INTENSIF, Vol.10 No.1 2026
ISSN: 2580-409X (Print) / 2549-6824 (Online)
DOI: https://doi.org/10.29407/intensif.v10i1.25294

Security Assessment Based on OWASP Top 10 Using SonarQube and ZAP on Export and Import Applications in the LNSW

Received:
9 June 2025
Accepted:
12 Agustus 2025
Published:
1 February 2026

[1*]Muhammad Wisnu, [2]Benfano Soewito
[1,2]Master of Computer Science, Universitas Bina Nusantara
E-mail: [1]muhammad.wisnu@binus.ac.id, [2]bsoewito@binus.edu

*Corresponding Author

Abstract— **Background**: The advancement of information and electronic systems has significantly transformed export and import processes. In Indonesia, the Lembaga National Single Window (LNSW) plays a pivotal role in facilitating international trade by integrating procedures and information related to exports, imports, and document flows. **Objective**: This study aims to assess the security of LNSW's export and import application by identifying vulnerabilities based on the Open Web Application Security Project (OWASP) Top 10 framework. It also compares the effectiveness of Static Application Security Testing (SAST) using SonarQube and Dynamic Application Security Testing (DAST) using ZAP (Zed Attack Proxy) in detecting various types of vulnerabilities. **Methods**: The analysis involved the use of SonarQube for source code scanning and ZAP for runtime testing. Each detected vulnerability was evaluated using the Common Vulnerability Scoring System (CVSS) to determine its severity level. Recommended mitigation strategies were provided accordingly. **Results**: A total of eight vulnerabilities were identified, comprising two High-severity and six Medium-severity issues. SonarQube proved more effective in detecting Identification and Authentication Failures (three instances), while ZAP excelled in identifying Vulnerable and Outdated Components (two instances). Notably, each tool uncovered four unique types of vulnerabilities that the other did not detect. **Conclusion**: These findings highlight the practical benefits of combining SAST and DAST techniques. By integrating both approaches, organizations can achieve a more comprehensive and reliable security assessment, ultimately leading to more resilient software systems.

Keywords—OWASP; ZAP; Cyber Security; SonarQube; SAST; Vulnerability Assessment

Corresponding Author:
Muhammad Wisnu,
Computer Science,
Universitas Bina Nusantara,
Email: muhammad.wisnu@binus.ac.id
Orchid ID: https://orcid.org/0009-0006-5040-7660

INTENSIF, Vol.10 No.1 2026
ISSN: 2580-409X (Print) / 2549-6824 (Online)
DOI: https://doi.org/10.29407/intensif.v10i1.25294

I. INTRODUCTION

In 2022, Indonesia recorded over 539 million anomalous network traffics, making it the top target for cyber anomalies in the region. Among the top threats were plaintext password transmissions, exploitation via HTTP OPTIONS, and Distributed Denial of Service (DDoS) attacks. Additionally, the Indonesian Security Incident Response Team (IDSIRTII) reported that government web applications were the most affected sector, with 855 defacement cases and 120 suspected incidents of vulnerabilities and data leaks [1]. These findings highlight the growing threat landscape targeting national digital infrastructure including systems that support vital functions such as trade and customs processing.

One institution at the center of Indonesia's digital trade infrastructure is the LNSW, which manages integrated applications for handling export, import, and document flows. As these systems become increasingly complex and digitally interconnected, their security becomes mission critical. A breach or compromise in LNSW systems could result in disrupted trade operations, data leaks, and reduced trust from international partners. Therefore, a systematic and comprehensive security assessment of these applications is urgently needed to ensure the resilience of Indonesia's international trade framework.

The rapid advancement of information technology brings both opportunities and challenges, particularly in the realm of cybersecurity. Cyberattacks such as web defacement, data theft, and unauthorized access threaten the confidentiality, integrity, and availability of information handled by government web applications [2]. Among the most prevalent vulnerabilities are Cross-Site Scripting (XSS) and SQL Injection (SQLi), which are commonly exploited by attackers to gain access to sensitive data or manipulate backend databases [3], [4]. These types of vulnerabilities have proven to be persistent and damaging, especially in public-facing systems.

Given this landscape, conducting comprehensive security assessments is essential for identifying and mitigating such threats. For LNSW, an institution responsible for managing critical export and import information, including customs and financial documentation, the stakes are especially high. Any compromise of its systems could result in data breaches, operational disruptions, and economic losses, ultimately undermining public trust and institutional credibility. In this context, vulnerability identification is conducted through systematic vulnerability assessments to obtain insight into potential security weaknesses within the application infrastructure using vulnerability scanning tools [5], [6], [7]. Therefore, ensuring the security of LNSW's application infrastructure is a strategic necessity.

Several prior studies have attempted to evaluate the security of web applications using the OWASP framework. For example, [8] and [9] employed ZAP as a Dynamic Application Security

ISSN: 2580-409X (Print) / 2549-6824 (Online)
DOI: https://doi.org/10.29407/intensif.v10i1.25294

Testing (DAST) tool to detect vulnerabilities such as SQL Injection, XSS, and CSRF. These studies demonstrated that DAST tools can identify high-severity vulnerabilities that pose risks of data breaches. However, both studies focused solely on external testing, without providing severity classification for each detected issue, which limits the ability to prioritize remediation efforts. Furthermore, ZAP's lack of access to source code and internal application logic constrained the depth of the findings.

It is known that each tool has its strengths and weaknesses in detecting certain vulnerabilities, and developers should not solely rely on DAST as the primary method for vulnerabilities detection. This is because DAST focuses only on interaction between the user and the application and does not perform internal analysis, which is done by static application security testing (SAST) tools like SonarQube [10]. To support this, combining other tools such as SonarQube and ZAP has been shown to provide more optimal results and improve the true positive rate in detecting security vulnerabilities within applications [11]. Prior research has indicated that DAST tools can identify numerous vulnerabilities across different severity levels, including critical issues such as SQL Injection and improper security configurations [12], [13]. It has also been observed that web applications lacking basic security measures are highly susceptible to attacks, potentially resulting in severe system compromise [14]. Furthermore, systematic reviews have emphasized that combining SAST and DAST approaches significantly enhances vulnerability detection capabilities compared to relying solely on a single method [15].

Previous studies on application security testing often rely solely on DAST tools, which lack internal system visibility and provide limited context for detected issues. While these approaches can identify runtime vulnerabilities, they are less effective in detecting certain categories of weaknesses and often omit severity scoring, making it difficult to prioritize remediation efforts.

However, these studies suffer from key limitations that this research seeks to address namely, the lack of severity scoring for each detected vulnerability and the reliance on a single testing tool, typically a DAST approach, without internal system visibility. These limitations reduce the effectiveness of prior analyses in prioritizing threats and understanding the full security posture of the application. This study therefore aims to provide a strategic and comparative security assessment of the LNSW export and import application by leveraging the complementary strengths of SAST (SonarQube) and DAST (OWASP ZAP) tools. By systematically integrating both testing methodologies, this research seeks to deliver a comprehensive identification and categorization of vulnerabilities, while highlighting the strengths and limitations of each approach in detecting different types of security issues.

INTENSIF, Vol.10 No.1 2026
ISSN: 2580-409X (Print) / 2549-6824 (Online)
DOI: https://doi.org/10.29407/intensif.v10i1.25294

II. RESEARCH METHOD

To achieve the research objectives, this study adopts a structured methodology consisting of the following main steps. These are visually summarized in Figure 1, which outlines the sequential phases carried out during this research.

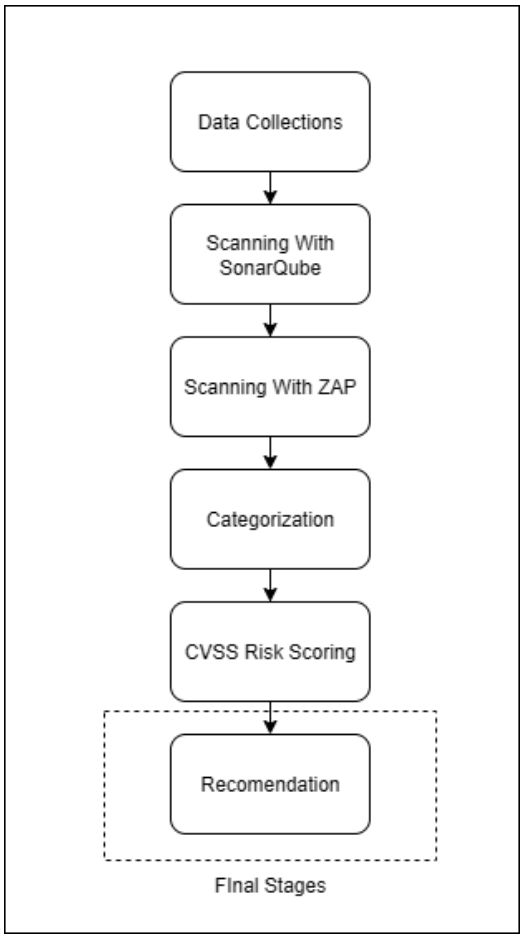

Fig 1. Methodology Steps

A. Data Collections

To represent and select relevant applications for security analysis, this study adopted an evidence-based selections strategy by accessing the internal analytics dashboard provided by LNSW. The dashboard displays live traffic statistics, including the total number of HTTP requests for each application, as recorded by the system's internal monitoring services. For this research, request data was retrieved covering a 90-day period from April to June 2024.

INTENSIF, Vol.10 No.1 2026
ISSN: 2580-409X (Print) / 2549-6824 (Online)
DOI: https://doi.org/10.29407/intensif.v10i1.25294

LNSW manages over 30 digital services, including API's and web applications. In line with the problem scope, this study focused on the top three applications with the highest number of user interactions, as measured by total request volume within the defined period. This approach assumed that applications with higher traffic are more likely to be exposed to potential security threats and therefore more critical for analysis.

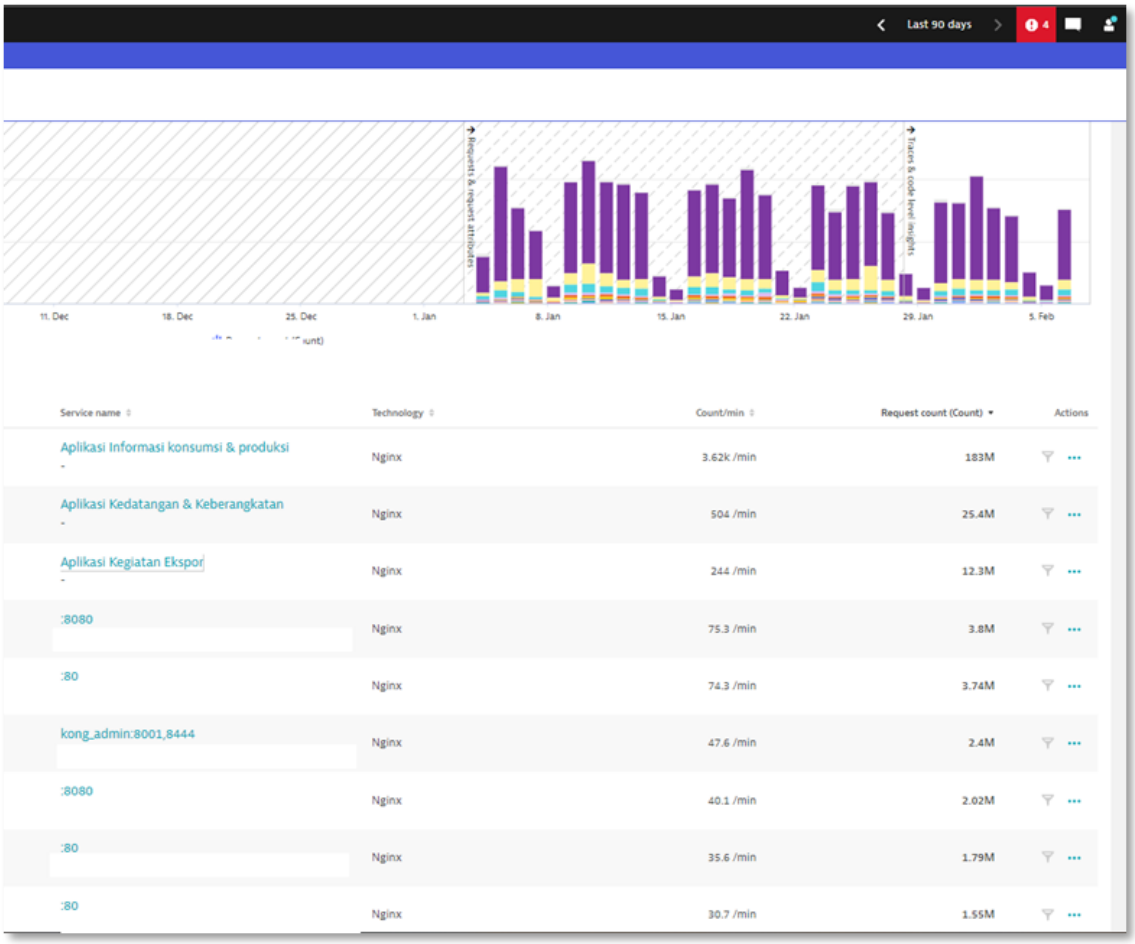

Fig 2. LNSW Internal Dashboard Displaying Application Request Volume for Period April-June 2024

As shown in Figure 2, the Aplikasi Informasil Konsumsi & Produksi recorded the highest request volume, followed by Aplikasi Keadatangan dan Keberangkatan with 25.4 million requests, and the Aplikasi Kegiatan Ekspor with 12.3 million requests. These three applications were selected for in-depth security scanning and vulnerability assesment.

INTENSIF, Vol.10 No.1 2026
ISSN: 2580-409X (Print) / 2549-6824 (Online)
DOI: https://doi.org/10.29407/intensif.v10i1.25294

B. Scanning Application Code with SonarQube

SonarQube is utilized in this study as a static code analysis tool to detect vulnerabilities and code quality issues by examining the application's source code without executing it [16], [17]. In this setup SonarQube was used with its default configuration and ruleset, without the addition of external plugins. The tools support over 30 programming languages [18], and in this study, it was applied to analyze JavaScript source code which is the primary language of the selected applications.

In this research, SonarQube was installed on a dedicated virtual machine within the LNSW development environment. It was integrated into the CI/CD pipeline using Gitlab CI, allowing for automated code analysis as supported by platform [19]. The analysis was triggered automatically upon every change or commit (full scan, enabling continuous monitoring of code quality. Figure 3 illustrates the integration of SonarQube within the CI/CD process.

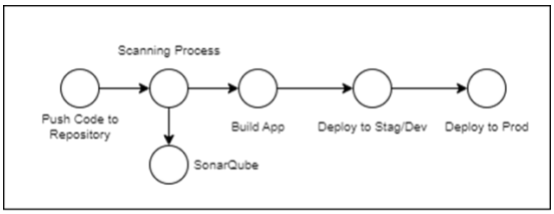

Fig 3. Pipeline Process

The scanning process generates categorized security issues that include the type of vulnerability, its location in the source code, and a severity level. Vulnerabilities are automatically mapped into OWASP Top 10 categories when applicable, such as XSS or SQL Injection, based on the rule match. This setup enables early identification of critical security flaws, including blocker bug and security hotspots, providing actionable insight for developer during the development cycle [20].

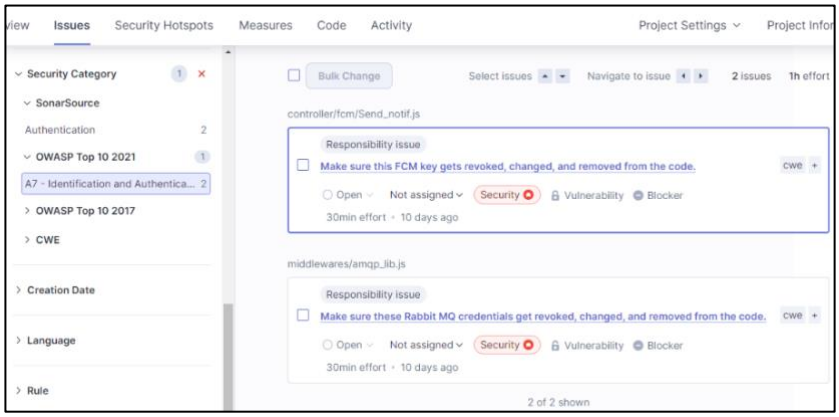

Fig 4. Scan Results by SonarQube

INTENSIF, Vol.10 No.1 2026
ISSN: 2580-409X (Print) / 2549-6824 (Online)
DOI: https://doi.org/10.29407/intensif.v10i1.25294

As shown in Figure 4, there are examples of two vulnerabilities with the Authentication type which fall under the OWASP category of Identification and Authentication Failures.

C. Scanning Application with ZAP

In this stage, scanning is conducted using ZAP to identify vulnerabilities present in the application. After downloading the application from its official site, ZAP is installed on a separate computer. Unlike SonarQube, which is integrated into the development server infrastructure, ZAP operates independently. It intercepts all requests and responses from the web server as part of a comprehensive testing process that includes information gathering, scanning, vulnerability exploitation, and reporting. Furthermore, ZAP allows users to modify transiting traffic, enabling deeper analysis and more thorough testing of potential vulnerabilities [21], [22]. ZAP can be enhanced to detect various types of SQL injection attacks such as Error-based, Union-based, Time-based blind, and Authentication Bypass using active rules scripts, which significantly improves its effectiveness in identifying critical web security issues [23]. Figure 5 provides an overview of how ZAP is utilized during the scanning process.

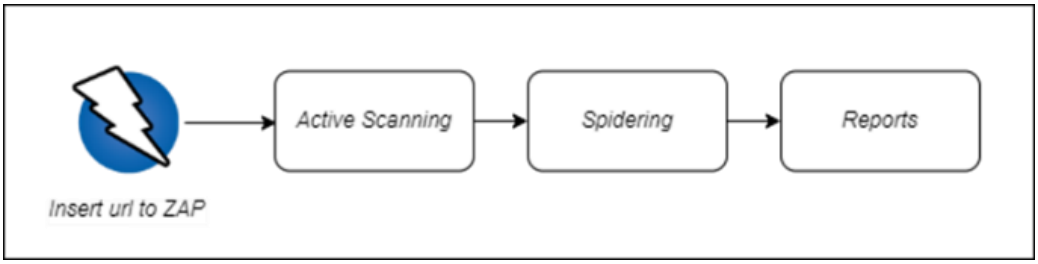

Fig 5. ZAP Scanning Process

D. Analysis of Scanning Results

After completing the scanning stage, the next step involves analyzing the vulnerabilities that were previously identified. In this step, the authors categorize types of attacks and describe the potential impacts of each vulnerability. The following are the steps taken to categorize the identified vulnerabilities, as depicted in Figure 6.

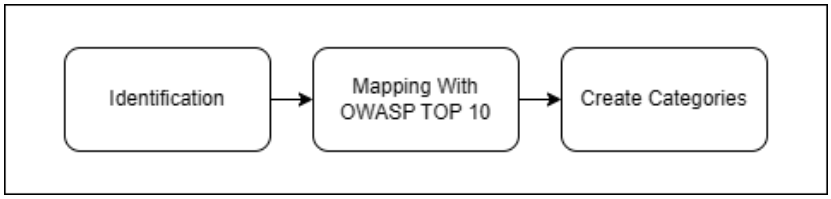

Fig 6. Security Categorization Step

ISSN: 2580-409X (Print) / 2549-6824 (Online)
DOI: https://doi.org/10.29407/intensif.v10i1.25294

Based on Figure 6, the Security Categorization Step involves the identification stage, where each vulnerability identified from the scanning results, such as SQL Injection and XSS, undergoes review. These vulnerabilities are then matched against the OWASP TOP 10 list to determine if they correspond to any of the listed ten categories. Subsequently, each vulnerability is assigned to the most appropriate category, for example, SQL Injection falls under the Injection category (A03:2021). In the final step, entries are recorded in a categorization table, followed by weighting using CVSS Risk Scoring. By categorizing the vulnerabilities, the author can provide appropriate recommendations, prioritize risks, and develop targeted mitigation strategies to prevent the exploitation of identified vulnerabilities, referring to OWASP guidelines [24].

E. CVSS Risk Scoring

Vulnerability ranking is performed after relevant information about each vulnerability has been identified. The purpose of this ranking is to classify the risk level and determine the handling priority of each vulnerability by assessing base score metrics. This process employs the Common Vulnerability Scoring System (CVSS) version 3.1 [25], [26], which assigns numerical values to vulnerabilities, enabling standardized evaluation of security risks and facilitating the identification of potential impacts associated with specific types of attacks.

The CVSS Risk Scoring Step is illustrated in Figure 7, showing the logical sequence from vulnerability identification to severity rating. The information regarding the impact and scenario of a vulnerability is obtained from the OWASP list and previous scan results. For example, if vulnerability is accessible via Network, the Attack Vector (AV) is assigned a value of 0.85, and if the attack has high complexity, the Attack Complexity (AC) is assigned a value of 0.44. Other parameters include Privileges Required (PR), User Interaction (UI), Scope (S), Confidentiality (C), Integrity (I), and Availability (A).

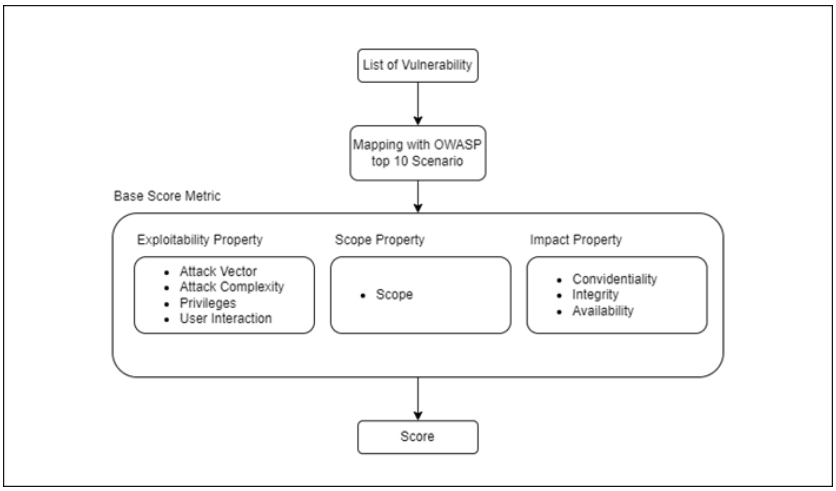

Fig 7. CVSS Risk Scoring Step

INTENSIF, Vol.10 No.1 2026
ISSN: 2580-409X (Print) / 2549-6824 (Online)
DOI: https://doi.org/10.29407/intensif.v10i1.25294

While Figure 7 presents the overall process for deriving CVSS risk scores, a more detailed understanding requires examining the possible values assigned to each metric in the Base Metric Group. Table 1 outlines these possible values for each metric, as defined by the CVSS 3.1 Specification Document [27]. These values are essential for ensuring consistency and accuracy when assessing the severity of vulnerabilities.

Table 1. Qualitative Severity Rating Scale

Metric Value	Possible Value	Values
Attack Vector (AV)	Physical (P)	0.20
	Network (N)	0.85
	Local (L)	0.55
	Adjacent (A)	0.62
Attack Complexity (AC)	Low (L)	0.77
	High (H)	0.44
Privileges (PR)	None (N)	0.85
	Low (L)	0.62
	High (H)	0.27/0.5
User Interaction (UI)	Required (R)	0.62
	None (N)	0.85
Scope (S)	Unchanged (U)	-
	Changed (C)	-
Confidentiality (C)	None (N)	0
	Low (L)	0.22
	High (H)	0.56
Integrity (I)	None (N)	0
	Low (L)	0.22
	High (H)	0.56
Availability (A)	None (N)	0
	Low (L)	0.22
	High (H)	0.56

The information regarding the impact and scenario of a vulnerability is obtained from the OWASP list and previous scan results. For example, if the vulnerability is accessible via Network, the Attack Vector is assigned a weight of 0.85, and if the attack have a high complexity, the Attack Complexity is given a value of 0.44, below is the formulas used to calculate the Impact Sub Score (ISS) (1), Impact (2), Exploitability (3), and the Base Score (4) in the CVSS 3.1 Framework.

$$ISS = 1 - [(1 - C) \times (1 - I) \times (1 - A)] \quad (1)$$

ISSN: 2580-409X (Print) / 2549-6824 (Online)
DOI: https://doi.org/10.29407/intensif.v10i1.25294

If Scope Unchanged: *(2)*

$$Impact = 6.42 \times ISS$$

If Scope Changed:

$$Impact = 7.52 \times (ISS - 0.029) - 3.25 \times (ISS - 0.02)^{15}$$

$$Exploitability = 8.22 \times AV \times AC \times PR \times UI \quad (3)$$

$$Base\ Score = \min(Impact + Exploitability, 10) \times Scope\ Modifier) \quad (4)$$

(Scope Modifier is 1.0 if scope unchanged, or 1.08 if scope is changed)

Each vulnerability is also represented using a CVSS vector string, which provides a compact, textual representation of its base metrics. For instance, the vector string AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H indicates a vulnerability that is exploitable over the network with low complexity, no required privileges, and high impact on confidentiality, integrity, and availability.

As an example, a vulnerability with vector string AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H would result in a base score of 9.8. The CVSS vector and base score were derived using the official NIST CVSS Calculator v3.1 [28].

For example, a vulnerability with a base score of 9.8 is considered Critical based on the CVSS scoring system. After obtaining the CVSS base score, a Qualitative Severity Rating is applied to classify the severity of the vulnerability into categories such as Low, Medium, High, or Critical [29]. Table 2 presents the score ranges used in this classification.

Table 2. Qualitative Severity Rating Scale

CVSS Score	Rating
0.0	None
0.1 - 3.9	Low
4.0 - 6.9	Medium
7.0 - 8.9	High
9.0-10.0	Critical

III. RESULT AND DISCUSSION

A. Scan Results

After scanning the applications using SonarQube and ZAP, the next step involves compiling a list of vulnerabilities identified during the scanning process into a table for assessment using the CVSS. Recommendations are then provided to prevent exploitation and close security gaps in the applications, following OWASP security guidelines [30]. The findings from both tools are summarized in Table 3, followed by a description of the differences in their detection capabilities.

INTENSIF, Vol.10 No.1 2026
ISSN: 2580-409X (Print) / 2549-6824 (Online)
DOI: https://doi.org/10.29407/intensif.v10i1.25294

Table 3. Vulnerabilities Identified by SonarQube and ZAP

ID	Vulnerabilities	Descriptions	Tools
V01	Directory Browsing	This vulnerability occurs when a web application is misconfigured, allowing unauthorized access to directories and files that should not be exposed to the public. A user can navigate through the directory structure and potentially discover sensitive files such as configuration files, log, or backup files.	ZAP
V02, V06	Vulnerable JS Library	This issue arises when an application uses outdated or unsupported JavaScript library that may contain known security vulnerabilities that have not been addressed due to lack of updates or patches from the developer. By using such components, the application is exposed to various risks, including code injections, XSS, and other exploits.	ZAP
V03	Server-Side Code Injection	This vulnerability occurs when an attacker can inject malicious code into the server-side environment, often through validation input or improper handling.	SonarQube
V04	Backup File Disclosure	Backup files are improperly stored or accessible on the server due to security misconfigurations. Backup files often contain sensitive data including source code, database credentials, configuration, and sensitive user information.	ZAP
V05, V07, V08	Authentication	This issue arises when sensitive data is hardcoded directly to the code. The data such as password, API keys, or secret data. Storing sensitive data to the code is dangerous because it can be easily exposed if the codebase is compromised.	SonarQube

While both tools uncovered a range of security issues, differences were observed in the types of vulnerabilities detected by each tool. For instance, ZAP was more effective in identifying outdated components and runtime-related issues, whereas SonarQube reported more vulnerabilities related to authentication and input validation. These discrepancies can be explained by the fundamental approaches used by each tool. ZAP operates as a DAST tool, which interacts with the application during execution and inspects HTTP responses and behaviors, making it suitable for detecting runtime and deployment-level flaws. In contrast, SonarQube functions as a SAST tool, analyzing source code without executing it. This allows it to identify logical flaws, insecure coding practices, and authentication issues directly within the codebase. Therefore, the combined use of both tools provides a more comprehensive assessment of the application's security posture.

INTENSIF, Vol.10 No.1 2026
ISSN: 2580-409X (Print) / 2549-6824 (Online)
DOI: https://doi.org/10.29407/intensif.v10i1.25294

At this stage, the result from previous process of identifying vulnerabilities are categorized according to OWASP. The author has grouped the types of attacks based on the list of vulnerabilities that correspond to the OWASP top 10 security risks. This categorization helps to better understand the severity and nature of each vulnerability in the context of web applications security. Figure 8 illustrates the type of vulnerability that fits within the most common and critical security risk.

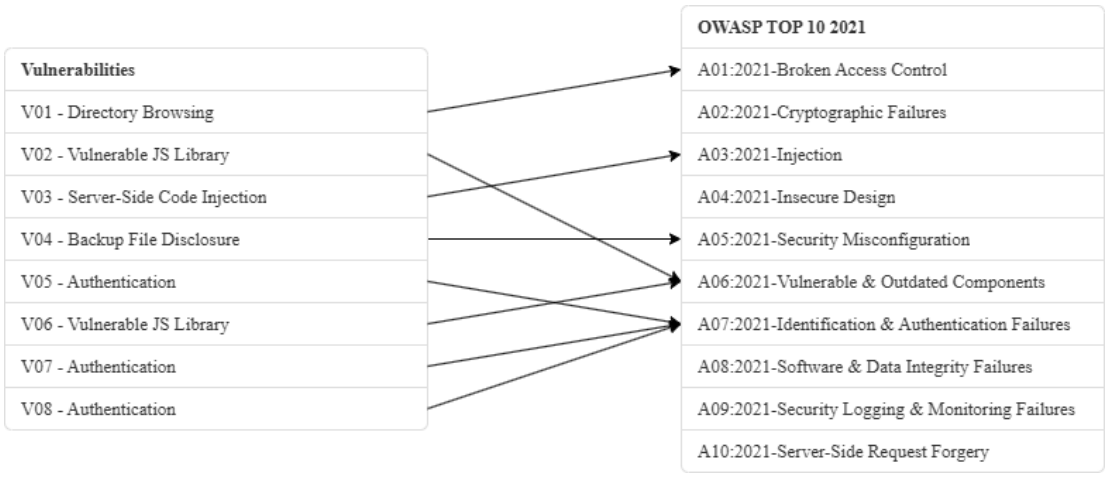

Fig 8. Category of Vulnerabilities Based on OWASP

Figure 8 illustrates the correlation between the identified application vulnerabilities (V01–V08) and the OWASP Top 10 2021 categories. This mapping provides a structured overview of the types of security weaknesses present in the analyzed applications.

As shown in the figure, several vulnerabilities are associated with A07:2021 – Identification and Authentication Failures. Specifically, vulnerabilities V05, V07, and V08 indicate the presence of hardcoded credentials in the codebase. The recurrence of this issue suggests a systemic flaw in how authentication data is managed and stored.

In addition, vulnerabilities V02 and V06 fall under A06:2021 Vulnerable and Outdated Components. These indicate the use of deprecated JavaScript libraries, which pose risks due to unpatched security flaws and the absence of maintenance from developers.

Vulnerability V03, categorized as A03:2021 Injection, highlights a server-side code injection issue, which typically arises from insufficient input validation and improper handling of user data on the server.

Furthermore, vulnerabilities V01 and V04 are linked to A05:2021 Security Misconfiguration. These include directory browsing and backup file disclosure both pointing to inadequate server configuration that exposes sensitive files to unauthorized users.

Overall, the classification reveals that most vulnerabilities are concentrated within authentication flaws, insecure component usage, and configuration errors. This trend reflects

INTENSIF, Vol.10 No.1 2026
ISSN: 2580-409X (Print) / 2549-6824 (Online)
DOI: https://doi.org/10.29407/intensif.v10i1.25294

common security concerns emphasized in the OWASP Top 10, underscoring the need for improved development practices, regular dependency updates, and secure configuration management.

Following the presentation of Figure 8, Table 4 provides a detailed summary of the vulnerabilities that have been categorized according to the OWASP Top 10. Each entry in the table shows the specific vulnerability, and the corresponding OWASP category.

Table 4. Scanning Results

ID	Vulnerabilities	Categories	Tools
V01	Directory Browsing	A01:2021-Broken Access Control	ZAP
V02, V06	Vulnerable JS Library	A06:2021-Vulnerable & Outdated Components	ZAP
V03	Server-Side Code Injection	A03:2021 – Injection	SonarQube
V04	Backup File Disclosure	A05:2021 – Security Misconfiguration	ZAP
V05, V07, V08	Authentication	A07:2021 – Identification & Authentication Failures	SonarQube

B. CVSS Results

After calculating the CVSS scores as described in the previous section, the results for each identified vulnerability are presented in Table 5. These results indicate the severity and urgency levels based on the calculated scores.

Table 5. Score Results

ID	AV	AC	PR	UI	S	C	I	A	Score	Rating
V03	0.55	0.77	0.5	0.85	-	0.56	0.56	0.56	8.2	High
V04	0.85	0.77	0.85	0.85	-	0.56	0	0	7.5	High
V02	0.85	0.77	0.85	0.85	-	0.22	0	0.22	6.5	Medium
V06	0.85	0.77	0.85	0.85	-	0.22	0	0.22	6.5	Medium
V05	0.55	0.77	0.85	0.85	-	0.56	0	0	6.2	Medium
V07	0.55	0.77	0.85	0.85	-	0.56	0	0	6.2	Medium
V08	0.55	0.77	0.85	0.85	-	0.56	0	0	6.2	Medium
V01	0.85	0.77	0.85	0.85	-	0.22	0	0	5.3	Medium

ISSN: 2580-409X (Print) / 2549-6824 (Online)
DOI: https://doi.org/10.29407/intensif.v10i1.25294

As shown in CVSS calculation results in Table 5, vulnerability V03 received a score of 8.2 (High) because it can be exploited over a network, has low attack complexity, requires no privileges (Privileges Required: None), and has a significant impact on both confidentiality and availability. In contrast, V04 scored 7.5 (High) with a moderate attack complexity, but it still poses a high risk due to its potential impact on data integrity and availability.

C. Security Recommendations

After calculating the CVSS and determining the priority, the next step is to provide recommendations are applied to help organizations safeguard their applications and prevent exploitation of the vulnerability. Table 6 presents the security recommendations.

Table 6. Security Recommendations Sorted by Rating

ID	Categories	Recommendation	Rating
V03	Injections	Implement server-side input validation, especially for dynamic query inputs, to mitigate the risk of injection vulnerabilities such as SQL Injection. Ensure that inputs are properly sanitized and validated before being processed.	High
V04	Security Misconfiguration	Change default passwords and remove any unused components or features from framework. This helps prevent potential vulnerabilities associated with default settings and unnecessary code to reducing the attack surface.	High
V02, V06	Vulnerable & Outdated Components	Update outdated components by either upgrading them or finding alternative components and regularly repeat the hardening process to ensure that systems remain resilient against emerging vulnerabilities.	Medium
V05, V07, V08	Identification & Authentication Failures	Avoid storing passwords, security keys, or sensitive data directly in the codebase, and implement two-factor authentication (2FA) to enhance security. These measures strengthen the protection of user credentials and sensitive information, reducing the risk of identification and authentication failures.	Medium
V01	Broken Access Control	Use a whitelist system, block all access by default, and only allow predefined resources to be publicly accessible. Regularly review and update a whitelist to ensure that only necessary resources remain exposed and monitor for any unauthorized access attempts to maintain a secure environment.	Medium

Based on the vulnerabilities presented in Table 6, each identified issue carries significant implications for LNSW's ability to facilitate secure and reliable national trade operations. For instance, injection vulnerabilities (V03) could compromise the integrity and confidentiality of

ISSN: 2580-409X (Print) / 2549-6824 (Online)
DOI: https://doi.org/10.29407/intensif.v10i1.25294

national trade data by enabling unauthorized manipulation or exfiltration of critical information, potentially leading to data breaches that undermine both operational accuracy and public trust. Security misconfigurations (V04) heighten the risk of unauthorized system access, which could disrupt trade processing or expose sensitive trade records to malicious actors. Outdated or vulnerable components (V02, V06) not only expand the attack surface but also threaten system availability, potentially delaying or halting the processing of import/export documentation and diminishing stakeholder confidence. Failures in identification and authentication mechanisms (V05, V07, V08) could result in credential theft, enabling impersonation of legitimate users and the submission of fraudulent trade transactions. Finally, weaknesses in access control (V01) may allow unauthorized entities to access restricted resources, leading to data leaks or operational manipulation. Collectively, these vulnerabilities represent risks not only to system security but also to LNSW's operational continuity, the integrity of national trade data, and the trust of both the public and international partners.

IV. CONCLUSION

The following conclusions can be drawn from the results of this study, the assessment of the application identified a total of eight vulnerabilities with varying severity levels, consisting of six categorized as Medium and two as High. The two High-severity vulnerabilities, with CVSS scores of 8.2 and 7.5, indicate significant security risks that require immediate remediation. SonarQube primarily detected vulnerabilities related to source code quality and secure coding practices, while ZAP identified issues related to runtime behavior and web application interactions. SonarQube detected four vulnerabilities across two types of security issues, while ZAP identified four vulnerabilities spanning three types. These results highlight the importance of employing multiple analysis tools with complementary focuses to gain a comprehensive view of potential security weaknesses. Addressing the identified vulnerabilities promptly can reduce the risk of exploitation and enhance the overall security posture of the application. Future research could explore broader testing scenarios or integrate additional security tools to further improve detection accuracy.

Author Contributions: *Muhammad Wisnu*: Was responsible for Conceptualization, Methodology, Writing Original Draft, Investigation, and Data Collections. *Benfano Soewito*: Was Responsible for Supervision, Writing – Review and Editing the Manuscript.

All authors have read and agreed to the published version of the manuscript.

Funding: This research received no specific grant from any funding agency.

Conflicts of Interest: The authors declare no conflict of interest.

ISSN: 2580-409X (Print) / 2549-6824 (Online)
DOI: https://doi.org/10.29407/intensif.v10i1.25294

Data Availability: The data that support the findings of this study are not publicly available due to privacy and security concerns related to the sensitivity of the application being assessed. Access to the data is restricted to authorized parties within the scope of this research.

Informed Consent: There were no human subjects.

Animal Subjects: There were no animal subjects.

ORCID:
Muhammad Wisnu: https://orcid.org/0009-0006-5040-7660
Benfano Soewito: https://orcid.org/0000-0002-3284-5433

REFERENCES

[1] Badan Siber dan Sandi Negara, "Lanskap Keamanan Siber Indonesia Tahun 2022," *in Laporan Hasil Monitoring.* Jakarta , Indonesia 2022.

[2] MOHD. Yusuf, Suryadi, Robi Hamid, *"Analisis Kejahatan Hacking Sebagai Bentuk Cyber Crime Dalam Sistem Hukum yang berlaku di Indonesia," JPDK*, vol. 4, no. 6 2022 doi: doi.org/10.31004/jpdk.v4i6.8685.

[3] A. Fadlil, I. Riadi, & F. Fachri, *"Mitigation Web Server for Cross-Site Scripting Attack Using Penetration Testing Method," IIETA*, vol. 12, no. 2, pp. 201-208, 2019, doi: 10.18280/ijsse.120208.

[4] A. Alanda, D. Satria, M. I. Ardhana, A. A. Dahlan, and H. A. Mooduto, "Web Application Penetration Testing Using SQL Injection Attack," *JOIV*, vol. 5, no. 3, pp. 320-326 2, pp. 201-208, 2019, doi: 10.30630/joiv.5.3.470.

[5] S. A. Khan, N. Azim, A. Iqbal, H. Abbas, and S. Qureshi, "Securing Web Applications: A Practical Approach to Mitigating OWASP Top 10 Vulnerabilities," *VFAST* vol. 13, no. 2, pp. 273–291, 2025, doi: 10.21015/vtse.v13i2.2145.

[6] Khanum, A., Qadir, S., & Jehan, S. A. Khanum, S. Qadir and S. Jehan, "OWASP-Based Assessment of Web Application Security," *Proc. 18th IEEE International Conference on Emerging Technologies (ICET), Peshawar, Pakistan,* pp. 240–243, 2023, doi: 10.1109/ICET59753.2023.10374730.

[7] A. M. Irzan and E. Sulistiyani, "Owasp Zap vs Arachni: Which One is Better in Vulnerability Assessment?," *Proc. 9th International Conference on Informatics and Computing (ICIC),* pp. 1–6, 2024, doi: 10.1109/ICIC64337.2024.10956935.

[8] M. Yunus, "Analisis Kerentanan Aplikasi Berbasis Web Menggunakan Kombinasi Security Tools Project Berdasarkan Framework OWASP," *JIIK*, vol. 24, no. 1, pp. 37-48, 2019, doi: 10.35760/ik.2019.v24i1.1988.

[9] M. H. Asep, D. Rifansyah, and D. F. Priambodo, "Penetration Testing Web XYZ Berdasarkan OWASP Risk Rating," *Teknika*, vol. 12, no. 1, pp. 33-46, 2023, doi: 10.34148/teknika.v12i1.571.

[10] G. Bennett, T. Hall, E. Winter, and S. Counsell, "Improving the Limited Performance of Static Application Security Testing (SAST) Tools," *EASE*, pp. 614-623, 2024, doi: 10.1145/3661167.366126

[11] C. Aparo, C. Bernardeschi, G. Lettieri, F. Lucattini and S. Montanarella, "An Analysis System to Test Security of Software on Continuous Integration-Continuous Delivery Pipeline," *IEEE*, pp. 58-67, 2023, doi: 10.1109/EuroSPW59978.2023.00012.

[12] A. Choiriyah and N. Qomariasih, "Security Analysis on Websites belonging to the Health Service Districts in Indonesia based on the Open Web Application Security Project (OWASP) Top 10 2021," *IEEE International Conference 2024*, doi: doi.org/10.1109/ICITCOM60176.2023.10442816.

[13] Nurbojatmiko, A. Lathifah, F. Bil Amri and A. Rosidah, "Security Vulnerability Analysis of the Sharia Crowdfunding Website Using OWASP-ZAP," *IEEE International Conference*, 2022, doi: doi.org/10.1109/CITSM56380.2022.9935837.

[14] M. Noman, M. Iqbal, and A. Manzoor, "A Survey on Detection and Prevention of Web Vulnerabilities," *International Journal of Advanced Computer Science and Applications*, 2020, 11(6), doi:https://doi.org/10.14569/IJACSA.2020.0110665.

[15] M. Aydos, Ç. Aldan, E. Coşkun, and A. Soydan, "Security testing of web applications: A systematic mapping of the literature," *IIETA*, vol. 34, no. 9, pp. 6775-6792, 2019, doi: doi.org/10.1016/j.jksuci.2021.09.018.

[16] M. A. A. Hilmi, A. Puspaningrum, Darsih, D. O. Siahaan, H. S. Samosir and A. S. Rahma, "Research Trends, Detection Methods, Practices, and Challenges in Code Smell:SLR", *IEEE Access*, 11, 129536-129551, 2023, doi:doi.org/10.1109/ACCESS.2023.3334258.

[17] I. R. Onyenweaku, M. S. Brown, M. Pelosi, and M. H. Shahine, "A sonarqube static analysis of the spectral workbench," *International Journal of Natural Science and Reviews*, p. 16, 2021, doi: 10.28933/ijnsr-2020-12-0605.

[18] J.-A. del-Hoyo-Gabaldon, A. Moreno-Cediel, E. Garcia-Lopez, A. Garcia-Cabot, and D. de-Fitero-Dominguez, "Automatic dataset generation for automated program repair of bugs and vulnerabilities through SonarQube," *SoftwareX*, Vol. 26, 2024, doi: doi.org/10.1016/j.softx.2024.101664.

[19] F. Lomio, S. Moreschini, and V. Lenarduzzi, "A machine and deep learning analysis among SonarQube rules, product, and process metrics for fault prediction," *ESE*, vol. 27, no. 189, pp. 1-57, 2022, doi: 10.1007/s10664-022-10164-z.

[20] D. Murtaza, R. Haider and F. Khan,"Comprehensive Security Analysis and Threat Mitigation Strategies for React.js Applications: Leveraging SonarQube for Robust Security Assurance," *IEEE 1st Karachi Section Humanitarian Technology Conference, Khi-HTC*, 2024, doi: doi.org/10.1109/KHI-HTC60760.2024.10482157.

[21] D. Priyawati, S. Rokhmah, and I. C. Utomo, "Website Vulnerability Testing and Analysis of Internet Management Information System Using OWASP," *In International Journal of Computer and Information System (IJCIS) Peer Reviewed-International Journal* (Vol. 03, Issue 03), 2022, e-ISSN: 2745-9659.

[22] M. D. Fadilah and S. Rochimah, "Security Evaluation of Insurance Portal Agency Information System Based on ISO/IEC 25010 Quality Standard Utilizing OWASP ZAP," *3rd International Conference on Intelligent Cybernetics Technology and Applications, ICICyTA 2023*, pp. 352–357, 2023, doi: 10.1109/ICICyTA60173.2023.10428701.

[23] S. Alazmi and D. C. de Leon, "Customizing OWASP ZAP: A Proven Method for Detecting SQL Injection Vulnerabilities," *Proceedings - 2023 IEEE 9th International Conference on Big Data Security on Cloud, IEEE International Conference on High Performance and Smart Computing, and IEEE International Conference on Intelligent Data and Security, BigDataSecurity-HPSC-IDS 2023*, pp. 102–106, 2023, doi: doi.org/10.1109/BigDataSecurity-HPSC-IDS58521.2023.00028.

[24] L. H. Riberu and A. W. R. Emanuel, "Vulnerability Testing and Analysis Using OWASP Top 10 on Academic Information System at University XYZ," *ICAAEEI,* 2024, doi: doi.org/10.1109/ICAAEEI63658.2024.10899162.

[25] R. Duraz, D. Espes, J. Francq, S. Vaton, "Using CVSS scores can make more informed and more adapted Intrusion Detection Systems," *Journal of Universal Computer Science*, 30(9), pp. 1244–1264, 2024, doi: doi.org/10.3897/jucs.131659.

[26] A. Balsam, M. Nowak, M. Walkowski, J. Oko and S. Sujecki, "Comprehensive comparison between versions CVSS v2.0, CVSS v3.x and CVSS v4.0 as vulnerability severity measures," *24th International Conference on Transparent Optical Networks (ICTON)*, Bari, Italy, 2024, pp. 1-4, doi: doi.org/10.1109/ICTON62926.2024.10647452.

ISSN: 2580-409X (Print) / 2549-6824 (Online)
DOI: https://doi.org/10.29407/intensif.v10i1.25294

[27] FIRST, "Common Vulnerability Scoring System v3.1: Specification Document," *Forum of Incident Response and Security Teams*. [Online]. Available: https://www.first.org/cvss/v3-1/specification-document. [Accessed: Dec. 7, 2024].

[28] NIST, "CVSS v3.1 calculator," *National Institute of Standards and Technology*. [Online]. Available: https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator. [Accessed: Dec. 7, 2024].

[29] P. Mell, J. Spring, D. Dugal, S. Ananthakrishna, F. Casotto, T. Fridley, C. Ganas, A. Kundu, P. Nordwall, V. Pushpanathan, D. Sommerfeld, M. Tesauro, and C. Turner, *"*Measuring the Common Vulnerability Scoring System base score equation,*" NIST*, no. 8409, pp. 1-43, 2022, doi: doi.org/10.6028/NIST.IR.8409.

[30] S. K. Lala, A. Kumar and S. T., "Secure Web development using OWASP Guidelines," *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*, Madurai, India, 2021, pp. 323-332, doi: doi.org/10.1109/ICICCS51141.2021.9432179.