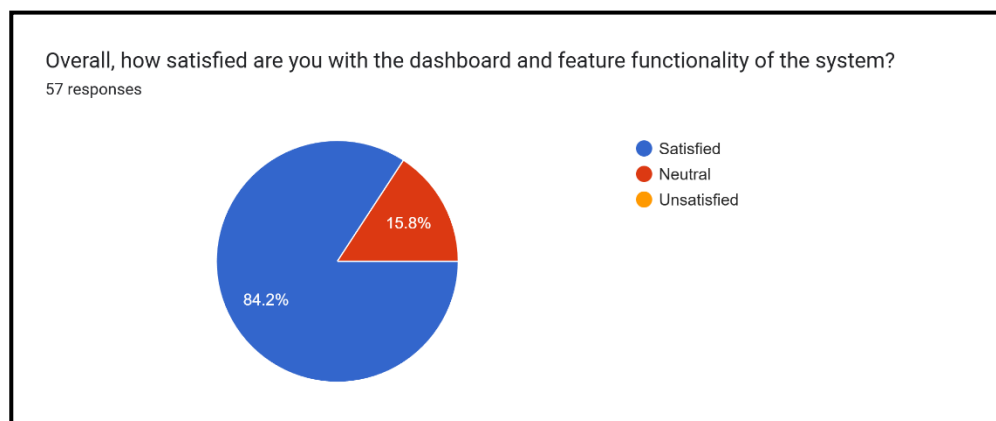


Figure 8.13 Result Questionnaire 12

Respondents showed a high level of satisfaction with the system and dashboard features, as evidenced by the 84.2% satisfaction with the system obtained. User satisfaction with the system features has translated into high user approval ratings as seen with respect to the interface, system navigation, and functionality. In a similar but different context, 15.8% of neutral respondents expressed no satisfaction with any issues that might cross the system, thus indicating a lack of possible exploration of system features. However, the absence of any respondents who rated the system as unsatisfactory indicated that the system, overall, met the expectations and expectations of the users, thus functioning as expected.

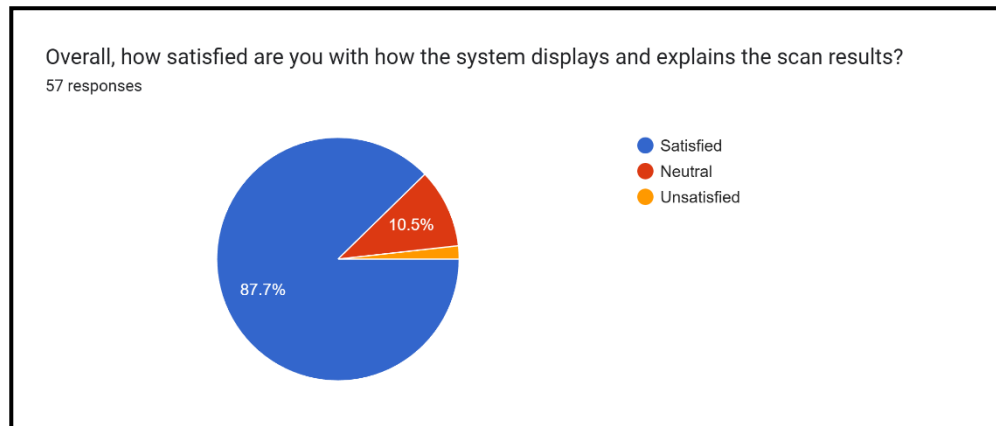


Figure 8.14 Result Questionnaire 13

87.7% of respondents were satisfied with the system's presentation of scan results and the system's explanation function. 10.5% of respondents remain neutral. This suggests that the system works reasonably well but may need some refinement towards tailoring the presentation of results for a subset of the target audience. 1.8% of respondents were dissatisfied, which is a small percentage overall. Overall, the data showed that most respondents thought the system presented the scan information clearly, and that it was structured in an easy-to-understand manner.

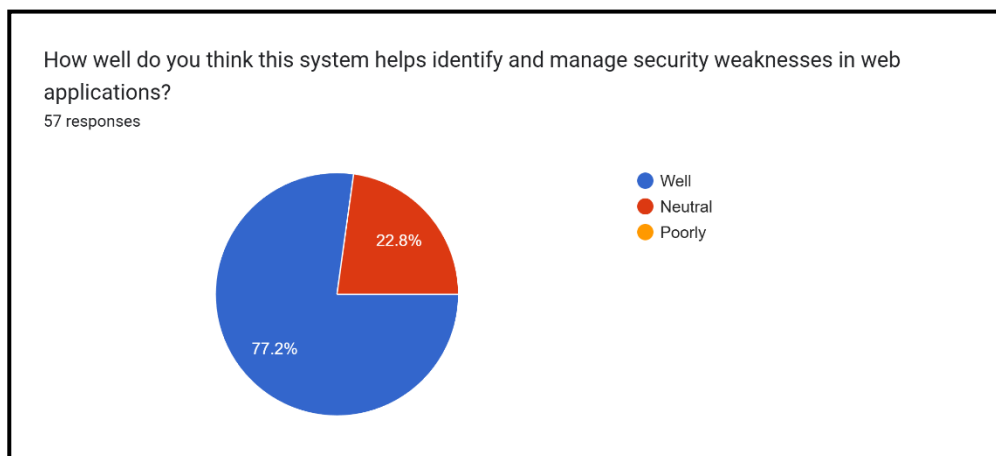


Figure 8.15 Result Questionnaire 14

Most respondents (77.2%) believe that the system scans and identifies security vulnerabilities and implements corrective actions effectively, indicating the system's scanning engine and reporting capabilities. The remaining 22.8% of respondents chose neutral, indicating that they recognize the impact of the system, although not fully reviewed or involved with the system's functioning. More significantly, the fact that no users rated the system low indicates a measure of their confidence in the system's ability to detect vulnerabilities and offer recommendations to improve website security.

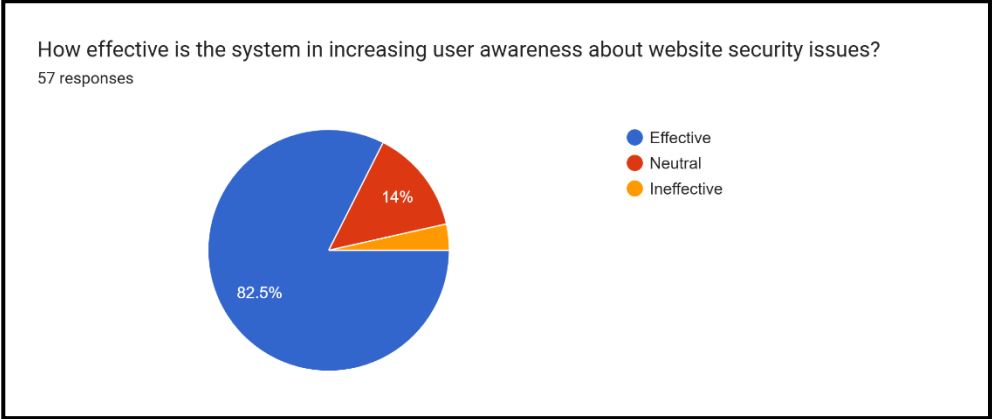


Figure 8.16 Result Questionnaire 15

Based on the feedback, a full 82.5% of respondents felt that the system helped users understand the risks associated with security issues by assisting in understanding scan results and explanations. On the other hand, 14% of respondent gave a neutral response, implying that the system was educational but may require additional exposure to educational content before it would be fully beneficial for some users. Only 3.5% of respondent felt that the system was ineffective in terms of education. The feedback clearly indicated that the system was able to assist users in understanding cybersecurity issues, educating them in an efficient and effective manner.

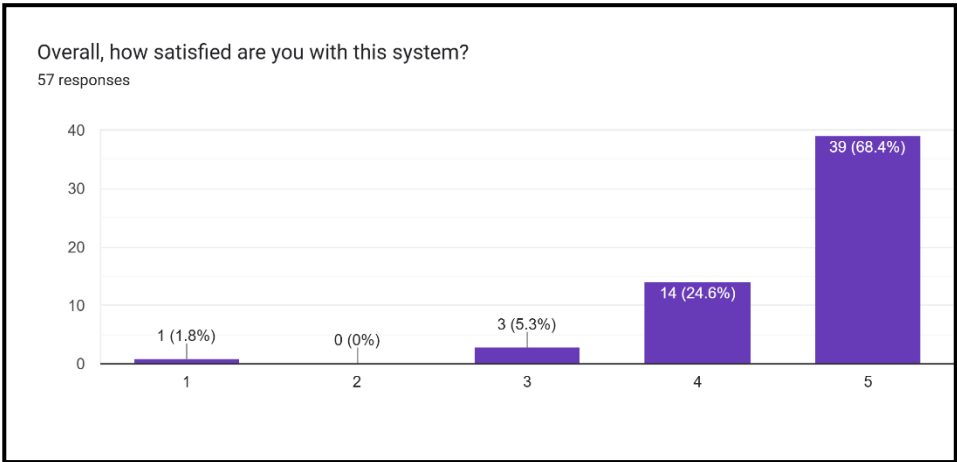


Figure 8.17 Result Questionnaire 16

The overall satisfaction of the respondents is reflected in the star rating, which averages 4.58 out of 5. A total of 68.4% of the respondents rated the system 5 stars, while 24.6% gave it a 4 star rating, which is a very positive overall rating. Only a respondents rated it lower than a 4 star rating, with a total of 5.3% rating it 3 stars, 1.8% rating it 1 star and 0% rating it 2 stars. The data proves that the users are indeed very satisfied with the system. The data proves that the project was successful in terms of meeting the overall satisfaction of the users.

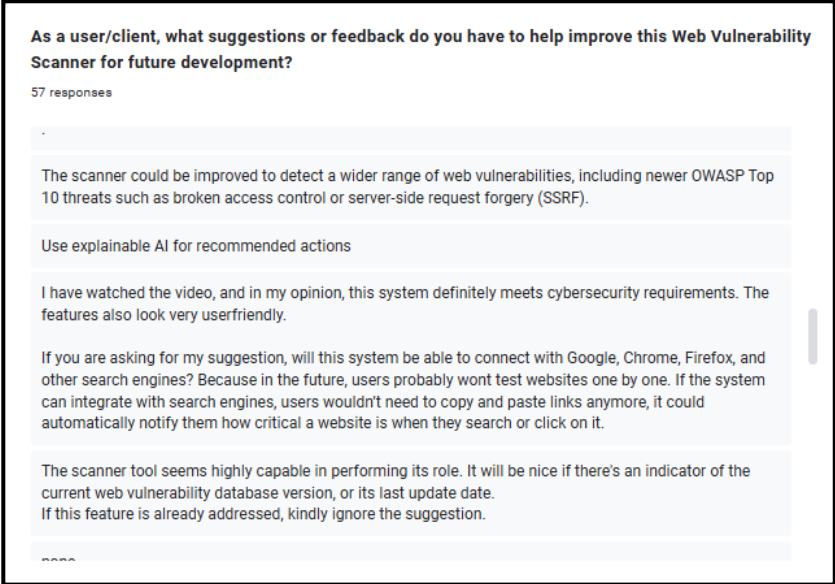


Figure 8.18 Result Questionnaire 17

Feedback from 57 open survey respondents focused on improving Web Vulnerability Scanner. Of the users who responded, many requested the incorporation of AI technology for improved automation, detection, recommendations, and overall intelligent feedback. AI was cited as an answer to many of the improvement requests centered around user interface (UI) feedback that focused on streamlining explanations and overall making the program more user-friendly. Among these requests, which were notable for their urgency, were for features that would allow the program to complete scans faster, work on a multi-threaded basis, offer a percentage-of-completion display, and incorporate a notification system that automatically alerts users when vulnerabilities are detected. A handful of respondents suggested adding the ability to detect newer vulnerabilities, including those listed in the latest version of the OWASP Top 10 list, and suggested that the program would benefit from incorporating a feedback loop to monitor and adjust the program’s performance on false positive detections. Regardless of feedback improvements, many survey respondents expressed their satisfaction by describing the vulnerability detection program as "good", "excellent", "user-friendly" and "perfect", which clearly illustrates the acceptance of the system and suggests an invaluable line of improvements.

8.3. Non-Functional Testing

Non-functional testing examines the performance and implementation of the system, and the operational quality of the Web Vulnerability Scanner by evaluating system performance, user experience, system security, and system compatibility. This type of testing measures whether the system is functioning optimally, and whether the system response time is efficient, user data is protected, and the system experience is seamless for users and system administrators. To ensure system reliability, security, and utilitarian value for real users in a variety of ecosystem situations, this project conducts non-functional testing.

8.3.1. Performance Testing

The speed and responsiveness of the system were checked during basic exercises such as logging in, loading pages, and performing vulnerability scans. Since the system interacts in real time with the scanning tools (ZAP and Nikto), the processing times needed to be verified. Respondents' comments indicated that 59.6% of users found the scanning time acceptable and 38.6% found it fast. This indicates that the system works efficiently and users have little, if any, wait time. Furthermore, users reported that the loading time of the dashboard, navigation through the system, and retrieving results were all smooth during testing.

Table 8.6 Performance Testing Table

Scenario	Expected Result	Actual Result	Status
Login page loading	Page loads within 3 seconds	Loaded in under 2 seconds	Pass
Scan execution	Scan completes within reasonable time	Scan time acceptable by users	Pass
Scan result loading	Results display without long waiting	Loaded instantly	Pass
Dashboard loading	All charts and stats load smoothly	Loaded smoothly	Pass

8.3.2. Security Testing

Security testing on the system includes system protection along with analysis of the system vulnerability scanning process. Given how this project aims to identify security vulnerabilities of various types of websites, it is appropriate that the scanner does its job accurately, can transparently show security vulnerabilities and does not overly falsely report security vulnerabilities. The following security tests were conducted:

a. System Security Testing

This testing verifies whether the system is protected from unauthorized access and whether users' data is secure.

Table 8.7 System Security Testing Table

Scenario	Expected Result	Actual Result	Status
Unauthorized user attempts to open admin-only pages	Access denied	System blocked access	Pass
OTP login verification	Wrong OTP rejected, correct OTP accepted	Works correctly	Pass
Verify user cannot see other company data	Only company-assigned data visible	Works correctly	Pass

b. Vulnerability Scanning Security Testing (Core Feature)

This testing ensures that the scanning engine which is OWASP ZAP and Nikto works correctly and provides accurate results.

Table 8.8 Vulnerability Scanning Security Testing Table

Scenario	Expected Result	Actual Result	Status
Run scan on purposely vulnerable website	Scanner detects common vulnerabilities (XSS, SQLi, etc.)	Vulnerabilities detected correctly	Pass
Scan non-vulnerable demo site	Scanner produces minimal or no false positives	Output as expected	Pass
ZAP scan integration	ZAP returns alerts and system stores them in the database	Successful	Pass
Nikto scan integration	Server headers, banner, and findings saved correctly	Successful	Pass
Grouping of duplicate ZAP alerts	Duplicate alerts grouped into one unique issue	Works correctly	Pass
Severity levels display	High/Medium/Low levels displayed correctly	Display accurate	Pass
Recommendation generation	Each vulnerability includes solution text	Provided correctly	Pass
Download report	PDF downloaded successfully	Works as intended	Pass

8.3.3. Usability Testing

Usability testing focused on how easy it was for participants to understand and use the system. Feedback showed that 82% of users had no problems navigating the system, 80% felt that the scanned reports were easy to understand, and over 89% agreed that the dashboard was easy to read and systematically organized. Such results attest to the fact that the interface is simple, clean, and suitable for users of all levels of technological background. The presence of functions such as the visualization of scan results, the ability to request a deep scan, and the download report button were also noted as valuable.

Table 8.9 Usability Testing Table

Scenario	Expected Result	Actual Result	Status
Navigation between dashboard, scan, and history	User can navigate easily	Works smoothly	Pass
Scan report readability	Users understand risk levels and issues	Users found it clear	Pass
Understanding of buttons & icons	No confusion during use	All users understood functions	Pass
Profile update usability	User can update info easily	Successful and simple	Pass

8.3.4. Compatibility Testing

Testing was done on the system's ability to function on different browsers and display resolutions. Given the web usage of the system, it was necessary to function in the browsers most frequently used in the organization's setting. No display or functionality issues were found when the system was accessed using Google Chrome, Mozilla Firefox and Microsoft Edge. All navigation functions, including logging into the system, scanning modules and opening the dashboard, were smooth on each of the browsers.

Table 8.10 Compatibility Testing Table

Scenario	Expected Result	Actual Result	Status
Google Chrome	Full system works properly	Works as expected	Pass
Mozilla Firefox	No UI or performance issues	Works as expected	Pass
Microsoft Edge	Features run normally	Works as expected	Pass
Responsive layout	Page adjusts to different screen sizes	Layout displayed correctly	Pass

8.4. Conclusion

Chapter 8 offers a comprehensive evaluation of the Web Vulnerability Scanner system and the achievement of its objectives and goals through various evaluation approaches and techniques. Accuracy, efficiency, and reliability, as well as user satisfaction, are the main focuses of the chapter's overview of the testing approach. Each component was examined separately in Unit Testing: login and authentication, OTP authentication, scan initiation, scan result generation, user profile and activity management, and updating user profile. Integration Testing demonstrates the harmonious functionality of the components, focusing on the interrelationship between the web application, Flask backend, SQLite database, OTP system, and external scan engines which is OWASP ZAP and Nikto. System Testing ensures that all integrated components are in complete working order and that the system complies with all functional requirements such as login, perform scans, report vulnerabilities, perform deep scans and download reports in smooth system operation.

This chapter provides the findings of User Acceptance Testing (UAT) conducted by supervisors and customers as the system is qualified through their real-world business applications. In UAT, users and administrators completed scenario-based tests and filled out usability, interface, functionality, and satisfaction rating questionnaires. Feedback from multiple participants conveyed overall satisfaction and appreciation for the system's user-centric interface, instant scanning, and smooth operation. Other tests were performed according to system attributes, Non-Functional Testing, which included system performance, security, usability, and cross-platform functionality. During performance testing, overall system response and scanning times were observed. Security testing included One-Time Password (OTP) method authentication, appropriate access controls, and various levels of system vulnerabilities denied. Usability testing included ease of navigation, and clarity of data presented from the system. Compatibility testing included verification of proper system functionality on various web browsers. Overall, these tests proved that the system functionality remained consistent, data was protected, provided efficient scanning, and the system met several requirements for use in business commerce and business promotion operations. Therefore, the system was operational. Overall, Chapter 8 presents an in-depth business usage scenario where system functionality meets system design and operational objectives, thereby meeting the customer's due diligence business needs.

9. PROJECT MANAGEMENT

9.1. Introduction

This chapter discusses the planning, management, and execution of the Web Vulnerability Scanner project during the development period. Since every activity during the development phase of the project will be scheduled, we, the project managers, explain the planning model, management tools, and management activities during the development phase of the project. Also, the chapter discusses the methodologies and processes of overcoming problems, the achievement of goals, and the flow of communication with the supervisor and the client. An ordered system was created, and the project was developed. The project was systematically completed. The quality control and project management systems developed during the project execution met the objectives.

9.2. Project Schedule

The project schedule is created to coordinate the development activities in an orderly and systematic manner. It also deals with ensuring that each activity is carried out according to the set schedule and the overall flow runs smoothly from analysis to testing and documentation phases. The schedule is monitored throughout the project and changes are made as needed as the project progresses, especially in phases that require more polish, such as in the integration of the scanning engine and in the polishing of the user interface. Having a timeline keeps the project on track and ensures that important milestones, such as the design of the system architecture and the development of the core modules and in the preparation of the final report and in the testing activities, are achieved within the stipulated time. This scheduling method contributes significantly to the successful completion of the Web Vulnerability Scanner system.

9.2.1. Work Breakdown Structure

The project's Work Breakdown Structure (WBS) is aligned with Agile Methodology, which breaks the development cycle into phases and captures the functionality in each phase for review and improvement in subsequent phases. The WBS captures and details all the major tasks and sub-tasks performed in each Agile phase so that development is sequential, manageable, and aligned with the project objectives.

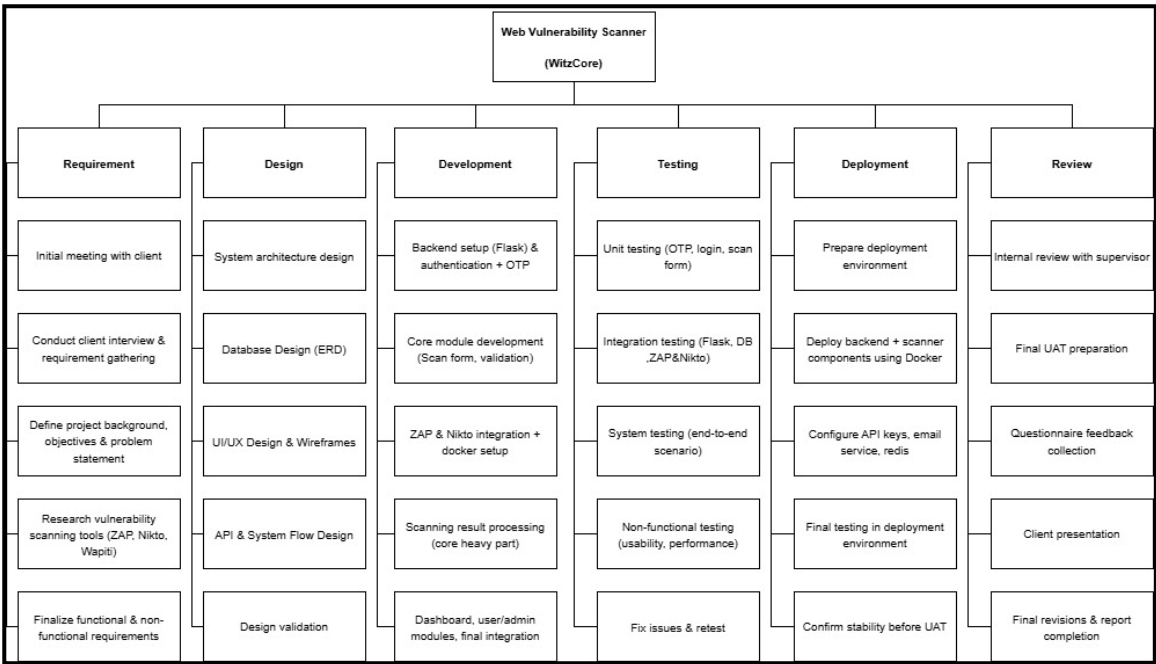


Figure 9.1 Work Breakdown Structure

The Work Breakdown Structure of this project was designed using Agile methodology, which divides the project into iterative phases, Requirements, Design, Development, Testing, Deployment and Review. Each phase contains defined activities that allow the project progress to progress at different rates in the Web Vulnerability Scanner (WitzCore). This WBS allows an overview of all activities performed throughout the project duration and provides an organized, systematic, manageable work that is aligned with the project objectives throughout the project lifecycle. This detailed WBS is essential for executing the project in an orderly manner, with a chronology of each phase of the project to stay on track.

9.2.2. Gantt Chart

The Gantt chart for this project shows the planning and time allocation for all Agile stages spread over a 3-month development period. The effort was completed in an iterative manner where development, testing and refinement overlapped in a cycle. There were adjustments made during the implementation of the scanning engines (ZAP and Nikto) which took longer to balance and prepare. Despite the minor slowdowns that were ongoing on the project, it was completed on time, and all key deliverables were achieved.

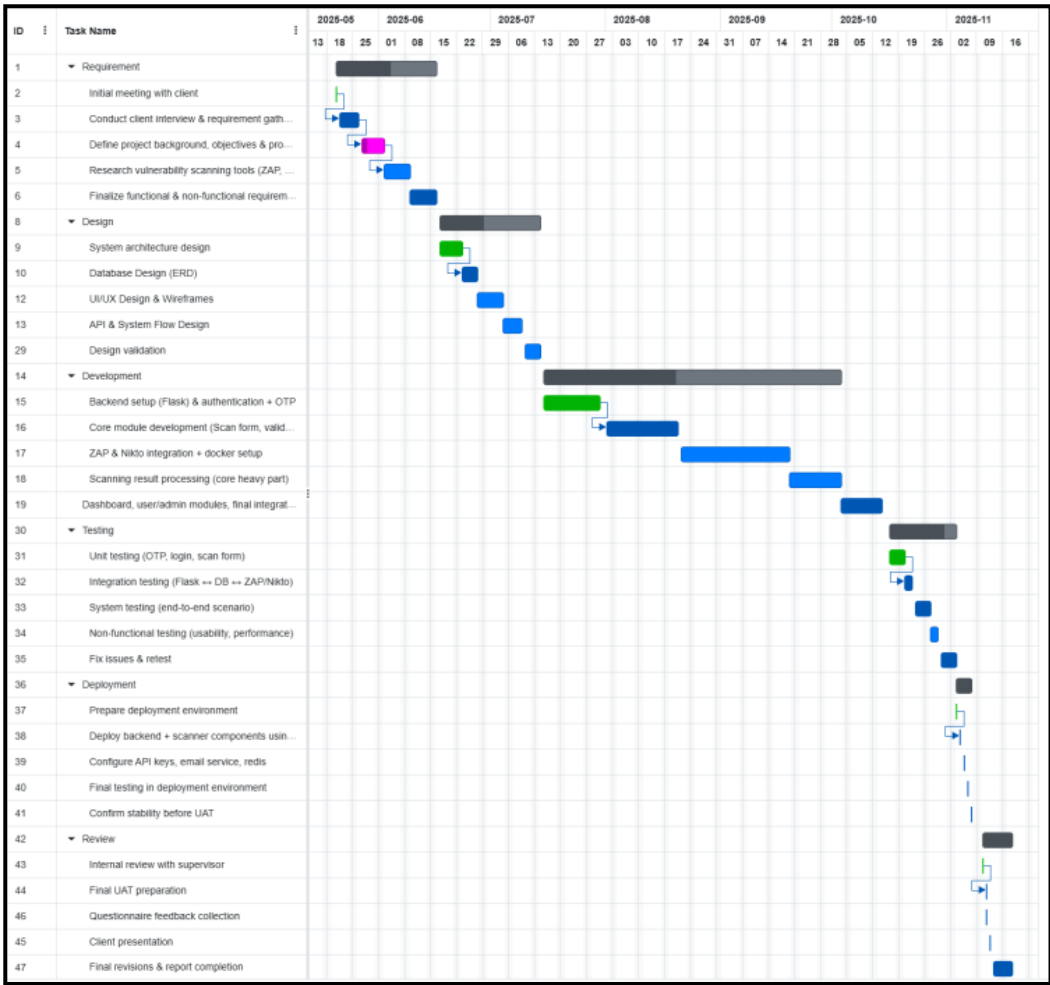


Figure 9.2 Gantt Chart

The project timeline is built and executed using Agile methodology between May 20 and November 11, 2025. The first stage is called the Requirements Phase, focusing on customer discussions, requirements gathering, and initial research to define and outline all functional and non-functional requirements. This is then followed by the Design Phase, which focuses on the system architecture, database design, wireframing, and setting up API workflows. The Development Phase is the longest, consisting of implementing the backend, OTP authentication and configuration, integrating both OWASP ZAP and Nikto, Docker containers, and creating all user and admin modules including the complex module that processes scan results. After the system development is fully completed, the Testing Phase begins, which includes unit testing, integration, system testing, and non-functional testing to verify that the system is working properly and remains stable. The system then moves to the Deployment Phase, which consists of deploying the system to a Docker environment, creating the final system configuration, and system reviews to ensure that the system continues to work properly. The project then moves into the Review Phase, which consists of internal reviews, customer reviews on November 9, 2025, collection of UAT questionnaire feedback on November 10 and 11, 2025, and then final refinement and report finalization

Table 9.1 Gantt Chart Table

ID	Task Name	Start Date	Finish Date	Duration
1	Requirement	2025-05-20	2025-06-16	30 days
2	Initial meeting with client	2025-05-20	2025-05-20	1 day
3	Conduct client interview & requirement gathering	2025-05-21	2025-05-26	6 days
4	Define project background, objectives & problem statement	2025-05-27	2025-06-02	7 days
5	Research vulnerability scanning tools (ZAP, Nikto, Wapiti)	2025-06-02	2025-06-09	8 days
6	Finalize functional & non-functional requirements	2025-06-09	2025-06-16	8 days
7	Design	2025-06-17	2025-07-14	31 days
8	System architecture design	2025-06-17	2025-06-23	7 days
9	Database Design (ERD)	2025-06-23	2025-06-27	5 days
10	UI/UX Design & Wireframes	2025-06-27	2025-07-04	8 days
11	API & System Flow Design	2025-07-04	2025-07-09	6 days
12	Design validation	2025-07-10	2025-07-14	5 days
13	Development	2025-07-15	2025-10-03	93 days
14	Backend setup (Flask) & authentication + OTP	2025-07-15	2025-07-30	16 days
15	Core module development (Scan form, validation)	2025-08-01	2025-08-20	20 days
16	ZAP & Nikto integration + docker setup	2025-08-21	2025-09-19	30 days
17	Scanning result processing (core heavy part)	2025-09-19	2025-10-03	15 days
18	Dashboard, user/admin modules, final integration	2025-10-03	2025-10-14	12 days
19	Testing	2025-10-16	2025-11-03	21 days
20	Unit testing (OTP, login, scan form)	2025-10-16	2025-10-20	5 days
21	Integration testing (Flask ↔ DB ↔ ZAP/Nikto)	2025-10-20	2025-10-22	3 days
22	System testing (end-to-end scenario)	2025-10-23	2025-10-27	5 days
23	Non-functional testing (usability, performance)	2025-10-27	2025-10-29	3 days
24	Fix issues & retest	2025-10-30	2025-11-03	5 days
25	Deployment	2025-11-03	2025-11-07	6 days
26	Prepare deployment environment	2025-11-03	2025-11-04	2 days
27	Deploy backend + scanner components using Docker	2025-11-04	2025-11-04	1 days
28	Configure API keys, email service, redis	2025-11-05	2025-11-05	1 days
29	Final testing in deployment environment	2025-11-06	2025-11-06	1 days
30	Confirm stability before UAT	2025-11-07	2025-11-07	1 days
31	Review	2025-11-10	2025-11-18	10 days
32	Internal review with supervisor	2025-11-10	2025-11-10	1 days
33	Final UAT preparation	2025-11-11	2025-11-11	1 days
34	Questionnaire feedback collection	2025-11-11	2025-11-11	1 days
35	Client presentation	2025-11-12	2025-11-12	1 days
36	Final revisions & report completion	2025-11-13	2025-11-18	6 days

9.3. Risk Management

Risk management is the process of predicting, analyzing, and preparing for problems that could threaten the success of a project. This requires predicting possible and impossible risks, considering their impact and probability of occurrence, and then developing a strategic plan to mitigate the negative impacts. For the Web Vulnerability Scanner project, risk management was essential, given the use of system security tools such as OWASP ZAP and Nikto, third-party integration, use of Docker, Celery task processors, and users with a dual-role environment (administrator and user). Each of these components poses certain technical, functional, and operational risks to the development of the security tools and to the system itself. These were identified during the requirements and design phase, and a risk mitigation plan, or mitigation plan, was developed to reduce and avoid delays. Some risks were present such as ZAP integration, Docker networking, configuration issues, and scan result parsing errors, but these were resolved with testing, debugging, and incremental improvements to the process. Risk management in this case guided the project positively in a direction that stayed on track and delivered a properly functioning system.

Table 9.2 Risk Management Table

Risk	Analysis	Mitigation
Integration issues with OWASP ZAP API and Nikto	High probability because the tools need to be properly configured, have API keys and network proxy settings. Failure to do so correctly may result in the scan failing or giving incomplete results.	Conducted early integration testing, validated official documentation, tested various configurations, and added fallback logic to make both scanners work.
Docker deployment and environment configuration problems	Medium risk due to the presence of multiple containers (Flask, Redis, ZAP). Incorrect configuration can stop the system from running or lead to network errors.	Completed container-level testing, configuration environment files, log monitoring, and optimized Docker Compose setup until stable.
Scan result parsing logic errors	High probability because ZAP and Nikto scan results are in a complex format. Poor parsing can result in vulnerabilities failing to be detected or duplicated.	Validated results with sample scan data, improved parsing capabilities, additional error handling, and debug output were tested repeatedly during development.
Database or data inconsistency	Medium probability due to the insertions (vulnerabilities, logs, notifications) can occur frequently and lead to the creation of fake connections or duplicate records.	Implemented additional validation, tested CRUD operations, and implemented SQLAlchemy relationships to ensure data integrity.

Security risks during testing	Low probability and high impact risks are possible because running a scan on an unauthorized site may violate the scanning policy.	Added warning messages to the system, limited scanning features to logged-in users, and ensured that scans were performed on approved test URLs.
Long scanning time for full scans (ZAP + Nikto)	The chances are high that a full vulnerability scan can take 20-50 minutes, depending on the size of the website. This can lead users to believe that the system is slow or can delay the generation of results.	Use asynchronous scanning with Celery to prevent UI lockups, provide scan status updates, show estimated scan time, and place a limit on scan duration to avoid extremely long scans.

The table above highlights the main risks observed in the preparation of the Web Vulnerability Scanner project, their analysis and the mitigation measures implemented to minimize the risks. It is expected that some risks may exist due to the security tools, including OWASP ZAP and Nikto, used by the system, the presence of several Docker containers and the complexity of the scan results. Technical issues such as problems with scanner integration, long scan times, parsing errors and database inconsistencies are considered high or medium risk as they can affect the accuracy and performance of the system. Different mitigation strategies have been designed against each risk, including initial testing, configuration improvements, data validation, alerts and asynchronous background tasks. These measures have been taken to make the project remain functional and stable despite the difficulties that will arise.

9.4. Conclusion

Chapter 9 discusses how project management facilitated the Web Vulnerability Scanner project from inception to completion. Using an Agile approach, the project was divided into six phases: Requirements, Design, Development, Testing, Deployment, and Review. This made the work more manageable and allowed for smoother changes, especially when faced with technical hurdles to integrate OWASP ZAP and Nikto, configure Docker, and process scan results. The Work Breakdown Structure and Gantt chart facilitated the planning of tasks, timelines, and dependencies, including customer interviews, back-end development, scanner integration, front-end design, testing, UAT, and final presentation. The chapter also discusses the importance of risk management, as potential risks were identified, including integration issues, long scan times, problems with parsing, data inaccuracies, and system configuration issues. Each risk was assessed and accompanied by a mitigation plan aimed at minimizing its negative impact. Several of these risks were demonstrated during the development phase, particularly those related to Docker and scan processing, and were addressed through debugging and testing. Concerns about scanner misuse prompted the implementation of warnings and limits on their use. The project remained on schedule, achieved its goals, and provided a stable and operational system because of the overall attention to planning, ongoing project guidance and evaluation, strategic risk management, and implementation of this system.

10. CONCLUSION

10.1. Introduction

This chapter provides a general conclusion of the Web Vulnerability Scanner project summarizing the main findings, achievements, and results realized during the development process. The project aimed to develop a simple and easy-to-use tool that would allow Netwitz clients to scan their websites for basic vulnerabilities, easily interpret the results, and notify Netwitz when they need further inspection. In this chapter, the project objectives will be restated, how each objective was met will be explained, and a reflection on the overall development process will be provided. It also discusses the constraints encountered in the project, offers potential improvements for future work, and provides a final opinion on the system. Through this project, a simple and easy-to-use security tool was created that helps in early identification of website vulnerabilities and increases security awareness among staff in SMEs.

10.2. Achievement

This section summarizes the project's achievements according to the initial goals. The final system successfully delivered a web-based vulnerability scanner with the ability to identify common website vulnerabilities, present results in a simple and understandable format, and provide secure access with various user privileges. User testing, interview feedback and questionnaire feedback revealed that the system has been effective in increasing user awareness and in facilitating Netwitz's operational needs. All objectives are discussed below.

10.2.1. Achievement of Objective 1: To develop a web-based vulnerability scanner for Netwitz clients

This has been achieved with all its objectives. This system gives users the ability to provide a site URL and run a simple vulnerability scan with the help of OWASP ZAP API and Nikto. The scanner can identify the most well-known vulnerabilities, including SQL Injection, Cross-Site Scripting (XSS), missing security headers, and server misconfiguration. Proof of success is presented with the help of Scan Page, Scan Results Page and Scan History Page which shows that the framework can perform scanning and save the results. In UAT, clients ensure that the scanning features are as intended and help them to detect vulnerabilities early. The survey results also showed that 87 percent of users were satisfied with the way the system presented and explained scan results. This shows that the initial goal has been achieved.

10.2.2. Achievement of Objective 2: To implement a secure login system with role-based access control

This has also been achieved. The system has a secure login system that authenticates passwords, OTPs and role-based access control (RBAC). There are two roles created, namely Administrator (Netwitz staff) and User (Netwitz clients), and the permissions of the roles are different. Administrators can manage users, view all scan activity, and process deep scan requests while customers can only perform scans and view their results. Evidence of this success can be represented in the User Management Module, Activity Log, and dashboard views limited to each role According to the questionnaire responses, it was also found that users believed that the system's login and roles were understandable and secure, which confirmed the achievement of this objective.

10.2.3. Achievement of Objective 3: To display scan results in a clear and understandable format

This goal has been achieved. This system presents scan results in a readable and understandable graphical format using charts, risk categories such as High, Medium and Low, and simplified explanations. Vulnerabilities are described and provided with suggested solutions, confidence levels, classifications, and reference links to allow users to learn more. The added features are a result of interview feedback with customers who have indicated that clear and readable results are necessary for them as they may not be technical. This achievement is also supported by survey findings as 82.5% of users reported that the system helped them understand website security better and the results were easy to read. This indicates that the system has achieved the goal of delivering understandable, meaningful, and user-friendly scan output to users.

10.3. Constraint and Limitation

During the development of Web Vulnerability Scanner, there were several constraints and limitations encountered that affected the development process and product. The biggest challenge is managing Nikto scan results, especially since Nikto results are not user-friendly and include long unstructured text. Each vulnerability had to be extracted, highlighted, and simplified before being displayed on the front end. This needs to be done manually and with a certain logic because Nikto does not provide clear recommendations. This means that each vulnerability requires further research to enable the system to show appropriate explanations and solutions. The design of the scanning results interface is also a challenge as it requires close formatting so that the results can be read and understood by users.

The second limitation is associated with scanning time. Scan times remain long despite advances such as subprocesses and threading, especially when ZAP and Nikto are both used simultaneously. It may take several minutes to scan a large website based on the site size, network performance, and hardware capacity. To avoid unnecessary delays, only selected Nikto checks were added. This improves performance but at the expense of scan depth. This suggests that the system is currently focused on faster output rather than very comprehensive scanning.

Finally, development has technical limitations, including Docker setup issues, ZAP API behavior, and inconsistent parsing results. There is also ZAP output with lengthy technical descriptions that need to be trimmed to read, and some Nikto results will sometimes display differently on scans, which needs to be handled further. This issue caused delays in development and required extensive debugging and testing. Despite the effectiveness of the system in the target application, these weaknesses reveal that additional optimizations, higher-end hardware, or more powerful scanning engines may be needed in the future to improve accuracy, performance, and user experience.

10.4. Future Work and Recommendation

According to the feedback provided by users, customers and supervisors during UAT and questionnaires, it is possible to consider some improvements that need to be introduced to further develop the Web Vulnerability Scanner. The implementation of artificial intelligence (AI) to improve detection accuracy, reduce false positives and offer smarter recommendations is one of the key recommendations. Some users are also recommended to use explainable AI so that the system explains why the vulnerability is risky and how to fix it in a clearer way. Additionally, users suggested deeper scanning, support for newer OWASP Top 10 vulnerabilities, and multi-threaded scanning to speed up the process. These improvements will contribute to making the system more sophisticated, intelligent and relevant to contemporary cybersecurity measures.

System usability and automation is another area of improvement. Several users appreciated the existing easy-to-use interface but recommended additional user interfaces and interactions and simplified explanations for non-technical end users. Some respondents suggested the inclusion of real-time notifications such as email alerts or system alerts if vulnerabilities are detected. There is also a suggestion to have scheduled automatic scans to ensure users don't have to manually type in the URL every time. One of the most innovative and future-oriented recommendations made by users was to integrate the system with the Google, Chrome and Firefox search engines. This means the system will be simpler and more proactive as the system will automatically scan websites as they are browsed, creating a simpler system and helping in making users aware of the level of risk.

It is also often noted that performance should be improved, as users demand shorter scan times, visible scan progress bars, and confirmation that the scanner will not overload the target site and does not cause any interference. Other suggestions are adding vulnerability database version information, improving hardware performance to enable faster response, and preparing the system for future market deployment. In general, this suggestion shows that users have high expectations for the system and are willing to use features that will make the system faster, smarter, and more automated. With these improvements, the system can transform into a more robust cybersecurity tool that is greater in coverage, more efficient, and has a better user experience.

10.5. Conclusion

Overall, this project was a good learning experience that allowed me to improve my technical and problem-solving skills. In the process of developing the Web Vulnerability Scanner system, I encountered several difficulties, specifically, working with the OWASP ZAP API, Nikto parsing logic, and Docker configurations. This problem usually leads to wasted time, unexpected interruptions, and time-consuming debugging. However, I was able to address each issue and successfully complete the system by researching the documentation, trying out various configurations, and refining the code step by step. Among the biggest lessons was the need to have clear requirements gathering and continuous testing as it helped me stay on schedule and ensure that the system was what the users really needed. As I would refine the project, I would allocate more time to fine-tune the UI, scan for performance optimizations, and study automation tools to speed up the scanning process. This project showed me that patience, adequate planning, and early testing are essential, and it has made me more confident in my ability to tackle real system development tasks.

Appendix A – Requirements Specification Document

1. Demonstration Video (Link)

Web Vulnerability Scanner (YouTube Link)

<https://youtu.be/sUCwaOXLHDY>

2. Project Source Code (Github Link)

Web Vulnerability Scanner Link

<https://github.com/zaezays/web-vuln-scanner.git>

11/16/25, 10:49 PM

Web Vulnerability Scanner – User Feedback Survey

Web Vulnerability Scanner – User Feedback Survey

Dear Participant,

Thank you for taking the time to participate in this survey. This questionnaire is part of a user feedback initiative for evaluating the **Web Vulnerability Scanner**, a system developed as part of my Final Year Project.

My name is **Nur Izzah Akmal Binti Mohd Restu**, a student of **CT206 Bachelor of Information Technology in Cyber Security** at **Universiti Poly-Tech Malaysia (UPTM)**.

The purpose of this system is to assist small business owners and general users in identifying potential security vulnerabilities within their websites in a simple and efficient manner.

All responses will remain strictly confidential and will be used solely for academic research purposes. Your feedback is highly valuable and sincerely appreciated, as it plays a crucial role in improving the functionality and effectiveness of this system.

Thank you once again for your support and participation.

1. Email *

2. What is your age group?

Mark only one oval.

- ☐ 18–24
- ☐ 25–34
- ☐ 35–44
- ☐ 45 and above

<https://docs.google.com/forms/d/1RprHXjYI7HAW3F6FDlia5YUtgZP963EGWGH1nbFso/edit>

1/6

11/16/25, 10:49 PM

Web Vulnerability Scanner – User Feedback Survey

3. Have you ever managed or owned a website for business or personal use?

Mark only one oval.

☐ Yes

☐ No

4. How often do you manage or interact with a website (e.g. update content, check visitors, use tools)?

Mark only one oval.

☐ Daily

☐ Weekly

☐ Monthly

☐ Rarely

☐ Never

5. Are you aware that websites can be attacked through vulnerabilities like SQL Injection and Cross-Site Scripting (XSS)?

Mark only one oval.

☐ Yes

☐ No

☐ Not sure

<https://docs.google.com/forms/d/1RprHXjYI7HAvW3F6FDlia5YUtgZP963EGWGH1nbFso/edit>

2/6

11/16/25, 10:49 PM

Web Vulnerability Scanner – User Feedback Survey

6. Have you ever experienced a security issue on your website (e.g., suspicious activity, broken links, data loss)?

Mark only one oval.

- ☐ Yes
☐ No
☐ Not sure

7. Have you ever used or heard of a website vulnerability scanner before?

Mark only one oval.

- ☐ Yes, I've used one
☐ I've heard of them but never used
☐ No, never heard of it

8. Do you think a system that can automatically scan your website for vulnerabilities would be useful?

Mark only one oval.

- ☐ Yes
☐ No
☐ Maybe

9. How important is it for you to receive clear explanations and suggestions when a security issue is found?

Mark only one oval.

- ☐ Very important
☐ Somewhat important
☐ Not important

<https://docs.google.com/forms/d/1RprHXjYI7HAvW3F6FDlia5YUtgZP963EGWGH1nbFso/edit>

3/6

11/16/25, 10:49 PM

Web Vulnerability Scanner – User Feedback Survey

10. Would a user-friendly interface (simple layout, clear buttons) encourage you to use the system more regularly?

Mark only one oval.

- ☐ Yes
☐ No
☐ Maybe

11. Would you like the system to show results using color codes (e.g., green = safe, red = high risk)?

Mark only one oval.

- ☐ Yes
☐ No
☐ Not Sure

12. How would you prefer to be notified if the system finds a problem?

Check all that apply.

- ☐ On the dashboard only (when I log in)
☐ By email
☐ No need for notification

13. Which basic features would be most useful to you in a website security scanner?

Check all that apply.

- ☐ Easy-to-understand scan results
☐ Clear status (safe, warning, danger)
☐ Option to re-scan the website anytime
☐ Option to download the scan result

<https://docs.google.com/forms/d/1RprHXjYI7HAvW3F6FDlia5YUtgZP963EGWGH1nbFso/edit>

4/6

11/16/25, 10:49 PM

Web Vulnerability Scanner – User Feedback Survey

14. How often would you prefer to scan your website using the system?

Check all that apply.

- ☐ Once a week
- ☐ Once a month
- ☐ Only when needed
- ☐ Not sure

15. Would you prefer short scan results with summaries or full technical details?

Mark only one oval.

- ☐ Summary only
- ☐ Full technical info
- ☐ Both

16. Do you think this system could help small businesses manage website security better?

Mark only one oval.

- ☐ Yes
- ☐ No
- ☐ Not Sure

This content is neither created nor endorsed by Google.

Google Forms

<https://docs.google.com/forms/d/1RprHXjYI7HAvW3F6FDlia5YUtgZP963EGWGH1nbFso/edit>

5/6

11/16/25, 10:50 PM

Post-Development Questionnaire: User Testing & Client Feedback

Post-Development Questionnaire: User Testing & Client Feedback

Dear Participant,

Thank you for participating in this post-development evaluation. This feedback form is designed to assess your experience and satisfaction after testing the **Web Vulnerability Scanner**, developed by **Nur Izzah Akmal Binti Mohd Restu**, a **CT206 Bachelor of Information Technology in Cyber Security** student at **Universiti Poly-Tech Malaysia (UPTM)**.

The system aims to provide an efficient, user-friendly solution for detecting web security vulnerabilities, especially for small businesses seeking affordable security assessment tools.

Your insights and comments will help evaluate the system's **functionality, usability, and readiness for deployment**. All information collected will be **strictly confidential** and used solely for academic evaluation.

Your feedback is vital for further improvement of this project.

Thank you for your time, support, and valuable participation.

* Indicates required question

1. Email *

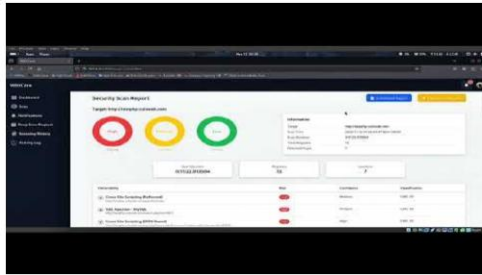
https://docs.google.com/forms/d/1O5VdsQeDvvvi8J-3IMv7Op-91CsmUGNaPTqKr8kj_q8/edit

1/7

11/16/25, 10:50 PM

Post-Development Questionnaire: User Testing & Client Feedback

Web Vulnerability Scanner (WitzCore)

<http://youtube.com/watch?v=gJtswoltitU>

Dashboard and System Features

This section evaluates how clear, informative, and useful the dashboard and main features are in helping users monitor and understand system activity.

2. **Do you feel that the login and access control features make the system secure?** *

Mark only one oval.

- ☐ Agree
☐ Neutral
☐ Disagree

3. **How clear and organized was the information displayed on the dashboard (e.g., total scans, completed scans, users, and organizations)?** *

Mark only one oval.

- ☐ Clear
☐ Neutral
☐ Unclear

https://docs.google.com/forms/d/1O5VdsQeDvvvi8J-3IMv7Op-91CsmUGNaPTqKr8kj_q8/edit

2/7

11/16/25, 10:50 PM

Post-Development Questionnaire: User Testing & Client Feedback

4. **How useful were the dashboard charts (such as vulnerability summary and trend graph) in helping you understand scan results?** *

Mark only one oval.

- ☐ Useful
- ☐ Neutral
- ☐ Not Useful

5. **Was it easy to navigate between different sections of the system (Dashboard, Scan, Notifications, etc.)?** *

Mark only one oval.

- ☐ Easy
- ☐ Neutral
- ☐ Difficult

6. **How well do you think the system features (e.g., Deep Scan Request, Notifications, Activity Log) support the overall scanning process?** *

Mark only one oval.

- ☐ Excellent
- ☐ Good
- ☐ Average
- ☐ Poor

Scan and Scanning Result

This section aims to assess how clearly the system presents vulnerability scan results. It focuses on whether the information, risk levels, and recommendations are easy for users to understand and take action on.

https://docs.google.com/forms/d/1O5VdsQeDvvvi8J-3IMv7Op-91CsmUGNaPTqKr8kj_q8/edit

3/7

11/16/25, 10:50 PM

Post-Development Questionnaire: User Testing & Client Feedback

7. Was it simple to start and run a website scan using the system? **Mark only one oval.*

- ☐ Agree
- ☐ Neutral
- ☐ Disagree

8. Was the scanning time reasonable based on the amount of information shown? **Mark only one oval.*

- ☐ Fast
- ☐ Acceptable
- ☐ Slow

9. How clear and easy to understand were the vulnerability descriptions shown in the scan report? **Mark only one oval.*

- ☐ Clear
- ☐ Neutral
- ☐ Unclear

https://docs.google.com/forms/d/1O5VdsQeDvvvi8J-3IMv7Op-91CsmUGNaPTqKr8kj_q8/edit

4/7

11/16/25, 10:50 PM

Post-Development Questionnaire: User Testing & Client Feedback

10. **Were the risk levels (High, Medium, Low) displayed in a way that helped you understand the severity of each issue?** *

Mark only one oval.

- ☐ Clear
- ☐ Neutral
- ☐ Confusing

11. **How useful were the recommendations or solutions provided for each vulnerability?** *

Mark only one oval.

- ☐ Useful
- ☐ Not Useful

12. **Was the overall layout of the scan report (charts, tables, and summaries) easy to read and interpret?** *

Mark only one oval.

- ☐ Easy
- ☐ Neutral
- ☐ Difficult

Overall Feedback

This section allows users to share their overall impressions and suggestions about the web vulnerability scanner system. Your feedback will help improve the system's performance, usability, and reporting features in the future. Please provide honest and constructive comments based on your experience.

https://docs.google.com/forms/d/1O5VdsQeDvvvi8J-3IMv7Op-91CsmUGNaPTqKr8kj_q8/edit

5/7

11/16/25, 10:50 PM

Post-Development Questionnaire: User Testing & Client Feedback

13. **Overall, how satisfied are you with the dashboard and feature functionality of the system?** *

Mark only one oval.

- ☐ Satisfied
☐ Neutral
☐ Unsatisfied

14. **Overall, how satisfied are you with how the system displays and explains the scan results?** *

Mark only one oval.

- ☐ Satisfied
☐ Neutral
☐ Unsatisfied

15. **How well do you think this system helps identify and manage security weaknesses in web applications?** *

Mark only one oval.

- ☐ Well
☐ Neutral
☐ Poorly

https://docs.google.com/forms/d/1O5VdsQeDvvi8J-3IMv7Op-91CsmUGNaPTqKr8kj_q8/edit

6/7

11/16/25, 10:50 PM

Post-Development Questionnaire: User Testing & Client Feedback

16. How effective is the system in increasing user awareness about website security issues? *

Mark only one oval.

☐ Effective

☐ Neutral

☐ Ineffective

17. Overall, how satisfied are you with this system? *

12345

☆

☆

☆

☆

☆

18. As a user/client, what suggestions or feedback do you have to help improve this Web Vulnerability Scanner for future development? *

This content is neither created nor endorsed by Google.

Google Forms

https://docs.google.com/forms/d/1O5VdsQeDvvvi8J-3IMv7Op-91CsmUGNaPTqKr8kj_q8/edit

7/7

Appendix B – User Manual



WEB VULNERABILITY SCANNER (WitzCore)

USER MANUAL

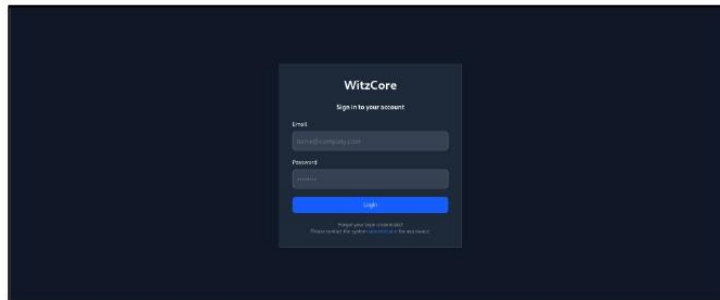
PREPARED BY
NUR IZZAH AKMAL BINTI MOHD RESTU
BACHELOR OF INFORMATION TECHNOLOGY IN CYBERSECURITY
FACULTY OF COMPUTING AND MULTIMEDIA
UNIVERSITI POLY-TECH MALAYSIA
10 NOVEMBER 2025

Table Of Contents

ADMIN USER MANUAL	3
1. Log into system	4
2. One Time Password Verification	4
3. Admin Dashboard Overview.....	5
4. My Profile Page.....	6
5. Performing a Security Scan	7
6. Security Scan Report.....	8
7. Notification.....	11
8. Scan History	12
9. User Management.....	12
10. Add New User.....	13
11. User Details	14
12. Edit User	14
13. Manage Deep Scan Requests	15
14. Activity Log	17
USER MANUAL	18
1. Log Into System.....	19
2. Administrator Contact Form.....	19
3. One Time Password Verification	20
4. User Dashboard Overview.....	20
5. My Profile Page.....	22
6. Performing a Security Scan	22
7. Security Scan Report.....	23
8. Notification.....	26
9. Deep Scan Request	26
10. Scan History	27
11. Activity Log	28

ADMIN USER MANUAL

1. Log into system



Getting Started

- Open the WitzCore website URL in browser.
- Enter your registered email and password that provide by admin at email.
- Click **Login** button
- If credentials are valid, an OTP number will be sent to your email.

Troubleshoot: If “User not registered”, “User not active” or “Invalid Password” appears, Contact the administrator or use the admin contact form.

2. One Time Password Verification

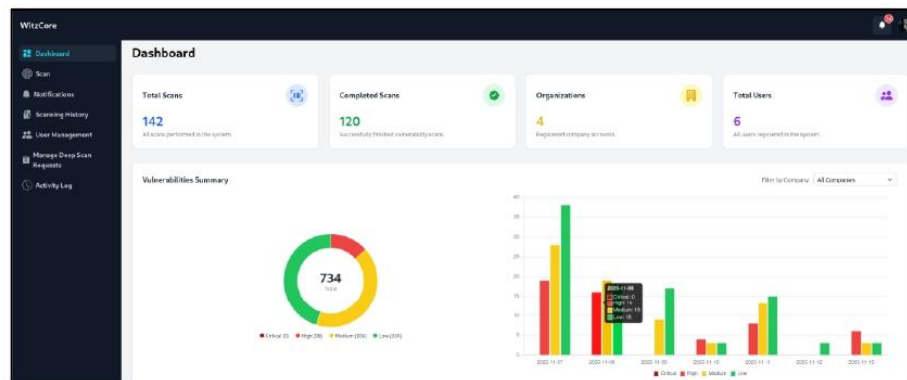


- After login, enter 6-digit OTP number that you received at email.
- Click **Verify OTP** button
- If OTP number expires, click **Resend OTP** to get new OTP number.

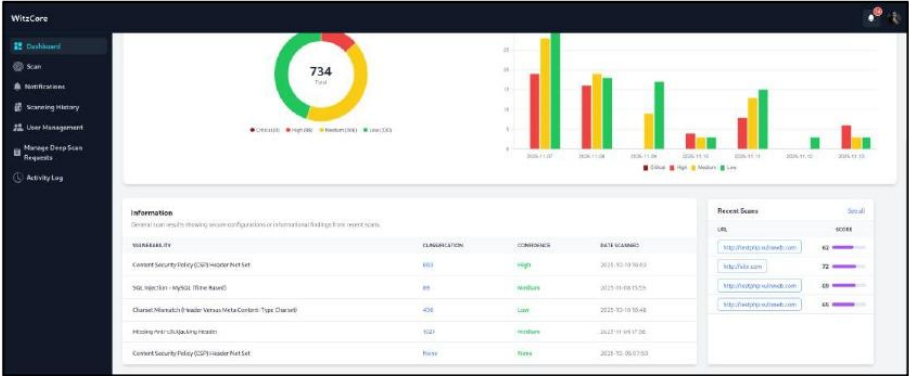
Tip: OTP has limited time, just only 5 minutes to verify it. Check spam folder if you do not receive it.

3. Admin Dashboard Overview

After logging in, the system will load the dashboard with summary cards, charts, vulnerability insights and recent scanned information

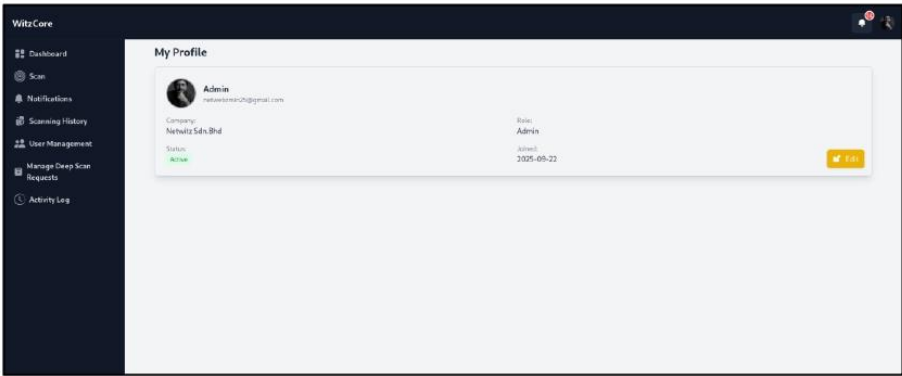


- **Summary card**
These 4 cards give you a quick real-time overview:
 - **Total scans:** Show the total number of scans performed by user
 - **Completed scans:** Display how many scans have finished successfully.
 - **Organizations:** Total number of registered companies
 - **Total Users:** Show the number of all users that registered.
- **Vulnerability Summary**
 - **Donut chart (Vulnerabilities by Severity):** Display the total number of vulnerabilities divided by: Critical, High, Medium and Low
 - **Bar chart (Vulnerabilities Over Time):** Show vulnerability counts by day and color represent severity:
 - Red Dark = Critical
 - Red = High
 - Yellow = Medium
 - Green = Low
 - **Company Filter:** Admin can filter all charts by company, you can choose a company to see their company's chart.



- Information Table**
This table provides a list of recent vulnerabilities that fall under the “Informational” category. Each Row displays:
 - Vulnerability Name
 - Classification (CWE)
 - Confidence Level (Critical/High/Medium/Low/Informational)
 - Date Scanned
- Recent Scan Table**
This table shows the latest scan performed by user
- Navbar**
At the top left, there is notification icon and user profile. Click the user profile, there you can see My Profile and Log out button.

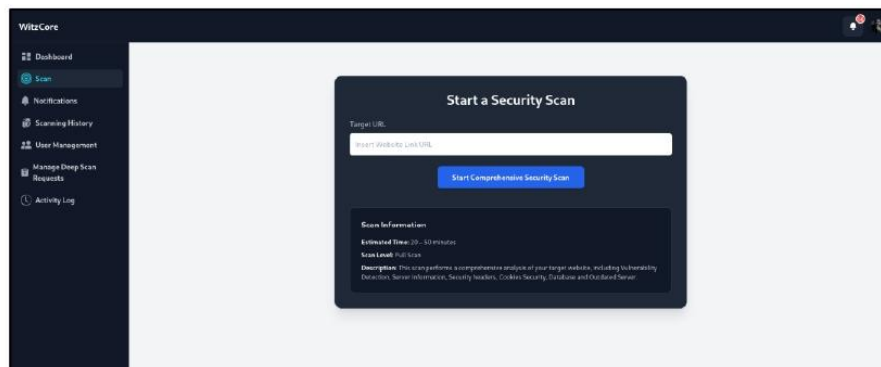
4. My Profile Page



The My Profile page allows you to view and update the personal account information. This section helps you verify your role, company, and status, and edit details such as username and profile picture.

5. Performing a Security Scan

This section explains how you can start a full security scan using the Web Vulnerability Scanner.



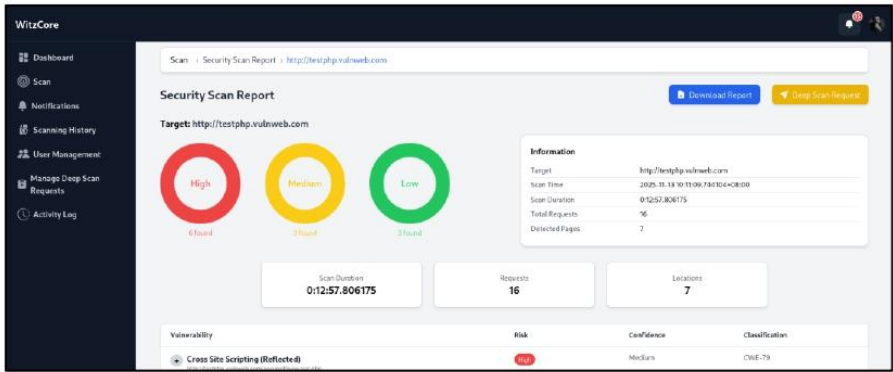
The Scan Page contains the following components:

- a. **Target URL Field:** This is where you type the **website address (URL)** that you want to scan.
- b. **Start Comprehensive Security Scan:** This button starts the full scan process. Once you clicked, the backend prepares scanning threads and sends requests to scanning engines.
- c. **Scan Information Box:** This section displays important information before scanning:
 - **Estimated Time:** 20–50 minutes depending on website size and response time.
 - **Scan Level:** Full Scan (includes crawling, spidering, header checks, server info, and vulnerability tests).
 - **Description:**
Explains that the scanner will check.

Warning: Ensure that you have permission to test the website URL.

6. Security Scan Report

The Security Scan Report page displays the full results of a website scan. You can review vulnerability details, scan statistics, risk levels, and recommended solutions



1. Risk Summary

You will see **three colored donut charts** that summarize the risks:

- **Red – High Risk:** Dangerous vulnerabilities
- **Yellow – Medium Risk:** Moderate vulnerabilities that affect security but require some conditions to exploit.
- **Green – Low Risk:** Minor issues with low impact.

Each chart shows the total number of findings for that severity

2. Scan Information Table

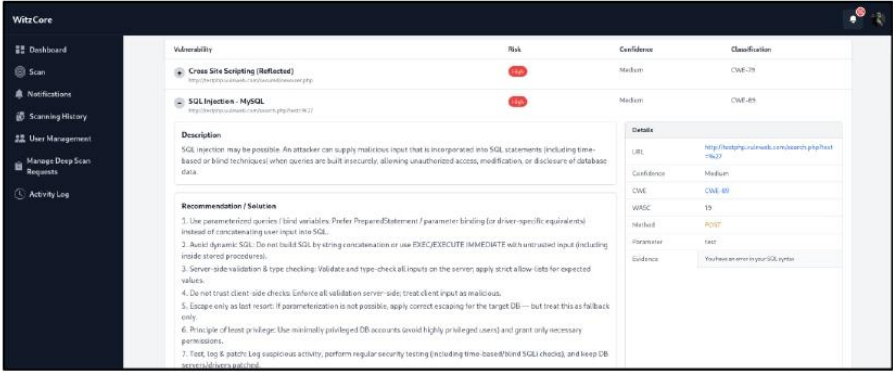
On the right, the Information panel displays scan metadata:

- **Target:** URL of the website that was scanned.
- **Scan Time:** When the scan started.
- **Scan Duration:** Total time taken to finish the scan.
- **Total Requests:** Number of HTTP requests sent to the website.
- **Detected Pages:** Number of web pages discovered during scanning.

3. Scan Duration, Requests and Locations

Below the donut charts, you can see card:

- **Scan Duration** – Exact length of the scan in HH:MM:SS.
- **Requests** – Total HTTP requests the scanner made.
- **Locations** – Number of endpoints or URLs tested.



4. Vulnerability Table

This table lists all vulnerabilities found during the scan:

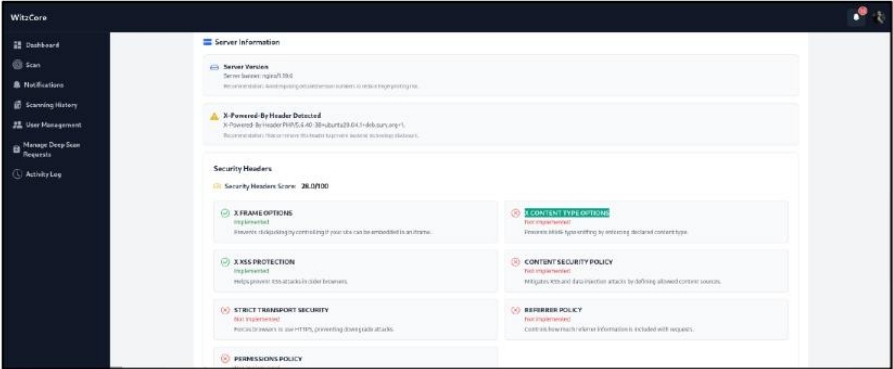
- **Vulnerability:** Name of the weakness (e.g., “Cross-Site Scripting”).
- **Risk:** High, Medium, Low or Informational.
- **Confidence:** How certain the scanner is about the finding (e.g., High, Medium).
- **Classification:** Technical reference code (CWE, OWASP).

You can expand each row to see more details.

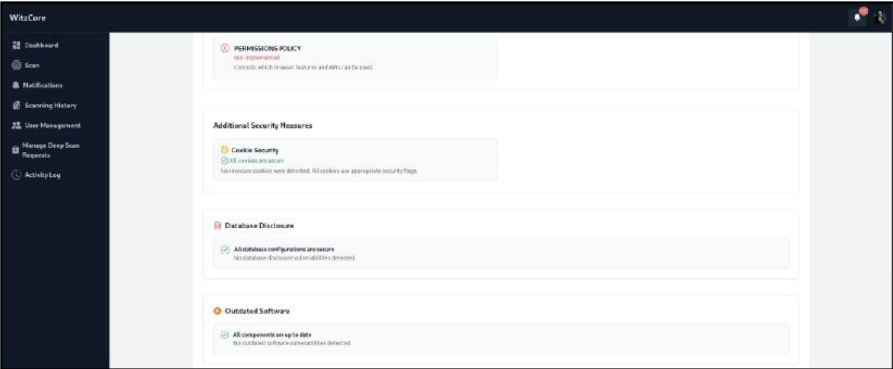
5. Detailed Vulnerability Breakdown

When you click on any vulnerability in the table, more information appears:

- a. **Description:** This section explains what vulnerability is and how attackers may exploit it.
- b. **Recommendation / Solution:** This section provides you step-by-step technical guidance on how to fix the issue.
- c. **References:** Links to official documentation or trusted cybersecurity websites. This allows you to read more about vulnerability.
- d. **Details Table:** This section shows you the technical information from the scanner
 - **URL:** Exact page where the vulnerability was found.
 - **Confidence:** Scanner certainty level.
 - **CWE:** (Common Weakness Enumeration) ID for technical reference.
 - **WASC :** Web Application Security Consortium category.
 - **Method:** HTTP method used (GET, POST).
 - **Parameter:** Input parameter that caused vulnerability.
 - **Evidence:** The response or output that proves the issue.



6. Server Information
- This section displays important server-level details that help identify potential weaknesses.
- Server Version: Shows the web server software version and recommendation
 - X-Powered-By Header: This header reveals what technology your website uses. If detected, you will see a warning.
7. Security Headers
- Security headers help protect your website against attacks by controlling how browsers behave. Each header will show either Implemented (green) or Not Implemented (red).



8. Additional Security Measures
- This section shows extra security checks beyond headers. These help ensure your website settings are safe.
- a. Cookie Security
- Shows whether your cookies use secure flags:
- Secure: Only sent over HTTPS
 - HttpOnly: Cannot be accessed by JavaScript
 - SameSite: Protects from cross-site request forgery (CSRF)
- If all cookies are secure, you will see a green “All cookies are secure”.

9. Database Disclosure

Check if your website accidentally exposes database information.

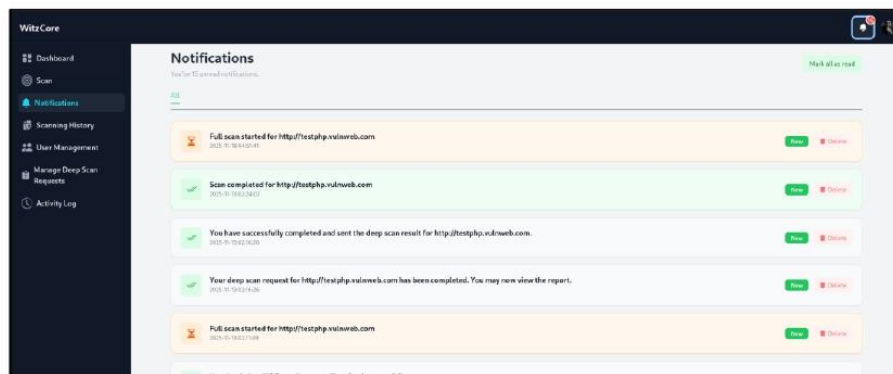
- **Safe:** No database configurations are exposed
- **If exposed:** Attackers can learn your system structure.

10. Outdated Software

Detects if your website uses outdated technologies.

- **Green:** All components up to date
- **Red:** Old versions detected

7. Notification



The Notifications page helps you keep track of important system activities

- Mark All as Read:** Click the button Mark all as read on the top-right. All unread notifications will be marked as read immediately.
- Delete Button:** Look for the notification you want to remove and then click the Delete icon. The notification will be removed permanently.

Message: Deleted notifications cannot be restored.

8. Scan History

The Scan History page provides a detailed record of all vulnerability scans performed through your system.

WitzCore

Dashboard

Scan

Notifications

Scanning History

User Management

Manage Deep Scan Requests

Activity Log

Scan History

View all your previous security scans and results.

TARGET	STATUS	STARTED	DURATION	RESULT	ACTION
http://example.vulnweb.com	Completed	Nov 15, 2025 10:11	0:02:57.80K175	View Details	
http://example.vulnweb.com	Completed	Nov 10, 2025 08:10	0:01:58.135545	View Details	
https://www.igtrm.edu.my	Completed	Nov 09, 2025 16:22	0:38:17.048558	View Details	
https://shop.beneluxgo.com/	Completed	Nov 09, 2025 12:31	2:04:30.371895	View Details	
http://testaggrnet.vulnweb.com/	Completed	Oct 13, 2025 09:26	0:06:17.685219	View Details	
http://testaggrnet.vulnweb.com/	Completed	Oct 13, 2025 08:18	0:00:34.911819	View Details	
http://example.vulnweb.com	Completed	Oct 12, 2025 18:07	0:02:05.831245	View Details	
http://example.vulnweb.com	Completed	Oct 11, 2025 09:22	0:02:48.437362	View Details	
http://example.vulnweb.com	Completed	Oct 12, 2025 02:43	0:00:08.890722	View Details	
http://example.vulnweb.com	Completed	Oct 11, 2025 18:07	0:03:08.360905	View Details	

1. **Target:** Displays the URL of the website that was scanned
2. **Status:** This shows whether the scan is Completed, In Progress or Failed.
3. **Started:** The exact timestamp when the scan began.
4. **Duration:** Shows how long the scan took to complete.
5. **Result:** Click View Details to open the complete scan report. The report provides all technical findings in an easy-to-read format.
6. **Action:** The red delete icon allows you to remove a scan entry.

9. User Management

The User Management module allows you to manage all registered users in the WitzCore system.

WitzCore

Dashboard

Scan

Notifications

Scanning History

User Management

Manage Deep Scan Requests

Activity Log

User Management

Company Filter

Showing 1 - 6 of 6 users

ADD NEW USER

PROFILE	USERS	COMPANY	ROLE	STATUS	CREATED	ACTIONS
	Isak isak.akmal10@gmail.com	RMA Tutor Hub Sdn Bhd	User	Inactive	2025-11-12	
	Mufida mufidamuz21@gmail.com	TDR (Malaysia) Sdn Bhd	User	Active	2025-10-13	
	Jack jackm123@gmail.com	Universiti Poly-Tech Malaysia	User	Active	2025-09-24	
	Adrian adrianadrian21@gmail.com	Netwotz Sdn Bhd	Admin	Active	2025-08-22	
	Mufis 602780727@student.upm.edu.my	Universiti Poly-Tech Malaysia	User	Active	2025-09-21	
	Mia miayaz2@gmail.com	Netwotz Sdn Bhd	Admin	Active	2025-09-12	

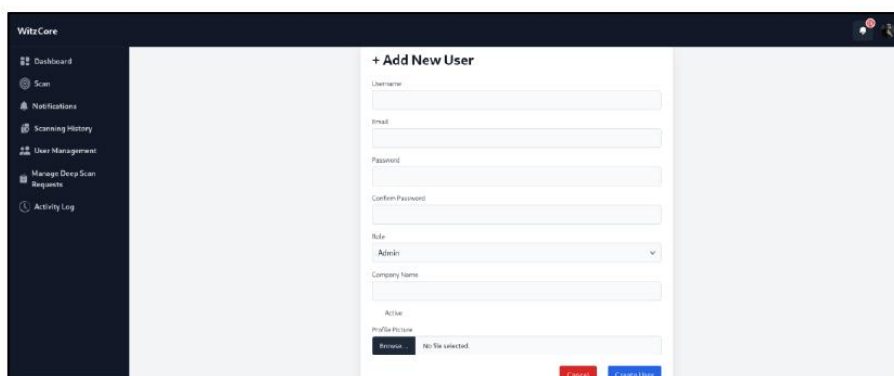
Page Overview

The User Management table contains these columns:

- **Profile:** Shows the user's profile picture
- **User:** Displays the user's name and email address.
- **Company:** Shows the organization the user belongs to.
- **Role:** The access level assigned to the user
- **Status:** Indicates whether the user is **Active** or **Not Active**
- **Created:** Shows the date when the account was created.
- **Actions:** Icons are provided to perform user management tasks
 - View: View full user profile
 - Edit: Update user details,
 - Delete: Remove user from the system
 - Add New user: Create a new user
- **Company Filter:** Select a specific company to filter the list and show users only from that organization.

10. Add New User

This section explains how you can create a new user account in the system



Fill in User Details

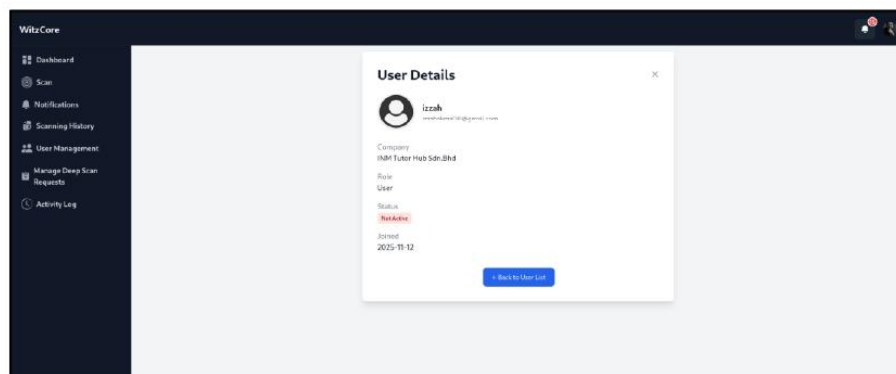
- **Username:** Enter the user's full name
- **Email:** Enter a valid email address. This email will be used for login and OTP verification.
- **Password:** Set a secure password for the user.
- **Confirm Password:** Re-enter the same password to avoid mistakes.
- **Role:** assign role to user

Important: Only assign Admin role to Netwitz staff.

- **Company Name:** Enter the user's company.

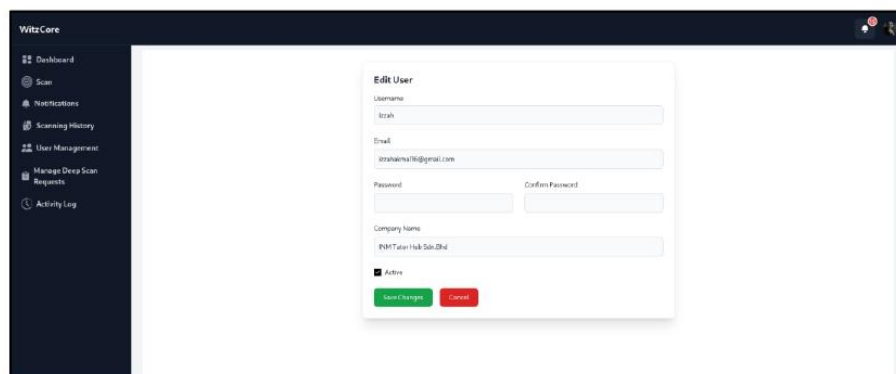
- **Active:** Tick the checkbox to activate the account. Uncheck it if you do not want.
- **Profile Picture:** To upload an image
- **Create User button:** Click Create User to create the account.
- **Cancel:** Click Cancel if you want to exit without saving.

11. User Details



To allow you to see a user's full profile, including role, company, status, and account creation date.

12. Edit User



To update the user's information such as username, password, company name, or activation status.

Update User Details:

- **Username:** Change the user username.
- **Email:** Update the user's login email.

- **Password / Confirm Password:** Optional, fill only if you want to reset the user's password.
- **Company Name:** Change the company the user belongs to.
- **Status (Active / Not Active):** Toggle the user's account availability.
- **Save Changes:** Click Save Changes to apply updates.

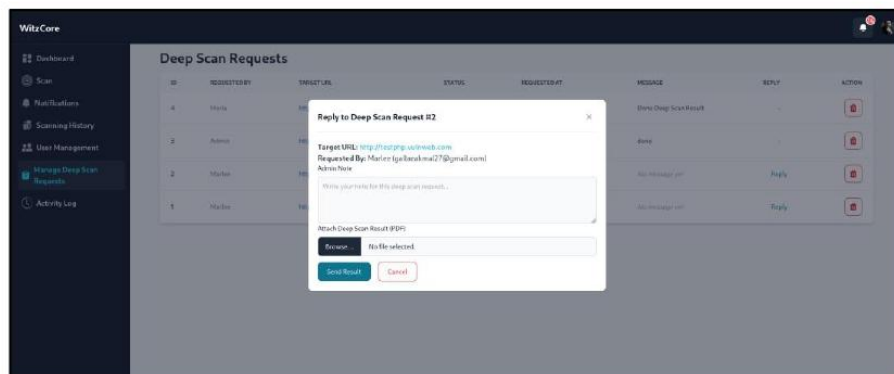
13. Manage Deep Scan Requests

ID	REQUESTED BY	TARGET URL	STATUS	REQUESTED AT	MESSAGE	REPLY	ACTION
4	Haris	http://thingsip.vulnweb.com	Completed	2025-01-12 07:45:10	Done Deep Scan Result	-	
5	Admin	http://thingsip.vulnweb.com	Completed	2025-01-12 07:45:10	Done	-	
2	Haris	http://thingsip.vulnweb.com	Pending	2025-01-09 08:01:38	No message yet	Reply	
1	Haris	http://thingsip.vulnweb.com	Pending	2025-01-08 05:50:00	No message yet	Reply	

This module allows you (administrators) to **view, manage, respond to, and close deep scan requests** submitted by clients. Deep scans are advanced manual analyses performed by Netwiz staff when clients request further investigation beyond the automated scan.

Deep Scan Requests Table

- **ID:** Unique request number assigned automatically.
- **Requested By:** The name & email of the client who submitted the request.
- **Target URL:** The website that requires a deep manual scan.
- **Status:** Pending/Completed
- **Requested At:** Timestamp of when the client submitted the request.
- **Message:** Any additional notes for admin.
- **Reply:** Admin's response status. If not replied yet, the button will appear.
- **Action:** Delete icon to remove a request if needed.



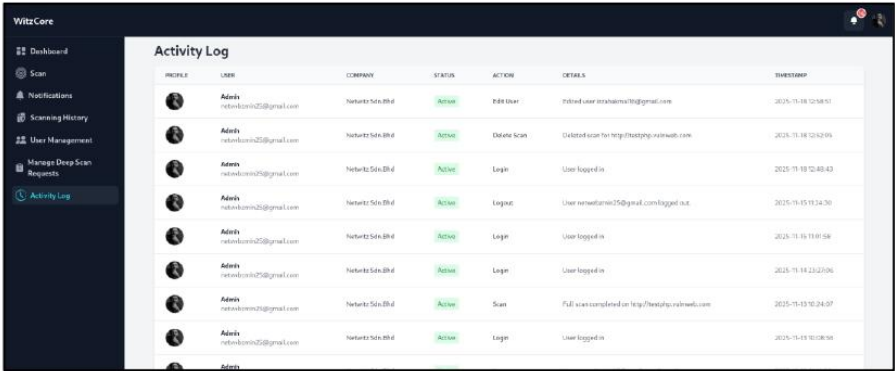
Responding to a Deep Scan Request

To reply to a request:

1. Click **Reply** on the row of the request you want to respond to.
2. A pop-up window will appear, showing:
 - **Target URL**
 - **Requested By (client name + email)**
 - **Admin Note** – a text box to write your response.
 - **Attachment Upload** – upload a PDF deep scan report (if available).
3. In the **Admin Note** field, type your message, such as:
 - Summary of your findings
 - Confirmation that the deep scan report is attached
4. Click **Browse...** to attach a PDF report.
 - PDF attachment is optional but recommended for detailed findings.
5. Click **Send Result**.

14. Activity Log

The Activity Log records every important action performed inside the system, including login attempts, scan activities, user management changes, and deep scan operations. This feature helps you monitor system usage, track security-related events, and detect unusual activity.



The screenshot shows the VitzCore Activity Log interface. It features a dark sidebar with navigation options: Dashboard, Scan, Notifications, Scanning History, User Management, Manage Deep Scan Requests, and Activity Log (which is highlighted). The main content area displays a table titled 'Activity Log' with the following columns: PROFILE, USER, COMPANY, STATUS, ACTION, DETAILS, and TIMESTAMP. The table contains several rows of activity logs, all with a status of 'Active'.

PROFILE	USER	COMPANY	STATUS	ACTION	DETAILS	TIMESTAMP
	Admin networks@vitzcore.com	Networks Sdn Bhd	Active	First User	Initial user networks@vitzcore.com	2025-11-18 12:58:51
	Admin networks@vitzcore.com	Networks Sdn Bhd	Active	Delete Scan	Deleted scan for http://testphp.vulnweb.com	2025-11-18 12:52:05
	Admin networks@vitzcore.com	Networks Sdn Bhd	Active	Login	User logged in	2025-11-18 12:45:43
	Admin networks@vitzcore.com	Networks Sdn Bhd	Active	Logout	User networks@vitzcore.com logged out	2025-11-15 11:24:30
	Admin networks@vitzcore.com	Networks Sdn Bhd	Active	Login	User logged in	2025-11-15 11:01:58
	Admin networks@vitzcore.com	Networks Sdn Bhd	Active	Login	User logged in	2025-11-14 22:22:06
	Admin networks@vitzcore.com	Networks Sdn Bhd	Active	Scan	Full scan completed on http://testphp.vulnweb.com	2025-11-13 10:24:37
	Admin networks@vitzcore.com	Networks Sdn Bhd	Active	Login	User logged in	2025-11-13 10:18:58

Activity Log Table

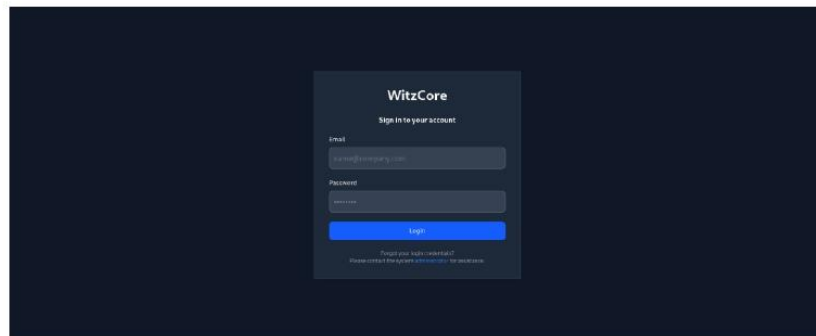
Each row represents one system action. Below are the columns and what they mean:

- **Profile:** Displays the user's profile picture.
- **User:** Name and email of the user who performed the action.
- **Company:** Shows the user's registered company
- **Status:** The user's account status
- **Action:** The type of activity performed
- **Details:** Additional information describing the action more clearly.
- **Timestamp:** The exact date and time when the action occurred.

This allows you to track every movement happening in the system.

USER MANUAL

1. Log Into System



Getting Started

1. Open the WitzCore website URL in browser.
2. Enter your registered email and password that provide by admin at email.
3. Click **Login** button
4. If credentials are valid, an OTP number will be sent to your email.

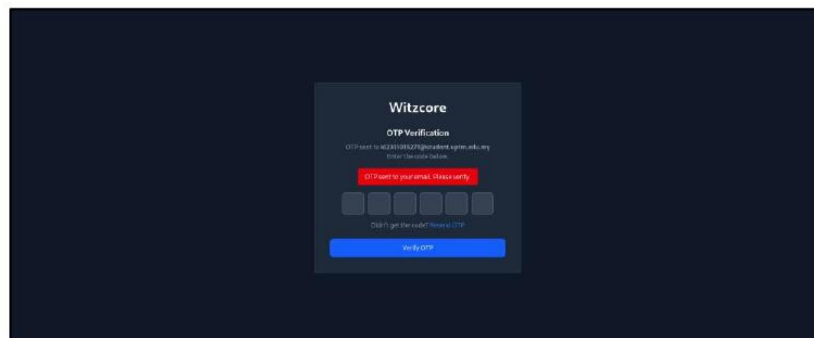
2. Administrator Contact Form



Troubleshoot: If "User not registered", "User not active" or "Invalid Password" appears, Contact the administrator or use the admin contact form.

Insert registered email and click button send request. It will notify the administrator

3. One Time Password Verification

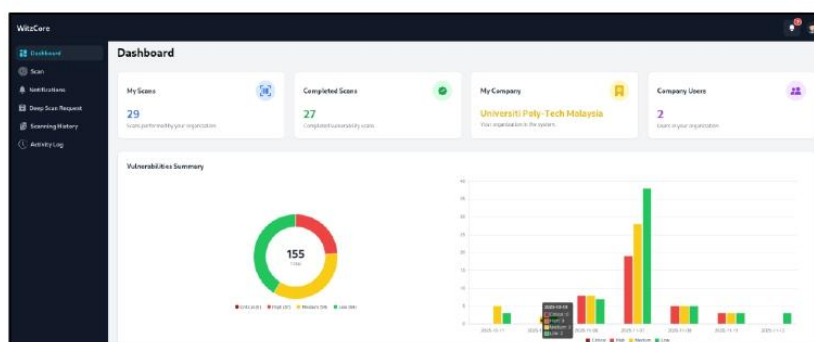


- After login, enter 6-digit OTP number that you received at email.
- Click **Verify OTP** button
- If OTP number expires, click **Resend OTP** to get new OTP number.

Tip: OTP has limited time, just only 5 minutes to verify it. Check spam folder if you do not receive it.

4. User Dashboard Overview

After logging in, the system will load the dashboard with summary cards, charts, vulnerability insights and recent scanned information



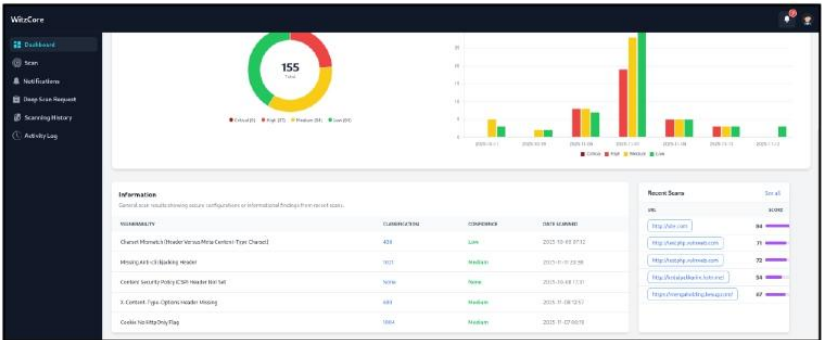
a. Summary card

These 4 cards give you a quick real-time overview:

1. **Total scans:** Show the total number of scans performed by user
2. **Completed scans:** Display how many scans have finished successfully.
3. **My Company:** Displays your registered company name in the system.
4. **Company Users:** Shows how many users in your organization have access to the system.

b. Vulnerability Summary

- **Donut chart (Vulnerabilities by Severity):** Display the total number of vulnerabilities divided by:
 - Critical
 - High
 - Medium
 - Low
- **Bar chart (Vulnerabilities Over Time):** Show vulnerability counts by day and color represent severity:
 - Red Dark = Critical
 - Red = High
 - Yellow = Medium
 - Green = Low



- c. Information Table**

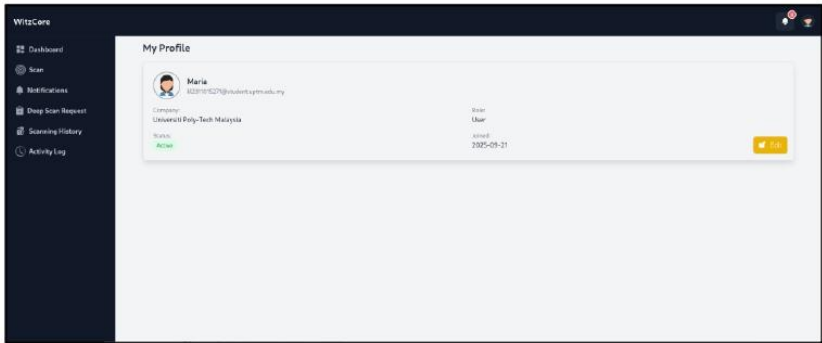
This table provides a list of recent vulnerabilities that fall under the “Informational” category. Each Row displays:

 - Vulnerability Name
 - Classification (CWE)
 - Confidence Level (Critical/High/Medium/Low/Informational)
 - Date Scanned
- d. Recent Scan Table**

This table shows the latest scan performed by user
- e. Navbar**

At the top left, there is notification icon and user profile. Click the user profile, there you can see My Profile and Log out button.

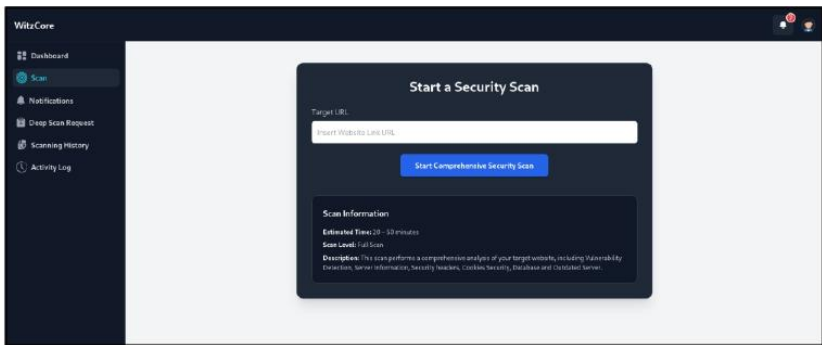
5. My Profile Page



The My Profile page allows you to view and update the personal account information. This section helps you verify your role, company, and status, and edit details such as username and profile picture

6. Performing a Security Scan

This section explains how you can start a full security scan using the Web Vulnerability Scanner.



The Scan Page contains the following components:

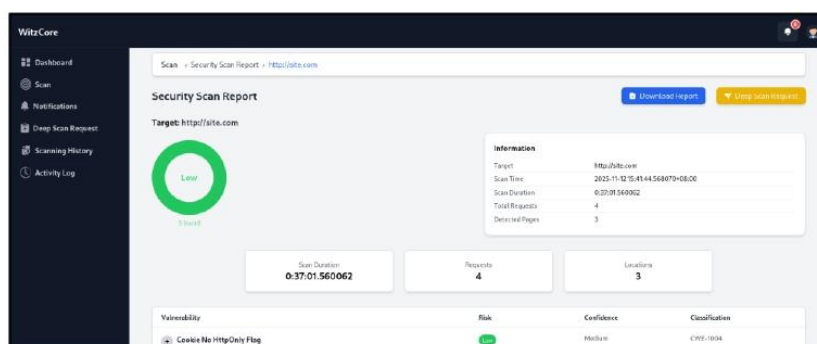
- a. **Target URL Field:** This is where you type the **website address (URL)** that you want to scan.
- b. **Start Comprehensive Security Scan:** This button starts the full scan process. Once you clicked, the backend prepares scanning threads and sends requests to scanning engines.
- c. **Scan Information Box:** This section displays important information before scanning:
 - **Estimated Time:** 20–50 minutes depending on website size and response time.
 - **Scan Level:** Full Scan (includes crawling, spidering, header checks, server info, and vulnerability tests).

- **Description:**
Explains that the scanner will check.

Warning: Ensure that you have permission to test the website URL.

7. Security Scan Report

The Security Scan Report page displays the full results of a website scan. You can review vulnerability details, scan statistics, risk levels, and recommended solutions



1. Risk Summary

You will see **three colored donut charts** that summarize the risks:

- **Red – High Risk:** Dangerous vulnerabilities
- **Yellow – Medium Risk:** Moderate vulnerabilities that affect security but require some conditions to exploit.
- **Green – Low Risk:** Minor issues with low impact.

Each chart shows the total number of findings for that severity

2. Scan Information Table

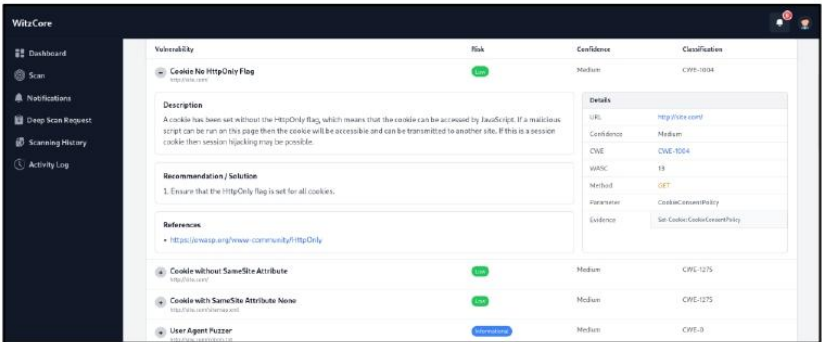
On the right, the Information panel displays scan metadata:

- **Target:** URL of the website that was scanned.
- **Scan Time:** When the scan started.
- **Scan Duration:** Total time taken to finish the scan.
- **Total Requests:** Number of HTTP requests sent to the website.
- **Detected Pages:** Number of web pages discovered during scanning.

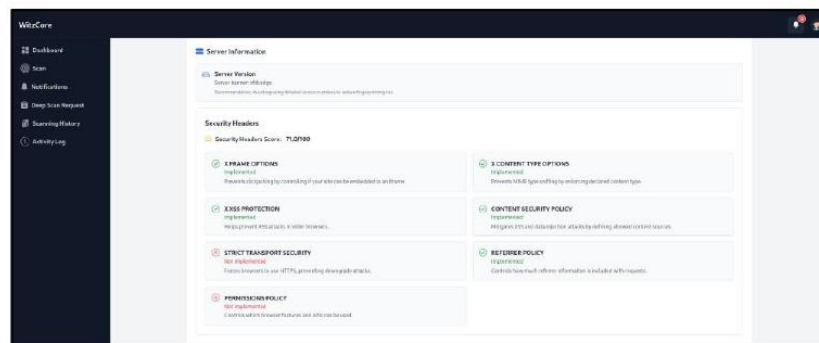
3. Scan Duration, Requests and Locations

Below the donut charts, you can see card:

- **Scan Duration** – Exact length of the scan in HH:MM:SS.
- **Requests** – Total HTTP requests the scanner made.
- **Locations** – Number of endpoints or URLs tested.



4. **Vulnerability Table**
- This table lists all vulnerabilities found during the scan:
- **Vulnerability:** Name of the weakness (e.g., “Cross-Site Scripting”).
 - **Risk:** High, Medium, Low or Informational.
 - **Confidence:** How certain the scanner is about the finding (e.g., High, Medium).
 - **Classification:** Technical reference code (CWE, OWASP).
- You can expand each row to see more details.
5. **Detailed Vulnerability Breakdown**
- When you click on any vulnerability in the table, more information appears:
- e. **Description:** This section explains what vulnerability is and how attackers may exploit it.
 - f. **Recommendation / Solution:** This section provides you step-by-step technical guidance on how to fix the issue.
 - g. **References:** Links to official documentation or trusted cybersecurity websites. This allows you to read more about vulnerability.
 - h. **Details Table:** This section shows you the technical information from the scanner
 - **URL:** Exact page where the vulnerability was found.
 - **Confidence:** Scanner certainty level.
 - **CWE:** (Common Weakness Enumeration) ID for technical reference.
 - **WASC :** Web Application Security Consortium category.
 - **Method:** HTTP method used (GET, POST).
 - **Parameter:** Input parameter that caused vulnerability.
 - **Evidence:** The response or output that proves the issue.



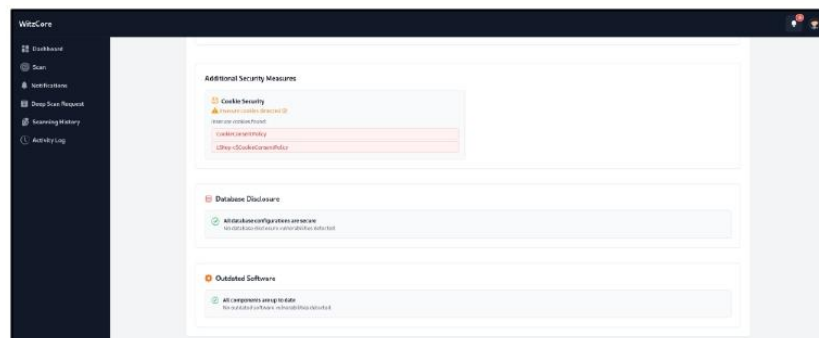
6. Server Information

This section displays important server-level details that help identify potential weaknesses.

- **Server Version:** Shows the web server software version and recommendation
- **X-Powered-By Header:** This header reveals what technology your website uses. If detected, you will see a warning.

7. Security Headers

Security headers help protect your website against attacks by controlling how browsers behave. Each header will show either Implemented (green) or Not Implemented (red).



8. Additional Security Measures

This section shows extra security checks beyond headers. This helps ensure your website settings are safe.

a. Cookie Security

Shows whether your cookies use secure flags:

- **Secure:** Only sent over HTTPS
- **HttpOnly:** Cannot be accessed by JavaScript
- **SameSite:** Protects from cross-site request forgery (CSRF)

If all cookies are secure, you will see a green “All cookies are secure”.

- b. Database Disclosure

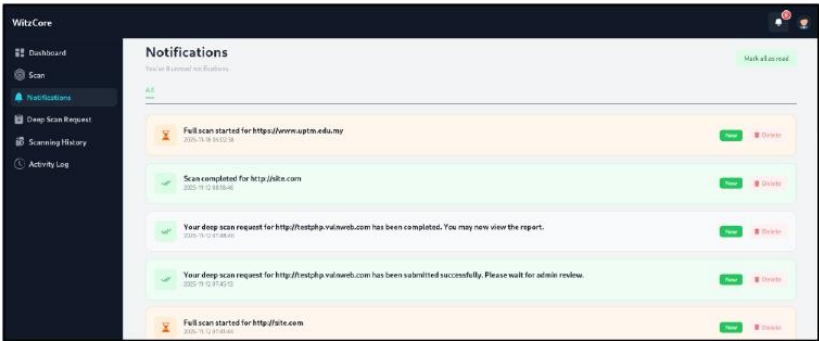
Check if your website accidentally exposes database information.

- **Safe:** No database configurations are exposed
 - **If exposed:** Attackers can learn your system structure.
- c. Outdated Software

Detects if your website uses outdated technologies.

- **Green:** All components up to date
 - **Red:** Old versions detected

8. Notification



The Notifications page helps you keep track of important system activities

- a. Mark All as Read:

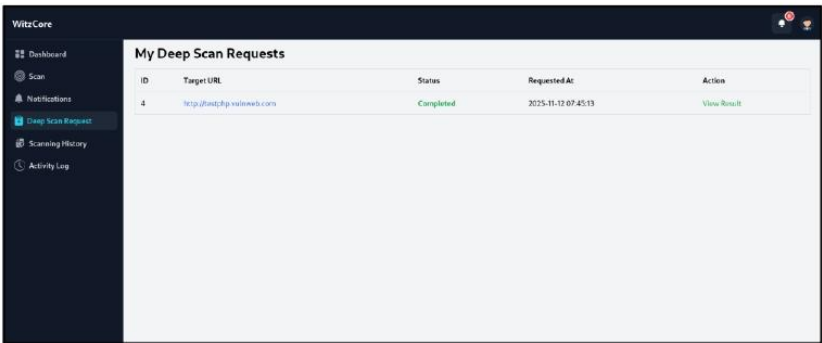
Click the button Mark all as read on the top-right. All unread notifications will be marked as read immediately.
- b. Delete Button:

Look for the notification you want to remove and then click the Delete icon. The notification will be removed permanently.
- Message:

Deleted notifications cannot be restored.

9. Deep Scan Request

This section allows you to view the status and results of deeper, manual security scans performed by Netwiz administrators.

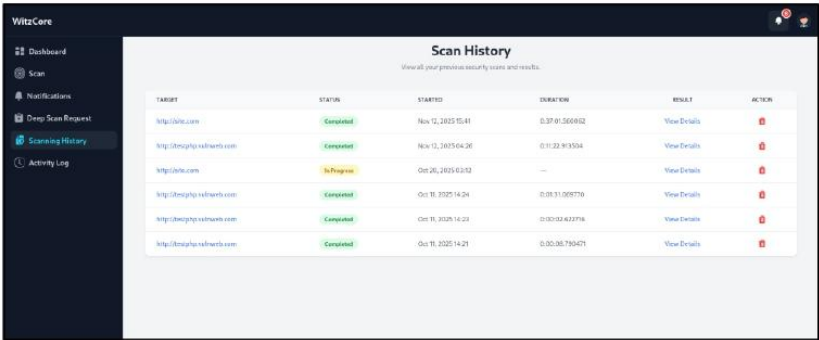


Deep Scan Request Table

- **ID:** A unique number for each request, used for tracking.
- **Target URL:** Shows the website you requested for a deeper scan. Clicking the link allows you to quickly review the scanned site.
- **Status:** Indicates the current progress of your request which is Pending/In Progress/Completed
- **Requested At:** Shows the exact date and time you submitted the request.
- **Action:** If the deep scan is completed, you will see, View Result to opens the final deep scan report

10. Scan History

The Scan History page provides a detailed record of all vulnerability scans performed through your system.



The screenshot shows the 'Scan History' page in the WitzCore interface. It features a sidebar with navigation options: Dashboard, Scan, Notifications, Deep Scan Request, Scanning History (selected), and Activity Log. The main content area displays a table titled 'Scan History' with the subtitle 'View all your previous security scans and results'. The table has six columns: TARGET, STATUS, STARTED, DURATION, RESULT, and ACTION. It lists several scan entries for the target 'http://demo.witzcore.com' with various statuses like 'Completed' and 'In Progress'.

TARGET	STATUS	STARTED	DURATION	RESULT	ACTION
http://demo.witzcore.com	Completed	Nov 12, 2025 15:41	0:37:41.588852	View Details	
http://demo.witzcore.com	Completed	Nov 12, 2025 04:26	0:11:22.913504	View Details	
http://demo.witzcore.com	In Progress	Oct 26, 2025 03:02	---	View Details	
http://demo.witzcore.com	Completed	Oct 11, 2025 14:24	0:08:31.089776	View Details	
http://demo.witzcore.com	Completed	Oct 11, 2025 14:23	0:10:02.622716	View Details	
http://demo.witzcore.com	Completed	Oct 11, 2025 14:21	0:30:36.750421	View Details	

- **Target:** Displays the URL of the website that was scanned
- **Status:** This shows whether the scan is Completed, In Progress or Failed.
- **Started:** The exact timestamp when the scan began.
- **Duration:** Shows how long the scan took to complete.
- **Result:** Click View Details to open the complete scan report. The report provides all technical findings in an easy-to-read format.
- **Action:** The red delete icon allows you to remove a scan entry.

11. Activity Log

The Activity Log records every important action performed inside the system, including login attempts, scan activities, user management changes, and deep scan operations. This feature helps you monitor system usage, track security-related events, and detect unusual activity.

WitzCore

Dashboard

Scan

Test Results

Deep Scan Report

Scanning History

Activity Log

Activity Log

Profile	User	Company	Status	Action	Details	Timestamp
	Heila	WitzCore Pte Ltd	Active	Change PC	Desktop location: https://www.witzcore.com	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59
	Heila	WitzCore Pte Ltd	Active	Login	User logged in	2025-11-18 11:00:59

Activity Log Table

Each row represents one system action. Below are the columns and what they mean:

- **Profile:** Displays the user's profile picture.
- **User:** Name and email of the user who performed the action.
- **Company:** Shows the user's registered company
- **Status:** The user's account status
- **Action:** The type of activity performed
- **Details:** Additional information describing the action more clearly.
- **Timestamp:** The exact date and time when the action occurred.

This allows you to track every movement happening in the system.



Security Scan Report

Target	http://testphp.vulnweb.com	Tools Used	OWASP ZAP, Nikto
Total Vulnerabilities	21	Scan Duration	0:11:22.913504
High Risk	3	Medium Risk	9
Low Risk	3	Completed At	2025-11-11 20:38:12.885428

Cross Site Scripting (Reflected) — High

Description:
Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user’s browser instance. Reflected XSS occurs when malicious input is immediately returned in the application’s response (e.g., via URL parameters or form submissions) and executed in the victim’s browser, allowing theft of cookies, account hijacking, or unauthorized redirection.

- Recommendation / Solution:**
- 1. Use secure frameworks/libraries: Employ libraries or frameworks that automatically handle encoding and escaping (e.g., OWASP ESAPI, Microsoft Anti-XSS).
 - 2. Contextual output encoding: Encode all untrusted output based on its context (HTML, JavaScript, attribute, or URL) before rendering.
 - 3. Input validation (allow-list): Validate and type-check input on the server using strict allow-lists; reject or sanitize unexpected values.
 - 4. Duplicate client checks server-side: Do not rely solely on client-side validation; enforce equivalent checks on the server.
 - 5. Specify character encoding: Always define a consistent page encoding (e.g., UTF-8) to avoid misinterpretation.
 - 6. Secure cookies: Use HttpOnly, Secure, and SameSite flags to protect session cookies from being accessed by scripts.
 - 7. Avoid unsafe reflection: Ensure user-supplied data is never reflected in responses without proper sanitization or encoding.
 - 8. Layered defense: Combine input validation, output encoding, secure frameworks, and cookie protection for effective XSS mitigation.

URL	http://testphp.vulnweb.com/guestbook.php
Method	POST
Confidence	Medium
CWE	79
WASC	8
Evidence	<scrIpt>alert(1);</scrIpt>

SQL Injection - MySQL — High**Description:**

SQL injection may be possible. An attacker can supply malicious input that is incorporated into SQL statements (including time-based or blind techniques) when queries are built insecurely, allowing unauthorized access, modification, or disclosure of database data.

Recommendation / Solution:

1. Use parameterized queries / bind variables: Prefer PreparedStatement / parameter binding (or driver-specific equivalents) instead of concatenating user input into SQL.
2. Avoid dynamic SQL: Do not build SQL by string concatenation or use EXEC/EXECUTE IMMEDIATE with untrusted input (including inside stored procedures).
3. Server-side validation & type checking: Validate and type-check all inputs on the server; apply strict allow-lists for expected values.
4. Do not trust client-side checks: Enforce all validation server-side; treat client input as malicious.
5. Escape only as last resort: If parameterization is not possible, apply correct escaping for the target DB — but treat this as fallback only.
6. Principle of least privilege: Use minimally privileged DB accounts (avoid highly privileged users) and grant only necessary permissions.
7. Test, log & patch: Log suspicious activity, perform regular security testing (including time-based/blind SQLi checks), and keep DB servers/drivers patched.

URL	http://testphp.vulnweb.com/search.php?test=%27
Method	POST
Confidence	Medium
CWE	89
WASC	19
Evidence	You have an error in your SQL syntax

Cross Site Scripting (DOM Based) — High**Description:**

Cross-site Scripting (XSS) is an attack technique that involves injecting and executing malicious code (commonly HTML or JavaScript) in a user's browser within the context of a trusted website. DOM-based XSS occurs when the vulnerability exists in client-side scripts that process data from an untrusted source (e.g., URL or DOM) without proper validation or encoding, allowing attackers to manipulate the page dynamically and execute arbitrary code.

Recommendation / Solution:

1. Use secure frameworks/libraries: Implement frameworks or libraries that automatically handle encoding and separation of data and code (e.g., OWASP ESAPI, Microsoft Anti-XSS).
2. Encode output properly: Apply the correct output encoding based on the context (HTML, attribute, JavaScript, or URL) before rendering untrusted data.
3. Validate input (allow-list): Use strict allow-list validation for all input data — length, type, and allowed characters. Reject or sanitize all other data.
4. Avoid client-side trust: Replicate client-side validation on the server to prevent bypassing.
5. Specify encoding: Always define a consistent character encoding (e.g., UTF-8) for all web pages to prevent interpretation-based vulnerabilities.
6. Set secure cookies: Use HttpOnly, Secure, and SameSite cookie attributes to mitigate session theft via XSS.
7. Avoid dangerous DOM operations: Prevent direct use of innerHTML, document.write(), eval(), or similar functions with untrusted data.
8. Defense-in-depth: Combine validation, encoding, secure cookie flags, and safe DOM handling to minimize XSS exposure and mitigate attacks effectively.

URL	http://testphp.vulnweb.com/cart.php?name=abc#
Method	POST
Confidence	High
CWE	79
WASC	8
Evidence	

Missing Anti-clickjacking Header — Medium**Description:**

The response does not protect against 'ClickJacking' attacks. It should include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options.

Recommendation / Solution:

1. Modern Web browsers support the Content-Security-Policy and X-Frame-Options HTTP headers. Ensure one of them is set on all web pages returned by your site/app.
2. If you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. Alternatively consider implementing Content Security Policy's "frame-ancestors" directive.

URL	http://testphp.vulnweb.com/
Method	GET
Confidence	Medium
CWE	1021
WASC	15
Evidence	

Content Security Policy (CSP) Header Not Set — Medium**Description:**

Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files.

Recommendation / Solution:

1. Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header.

URL	http://testphp.vulnweb.com/
Method	GET
Confidence	High
CWE	693
WASC	15
Evidence	

Absence of Anti-CSRF Tokens — Medium**Description:**

No Anti-CSRF tokens were found in an HTML submission form. A Cross-Site Request Forgery (CSRF) attack forces a victim's browser to send unauthorized requests to a trusted web application where the victim is authenticated. This occurs when forms or URLs use predictable actions without unique, verifiable tokens, allowing attackers to perform actions as the victim.

Recommendation / Solution:

1. Implement anti-CSRF tokens: Use vetted libraries or frameworks (e.g., OWASP CSRFGuard) to automatically generate and validate CSRF tokens.
2. Use unique nonces: Include an unpredictable, per-request or per-session token in each form and verify it upon submission.
3. Avoid unsafe methods: Do not use GET for state-changing actions; use POST or other secure methods with token validation.
4. Combine with XSS prevention: Fix all XSS issues since attacker-controlled scripts can bypass CSRF protections.
5. Reconfirm sensitive actions: Require user confirmation or re-authentication for critical operations.
6. Check request origin (optional): Validate the Referer or Origin header where feasible, but be aware of privacy limitations.
7. Use secure session management: Employ secure frameworks (e.g., ESAPI Session Management) with built-in CSRF protection mechanisms.

URL	http://testphp.vulnweb.com/
Method	GET
Confidence	Low
CWE	352
WASC	9
Evidence	<form action="search.php?test=query" method="post">

Server Leaks Version Information via "Server" HTTP Response Header Field — Low**Description:**

The web/application server is leaking version information via the "Server" HTTP response header. Access to such information may facilitate attackers identifying other vulnerabilities your web/application server is subject to.

Recommendation / Solution:

1. Ensure that your web server, application server, load balancer, etc. is configured to suppress the "Server" header or provide generic details.

Confidence	High
CWE	497
WASC	13
Evidence	nginx/1.19.0

X-Content-Type-Options Header Missing — Low**Description:**

The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.

Recommendation / Solution:

1. Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages.
2. If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.

URL	http://testphp.vulnweb.com/
Method	GET
Confidence	Medium
CWE	693
WASC	15
Evidence	

Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s) — Low**Description:**

The web/application server is leaking information via one or more "X-Powered-By" HTTP response headers. Access to such information may facilitate attackers identifying other frameworks/components your web application is reliant upon and the vulnerabilities such components may be subject to.

Recommendation / Solution:

1. Ensure that your web server, application server, load balancer, etc. is configured to suppress "X-Powered-By" headers.

Confidence	Medium
CWE	497
WASC	13
Evidence	X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1

Charset Mismatch (Header Versus Meta Content-Type Charset) — Info**Description:**

This check identifies responses where the HTTP Content-Type header declares a charset different from the charset defined by the body of the HTML or XML. When there's a charset mismatch between the HTTP header and content body Web browsers can be forced into an undesirable content-sniffing mode to determine the content's correct character set. An attacker could manipulate content on the page to be interpreted in an encoding of their choice. For example, if an attacker can control content at the beginning of the page, they could inject script using UTF-7 encoded text and manipulate some browsers into interpreting that text.

Recommendation / Solution:

1. Force UTF-8 for all text content in both the HTTP header and meta tags in HTML or encoding declarations in XML.

URL	http://testphp.vulnweb.com/
Method	GET
Confidence	Low
CWE	436
WASC	15
Evidence	

Modern Web Application — Info**Description:**

The application appears to be a modern web application. If you need to explore it automatically then the Ajax Spider may well be more effective than the standard one.

Recommendation / Solution:

1. This is an informational alert and so no changes are required.

URL	http://testphp.vulnweb.com/AJAX/index.php
Method	GET
Confidence	Medium
CWE	-1
WASC	-1
Evidence	titles

User Controllable HTML Element Attribute (Potential XSS) — Info**Description:**

This check looks at user-supplied input in query string parameters and POST data to identify where certain HTML attribute values might be controlled. This provides hot-spot detection for XSS (cross-site scripting) that will require further review by a security analyst to determine exploitability.

Recommendation / Solution:

1. Validate all input and sanitize output it before writing to any HTML attributes.

URL	http://testphp.vulnweb.com/search.php?test=query
Method	POST
Confidence	Low
CWE	20
WASC	20
Evidence	

Authentication Request Identified — Info**Description:**

The given request has been identified as an authentication request. The 'Other Info' field contains a set of key=value lines which identify any relevant fields. If the request is in a context which has an Authentication Method set to "Auto-Detect" then this rule will change the authentication to match the request identified.

Recommendation / Solution:

1. This is an informational alert rather than a vulnerability and so there is nothing to fix.

URL	http://testphp.vulnweb.com/secured/newuser.php
Method	POST
Confidence	Low
CWE	-1
WASC	-1
Evidence	upass

Server Version — Info**Description:**

Server banner: nginx/1.19.0

Recommendation / Solution:

1. Ensure server software is up-to-date.

X-Powered-By Header Detected — Info**Description:**

X-Powered-By Header PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1.

Recommendation / Solution:

1. Hide or remove X-Powered-By header to prevent info disclosure.

/: — Medium**Description:**

The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type. See: <https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/>

Recommendation / Solution:

1. Implement missing HTTP security headers such as X-Frame-Options, X-Content-Type-Options, Referrer-Policy, Content-Security-Policy (CSP), and Permissions-Policy.

/: — Medium**Description:**

missing header: strict-transport-security. See: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security>

Recommendation / Solution:

1. Implement missing HTTP security headers such as X-Frame-Options, X-Content-Type-Options, Referrer-Policy, Content-Security-Policy (CSP), and Permissions-Policy.

/: — Medium**Description:**

missing header: content-security-policy. See: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP>

Recommendation / Solution:

1. Implement missing HTTP security headers such as X-Frame-Options, X-Content-Type-Options, Referrer-Policy, Content-Security-Policy (CSP), and Permissions-Policy.

/: — Medium**Description:**

missing header: permissions-policy. See: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Permissions-Policy>

Recommendation / Solution:

1. Implement missing HTTP security headers such as X-Frame-Options, X-Content-Type-Options, Referrer-Policy, Content-Security-Policy (CSP), and Permissions-Policy.

/: — Medium**Description:**

missing header: x-content-type-options. See: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options>

Recommendation / Solution:

1. Implement missing HTTP security headers such as X-Frame-Options, X-Content-Type-Options, Referrer-Policy, Content-Security-Policy (CSP), and Permissions-Policy.

/: — Medium**Description:**

missing header: referrer-policy. See: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referrer-Policy>

Recommendation / Solution:

1. Implement missing HTTP security headers such as X-Frame-Options, X-Content-Type-Options, Referrer-Policy, Content-Security-Policy (CSP), and Permissions-Policy.

Appendix C – Turnitin Result

*% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Disclaimer

Our AI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (it may misidentify writing that is likely AI generated as AI generated and AI paraphrased or likely AI generated and AI paraphrased writing as only AI generated) so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?

The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?

Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.







Page 2 of 157 - Integrity Overview

Submission ID: trncoid::1:3416829075

9% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

156 Not Cited or Quoted: 8%

Matches with neither in-text citation nor quotation marks

11 Missing Quotations: 0%

Matches that are still very similar to source material

41 Missing Citation: 1%

Matches that have quotation marks, but no in-text citation

8 Cited and Quoted: 0%

Matches with in-text citation present, but no quotation marks

Top Sources

5%

Internet sources

3%

Publications

6%

Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

FYP4085

Page 193 of 208



Match Groups

- **156 Not Cited or Quoted: 8%**
Matches with neither in-text citation nor quotation marks
- **11 Missing Quotations: 0%**
Matches that are still very similar to source material
- **41 Missing Citation: 1%**
Matches that have quotation marks, but no in-text citation
- **8 Cited and Quoted: 0%**
Matches with in-text citation present, but no quotation marks

Top Sources

- 5% ■ Internet sources
- 3% ■ Publications
- 6% ■ Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Student papers	
	Kolej Universiti Poly-Tech MARA	2%
2	Student papers	
	Asia Pacific University College of Technology and Innovation (UCTI)	1%
3	Student papers	
	Universiti Poly-Tech Malaysia	<1%
4	Student papers	
	Southern New Hampshire University - Continuing Education	<1%
5	Student papers	
	University of Western Sydney	<1%
6	Internet	
	www.coursehero.com	<1%
7	Student papers	
	University of Greenwich	<1%
8	Student papers	
	Kolej Poly-Tech MARA Kuala Lumpur	<1%
9	Internet	
	umpir.ump.edu.my	<1%
10	Student papers	
	American Public University System	<1%

11	Publication	Kodi A. Cochran. "Cybersecurity Essentials", Springer Science and Business Media ...	<1%
12	Student papers	University of Birmingham	<1%
13	Internet	dione.lib.unipi.gr	<1%
14	Internet	estia.hua.gr	<1%
15	Internet	moldstud.com	<1%
16	Student papers	University of Winchester	<1%
17	Student papers	Noroff University College	<1%
18	Student papers	University of South Australia	<1%
19	Internet	dev.to	<1%
20	Student papers	University of Bradford	<1%
21	Publication	Abhay Bhargav, B. V. Kumar. "Secure Java - For Web Application Development", C...	<1%
22	Student papers	PSB Academy (ACP eSolutions)	<1%
23	Internet	www.cliffsnotes.com	<1%
24	Publication	Anthony P. Ambler, John W. Sheppard. "Realizing Complex System Design", CRC P...	<1%

25	Student papers	Manchester Metropolitan University	<1%
26	Student papers	Hogeschool Utrecht - Tii	<1%
27	Publication	Keshav Kaushik, Mariya Ouassa, Aryan Chaudhary. "Advanced Techniques and A...	<1%
28	Student papers	UNITEC Institute of Technology	<1%
29	Internet	www.archimetric.com	<1%
30	Internet	www.studysmarter.co.uk	<1%
31	Internet	prometteursolutions.com	<1%
32	Publication	Adesina-Ojo, Ayodele Adeola. "Towards the Formalisation of Object-Oriented Met...	<1%
33	Publication	Castro, Ivo Loureiro. "Enhancing Health and Fitness: A Hybrid Recommender Syst...	<1%
34	Internet	www.slideshare.net	<1%
35	Publication	Almeida, Miguel Craveiro Martins de. "Bluetooth Hotspots for Smart Spaces Inter...	<1%
36	Student papers	Adtalem Global Education	<1%
37	Publication	Rafay Baloch. "Web Hacking Arsenal - A Practical Guide to Modern Web Pentestin...	<1%
38	Publication	Truong Duy Dinh, Tran Duc Le, ThiThu Ha Nguyen, Hoang Giang Do. "A Lightweig...	<1%

39	Publication	Wasswa Shafik, Adel Ben Youssef, Chithirai Pon Selvan, Pushan Kumar Dutta. "Su...	<1%
40	Internet	csiac.dtic.mil	<1%
41	Internet	ir.unimas.my	<1%
42	Internet	ukzn-dspace.ukzn.ac.za	<1%
43	Publication	Christoforos Ntantogian, Panagiotis Bountakas, Dimitris Antonaropoulos, Consta...	<1%
44	Publication	Jamuna S. Murthy, G. M. Siddesh, K. G. Srinivasa. "Cloud Security - Concepts, Appli...	<1%
45	Internet	open.uct.ac.za	<1%
46	Internet	silo.tips	<1%
47	Internet	steamcommunity.com	<1%
48	Internet	unsworks.unsw.edu.au	<1%
49	Student papers	Asia Pacific Institute of Information Technology	<1%
50	Student papers	Collin County Community College	<1%
51	Student papers	Dublin City University	<1%
52	Student papers	Massey University	<1%

53	Student papers	TechKnowledge	<1%
54	Student papers	American Intercontinental University Online	<1%
55	Student papers	Federal University of Technology	<1%
56	Student papers	Strathmore University	<1%
57	Student papers	Instituto Superior de Artes, Ciencias y Comunicación IACC	<1%
58	Publication	Rafay Baloch. "Ethical Hacking and Penetration Testing Guide", CRC Press, 2017	<1%
59	Student papers	University of Maryland, University College	<1%
60	Internet	www.geeksforgeeks.org	<1%
61	Internet	www.vpnunlimited.com	<1%
62	Publication	Manoj Kumar, Tanweer Ali, Jaume Anguera, Suman Lata Tripathi. "Emerging Tech...	<1%
63	Student papers	University of Sydney	<1%
64	Student papers	University of Westminster	<1%
65	Internet	technodocbox.com	<1%
66	Internet	www.coursera.org	<1%

81	Internet	www.etd.dbu.edu.et	<1%
82	Internet	www.fynd.academy	<1%
83	Internet	www.itema-conference.com	<1%
84	Internet	6b.digital	<1%
85	Publication	Dan C. Marinescu, Craig Lee. "Process Coordination Ubiquitous Computing", CRC ...	<1%
86	Publication	Mounia Zaydi, Youness Khouidfi, Bouchaib Nassereddine, Justin Zhang. "Agile Se...	<1%
87	Internet	eprints.soas.ac.uk	<1%
88	Internet	lrd.yahooapis.com	<1%
89	Internet	medium.com	<1%
90	Internet	www.diva-portal.org	<1%
91	Internet	www.getambassador.io	<1%
92	Publication	"Decision Support System for Determining the Best Lecturer at Wiraraja Universit...	<1%
93	Publication	"Proceedings of 3rd International Conference on Smart Computing and Cyber Sec...	<1%
94	Publication	"Proceedings of International Conference on Generative AI, Cryptography and Pr...	<1%

95	Internet	"Sixty Percent Conceptual Design Report: Enterprise Accountability System for Cl...	<1%
96	Publication	Anna Wimbauer, Luca Muscariello, Jacques Samain, Lion Steger, Kilian Glas, Max ...	<1%
97	Publication	Gary L. Richardson, Deborah Sater Carstens. "Project Management Tools and Tec...	<1%
98	Publication	Luís, Rafael Rainho. "Urso Verde - Privacy, Security and Management for a Smart ...	<1%
99	Publication	Mudau, Thendo David. "Improving Bulk Water Pump Station Policies and Operati...	<1%
100	Publication	Vinay Aseri, Sumit Kumar Choudhary, Adarsh Kumar. "Cyber Security, Forensics a...	<1%
101	Internet	composable-security.com	<1%
102	Internet	discjockeyparty.com	<1%
103	Internet	dn790009.ca.archive.org	<1%
104	Internet	docplayer.net	<1%
105	Internet	download.bibis.ir	<1%
106	Internet	dspace.cuni.cz	<1%
107	Internet	eprints.ost.ch	<1%
108	Internet	linuxsecurity.com	<1%

turnitin		Page 11 of 157 - Integrity Overview	Submission ID: 13416829075
109	Internet	open-innovation-projects.org	<1%
110	Internet	pentestgodmod.readthedocs.io	<1%
111	Internet	skat.lhmc.us	<1%
112	Internet	www.documents.philips.com	<1%
113	Internet	www.rangeforce.com	<1%
114	Internet	www.testdevlab.com	<1%
115	Publication	Abreu, Leonor Sofia Rosário. "Relatório do Estágio Pedagógico de Educação Física..."	<1%
116	Publication	Kesavan, Krishnan. "A Cognitive Mapping Approach in Real-Time Haptic Renderin..."	<1%
117	Publication	Pedro Henrique Barcha Correia. "Carbono 21: promovendo florestamento utilizan..."	<1%
118	Publication	Sayed Elham Sadat, Mohammed Fahim Naseri, Khosraw Salamzada. "Identifying ..."	<1%
119	Publication	Yassine Maleh. "Web Application PenTesting - A Comprehensive Guide for Profess..."	<1%

Appendix D – Logbook

CT206 / BACHELOR OF INFORMATION TECHNOLOGY in CYBER SECURITY (HONOURS)



FACULTY OF COMPUTING & MULTIMEDIA (FCOM)

FINAL YEAR PROJECT O1

FYP 4074

LOG BOOK

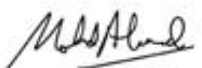
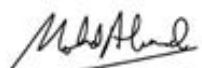
STUDENT'S NAME : NUR IZZAH AKMAL BINTI MOHD RESTU

ID NO. : AM2311015271

SUPERVISOR : EN. MOHD AKMAL BIN MOHD AZMER

PROJECT TITLE : WEB VULNERABILITY SCANNER

CT206 / BACHELOR OF INFORMATION TECHNOLOGY in CYBER SECURITY (HONOURS)

Week		Agenda	Next Agenda	Signature (Supervisor / Coordinator)
19/5/25 – 23/5/25	1	Brainstorming, Introduction to the course, Title selection and find Supervisor.	Consultation on title project, technology stack and purpose of the project at supervisor	
26/5/25- 30/5/25	2	Consultation on title project, technology stack and purpose of the project at supervisor	Register title on FYPMS, Find Client for the project and Make Project Proposal	
2/6/25- 6/2/25	3	Register title on FYPMS, Find Client for the project and Make Project Proposal	Submit Project Proposal at coordinator , supervisor and FYPMS and Start doing project report	
9/6/25- 13/6/25	4	Submit Project Proposal at coordinator , supervisor and FYPMS and Start doing project report	Submission of project report chapter 1 to coordinator and supervisor, Continue chapter 3,4 and make a questionnaire using Google forms and finalize to user,	
16/6/25- 20/6/25	5	Submission of project report chapter 1 to coordinator and supervisor, Continue chapter 3,4 and make a questionnaire using Google forms and finalize to user,	Submission chapter 2 at coordinator, Collect data and do analysis from questionnaire, and continue doing chapter 4 and 5.	
23/6/25 – 27/6/25	6	Submission chapter 2 at coordinator, Collect data and do analysis from questionnaire, and continue doing chapter 4 and 5	Make slide presentation for presentation with supervisor and examiner on 4/7/25, Contact examiner and supervisor to set date to present	
30/6/25 – 9/7/25	7	Make slide presentation for presentation with supervisor and examiner on 4/7/25, Contact examiner and supervisor to set date to present	Submit Project Report and Logbook to examiner, coordinator and supervisor via email and classroom	

CT206 / BACHELOR OF Information Technology (Honours) in Cyber Security



FACULTY OF COMPUTING & MULTIMEDIA (FCOM)

FINAL YEAR PROJECT 2

FYP4085

LOG BOOK

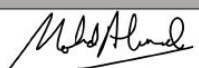
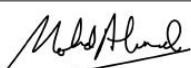
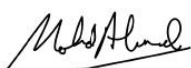
STUDENT'S NAME : NUR IZZAH AKMAL BINTI MOHD RESTU

ID NO. : AM2311015271

SUPERVISOR : MOHD AKMAL BIN MOHD AZMER

PROJECT TITLE : WEB VULNERABILITY SCANNER

CT206 / BACHELOR OF Information Technology (Honours) in Cyber Security

Week		Agenda	Next Agenda	Signature (Supervisor / Coordinator)
5 August 2025	1	API and Scanning tool integration in system	Google Meet with supervisor to update project progress	
14 August 2025	2	Google Meet with supervisor to update project progress	Security Development progress	
21 August 2025	3	Security Development progress	Submission chapter 6 report	
29 August 2025	4	Submission chapter 6 report	Frontend Development progress	
5 September 2025	5	Frontend Development progress	Progress Report chapter 7	
9 September 2025	6	Progress Report chapter 7	Configure nikto scanning result error that not show at frontend system	
23 September 2025	7	Configure nikto scanning result error that not show at frontend system	Submission chapter 7 report	
29 September 2025	8	Submission chapter 7 report	Review error in system and configure the error in system	
7 October 2025	9	Review error in system and configure the error in system	Discussion with supervisor about scanning time of system	
14 October 2025	10	Discussion with supervisor about scanning time of system	Meet with supervisor to show the system and sign acknowledgement letter	
21 October 2025	11	Meet with supervisor to show the system and sign acknowledgement letter	Presentation with client and conduct questionnaire for feedback user	
30 October 2025	12	Presentation with client and conduct questionnaire for feedback user and blast it	Meet with Supervisor to review the poster before presentation FYP 2	
5 November 2025	13	Meet with Supervisor to review the poster before presentation FYP 2	Presentation FYP 2	
13 November 2025	14	Presentation FYP 2	Submission Report, Slide, Poster and Rubric to supervisor and examiner	

References

- Abrams, L. (2025, September 25). Kali Linux 2025.3 released with 10 new tools, Wi-Fi enhancements. BleepingComputer. <https://www.bleepingcomputer.com/news/security/kali-linux-20253-released-with-10-new-tools-wifi-enhancements/>
- Advisory. (2025). *MyCERT Advisory - Security Best practices in protecting against ransomware threats*. CyberSecurity Malaysia. <https://mycert.org.my/portal/advisory?id=MA-1337.062025>
- Author, V., & Author, V. (2024, January 16). *Burp Suite, the tool dedicated to web application security*. VAADATA - Ethical Hacking Services. <https://www.vaadata.com/blog/introduction-to-burp-suite-the-tool-dedicated-to-web-application-security/>
- Beschokov, M. (2025, June 16). *OWASP ZAP - ZEd Attack Proxy*. Wallarm. <https://www.wallarm.com/what/owasp-zap-zed-attack-proxy>
- Canonical. (2019c, April 4). *4th April 2019, London, UK – As of today, Microsoft Visual Studio Code is available for Linux as a snap, providing seamless auto-updates for its users. Visual Studio Code, a free, lightweight code editor, has redefined editors for building modern web and cloud applications, with built-in support for debugging, task running, and version c [. . .]*. Canonical. <https://canonical.com/blog/visual-studio-code-launches-as-a-snap>
- Career Technology Cyber Security India Pvt. Ltd. (2023, November 20). *Wapiti: Website Scanning Tool - Career Technology Cyber Security India Pvt. Ltd. - Medium*. Medium. <https://medium.com/@careertechnologymiraroad/wapiti-website-scanning-tool-7c8bd8255e68>
- Codecademy. (n.d.). *What is SQLite?* Codecademy. <https://www.codecademy.com/article/what-is-sqlite>
- Cyber Security Asean. (2024c, October 30). *Kaspersky reveals cyber threats growing in Malaysia amidst data centre expansion*. <https://cybersecurityasia.net/kaspersky-growing-cyber-threats-in-malaysia/>
- D, B. (2024, February 25). *Why is Role-based Access Control [RBAC] Important for Cybersecurity?* Neumetric. <https://www.neumetric.com/role-based-access-control-rbac-for-cybersecurity/>
- File:Python logo and wordmark.svg - Wikimedia Commons. (2008, August 6). https://commons.wikimedia.org/wiki/File:Python_logo_and_wordmark.svg
- File:CSS3 logo and wordmark.svg - Wikimedia Commons. (2016, May 30). https://commons.wikimedia.org/wiki/File:CSS3_logo_and_wordmark.svg
- GeeksforGeeks. (2025, May 28). *Functional vs. Non Functional Requirements*. GeeksforGeeks. <https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/>
- Chu, D. (2024b, August 12). *What are the common methods for data gathering?* Secoda. <https://www.secoda.co/learn/what-are-the-common-methods-for-data-gathering>
- GeeksforGeeks. (2025a, April 8). *Flask Tutorial*. GeeksforGeeks. <https://www.geeksforgeeks.org/python/flask-tutorial/>
- GeeksforGeeks. (2025a, February 21). *What is Burp Suite?* GeeksforGeeks. <https://www.geeksforgeeks.org/ethical-hacking/what-is-burp-suite/>

Gomez, B. (2024, July 5). *Protect your websites with Wapiti: a Comprehensive Security Audit*. Linux Security. <https://linuxsecurity.com/features/complete-guide-to-using-wapiti-web-vulnerability-scanner-to-keep-your-web-applications-websites-secure>

Itema. (n.d.). *USE OF "OWASP TOP 10" IN WEB APPLICATION SECURITY - ITEMA 2025*. ITEMA 2025. <https://www.itema-conference.com/itema-2020-25/>

Iqbal, M. A. (2025, February 11). Flask: A Comprehensive Guide. DEV Community. <https://dev.to/atifwattoo/flask-a-comprehensive-guide-19mm>

Kalkman, P. (2023, March 7). Continuously pentesting with the OWASP ZAP API and Python. *Medium*. <https://python.plainenglish.io/continuously-pentesting-with-the-owasp-zap-api-and-python-fd66b574aca>

Klie, F. R. (2025, November 9). A Comprehensive Guide to SQLite Database: Lightweight, powerful, and easy to use. DEV Community. <https://dev.to/farhadrahimiklie/a-comprehensive-guide-to-sqlite-database-lightweight-powerful-and-easy-to-use-21po>

K, P. (2025b, May 24). OWASP ZAP Tutorial: A Comprehensive guide for DevSecOPs - DevSecOPS School. DevSecOps School. <https://devsecopsschool.com/blog/owasp-zap-tutorial-a-comprehensive-guide-for-devsecops/> Amal Tom Parakkaden. (2023, May 23). Nikto: A Comprehensive Web Vulnerability Scanner - Amal Tom Parakkaden - Medium. Medium; Medium. <https://medium.com/@amaltomparakkaden/nikto-a-comprehensive-web-vulnerability-scanner-bb9f0ec105e4>

Kovalenko, O. (2025, January 6). Common Web Application Vulnerabilities in 2025 | Iterasec. *Your pragmatic cybersecurity partner*. <https://iterasec.com/blog/common-web-application-vulnerabilities/>

Lebbos, G., PhD. (2023, October 29). The Evolution of JavaScript: A Journey from ES1 to the Latest Version (Part 1). <https://www.linkedin.com/pulse/evolution-javascript-journey-from-es1-latest-version-part-lebbos-za9fe>

Marler, K. (2025, May 27). The Best Practices for using OWASP ZAP in Web API Security. *StackHawk, Inc.* <https://www.stackhawk.com/blog/guide-to-zap-application-security-testing/>

Okoli, V. (2023, October 30). Docker: Explained Simply for a 10-Year-Old — The Magic Box for Computer Programs. Medium; AWS in Plain English. <https://aws.plainenglish.io/docker-explained-simply-for-a-10-year-old-the-magic-box-for-computer-programs-94452b930d6b>

OWASP Zed Attack Proxy Scan - Visual Studio Marketplace. (n.d.). <https://marketplace.visualstudio.com/items?itemName=kasunkodagoda.owasp-zap-scan>

Protected, E. (2025, February 3). Comprehensive Guide to Creating Use Case Diagrams with Visual Paradigm - ArchiMetric. ArchiMetric. <https://www.archimetric.com/comprehensive-guide-to-creating-use-case-diagrams-with-visual-paradigm/>

SentinelOne. (2025, May 19). *7 types of SQL injection attacks & How to prevent them?* SentinelOne. <https://www.sentinelone.com/cybersecurity-101/cybersecurity/types-of-sql-injection/>

Sillam, Y. (2023, December 21). *What is SQL Injection | SQLI Attack Example & Prevention Methods | Imperva*. Learning Center. <https://www.imperva.com/learn/application-security/sql-injection-sqli/>

Staff, C. (2025, May 19). *Visual Studio Code: A guide to Microsoft's free code editor*. Coursera. <https://www.coursera.org/articles/visual-studio-code>

Sutherland, S. (2025, February 20). *Hacking SQL Server Stored Procedures – Part 3: SQL Injection*. NetSPI. <https://www.netspi.com/blog/technical-blog/network-pentesting/hacking-sql-server-stored-procedures-part-3-sqli-and-user-impersonation/>

SQL Injection Prevention - OWASP Cheat Sheet Series. (n.d.). https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

SQLite Tutorial - An easy way to master SQLite fast. (2024, May 5). SQLite Tutorial. <https://www.sqlitetutorial.net/>

Types of SQL injection? (2025, March 5). Acunetix. <https://www.acunetix.com/websitesecurity/sql-injection2/>

Waf, N. (2021, December 15). Wapiti – free web-application vulnerability scanner. *Medium*. <https://nemesida-waf.medium.com/wapiti-free-web-application-vulnerability-scanner-ce7712adf644>

Wikipedia contributors. (2025, November 11). HTML. Wikipedia. <https://en.wikipedia.org/wiki/HTML>

What is SQL Injection (SQLi) and How to Prevent Attacks. (2025, February 17). Acunetix. <https://www.acunetix.com/websitesecurity/sql-injection/>

What OWASP ZAP can do, and when to use it. (n.d.). Jit. <https://www.jit.io/resources/owasp-zap>
Wapiti: a Free and Open-Source web-application vulnerability scanner in Python. (n.d.). <https://wapiti-scanner.github.io/>

What is cross-site scripting (XSS) and how to prevent it? | *Web Security Academy*. (n.d.). <https://portswigger.net/web-security/cross-site-scripting>

What is Cross-site Scripting (XSS): prevention and fixes. (2025, February 18). Acunetix. <https://www.acunetix.com/websitesecurity/cross-site-scripting/>

What is Cross site scripting (XSS) and how does it work? | *Black Duck*. (n.d.). <https://www.blackduck.com/glossary/what-is-cross-site-scripting.html>

What is Use Case Diagram? (n.d.). <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/>

Zed Attack Proxy (ZAP) Daniel W -OWASP Chapter Lead. (n.d.). <https://owasp.org/www-chapter-dorset/assets/presentations/2020-01/20200120-OWASPDorset-ZAP-DanielW.pdf>

GeeksforGeeks. (2024, July 4). *User interface design*. GeeksforGeeks. <https://www.geeksforgeeks.org/software-engineering/software-engineering-user-interface-design/>

GeeksforGeeks. (2025, July 23). *Database design in DBMS*. GeeksforGeeks. <https://www.geeksforgeeks.org/dbms/database-design-in-dbms/>