International Journal of Safety and Security Engineering
Vol. 15, No. 12, December, 2025, pp. 2483-2495
Journal homepage: http://iieta.org/journals/ijsse

Security Engineering of Open Web Application Security Project-Based Automated System for Web Vulnerability Detection and Mitigation

Irawan Afrianto[1*], Ismayani Setyaningrum[1], Sufa Atin[1], Eddy Prasetyo Nugroho[2], Girindro Pringgo Digdo[3]

[1] Department of Informatics Engineering, Universitas Komputer Indonesia, Bandung 40132, Indonesia
[2] Department of Computer Science, Universitas Pendidikan Indonesia, Bandung 40154, Indonesia
[3] CyberArmyID – PT Global Inovasi Siber Indonesia, Bandung 40164, Indonesia

Corresponding Author Email: irawan.afrianto@email.unikom.ac.id

https://doi.org/10.18280/ijsse.151206

Received: 11 October 2025
Revised: 9 December 2025
Accepted: 21 December 2025
Available online: 31 December 2025

Keywords:
automated monitoring, Model-View-Controller architecture, Open Web Application Security Project Zed Attack Proxy Application Programming Interface, security engineering, vulnerability assessment, vulnerability mitigation, web application security

ABSTRACT

Web application vulnerabilities present escalating security risks to organizational data integrity, particularly within public sector infrastructure, where legacy systems often lack robust defense mechanisms. To address this challenge, this study engineers an automated web vulnerability monitoring system that seamlessly integrates the Open Web Application Security Project Zed Attack Proxy Application Programming Interface (OWASP ZAP API) within a Model-View-Controller (MVC) architecture. Adopting an Iterative Development Life Cycle (IDLC), the system was rigorously validated through Black-box, White-box, and User Acceptance Testing (UAT) to ensure functional reliability. The system's practical efficacy was evaluated via a comparative analysis against the standard OWASP ZAP desktop interface using "Laplakgar2022," an active government performance reporting application, as a real-world test subject. The empirical results demonstrate that the proposed system achieves a 20% reduction in scanning duration by optimizing scanning policies, while simultaneously maintaining a 100% detection rate for critical High and Medium-risk vulnerabilities, specifically Structure Query Language (SQL) Injection and Cross-Site Scripting (XSS). Furthermore, UAT results indicate a high usability score of approximately 80%, confirming that the system effectively mitigates alert fatigue and assists development teams in prioritizing remediation. Future research will focus on integrating Artificial Intelligence (AI) for predictive threat modeling and expanding validation to distributed cloud environments.

1. INTRODUCTION

Along with the development of systems and information, as well as the use of websites in Indonesia, cyber threats continue to evolve and increase [1]. One of the facts obtained based on data from the Indonesian Cyber Security Landscape in 2022 published by Indonesia National Cyber and Crypto Agency (BSSN), suggests that there are the Top 5 vulnerabilities with critical levels that can have a significant impact on the security of an application; this can have a detrimental impact, such as the creation of backdoors, malicious file insertion, and data leakage [2]. The Top 5 vulnerabilities analyzed by Information Technology Security Assessment (ITSA) throughout 2022 are Insecure Direct Object Reference (IDOR), Structure Query Language (SQL) Injection, Privilege Escalation, Broken Access Control, and Cross-Site Scripting (XSS) [3, 4].

Research in 2020 revealed that security is often still a secondary priority for many system and network developers. To address this, previous studies have proposed various testing tools. For instance, the Bangkolo application [5] utilized the ISAF framework with a Hybrid Apps approach for security testing, while Ayeni et al. [6] designed an application specifically targeting Phishing, XSS, and SQL Injection vulnerabilities. However, these existing solutions primarily address the detection aspect in isolation or within limited scopes. They often lack a fully integrated, automated workflow that seamlessly couples' vulnerability scanning with immediate, actionable mitigation strategies for a broad spectrum of web threats. This research bridges this gap by engineering a comprehensive security monitoring system based on the Open Web Application Security Project Zed Attack Proxy Application Programming Interface (OWASP ZAP API). Unlike previous tools, the proposed application not only automates the recognition and scanning phases but also generates detailed reports with specific code-level remediation recommendations.

Recent developments in automated vulnerability detection have increasingly focused on enhancing scanning efficiency and accuracy. Several comparative studies have evaluated the performance of open-source and commercial scanners. For instance, Hafez and Almustafa [7] examine an automated system capable of detecting and mitigating SQL injection and XSS attacks. Abualese and Al-Rousan [8] conducted a comparative analysis of six web vulnerability scanners, highlighting that while tools like OWASP ZAP are highly effective for general detection, they often differ significantly

in scan duration and false positive rates compared to commercial counterparts like Burp Suite. Similarly, Sharma and Pahuja [9] emphasized the importance of choosing the right open-source tools for specific network environments to minimize detection latency. Beyond traditional scanning, recent research has explored integrating Artificial Intelligence (AI) to improve detection capabilities. Moreira et al. [10] proposed a hybrid framework combining dynamic scanning with machine learning to identify risks in real-time, addressing the limitations of static rule-based engines. Furthermore, surveys by Yang et al. [11] and Yitagesu et al. [12] indicate a growing trend towards Deep Learning-aided detection systems, which aim to automate the identification of complex code vulnerabilities that manual audits might miss.

By streamlining the process from detection to mitigation, this system supports the ultimate goal of security gap analysis: to effectively detect threats that compromise the confidentiality, integrity, and availability of data [13, 14].

OWASP is a leading organization that focuses on the security of web-based applications, where OWASP itself has reliable documentation, tools, and methods to support security gap testing [15]. Web application developers often use the OWASP method to improve and determine a web-based application's security level [16]. The OWASP method is open source, comprehensive, highly reliable, and easy to understand [17].

The security monitoring application is built using the API provided by OWASP, namely the ZAP API [18], to support website application vulnerability testing by performing automatic vulnerability scanning. The security monitoring application can also provide scan results based on the name of the vulnerability and risk level. It is also a repair solution with references and classifications based on the OWASP Top 10. This can be used as awareness and repair recommendations by the standards set by OWASP [19, 20].

Based on this introduction, there are research questions (RQs) that will be resolved in this research activity, namely:

1. RQ1: What are the most common vulnerabilities in web applications tested using OWASP-based security monitoring applications?
2. RQ2: How practical is the developed security monitoring application in detecting vulnerabilities compared to other vulnerability scanning tools, such as OWASP ZAP?
3. RQ3: How can the user interface of a security monitoring application affect the user experience in performing vulnerability testing?

From the RQs, there are several research objectives (ROs) that can be obtained, namely:

1. RO1: Identify and analyze common types of vulnerabilities in web applications through testing using OWASP-based security monitoring applications.
2. RO2: Evaluate the effectiveness of the developed security monitoring application in detecting vulnerabilities compared to existing vulnerability scanning tools.
3. RO3: Analyze the impact of the user interface on user experience in performing vulnerability testing.

This development is based on widely recognized security principles, such as the Confidentiality, Integrity, Availability (CIA TRIAD), Vulnerability Assessment and Penetration Testing (VAPT), and Pretty Good Privacy (PGP), which are designed to meet increasingly complex information security needs [21-24]. Based on the objectives of this research, this study aims to develop a web-based application integrated with the OWASP ZAP API to streamline the scanning and reporting process. This research represents a strategic, practical contribution to the field of security engineering by establishing a systematic and automated approach to web vulnerability management. Through the development of this system, the study provides an actionable methodology for the effective detection and mitigation of cyber threats in web applications, directly addressing the technical and procedural requirements of modern cybersecurity. The following sections will discuss the methodology, system design, and evaluation of the proposed system.

2. RESEARCH METHODOLOGY

The research methodology adopts a comprehensive Iterative Development Life Cycle (IDLC) approach [25], integrated with a specific strategy for automated security testing. As illustrated in Figure 1, the research framework is structured into six sequential phases: Planning and Analysis, System Design, Implementation and Integration, Reporting and Classification, Testing and Evaluation, and Deployment. This structure ensures a continuous feedback loop for refining the system's accuracy and performance.

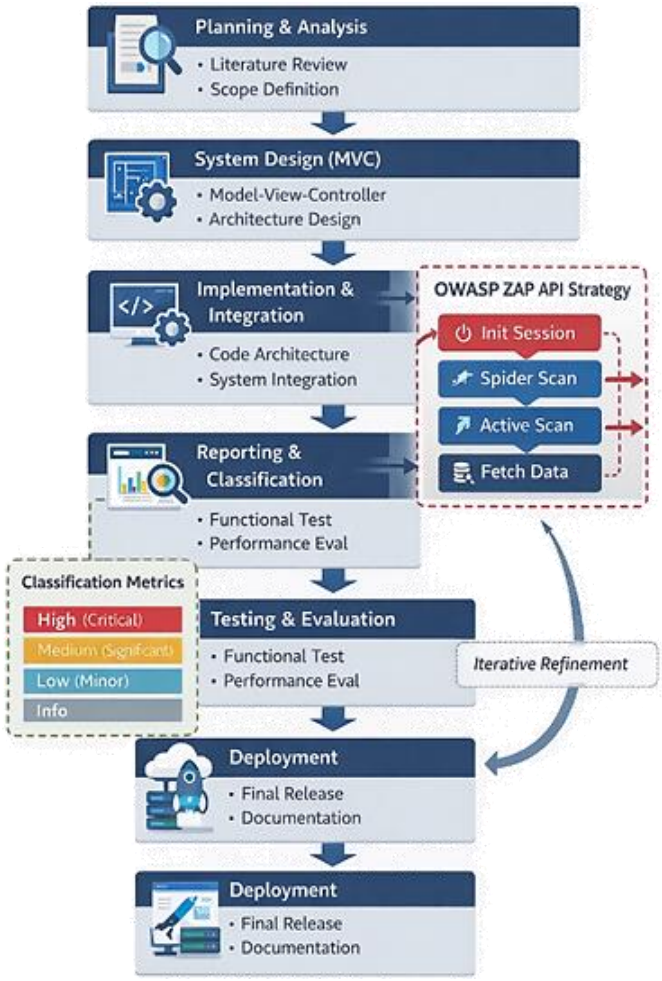

Figure 1. Flow of research

2.1 Planning and requirement analysis

The initial phase focuses on defining the scope of security

monitoring. The functional requirements were established to target the OWASP Top 10 vulnerabilities, which serve as the global standard for web application security. During this phase, the system specifications were defined to include automated crawling, active scanning, and the generation of remediation-focused reports.

2.2 System design (Model-View-Controller architecture)

The system architecture is designed using the Model-View-Controller (MVC) pattern to separate the internal representation of information from the way information is presented to and accepted from the user.

a. Model: Manages the data structure for scan results and user configurations.
b. View: Developed using Jinja2 templates within the Flask framework to provide a user-friendly dashboard for monitoring scan progress and viewing reports.
c. Controller: Built using Python, acting as the intermediary that processes user requests and interacts with the external OWASP ZAP engine.

2.3 Implementation and Open Web Application Security Project Zed Attack Proxy Application Programming Interface integration

This phase constitutes the core technical development. The system integrates the OWASP ZAP in daemon mode via its REST API, utilizing the python-owasp-zap-v2.4 library. The integration strategy follows a strict automated workflow:

a. Session initialization: The system establishes a connection to the ZAP daemon using a unique API key to ensure authorized access.
b. Spidering (crawling): The zap.spider.scan endpoint is triggered to index the target URL. The recursion depth is configured to 5 levels to balance thoroughness with scanning speed.
c. Active scanning: Upon completion of the spidering process, the system initiates the active scan module (zap.ascan.scan). This module injects specific payloads to simulate attacks (e.g., SQL Injection, XSS) against the identified endpoints.
d. Data retrieval: The system polls the API for status updates and retrieves the final alerts using zap.core.alerts in JSON format for processing.

2.4 Reporting and vulnerability classification metrics

The raw data obtained from the ZAP API is parsed and mapped into actionable insights. To ensure clarity, the system classifies vulnerabilities based on the OWASP Risk Rating Methodology [26]. The classification metrics are defined as follows:

a. High risk: Critical vulnerabilities (e.g., SQL Injection, Remote Code Execution) that could lead to system compromise.
b. Medium risk: Significant issues (e.g., XSS, Sensitive Data Exposure) that allow unauthorized access or data modification.
c. Low risk: Minor configuration errors (e.g., Missing Security Headers) with limited impact.
d. Informational: Observations regarding best practices (e.g., Information Leakage in comments).

2.5 Testing and evaluation

The system undergoes rigorous testing to validate its functionality, internal logic, and performance.

a. Black box testing: Conducted to validate the functional requirements of the system, ensuring that all modules (login, history, scanning, reporting) operate as intended from an end-user perspective.
b. White box testing: Performed to examine the internal logic and code structure of the application. This testing focuses on verifying the execution paths, conditions, and loops within the Python/Flask controller to ensure the code operates efficiently and handles errors correctly.
c. Comparative evaluation: The system's detection capabilities are benchmarked against the standard OWASP ZAP GUI. Key performance metrics include scan duration (efficiency) and detection rate (accuracy).

3. RESULTS AND DISCUSSION

The security monitoring application aims to support users in testing web application security gaps. This application was built using the API provided by OWASP, namely the OWASP ZAP API; the security monitoring application is built with a website that aims to make it easier for users to access the application in real-time.

3.1 System overview

The system is developed as shown in Figure 2. Users can conduct security gap testing by inputting target addresses such as URLs or IP addresses. Next, the application will perform an identification (scanning) process based on the OWASP standardization. After the identification and testing process, the application will provide identification results that the user can obtain to configure security for the web application. The identification results or scan results obtained will be displayed based on the name of the vulnerability, the level of risk, and the recommended solution, which is equipped with references and classifications based on the OWASP Top 10, which can be used as user awareness of the security system.

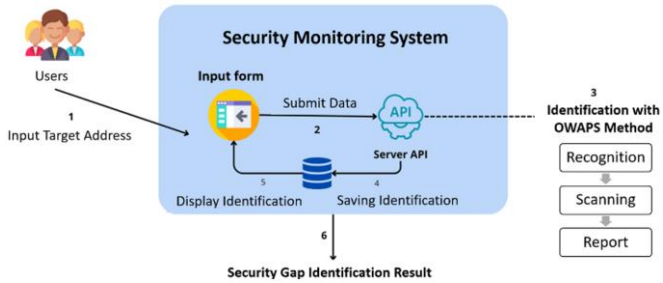

Figure 2. The system description

3.2 System architecture

The system architecture in the security monitoring application aims to provide an overview of the application framework. The details of the system architecture can be seen in Figure 3. The system was built with the Python language and the Flask framework, a microframework that is the

primary basis for making the system. OWASP standardization will be implemented in the security gap identification process using the Zap Python client. This library can run functions on the OWASP ZAP API. OWASP ZAP API provides functions that can be used to perform the identification process and vulnerability testing as provided by ZAP, which can support vulnerability testing of web-based applications, which consists of recognition, scanning, and report generation phases [27].

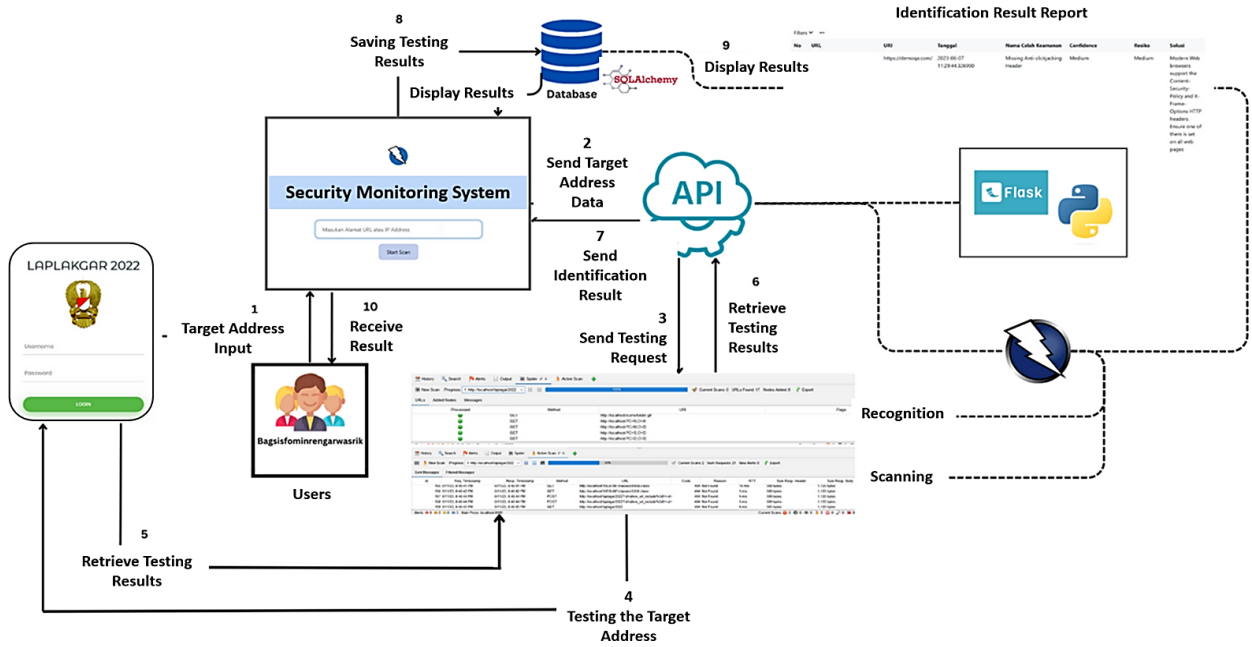

Figure 3. The system architecture

3.3 Technology architecture

The technology architecture aims to provide an overview of the technology used to support application development. The details of the technology architecture can be seen in Figure 4. On the front side, it functions to display the features that connect users with the system. The front end is built using the Jinja framework and Bootstrap. Meanwhile, the backend functions to process the recognition phase, scanning, and report generation. The backend uses the Flask framework, SQLAlchemy database, and ZAP Python library. The front and back ends are connected to the OWASP ZAP API.

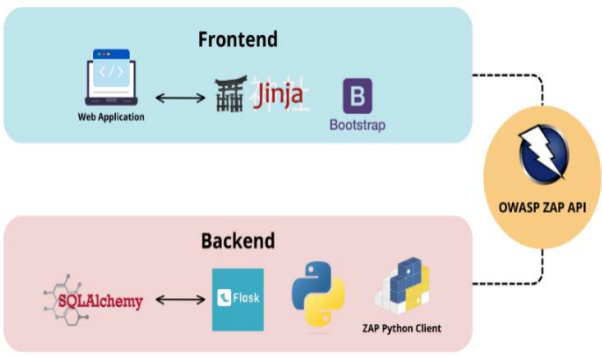

Figure 4. The technology architecture

3.4 The Open Web Application Security Project analysis

The application will perform scans that run according to the OWASP method: recognition, scanning, and reporting.

3.4.1 Recognition phase subsystem

The recognition phase subsystem functions to collect information and identify potential vulnerabilities in the tested application. The detailed stages of the recognition process can be seen in Figure 5.

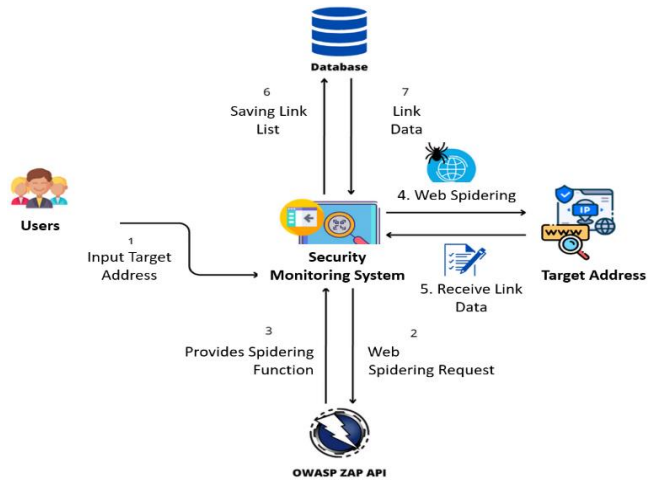

Figure 5. Recognition phase subsystem

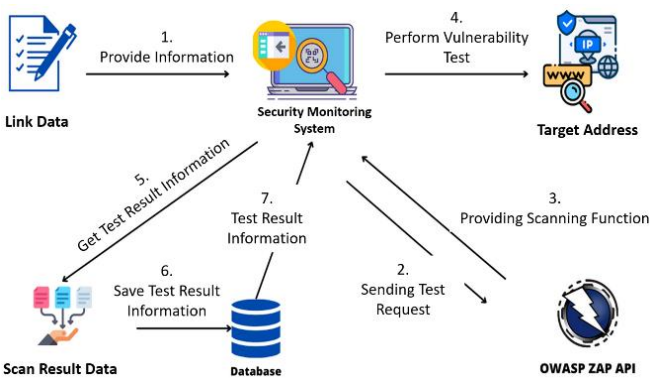

Figure 6. Scanning phase subsystem

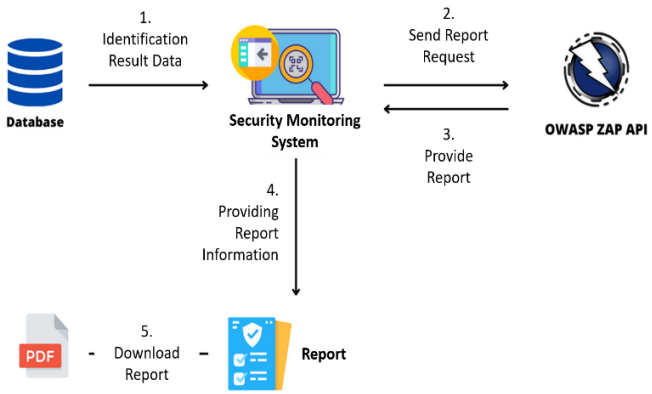

Figure 7. Report phase subsystem

3.4.2 Scan phase subsystem

The scanning phase subsystem serves to scan the target application to identify vulnerabilities. This phase provides knowledge related to the weaknesses that exist in the application. The scanning phase subsystem can be seen in Figure 6.

3.4.3 Report phase subsystem

The report subsystem is the last phase in the system vulnerability identification process. This phase serves to present the results of security testing that has been carried out in the previous phase. The report presented contains vulnerability findings, risk analysis, and solutions or recommendations for improvement that are clearer and more structured. The report phase subsystem is described in Figure 7.

3.5 The system modelling

The Use case diagrams mainly explain activities or communication in security monitoring applications, where there are nine activities and four actors in the application being built. As for the details, it can be seen in Figure 8.

A Class diagram mainly defines the relationship between classes, attributes, and methods supporting security monitoring applications' development. The details of the class diagram in the security monitoring application are described in Figure 9.

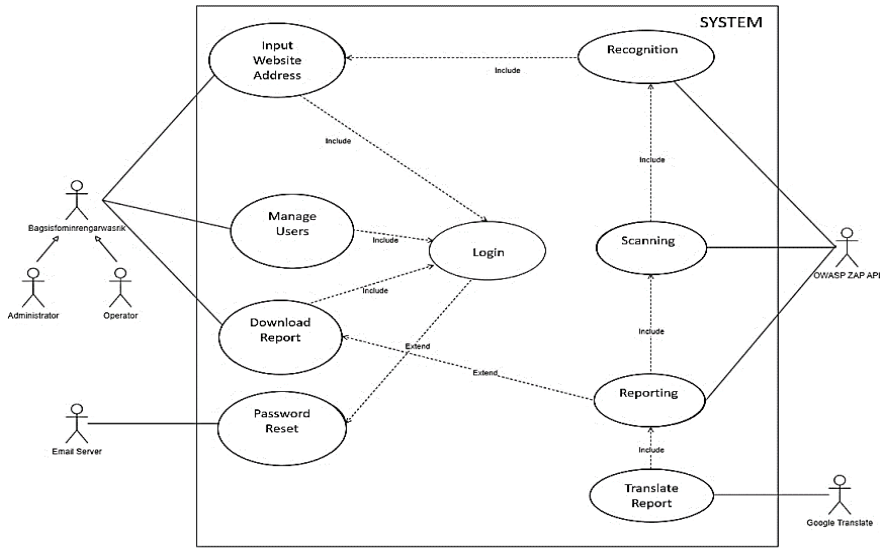

Figure 8. Use case modelling

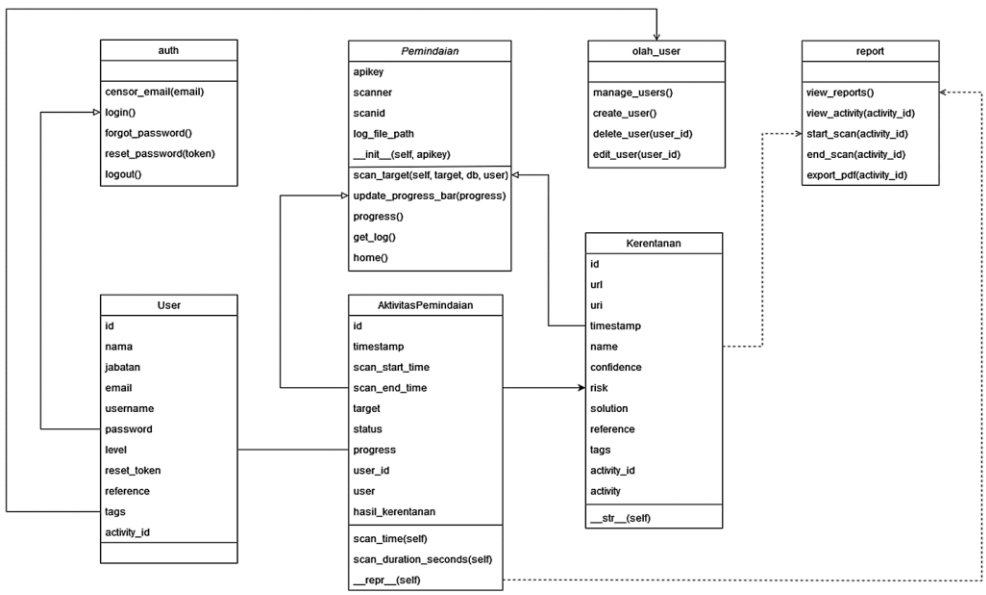

Figure 9. Class diagram modelling

3.6 The system interface

The results of the interface implementation of the security monitoring application with OWASP standardization in the recognition subsystem section can be seen in Figure 10, which shows that in this process, the security monitoring application will collect relevant information to be used in the following process.

Figure 11 shows the scanning process, wherein the security monitoring application will perform a vulnerability testing process on the target address. Figure 12 shows the scan results, with details of the scan date, target address, status, and action details.

Figure 13 shows the scan results, with activity details, target address details, vulnerability names, risks, solutions, and references that users can use to improve the application security system.

Figure 10. Interface of reconnaissance Open Web Application Security Project (OWASP)

Figure 11. Interface of scanning the Open Web Application Security Project (OWASP)

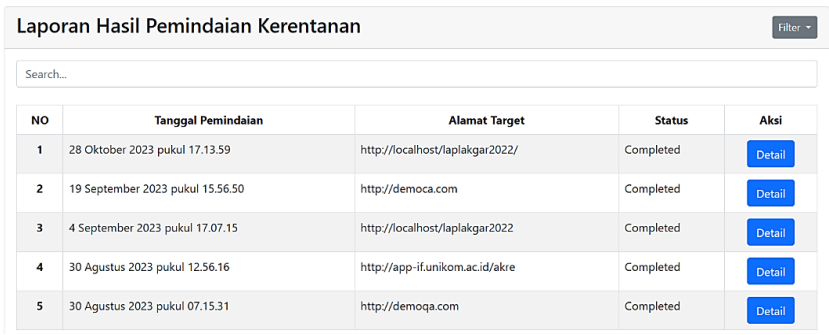

Figure 12. Interface of the scan result

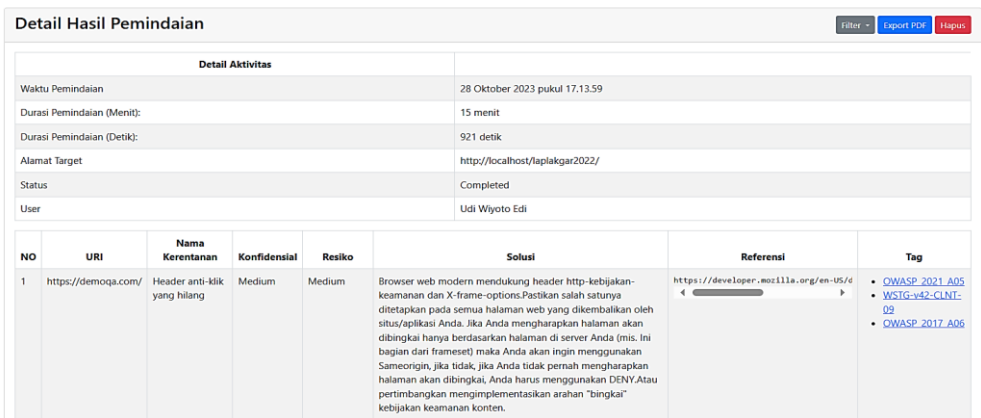

Figure 13. Interface of vulnerability detailed result

3.7 System testing

The testing is done by testing each security monitoring application system function. This test finds errors and determines whether the application developed has met the criteria and follows the objectives and needs of the software [28]. This test is divided into API testing, performance testing, whitebox testing, blackbox testing, and vulnerability finding comparison testing.

3.7.1 The Application Programming Interface's testing

This test aims to test the API functions implemented in the application. API testing aims to determine whether the system functions can provide responses and requests properly [29]. This test consists of testing the recognition and scanning (Figure 14) using Postman tools. This test aims to test the recognition function, which retrieves data or information on the target website. Based on the test results, the response received after running the recognition function endpoint was successfully executed, and the recognition results were obtained, as shown in Figure 15; thus, testing the recognition function was declared victorious.

3.7.2 The performance testing

Performance testing of security monitoring applications is carried out using the Postman tool. This performance test is carried out to evaluate application performance under high load to identify limitations, weak points, and response times required by applications or services when faced with realistic load conditions [30] (Figure 16).

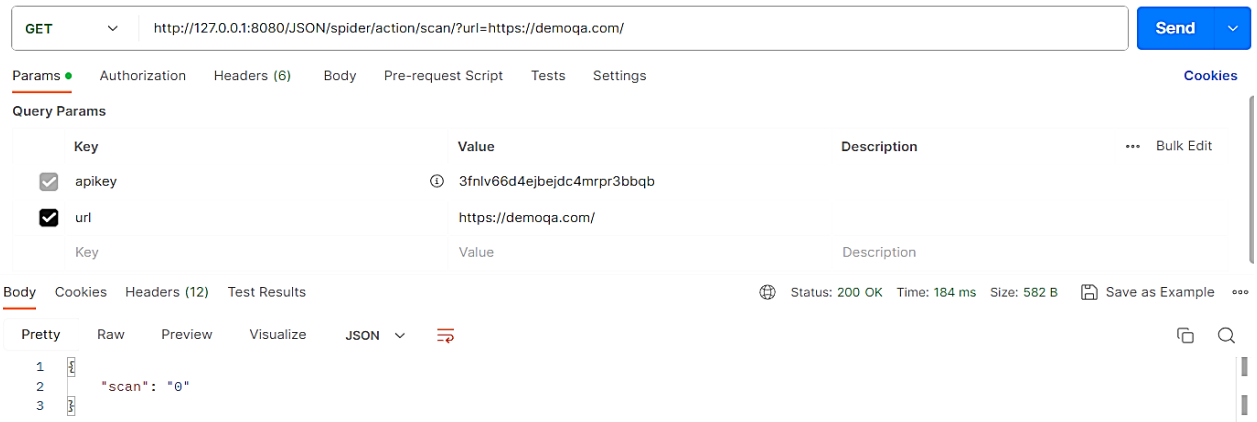

Figure 14. Recognition and scanning Application Programming Interface (API) testing

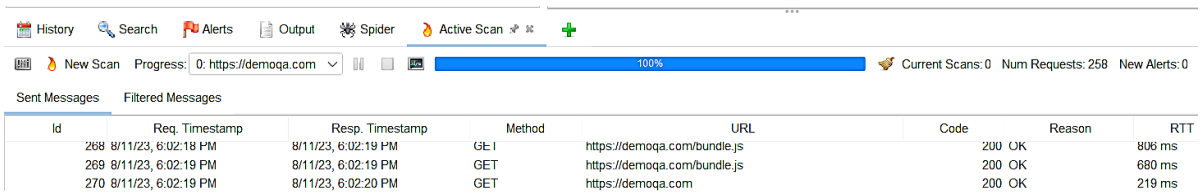

Figure 15. Result of Application Programming Interface (API) testing

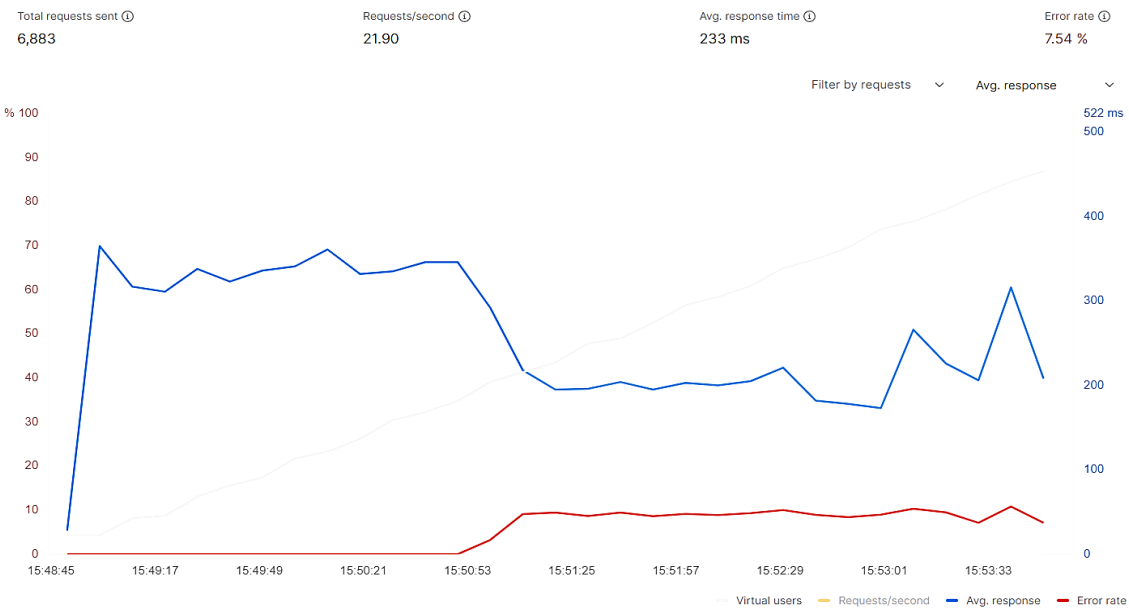

Figure 16. Performance testing

Based on the test results, by requesting as many as 20 virtual users within 5 minutes, the data obtained shows that the performance test of the security monitoring application shows promising results. It can be seen in Figure 16 that the results of the number of requests, average response time, maximum and minimum response time, request success, and error response show good performance values and have no errors. However, for the request to forget the password, there is an 83.04% error; this shows that the SMTP server cannot handle many requests at the same time. This is a natural thing because the request to forget the password should not be done more than once at a time.

3.7.3 The whitebox testing

Whitebox testing focuses on observing program code, which is then analyzed to determine possible errors. This test is known as a structural, logic-based, and transparent test. The details of the whitebox testing results are shown in Table 1.

Based on the results of testing using the whitebox method that has been carried out on the program code for each process, which includes the login process, scanning, and scanning results, the results obtained based on the calculation of cyclomatic complexity with the comparison of paths found in each program logic have valid results. Thus, whitebox testing on security monitoring applications is declared victorious and acceptable.

3.7.4 The blackbox testing

The blackbox testing method focuses on external functions and behaviors such as input and output to system functionality [31]. The test results using the blackbox method are shown in Table 2.

Based on testing with the blackbox method that has been carried out, it is obtained that the functions of Login, reset password, Home, and Scan Results have run well, have no bugs, and provide a response as expected.

Table 1. Whitebox testing results

No.	Flowgraph	Cyclomatic Complexity	Number of Paths Found	Path	Description
1	Login	3	3	1-2-3-4-5-6-7-8-9	Valid
				1-2-3-8-9	Valid
				1-2-3-4-5-6-4-5-6-7-8-9	Valid
2	Home (Scanning)	1	1	1-2-3-4-5-6-7-8-9	Valid
	Home (Progress Bar)	1	1	1-2-3-4-5-6-7	Valid
	Home (Clear Log)	1	1	1-2-3-4	Valid
3	Scan Results (View)	3	3	1-2-3-4-5-7-8-9	Valid
				1-2-3-4-5-6-9	Valid
				1-2-3-7-8-9	Valid
	Scan Result (View Details)	3	3	1-2-3-4-5-6-8-9-10-11	Valid
				1-2-3-4-8-9-10-11	Valid
				1-2-3-4-5-6-7-11	Valid
	Scan Result (Delete)	1	1	1-2-3-4-5-6	Valid
	Scan Result (Download)	1	1	1-2-3-4-5-6-7-8-9	Valid

Table 2. Blackbox testing results

No	Test Menu	Input	Test Point	Test Status
1	Login	Username and password	Blank data input Wrong data input Correct data input	Accepted
2	Reset Password	E-mail address data, new password	Blank data input Wrong data input Correct data input	Accepted
3	Home	Data address Target	Correct target address input Wrong target address input Blank data input Display scanning progress Delete scan log	Accepted
4	Scan Results	Click the "Detail" button Click the "Delete" button Click the "Download" button	Displays a list of overall scan results Displays a list of scan results according to a specific time Delete the scan results according to a specific time Download the scan results in PDF format	Accepted Accepted Accepted Accepted

3.7.5 System testing results comparison

To rigorously validate the practical utility of the proposed system, a comparative performance analysis was conducted against the industry-standard OWASP ZAP (v2.11.1). Unlike theoretical simulations often found in similar studies, this research utilized a real-world case study to ensure the relevance of the findings. The security audit was executed on "Laplakgar2022" (http://localhost/laplakgar2022/), shown in Figure 17, an active government performance reporting application built on web-based language programming, PHP, and database server MySQL. This testing was performed in a controlled local environment (Intel Core i5, 8GB RAM) to isolate the system's scanning logic from external variables.

The comparative evaluation focused on two critical metrics: Scanning Duration (Efficiency) and Vulnerability Detection Rate (Effectiveness). The quantitative results of this head-to-head comparison are summarized in Table 3.

As detailed in Table 3, the proposed system demonstrated superior efficiency by completing the comprehensive scan in 4 minutes, marking a 20% reduction compared to the 5 minutes required by the benchmark tool. This efficiency gain is attributed to the system's optimized API configuration,

which was specifically tuned to the target's technology stack (PHP/MySQL), thereby bypassing redundant checks for irrelevant technologies (e.g., IIS or Java JSP) that typically slow down the default ZAP scan.

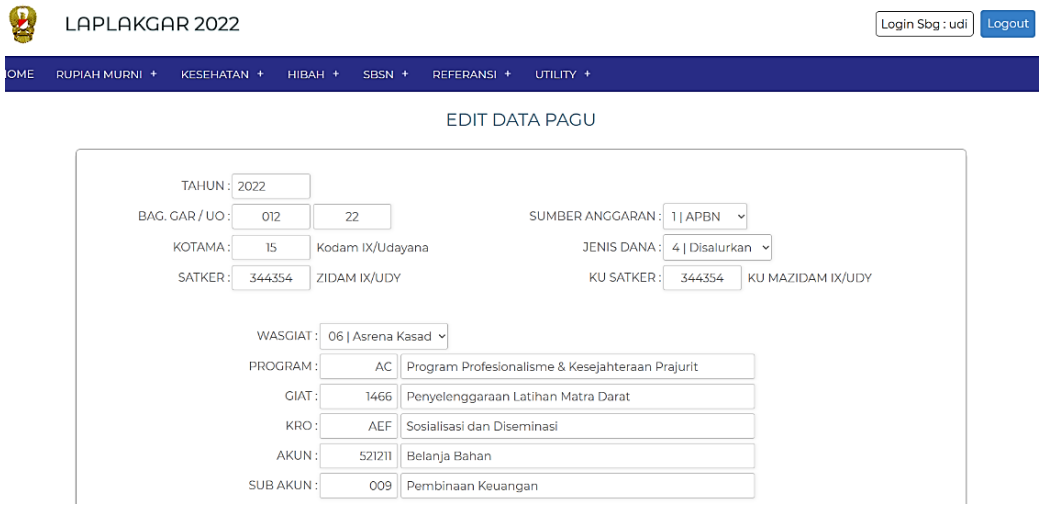

Figure 17. Laplakgar2022 testing application

Table 3. Performance comparison: Proposed system vs. OWASP ZAP

Metric	Proposed System	OWASP ZAP (Benchmark)	Difference
Scan Duration	4 minutes	5 minutes	20% Faster
High Risk Findings	1 (SQL Injection)	1 (SQL Injection)	Match
Medium Risk Findings	1 (XSS)	1 (XSS)	Match
Low/Info Findings	4	6	-2 Findings
Total Findings	6	8	-2 Findings

Note: OWASP ZAP = Open Web Application Security Project Zed Attack Proxy; SQL = Structure Query Language; XSS = Cross-Site Scripting.

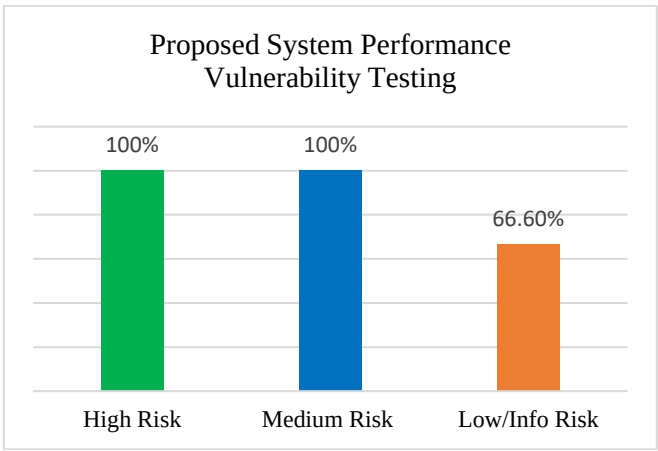

Figure 18. Vulnerability identification testing

In terms of detection accuracy, the system proved highly effective by successfully identifying all critical vulnerabilities present in the government application, specifically High-Risk SQL Injection and Medium-Risk XSS. Although the benchmark tool reported a higher total number of alerts (8 vs. 6), an analysis reveals that the "missing" findings in the proposed system were merely low-priority "Informational" alerts. By filtering out this noise, the developed system reduces alert fatigue, enabling developers to prioritize and remediate high-impact security threats more effectively. While these results highlight the system's efficiency and accuracy, it is important to acknowledge the limitations of the testing environment. Conducting the audit on a local server (localhost) excludes external factors such as Web Application Firewalls (WAF) and network jitter, which are common in live production scenarios. However, by validating the core detection engine against a real-world government application rather than a synthetic dummy target, this study establishes a strong baseline for the system's capability to audit public sector software infrastructure.

Based on the overall system tests, the test results show that the system has identified vulnerabilities at the high risk, medium risk, and informative risk levels. Based on the scenario created, the system can identify vulnerabilities at 100% at the high-risk level, 100% at the medium risk level, and 66.6% at the informative risk level (Figure 18).

3.7.6 Improvement recommendation

The developed system can provide recommendations for improvements to the evaluated web. Fixing vulnerability findings aims to fix the security holes found according to the identification results with security monitoring applications and ZAP. The system prioritizes improvements for the medium level, namely Content Security Policy (CSP) Header Not Set and Missing Anti Clickjacking Header.

```
<?php
header("Content-Security-Policy:     default-src
'self'; script-src 'self'");
?>
<script language="JavaScript">
    ;
    document.location = "login.php"
</script>
```

Figure 19. Source code recommendation on Content Security Policy (CSP) header

To make improvements based on the information obtained when scanning, namely the CSP Header Not Set, the system provides recommendations by adding the CSP header to the index.php file. This aims to protect application security and minimize the risk of XSS attacks. The results of the CSP Header Not Set recommendation are shown in the source code (Figure 19).

```
<?php
// Set Content Security Policy
header("Content-Security-Policy: frame-ancestors
'self'; default-src 'self'; script-src 'self'");

// Set anti-clickjacking X-Frame-Options header
header("X-Frame-Options: SAMEORIGIN");
?>
```

Figure 20. Source code improvement recommendations on the missing anti-clickjacking header

Meanwhile, the system recommends adding one to fix the missing anti-clickjacking header. This aims to protect the application from clickjacking attacks. The following shows the source code of the recommendation that has been added to the anti-clickjacking header in Figure 20.

After system improvements based on the recommendations given and tested again on the vulnerability monitoring system built, no vulnerabilities related to the CSP Header Not Set and Missing Anti Clickjacking Header were found. This shows that the recommendations provided by the vulnerability monitoring system provide the right solution to resolve existing system vulnerabilities.

3.7.7 The User Acceptance Testing

After the simulation is tested in the developer environment, the following application will be tested by involving end users to find out whether this application has met user needs using User Acceptance Testing (UAT). UAT is a test that involves interaction with end users and the system, and it aims to ensure that the features in the software are running correctly and by the user's needs [32]. UAT is done by giving questionnaires to all users of the security monitoring application. The goal is to determine that the built system or application meets the needs and objectives of UAT. This questionnaire method is given to 2 types of respondents (admin and operator) with 20 questions, as in Table 4. The results of each question will be presented using a Likert scale consisting of 5 scales.

Table 4. User Acceptance Testing (UAT) questionnaire

Question Type	Question Items
1. User Interface Testing	Is the interface of this security monitoring software easy to understand?
	Is the interface of this security monitoring software attractive?
	Is the home menu for scanning as expected?
	Is the scan result menu as expected?
	Is the login menu as expected?
	Is the profile menu as expected?
2. Processing Testing	Is the process on the home menu during scanning as expected?
	Did the scanning process run appropriately without any problems?
	Is the process on the scan result menu as expected?
	Is the process of downloading the scan results without problems and running as expected?
	Is the process on the login menu as expected?
	Is the process on the profile menu as expected?
3. Functional Testing	Is this security monitoring application as expected?
	Is there no error when the application is run?
	Can this simulation help users to identify web vulnerabilities?
	Does this application provide solutions to overcome the vulnerabilities found?
	Does this application provide solutions to overcome the vulnerabilities found?
	Is the scan report generated as expected?
	Is this application easy to understand and use?
	Can this application help improve application security systems?

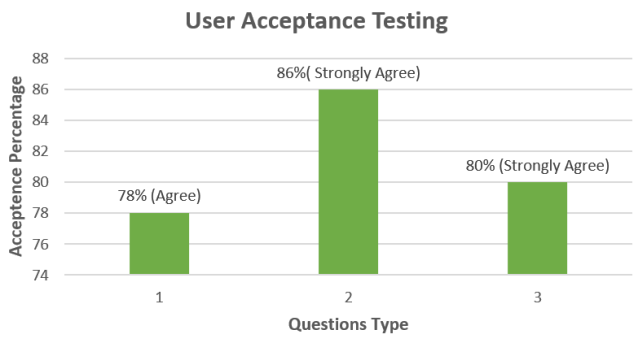

Figure 21. User Acceptance Testing (UAT) result

The evaluation is done by providing five answer options for each statement using a Likert scale of 1 to 5. The answer options include Strongly Disagree (SD), Disagree (D), Neutral (N), Agree (A), and Strongly Agree (SA). A value of 1 means Strongly Disagree, and five means Strongly Agree. The results obtained from UAT show that the first type of question is on an agreed scale, the second type of question is on a strongly agreed scale, and the third question is on an agreed scale, as shown in Figure 21.

Based on the ROs targeted in this study, the findings obtained in the study include the following:

1. This research successfully identified several frequently occurring vulnerabilities, such as SQL Injection, XSS, and IDOR. The data obtained shows the frequency of occurrence of each vulnerability in the tested applications, providing a clear insight into the most significant threats.
2. The test results show that the developed OWASP-based security monitoring application is capable of detecting system vulnerabilities, comparable to other vulnerability scanning tools, such as OWASP ZAP. This shows that the developed application is practical

and provides added value in terms of detection speed and accuracy.

3. This research showed that an intuitive and easy-to-use user interface significantly improved the user experience when performing vulnerability testing. Feedback from users indicated that a good interface design helped them understand the scan results and implement remediation recommendations more efficiently.

This performance is noteworthy and warrants a comparative discussion with similar studies in the field. Firstly, Aslan et al. [33] conducted a comprehensive review of various web application vulnerability assessment tools, emphasizing the importance of detection accuracy and efficiency. Their findings suggest that many existing tools struggle with false positives and negatives, hindering developers' trust in automated solutions. In contrast, the current research's application demonstrates a clear advantage in accurately identifying vulnerabilities, particularly at the medium level. This suggests that the application may offer a more reliable alternative for developers seeking to enhance their web application security.

Furthermore, Amrollahi et al. [34] and Shree and Dhanalakshmi [35] explored the integration of AI and machine learning techniques to improve vulnerability detection. Their study highlights that machine learning can significantly enhance the accuracy of vulnerability assessments by learning from previous data. While the current research does not explicitly incorporate machine learning, the recommendation for future work is to explore such technologies. The potential for integrating machine learning further elevates the detection capabilities of the developed application, particularly in identifying low and informative vulnerabilities. Ferrara et al. [36] focused on automated vulnerability detection through static analysis in a different approach. Their research indicates that static analysis tools can effectively identify vulnerabilities early in the development process, which is crucial for preventing security issues before deployment. While effective in identifying vulnerabilities post-development, the current research's application could benefit from incorporating static analysis techniques to enhance its overall effectiveness. This integration could improve the detection rates for low and informative vulnerabilities, which currently stand at 75% and 50%, respectively. Kumar et al. [37] introduced a novel approach using fuzzy logic for web application security testing. Their findings suggest that fuzzy logic can provide a more nuanced evaluation of vulnerabilities, accommodating the inherent uncertainties in security assessments.

The current research's application, while effective, could explore the incorporation of fuzzy logic methodologies to enhance its vulnerability assessment capabilities. This could improve detection rates, particularly for vulnerabilities that may not fit neatly into predefined categories. Meanwhile, the research of Salehi and Tavakoli [38] and Teng and Zhang [39] emphasizes information security in diverse transmission environments and media. Lastly, Al Anhar and Suryanto [40] evaluated the effectiveness of OWASP ZAP, a widely recognized security scanner, in detecting vulnerabilities in modern web applications. Their study found that while OWASP ZAP is effective, it requires continuous updates to keep pace with evolving threats. The current research's application builds upon the OWASP framework, suggesting it may face similar challenges. However, the high detection rates achieved in the current study indicate that the developed application may be better positioned to adapt to new vulnerabilities, provided it undergoes regular updates and enhancements.

4. CONCLUSIONS

This study successfully engineered an automated web vulnerability monitoring system that integrates the OWASP ZAP API into a streamlined detection-to-mitigation workflow. The system's robustness was rigorously validated through a multi-layered testing approach. Black-box and White-box testing confirmed the system's functional compliance and internal logical integrity, ensuring that all modules operate error-free. Furthermore, UAT demonstrated strong usability, achieving an average user approval rating of approximately 80%, indicating that the system's interface and reporting features align well with user needs. In terms of performance, the validation conducted on a real-world government application ("Laplakgar2022") revealed that the proposed solution matches the industry-standard benchmark with a 100% detection rate for critical High and Medium-risk vulnerabilities (specifically SQL Injection and XSS). Meanwhile, the system identified 66.6% of the Low-risk and Informational findings detected by the benchmark; this variance highlights the tool's capability to filter out non-critical noise, allowing developers to focus on actionable threats. Additionally, the system demonstrated a 20% reduction in scan duration (4 minutes vs. 5 minutes) compared to the standard tool.

Unlike the standard OWASP ZAP desktop interface, which provides comprehensive but often overwhelming data for non-security specialists, this developed application offers a distinct practical value proposition. Practitioners would prefer this tool over directly using OWASP ZAP in scenarios where rapid, automated assessments are required by development teams who need immediate, code-level remediation guidance without configuring complex scanning policies. By prioritizing high-risk threats and filtering out minor alerts, the system reduces alert fatigue and accelerates the patch management process for legacy web infrastructure.

However, it is important to contextualize these contributions within the study's limitations. The current validation was conducted on a single target application in a controlled local environment, which may not fully represent the complexity of large-scale distributed systems or cloud-native architectures protected by advanced firewalls. Therefore, while the system proves effective for its specific use case, broader generalizations regarding its performance require further extensive testing. Future research will address these constraints by expanding the test scope to diverse production environments and integrating AI to enhance the detection logic against evolving sophisticated attacks.

ACKNOWLEDGMENT

The researcher would like to thank the Directorate of Research, Technology, and Community Service (DRTPM) - Directorate General of Higher Education, Research, and Technology - Ministry of Education, Culture, Research, and Technology with Contract Number 106/E5/PG.02.00.PL/2024 for funding this research activity on the Regular Fundamental Research Grant (PFR) Year 2024 as well as Universitas

Komputer Indonesia which has provided facilities and infrastructure to support this program.

REFERENCES

[1] Anwary, I. (2022). The role of public administration in combating cybercrime: An analysis of the legal framework in Indonesia. International Journal of Cyber Criminology, 16(2): 216-227.

[2] Syahlan, Z. (2023). Implementation of information technology for the Indonesian navy in dealing with cyber threats. International Journal of Social and Management Studies, 4(4): 1-9.

[3] Ramadhani, E.H., Suyitno, D., Suryantoro, S. (2024). Information technology security assessment (ITSA) methodology for web-based E-government. AIP Conference Proceedings, 2838(1): 070002. https://doi.org/10.1063/5.0179775

[4] Kollepalli, R.P.K., Reddy, M.J.S., Sai, B.L., Natarajan, A., Mathi, S., Ramalingam, V. (2024). An experimental study on detecting and mitigating vulnerabilities in web applications. International Journal of Safety & Security Engineering, 14(2): 523-532. https://doi.org/10.18280/ijsse.140219

[5] Hariyadi, D., Fazlurrahman, F., Wijayanto, H. (2020). Bangkolo: Vulnerability identification application based on hybrid apps. Cyber Security dan Forensik Digital, 3(1): 39-44. https://doi.org/10.14421/csecurity.2020.3.1.2027

[6] Ayeni, B.K., Sahalu, J.B., Adeyanju, K.R. (2018). Detecting cross-site scripting in web applications using fuzzy inference system. Journal of Computer Networks and Communications, 2018(1): 8159548. https://doi.org/10.1155/2018/8159548

[7] Hafez, A.E., Almustafa, M.M. (2024). Detecting security vulnerabilities in web applications: A proposed system. International Journal of Safety & Security Engineering, 14(6): 1933-1940. https://doi.org/10.18280/ijsse.140627

[8] Abualese, H., Al-Rousan, T. (2023). A comparative study of web application security scanners for vulnerability detection. i-Manager's Journal on Software Engineering, 17(4): 1-8. https://doi.org/10.26634/jse.17.4.19813

[9] Sharma, I., Pahuja, V. (2023). Comparative analysis of open-source vulnerability assessment tools for campus area network. In 2023 International Conference on Emerging Smart Computing and Informatics (ESCI), Pune, India, pp. 1-6. https://doi.org/10.1109/ESCI56872.2023.10100030

[10] Moreira, D., Seara, J.P., Pavia, J.P., Serrão, C. (2024). Intelligent platform for automating vulnerability detection in web applications. Electronics, 14(1): 79. https://doi.org/10.3390/electronics14010079

[11] Yang, Y.J., Zhou, X., Mao, R.F., Xu, J.W., Yang, L.X., Zhang, Y., Shen, H.F., Zhang, H. (2025). DLAP: A deep learning augmented large language model prompting framework for software vulnerability detection. Journal of Systems and Software, 219: 112234. https://doi.org/10.1016/j.jss.2024.112234

[12] Yitagesu, S., Xing, Z., Zhang, X., Feng, Z., Bi, T., Han, L., Li, X. (2025). Systematic literature review on software security vulnerability information extraction. ACM Transactions on Software Engineering and Methodology. https://doi.org/10.1145/3745026

[13] Duggineni, S. (2023). Impact of controls on data integrity and information systems. Science and Technology, 13(2): 29-35. https://doi.org/10.5923/j.scit.20231302.04

[14] Arenas, L.A., Yactayo-Arias, C., Quispe, S.R., Sandoval, J.L. (2023). Leveraging security modeling and information systems audits to mitigate network vulnerabilities. International Journal of Safety & Security Engineering, 13(4): 763-771. https://doi.org/10.18280/ijsse.130420

[15] Jahanavi, G., Mubeen, T., Aishwarya, R., Yogitha, R. (2023). Cloud computing using OWASP: Open Web Application Security Project. In 2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, pp. 740-743. https://doi.org/10.1109/ICICCS56967.2023.10142457

[16] Lala, S.K., Kumar, A. (2021). Secure web development using OWASP guidelines. In 2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, pp. 323-332. https://doi.org/10.1109/ICICCS51141.2021.9432179

[17] Gandikota, P.S.S.K., Valluri, D., Mundru, S.B., Yanala, G.K., Sushaini, S. (2023). Web application security through comprehensive vulnerability assessment. Procedia Computer Science, 230: 168-182. https://doi.org/10.1016/j.procs.2023.12.072

[18] Patil, S., Rao, M., Misal, L., Phaldesai, D., Shivsharan, K. (2023). A review of the OW ASP top 10 web application security risks and best practices for mitigating these risks. In 2023 7th International Conference on Computing, Communication, Control and Automation (ICCUBEA), Pune, India, pp. 1-8. https://doi.org/10.1109/ICCUBEA58933.2023.10392030

[19] Upadhyay, D., Ware, N.R. (2023). Evolving trends in web application vulnerabilities: A comparative study of OWASP Top 10 2017 and OWASP Top 10 2021. International Journal of Engineering Technology and Management Sciences, 6(7): 262-269. https://doi.org/10.46647/ijetms.2023.v07i06.038

[20] Aljabri, M., Aldossary, M., Al-Homeed, N., Alhetelah, B., Althubiany, M., Alotaibi, O., Alsaqer, S. (2022). Testing and exploiting tools to improve OWASP top ten security vulnerabilities detection. In 2022 14th International Conference on Computational Intelligence and Communication Networks (CICN), Al-Khobar, Saudi Arabia, pp. 797-803. https://doi.org/10.1109/CICN56167.2022.10008360

[21] Chitadze, N. (2023). Basic principles of information and cyber security. In Analyzing New Forms of Social Disorders in Modern Virtual Environments, pp. 193-223. https://doi.org/10.4018/978-1-6684-5760-3.ch009

[22] Buede, D.M., Miller, W.D. (2024). The Engineering Design of Systems: Models and Methods. John Wiley & Sons.

[23] Rafik, H., Ettaoufik, A., Maizate, A. (2024). Securing medical data exchange: A decentralized approach based on the e-IPGPChain framework. International Journal of Safety & Security Engineering, 14(3): 815-829. https://doi.org/10.18280/ijsse.140314

[24] Senturk, O., Baghirov, A. (2024). The impact of blockchain technology on reducing cyber risks. Journal of Organizations, Technology and Entrepreneurship, 2(2): 122-135. https://doi.org/10.56578/jote020205

[25] Heryandi, A., Afrianto, I. (2019). Online diploma supplement information system modelling for Indonesian higher education institution. IOP Conference Series: Materials Science and Engineering, 662(2): 022092. https://doi.org/10.1088/1757-899X/662/2/022092

[26] Sakthivel, M., Sivanantham, S., Bharathiraja, N., Krishna, N.B., Kamalraj, R., Kumar, V.S. (2024). Ensuring web application security: An OWASP driven development methodology. In 2024 International Conference on Knowledge Engineering and Communication Systems (ICKECS), Chikkaballapur, India, pp. 1-7. https://doi.org/10.1109/ICKECS61492.2024.10617156

[27] Kellezi, D., Boegelund, C., Meng, W. (2021). Securing open banking with model-view-controller architecture and OWASP. Wireless Communications and Mobile Computing, 2021(1): 8028073. https://doi.org/10.1155/2021/8028073

[28] Priyawati, D., Rokhmah, S., Utomo I.C. (2022). Website vulnerability testing and analysis of website application using OWASP. International Journal of Computer and Information System (IJCIS), 3(3): 142-147. https://doi.org/10.29040/ijcis.v3i3.90

[29] Idris, M., Syarif, I., Winarno, I. (2022). Web application security education platform based on OWASP API security project. EMITTER International Journal of Engineering Technology, 10(2): 246-261. https://doi.org/10.24003/emitter.v10i2.705

[30] Sarpong, P.A., Larbi, L.S., Paa, D.P., Abdulai, I.B., Amankwah, R., Amponsah, A. (2021). Performance evaluation of open source web application vulnerability scanners based on OWASP benchmark. International Journal of Computer Applications, 174(18): 15-22. https://doi.org/10.5120/IJCA2021921070

[31] Afrianto, I., Heryandi, A., Finandhita, A., Atin, S. (2020). Prototype of e-document application based on digital signatures to support digital document authentication. IOP Conference Series: Materials Science and Engineering, 879(1): 012042. https://doi.org/10.1088/1757-899X/879/1/012042

[32] Afrianto, I., Heryandi, A., Finandhita, A. (2021). User acceptance test for digital signature application in academic domain to support the COVID-19 work from home program. IJISTECH (International Journal of Information System and Technology), 5(3): 270-280. https://doi.org/10.30645/ijistech.v5i3.132

[33] Aslan, Ö., Aktuğ, S.S., Ozkan-Okay, M., Yilmaz, A.A., Akin, E. (2023). A comprehensive review of cyber security vulnerabilities, threats, attacks, and solutions. Electronics, 12(6): 1333. https://doi.org/10.3390/electronics12061333

[34] Amrollahi, M., Hadayeghparast, S., Karimipour, H., Derakhshan, F., Srivastava, G. (2020). Enhancing network security via machine learning: Opportunities and challenges. In Handbook of Big Data Privacy, pp. 165-189. https://doi.org/10.1007/978-3-030-38557-6_8

[35] Shree, S.V., Dhanalakshmi, S. (2024). Information security by employing RSA algorithm and graph labeling. IAENG International Journal of Applied Mathematics, 54(12): 2563-2568.

[36] Ferrara, P., Mandal, A.K., Cortesi, A., Spoto, F. (2021). Static analysis for discovering IoT vulnerabilities. International Journal on Software Tools for Technology Transfer, 23(1): 71-88. https://doi.org/10.1007/s10009-020-00592-x

[37] Kumar, R., Khan, A.I., Abushark, Y.B., Alam, M.M., Agrawal, A., Khan, R.A. (2020). An integrated approach of fuzzy logic, AHP and TOPSIS for estimating usable-security of web applications. IEEE Access, 8: 50944-50957. https://doi.org/10.1109/ACCESS.2020.2970245

[38] Salehi, M.R.D., Tavakoli, H. (2022). Achieving secure communication over wiretap channels using the error exponent of the polar code. Engineering Letters, 30(1): 1.

[39] Teng, W., Zhang, Y. (2024). STRay: A model for prohibited item detection in security check images. Engineering Letters, 32(10): 1854.

[40] Al Anhar, A., Suryanto, Y. (2021). Evaluation of web application vulnerability scanner for modern web application. In 2021 International Conference on Artificial Intelligence and Computer Science Technology (ICAICST), Yogyakarta, Indonesia, pp. 200-204. https://doi.org/10.1109/ICAICST53116.2021.9497831