Progresif: Jurnal Ilmiah Komputer
https://ojs.stmik-banjarbaru.ac.id/index.php/progresif/index
Jl. Ahmad Yani, K.M. 33,5 - Kampus STMIK Banjarbaru
Loktabat – Banjarbaru (Tlp. 0511 4782881), e-mail: puslit.stmikbjb@gmail.com
e-ISSN: 2685-0877

Security Assessment of Open-Source Village Governance Systems: A Case Study of OpenSID

DOI: http://dx.doi.org/10.35889/progresif.v22i1.3560

Ikhsan Fanani[1*], Nana Sujana[2], Muhamad Hilmansyah Susanta[3]
Teknologi Komputer, Politeknik Pajajaran ICB Bandung, Bandung, Indonesia
*e-mail *Corresponding Author:* ikhsan.fanani@poljan.ac.id

Abstract
Indonesia's OpenSID platform manages sensitive citizen data across thousands of rural administrative units, yet no empirical security assessment exists in academic literature. This study addresses this gap through comprehensive security evaluation using Static Application Security Testing (SAST) and Software Composition Analysis (SCA), with findings mapped to OWASP Top 10 and scored using CVSS v3.1. Analysis identified 402 raw findings, with 170 (42.3%) confirmed as true positives after manual validation. Broken Access Control (105 findings) and Injection vulnerabilities (26 findings) were predominant, with seven Critical or High severity issues detected, including path traversal and known CVE dependencies. The 57.7% false positive rate emphasizes the necessity of manual validation alongside automated scanning. This research provides the first structured security audit of Indonesian governance software and recommends adopting GitHub-native security tools and formal vulnerability disclosure policies.
Keywords: *OpenSID; Security Assessment; OWASP Top 10; Static Application Security Testing; Village Information System*

Abstrak
Platform OpenSID Indonesia mengelola data sensitif warga di ribuan unit administrasi pedesaan, namun belum ada penilaian keamanan empiris dalam literatur akademik. Penelitian ini mengisi kesenjangan tersebut melalui evaluasi keamanan komprehensif menggunakan *Static Application Security Testing* (SAST) dan *Software Composition Analysis* (SCA), dengan temuan dipetakan ke OWASP Top 10 dan dinilai menggunakan CVSS v3.1. Analisis mengidentifikasi 402 temuan mentah, dengan 170 (42,3%) dikonfirmasi sebagai *true positive* setelah validasi manual. *Broken Access Control* (105 temuan) dan kerentanan *Injection* (26 temuan) mendominasi, dengan tujuh masalah tingkat keparahan *Critical* atau *High* terdeteksi, termasuk path traversal dan dependensi CVE yang diketahui. Tingkat *false positive* 57,7% menekankan pentingnya validasi manual bersama pemindaian otomatis. Riset ini menyediakan audit keamanan terstruktur pertama untuk perangkat lunak tata kelola Indonesia dan merekomendasikan adopsi tools keamanan GitHub-native serta kebijakan pengungkapan kerentanan formal.
Kata kunci: *OpenSID; Asesmen Keamanan; OWASP Top 10; Static Application Security Testing; Sistem Informasi Desa*

1. Introduction

The security of digital government platforms has emerged as a critical concern in the global push toward e-governance. As public services transition to digital systems, the volume of sensitive citizen data—including personally identifiable information (PII), financial records, and administrative correspondence—continues to grow. Research on e-government cybersecurity has established that digitization, particularly in developing countries, poses significant challenges in protecting sensitive information from escalating cyber threats [1]. In the context of village-level governance, these concerns are amplified by the typical absence of dedicated IT security personnel and limited cybersecurity budgets. Indonesia’s Desa Digital initiative, which seeks to modernize administrative services across tens of thousands of villages, epitomizes both the promise and the peril of this transformation. The software security engineering literature emphasizes that security must be integrated throughout the software development lifecycle [2]; however, for community-driven open-source governance platforms operating under resource constraints, this principle is difficult to implement without empirical data on existing vulnerabilities.

At the core of Indonesia’s village digitization lies OpenSID (Sistem Informasi Desa), an open-source village information system maintained by the Perkumpulan Desa Digital Terbuka community [3]. The platform’s GitHub repository has accumulated over 1,200 stars and 1,100 forks, reflecting substantial adoption. Built on PHP with the CodeIgniter framework, OpenSID handles sensitive operations including citizen data management (national identity numbers, family records, land ownership data), correspondence generation, financial reporting, and statistical reporting. Prior studies have documented OpenSID’s deployment across diverse Indonesian villages as a foundational digital governance tool [4], [5], [6]. Despite this critical role, the platform has not been subjected to a formal security assessment in the academic literature. This is problematic because the citizen records constitute highly sensitive PII, the open-source codebase exposes internal logic to potential adversaries, and the platform is typically deployed without compensating security controls such as web application firewalls or intrusion detection systems. The combination of sensitive data, broad deployment, and limited security oversight creates a risk profile that demands empirical investigation.

Prior research on e-government security evaluation has employed various methodologies. Within Indonesia, studies have relied on dynamic testing approaches. Darojat et al. [7] assessed local and village government websites using NIST SP 800-115 and OWASP with web vulnerability scanners but did not examine source code. Fauzi et al. [8] analyzed village government website security using OWASP and PTES, identifying open directories and weak headers, though limited to black-box testing of a single instance. Ariyadi et al. [9], Hidayatulloh and Saptadiaji [10], and Dirgahayu et al. [11] similarly conducted penetration testing on government and academic websites using OWASP-based methodologies. Ghozali et al. [12] applied OWASP Risk Rating to assess web application vulnerabilities. A common limitation across these studies is their exclusive reliance on dynamic testing, which cannot systematically identify insecure coding patterns embedded in source code.

In the international literature, static application security testing (SAST) has demonstrated significant potential for source-code-level vulnerability detection. Nunes et al. [13] benchmarked multiple SAST tools for web security and found significant detection rate variation. Li et al. [14] compared SAST tools for Java, establishing that no single tool dominates across all vulnerability types. Bennett et al. [15] evaluated four SAST tools including Semgrep, finding detection rates of 11.2%–26.5% with defaults; custom Semgrep rules improved detection by 181% to 44.7%. Hashmat et al. [16] ran 24 tools across nearly 5,000 repositories, correlating higher criticality scores with more detected errors. For PHP specifically, Kree et al. [17] demonstrated 86% precision for Semgrep across seven OWASP classes in 300 PHP projects. Luo et al. [18] developed TChecker for PHP taint analysis, Medeiros and Neves [19] showed that coding style influences vulnerability detectability, and Al Azhar [20] achieved 88.8% accuracy for SQL injection detection in CodeIgniter. Bermejo Higuera et al. [21] demonstrated that combining SAST, DAST, and IAST significantly improves OWASP Top 10 detection coverage, while Elder et al. [22] confirmed that combining multiple techniques outperforms any single approach. On the software composition analysis (SCA) front, the Synopsys OSSRA report [23] found that 84% of codebases contained at least one known vulnerability. Cruz et al. [24] reviewed open-source SCA solutions, and Zahan et al. [25] highlighted cascading risks from vulnerable transitive dependencies.

Despite the maturity of these methods, a critical gap persists: no study has applied white-box static analysis and software composition analysis to an Indonesian community-driven governance platform. Indonesian e-government security research remains confined to dynamic testing, which cannot identify deeply embedded insecure coding patterns or vulnerable dependency chains. International SAST research, meanwhile, has not addressed the specific context of village governance systems in developing countries—systems maintained by volunteer communities under significant resource constraints.

This study addresses the identified gap by proposing a white-box security evaluation methodology combining Static Application Security Testing (SAST) using Semgrep with Software Composition Analysis (SCA) using Trivy, with findings mapped to the OWASP Top 10 (2025 revision) [26] and scored using CVSS v3.1 [27] and a probability-impact risk matrix. This integration overcomes the limitations of prior Indonesian e-government security studies by performing source-code-level analysis capable of detecting insecure coding patterns—injection flaws, path traversal, insecure function usage, and cryptographic weaknesses—invisible to network-level scanners. The SCA component addresses third-party dependency risks shown to affect the vast majority of modern codebases [23], [25] but previously unexamined in Indonesian governance software.

The novelty of this research is threefold. First, it provides the first empirical security audit of an Indonesian governance software platform in the academic record, complementing existing penetration-testing-based studies [7], [8], [9], [10], [11], [12] with a white-box perspective absent from this domain. Second, it quantifies OpenSID's vulnerability landscape using internationally recognized frameworks (OWASP Top 10 and CVSS v3.1), establishing a baseline for future improvements and cross-study comparison. Third, it demonstrates a reproducible methodology combining automated static analysis with manual validation—validated in recent research [15], [22]—adaptable to similar governance projects. The integration of SAST, SCA, and multi-framework risk scoring applied to a real-world governance platform represents a state-of-the-art contribution bridging mature security evaluation methodologies and the understudied domain of Indonesian village governance software.

2. Methodology

2.1. Subject of Study

The subject of this security assessment is OpenSID (Open Source Sistem Informasi Desa) platform [3], an open-source Village Information System developed and maintained by the Indonesian open-source community. The specific version analyzed in this study is version 2601.0.0 from the "Umum" branch, identified by commit hash cef7c2d. OpenSID is build using PHP with the CodeIgniter framework and incorporates various JavaScript libraries for frontend functionality. The platform includes modules for population management, correspondence handling, financial reporting, and geographic information system (GIS) capabilities. The "Umum" branch represents the publicly available version suitable for standard village administrative needs.

2.2. Tools and Techniques

This study employed two complementary security analysis tools to comprehensively assess the OpenSID platform. Semgrep, an open-source static analysis tool, operates through pattern-matching against a configurable set of security rules [15], [17]. Unlike traditional SAST tools that rely on full compilation or complex data-flow analysis, Semgrep uses a lightweight, syntax-aware approach that enables rapid scanning of large codebases. Its effectiveness for PHP vulnerability detection has been empirically validated, with Kree et al. [17] demonstrating 86% precision across seven OWASP classes in PHP applications. In this study, Semgrep was configured with community-maintained rule sets targeting common PHP and JavaScript vulnerabilities, including injection flaws, insecure function usage (such as eval() and exec()), path traversal, insecure deserialization, and cryptographic weaknesses. Each finding produced by Semgrep includes a confidence level (HIGH, MEDIUM, or LOW) reflecting the tool's certainty that the detected pattern represents a genuine vulnerability.

Complementing the static analysis, Trivy was utilized as an open-source vulnerability scanner developed by Aqua Security that specializes in identifying known vulnerabilities in operating system packages, container images, and application dependencies [24]. Trivy performed Software Composition Analysis (SCA) by scanning the project's composer.lock file to identify known Common Vulnerabilities and Exposures (CVEs) in third-party PHP libraries. Unlike SAST, which examines proprietary code for insecure patterns, SCA focuses on the supply chain risk introduced by external dependencies, a concern amplified by reports showing that the vast majority of modern codebases contain at least one vulnerable open-source component [23].

2.3. Evaluation Metrics

Three complementary evaluation frameworks were employed to categorize, score, and prioritize the identified findings.

OWASP Top 10. The OWASP Top 10 (2025 revision) [26] was used as the primary categorization framework. Each identified vulnerability was mapped to the most relevant OWASP category, enabling a structured overview of the types of security weaknesses present in the codebase, consistent with categorization approaches used in prior studies [12], [21]. For findings related to third-party dependencies identified prior to the 2025 revision, the corresponding 2021 categories were retained for accuracy.

CVSS v3.1. The Common Vulnerability Scoring System version 3.1 [27] was applied where applicable, particularly for findings with assigned CVE identifiers. CVSS provides a standardized numerical score (0.0 to 10.0) reflecting the severity of a vulnerability based on factors such as attack vector, complexity, privileges required, and impact on confidentiality, integrity, and availability.

Risk = Likelihood × Impact. For findings without formal CVE identifiers (i.e., the majority of SAST findings), risk was quantified using a probability-impact matrix [2], [12]. Likelihood and Impact were each assessed on a qualitative scale (Low = 1, Medium = 2, High = 3), yielding a Risk Score ranging from 1 (lowest) to 9 (highest). This approach enabled the prioritization of findings that lack standardized CVSS scores.

Table 1. Probability-Impact Risk Matrix

	Low Impact (1)	Med. Impact (2)	High Impact (3)
Low Likelihood (1)	1	2	3
Med. Likelihood (2)	2	4	6
High Likelihood (3)	3	6	9

2.4. Validation Process

All raw findings were subjected to manual validation to distinguish true positives from false positives, following the empirical validation practices recommended in the SAST evaluation literature [15], [22]. The validation process involved: (1) tracing reported variables to their source; (2) checking for the presence of sanitization or validation functions (e.g., htmlspecialchars(), basename()); (3) verifying whether input was validated against a whitelist; (4) examining framework-level protections such as CodeIgniter middleware and CSRF tokens; (5) evaluating the execution context (e.g., whether the code was reachable only by authenticated administrators); and (6) confirming that the flagged code was reachable in a production deployment. Common false positive patterns identified included variables sourced from database identifiers rather than user input, hardcoded values erroneously flagged, routes protected by framework authentication middleware, and test or example code not present in production builds.

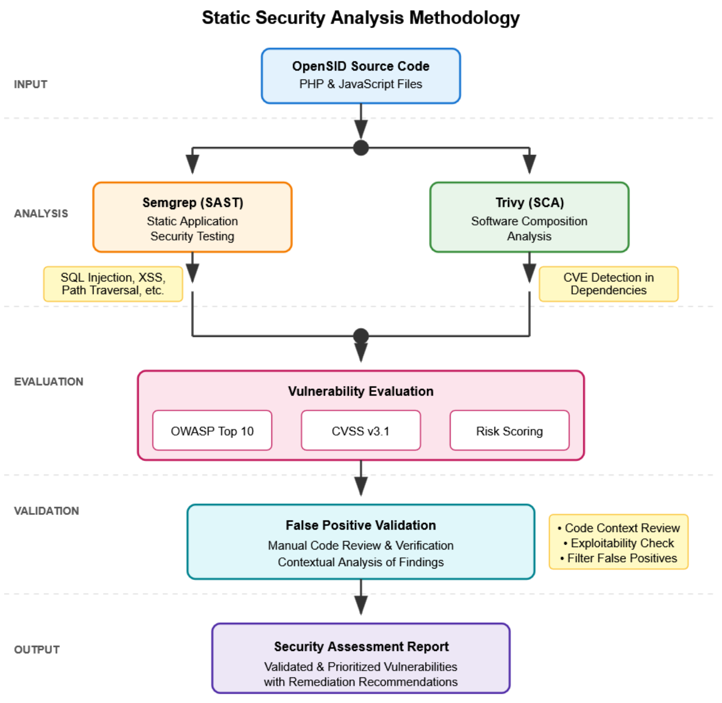
Static Security Analysis Methodology
INPUT
OpenSID Source Code
PHP & JavaScript Files
ANALYSIS
Semgrep (SAST)
Static Application Security Testing
Trivy (SCA)
Software Composition Analysis
SQL Injection, XSS, Path Traversal, etc.
CVE Detection in Dependencies
EVALUATION
Vulnerability Evaluation
OWASP Top 10
CVSS v3.1
Risk Scoring
VALIDATION
False Positive Validation
Manual Code Review & Verification
Contextual Analysis of Findings
• Code Context Review
• Exploitability Check
• Filter False Positives
OUTPUT
Security Assessment Report
Validated & Prioritized Vulnerabilities
with Remediation Recommendations

Figure 1. Static Security Analysis Methodology

3. Result and Discussion

3.1. Vulnerability Overview

The combined scanning effort using Semgrep and Trivy produced a total of 402 raw findings. Of these, 400 originated from Semgrep's static pattern-matching analysis of the PHP and JavaScript source code, and 2 originated from Trivy's dependency scan of the composer.lock manifest. The distribution of findings priority level and tool is summarized in Table 2 and Table 3.

Table 2. Security Findings by Source Tool

Tool	Findings Count	Percentage
Semgrep (SAST)	400	99.50%
Trivy (SCA)	2	0.50%
Total	**402**	**100%**

Table 3. Security Findings by PriorityTool

Priority	Count	Percentage
Critical	10	2.49%
High	3	0.75%
Medium	80	19.90%
Low	309	76.87%
Total	**402**	**100%**

The predominance of Semgrep findings (99.50%) reflects the comprehensive nature of static code analysis compared to dependency scanning. The relatively low number of dependency vulnerabilities (2) suggests that the OpenSID project maintains reasonably current dependencies, though the identified vulnerabilities are significant in severity.

3.2. False Positive Analysis

Manual validation determined that 232 of the 402 raw findings (57.7%) were false positives, leaving 170 confirmed true positives (42.3%). The false positive distribution was not uniform across confidence levels. As shown in Table 4, findings with HIGH confidence exhibited a substantially lower false positive rate (41.7%) compared to those with LOW confidence (61.2%), validating the utility of the tool's confidence scoring as a triage mechanism.

Table 4. Semgrep Findings by Confidence Level

Confidence	Count	Percentage
HIGH	10	2.50%
MEDIUM	52	13.00%
LOW	338	84.50%
Total	**400**	**100%**

The overall false positive rate of 57.7% is within the range commonly reported in the literature for SAST tools. Bennett et al. [15] observed that individual SAST tools detected between 11.2% and 26.5% of known vulnerabilities using default configurations, with substantial variation between tools. Nunes et al. [13] similarly reported significant false positive rates in their benchmarking study of static analysis tools for web security. The most common false positive patterns observed in this study included: file operations on paths derived from database records rather than user-controllable input; administrative routes protected by authentication middleware that the static analyzer could not model; bidirectional Unicode character detections in internationalization files (Arabic locale data); and hardcoded bcrypt hashes used as default seeds in migration scripts.

3.3. OWASP Top 10 Breakdown

After false positive removal, the 170 confirmed true positives were distributed across the OWASP Top 10 categories as shown in Table 5. Broken Access Control (A01) was the dominant category, accounting for 61.8% of all confirmed findings. Injection (A05) was the second most prevalent at 15.3%, followed by Cryptographic Failures (A04) at 8.8%.

Table 5. Validated Findings by OWASP Top 10 Category

OWASP Category	Count	%
A01:2025 - Broken Access Control	105	61.8%
A05:2025 - Injection	26	15.3%
A04:2025 - Cryptographic Failures	15	8.8%
A08:2025 - Software or Data Integrity Failures	9	5.3%
A02:2025 - Security Misconfiguration	9	5.3%
A03:2025 - Software Supply Chain Failures	2	1.2%
A07:2025 - Authentication Failures	1	1.2%
A06:2025 - Insecure Design	1	0.6%
N/A (Informational)	2	0.6%
Total	**170**	**100%**

The prevalence of Broken Access Control findings (61.8% of confirmed true positives) mirrors global trends reported in the OWASP Top 10 [26], where A01 has consistently been the most prevalent vulnerability category. This is primarily attributable to the widespread use of the unlink() function (76 true positive instances) and tainted filename operations (24 instances) across the codebase. While many of these operations may be mitigated by the application's authentication layer in practice, the static analyzer correctly flags them as potential risks in the event that access control mechanisms are bypassed or misconfigured. The Injection category is driven by the use of eval() (16 instances) and exec() (6 instances) functions, which represent inherently dangerous programming patterns, consistent with the findings of Medeiros and Neves [19] on how coding patterns influence vulnerability profiles in PHP applications.

3.4. Critical and High Severity Findings

Five findings were confirmed as true positives with Critical or High priority after manual validation. These findings warrant immediate attention from the development team and are detailed in Table 6.

Table 6. Critical and High-Severity True Positive Findings

Priority	Vulnerability Type	OWASP	Risk
Critical	Base Convert Precision Loss (2)	A02:2025	1
Critical	Tainted Path Traversal (3)	A01:2025	4
High	Tainted URL Connection (1)	A01:2025	6
High	CVE-2025-64500 (1)	A03:2025	9
Medium	CVE-2026-24739 (1)	A03:2025	9

Three of the five Critical true positives are tainted path traversal vulnerabilities located within the Responsive File Manager component. These findings involve user-controllable input reaching file system operations without adequate sanitization. In a worst-case scenario, an attacker could exploit these vulnerabilities to read arbitrary files on the server (including configuration files containing database credentials), upload malicious files to arbitrary directories, or delete critical system files. The concentration of high-severity findings in this single module suggests that the Responsive File Manager component introduces disproportionate risk to the platform, a pattern observed in prior studies where third-party components were found to be responsible for outsized proportions of vulnerability exposure [23], [25].

Another critical finding involves a Server-Side Request Forgery (SSRF) pattern in which a user-supplied URL is passed directly to a connection function. This could allow an attacker to force the server to make requests to internal services, potentially bypassing firewalls and accessing resources on the internal network. The risk score of 6 (Medium Likelihood, High Impact) reflects the severity of this vector, particularly in deployment environments where the OpenSID server may have access to internal databases or administrative interfaces.

The highest individual risk score in the dataset (9, corresponding to High Likelihood and High Impact) was assigned to CVE-2025-64500, a known vulnerability in the symfony/http-foundation package identified through Trivy's SCA scan [28]. This vulnerability carries a CVSS score of 7.3 [27], placing it in the High severity range. A second dependency vulnerability, CVE-2026-24739 in symfony/process (CVSS 6.3, Medium severity), was also identified. Both findings underscore the importance of maintaining current dependency versions and monitoring vulnerability advisories for third-party libraries, consistent with the supply chain risk findings of the OSSRA report [23] and the dependency analysis research of Zahan et al. [25].

Additionally, findings related to the use of PHP's base_convert() function were identified, which can silently lose precision when handling large integers due to floating-point conversion. While the risk score is low (1), this represents a security misconfiguration that could lead to URL collision or identifier predictability in certain modules.

3.5. Discussion

The results reveal several important patterns with both practical and scientific significance. The dominance of Broken Access Control (61.8%) mirrors global OWASP Top 10 trends [26], consistent with Kree et al.'s [17] observations across 300 PHP projects. This finding contributes empirical evidence that OWASP vulnerability distributions are not merely artifacts of enterprise software development but reflect fundamental patterns in PHP web application construction that persist even in resource-constrained, volunteer-maintained governance codebases. The OpenSID codebase thus shares common security challenges with the broader PHP ecosystem, extending the generalizability of OWASP benchmarks to the understudied domain of community-driven governance software in developing countries.

The concentration of four out of seven Critical/High-severity findings in the rfm/ (Responsive File Manager) module provides granular empirical support for the theoretical concerns raised in the supply chain security literature [23], [25]. While prior studies have demonstrated dependency risks at a macro level, this finding refines the understanding of supply chain risk in open-source governance platforms: the dominant threat vector is not merely transitive dependency vulnerabilities but deeply integrated third-party modules whose security posture diverges from that of the host application. This has practical implications—the module should be subjected to dedicated review, sandboxed, or replaced—and conceptual implications for how security risk models should weight component integration depth in resource-constrained projects.

The 57.7% false positive rate, while within expected SAST ranges [15], [13], [29], contributes the first quantitative measurement for SAST applied to a PHP governance application with CodeIgniter-specific patterns. The identification of framework-specific false positive sources—middleware-protected routes, database-derived path variables, internationalization file detections—enriches the knowledge base on how PHP framework architectures interact with static analysis tools. This evidence is valuable for researchers and practitioners seeking to calibrate SAST rule sets for similar government applications, as emphasized by Elder et al. [22] regarding the necessity of manual validation for empirical rigor.

More broadly, this study provides a few addition to the existing literature. First, by introducing white-box static analysis to the Indonesian e-government security domain—previously confined to dynamic testing [7], [8], [9], [10], [11], [17]—it empirically demonstrates that source-code-level assessment reveals critical vulnerabilities (path traversal, injection patterns) invisible to network-level scanners, extending the conceptual understanding that comprehensive e-government security requires multi-layered evaluation combining static and dynamic techniques [21], [22]. Second, the integrated methodology—SAST, SCA, OWASP Top 10, CVSS v3.1, and probability-impact scoring within a single pipeline—constitutes a reproducible framework that future researchers can adopt for similar governance platforms worldwide, enabling cross-study comparison and broader evidence accumulation. Third, the identification of seven Critical/High-severity vulnerabilities in a platform deployed across thousands of villages without prior detection provides concrete empirical evidence for the development of e-government cybersecurity policies in the Global South, addressing the data scarcity noted by Figueroa et al. [1] and underscoring the urgency of integrating proactive security assessment into the governance software lifecycle.

4. Conclusion and Future Work

4.1. Conclusion

This study has presented a systematic security assessment of the OpenSID village governance platform [3], employing Semgrep for static application security testing [15], [17] and Trivy for software composition analysis [24]. The evaluation of the Umum branch at Version v2601.0.0 revealed 402 raw findings, of which 170 were confirmed as true positives after manual validation, yielding a false positive rate of 57.7%.

The confirmed findings were dominated by Broken Access Control vulnerabilities (105 findings, 61.8%), followed by Injection (26 findings, 15.3%) and Cryptographic Failures (15 findings, 8.8%). Seven findings were classified as Critical or High severity, with the most severe cluster located in the Responsive File Manager module (rfm/) and one high-risk dependency vulnerability in symfony/http-foundation (CVSS 7.3) [28].

This research fills a documented gap in the Indonesian academic literature by providing the first empirical security audit of a community-driven governance software platform tailored to the Indonesian context. The methodology employed—combining automated scanning with manual validation and risk quantification using the OWASP Top 10 [26], CVSS v3.1 [27], and a probability-impact matrix—provides a reproducible framework for future assessments of similar systems.

4.2. Recommendations

Based on the findings of this study, the following recommendations are proposed for the OpenSID development community.

Adopt GitHub-native security tooling. As a community-driven open-source project hosted on GitHub, OpenSID is well-positioned to leverage GitHub's built-in security features at no cost. The project should enable Dependabot alerts and security updates to automatically monitor composer.lock for known CVEs, addressing the supply chain vulnerabilities identified in this study (Section 4.4.3). Additionally, integrating Semgrep or CodeQL via GitHub Actions for automated code scanning on pull requests would enable contributors to identify insecure patterns before they are merged, without requiring a formal CI/CD infrastructure. These capabilities are freely available for public repositories and represent the most practical path to continuous security monitoring for the project [15], [24].

Establish a vulnerability disclosure policy. The project should publish a SECURITY.md file in the repository root, defining a clear process for external security researchers to report vulnerabilities responsibly. This is a widely-adopted practice in the open-source ecosystem and would formalize the ad hoc security discussions currently taking place across community forums. Given that OpenSID handles sensitive citizen data across thousands of villages, a structured disclosure process is essential for maintaining the trust of deploying administrations [1], [2].

Prioritize the Responsive File Manager module for security hardening. The rfm/ directory accounted for four of the seven Critical and High-severity findings (Sections 4.4.1 and 4.4.2). The development team should conduct a dedicated security audit of this module, with consideration given to replacing it with a more actively maintained and security-hardened file management alternative, or sandboxing it with stricter access controls and input validation.

Develop contributor security guidelines. A contributor-facing document outlining secure coding practices specific to the OpenSID codebase—such as mandatory use of parameterized queries, avoidance of eval() and exec(), and proper path sanitization—would help prevent the introduction of the vulnerability patterns identified in this study. This document could be integrated into the existing contribution workflow as a pull request checklist.

4.3. Future Work

Several avenues for future research emerge from this study. First, a Dynamic Application Security Testing (DAST) assessment using tools such as OWASP ZAP would complement the static analysis performed here by identifying runtime vulnerabilities not detectable through source code inspection alone [21]. Second, a comparative study across multiple versions of OpenSID would enable longitudinal analysis of the platform's security trajectory. Third, extending the assessment to the Premium branch and associated API endpoints would provide a more complete picture of the platform's attack surface. Fourth, conducting a deployment configuration review of live OpenSID instances (with appropriate ethical approvals) would reveal whether theoretical vulnerabilities are mitigated or exacerbated by real-world server configurations, following the assessment approaches documented by Darojat et al. [7] and Fauzi et al. [8]. Finally, a similar assessment methodology could be applied to other Indonesian governance platforms to benchmark the security maturity of the broader Desa Digital ecosystem.

4.4. Limitations

This study has several limitations that should be acknowledged. The assessment was limited to static analysis and did not include dynamic testing or penetration testing of running instances. The evaluation focused on a single version (v2601.0.0) and may not reflect the security posture of earlier or subsequent releases. False positive validation, while systematic, involved subjective judgment that may introduce classification errors. Additionally, the study did not assess security configurations of deployed instances, which may vary significantly from the default codebase.

References

[1] V. Figueroa, L. E. S. Crespo, A. Santos-Olmo, D. G. Rosado and E. Fernández-Medina, "Building a holistic cybersecurity framework for e-Government based on a systematic analysis of proposals," *International Journal of Information Security,* vol. 24, no. 3, p. 121, 2025.

[2] G. McGraw, Software Security: Building Security In, Addison-Wesley, 2006.

[3] Perkumpulan Desa Digital Terbuka, "OpenSID Repository," [Online]. Available: https://github.com/OpenSID/OpenSID. [Accessed 23 January 2026].

[4] L. Hadjaratie, R. Yusuf, M. Polin, A. Lahinta, A. Dwinanto, M. Mokoginta and M. A. R. N. Fauzan, "Sosialisasi dan Pelatihan Aplikasi Sistem Informasi Desa Berbasis Web Menggunakan OpenSID di Desa Bilolantunga," *Devotion,* vol. 2, no. 2, pp. 18-22, 2023.

[5] C. Rahmad, A. D. W. Sumari, A. P. Kirana, M. Z. Abdullah and S. E. Sukmana, "Penerapan Sistem Informasi Administratif Desa Ngijo Kecamatan Karangploso Kabupaten Malang menggunakan OpenSID," *Bhakti Persada,* vol. 8, no. 1, pp. 1-8, 2022.

[6] R. Fitri, A. N. Asyikin and A. S. B. Nugroho, "Pengembangan Sistem Informasi Desa untuk Menuju Tata Kelola Desa yang Baik Berbasis TIK," *Positif,* vol. 3, no. 2, pp. 99-105, 2017.

[7] E. Z. Darojat, E. Sediyono and I. Sembiring, "Vulnerability Assessment Website E-Government dengan NIST SP 800-115 dan OWASP Menggunakan Web Vulnerability Scanner," *JSINBIS (Jurnal Sistem Informasi Bisnis),* vol. 12, no. 1, pp. 36-44, 2022.

[8] R. M. Fauzi, R. Hermawan, D. R. Adhy and S. Maesaroh, "Analisis Kerentanan Keamanan Web Menggunakan Metode OWASP Dan PTES di Web Pemerintahan Desa XYZ," *Power Elektronik,* vol. 13, no. 2, pp. 225-231, 2024.

[9] T. Ariyadi, H. Fadli, T. Akbar and M. B. Prihandoko, "Implementasi OWASP untuk Analisis Kerentanan dan Keamanan pada Sistem Informasi Akademik Terintegrasi Universitas Bina Darma," *STORAGE,* vol. 4, no. 1, pp. 1-7, 2025.

[10] S. Hidayatulloh and D. Saptadiaji, "Penetration Testing pada Website Universitas ARS Menggunakan Open Web Application Security Project (OWASP)," *Jurnal Algoritma,* vol. 19, no. 1, pp. 77-86, 2021.

[11] R. T. Dirgahayu, Y. Prayudi and A. Fajaryanto, "Penerapan Metode ISSAF dan OWASP versi 4 Untuk Uji Kerentanan Web Server," *Jurnal Ilmiah NERO,* vol. 1, no. 3, pp. 190-197, 2015.

[12] B. Ghozali, Kusrini and Sudarmawan3, "Mendeteksi Kerentanan Keamanan Aplikasi Website Menggunakan Metode Owasp (Open Web Application Security Project) untuk Penilaian Risk Rating," *Citec Journal,* vol. 4, no. 4, pp. 264-275, 2017.

[13] P. Nunes, I. Medeiros, J. Fonseca, N. Neves, M. Correia and M. Vieira, "Benchmarking Static Analysis Tools for Web Security," *IEEE Transactions of Reliability,* vol. 67, no. 3, pp. 1159-1175, September 2018.

[14] K. Li, S. Chen, L. Fan, R. Feng, H. Liu, C. Liu, Y. Liu and Y. Chen, "Comparison and Evaluation on Static Application Security Testing (SAST) Tools for Java," in *Proceedings of the 31st ACM Joint European So□ware Engineering Conference and Symposium on the Foundations of So□ware Engineering*, San Francisco, 2023.

[15] G. Bennett, T. Hall, E. Winter and S. C. A. S. T. (. Tools, "Semgrep*: Improving the Limited Performance of Static," in *Proceedings of the 28th International Conference on Evaluation and Assessment in So□ware Engineering*, Salerno, Italy, 2024.

[16] F. Hashmat, Z. A. Aljaali, M. Shen and A. Machiry, "Insights from Running 24 Static Analysis Tools on Open Source Software Repositories," in *International Conference on Information Systems Security*, Jaipur, India, 2024.

[17] L. Kree, R. Helmke and E. Winter, "Using Semgrep OSS to Find OWASP Top 10 Weaknesses in PHP Applications: A Case Study," in *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2024)*, Lausanne, Switzerland, 2024.

[18] C. Luo, P. Li and W. Meng, "TChecker: Precise Static Inter-Procedural Analysis for Detecting Taint-Style Vulnerabilities in PHP Applications," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CA, Los Angeles, USA, 2022.

[19] I. Medeiros and N. Neves, "Effect of Coding Styles in Detection of Web Application Vulnerabilities," in *2020 16th European Dependable Computing Conference (EDCC)*, Munich, Germany, 2020.

[20] M. F. A. Azhar and R. Harwahyu, "Detection of SQL Injection Vulnerability in CodeIgniter Framework Using Static Analysis," *Multitek Indonesia,* vol. 17, no. 1, pp. 69-78, July 2023.

[21] F. M. Tudela, J.-R. B. Higuera, J. B. Higuera, J.-A. S. Montalvo and M. I. Argyros, "On Combining Static, Dynamic and Interactive Analysis Security Testing Tools to Improve OWASP Top Ten Security Vulnerability Detection in Web Applications," *Applied Sciences,* vol. 10, no. 24, p. 9119, December 2020.

[22] S. Elder, N. Zahan, R. Shu, M. Metro, V. Kozarev, T. Menzies and L. Williams, "Do I Really Need All This Work to Find Vulnerabilities? An Empirical Case Study Comparing Vulnerability Detection Techniques on a Java Application," *Empirical Software Engineering,* vol. 27, no. 6, p. 154, 2022.

[23] Synopsys, "2024 Open Source Security and Risk Analysis (OSSRA) Report," Synopsys Cybersecurity Research Center, 2024.

[24] D. B. Cruz, J. R. Almeida and J. L. Oliveira, "Open Source Solutions for Vulnerability Assessment: A Comparative Analysis," *IEEE Access,* vol. 11, pp. 100234-100255, September 2023.

[25] N. Zahan, T. Zimmermann, P. Godefroid, B. Murphy, C. Maddila and L. Williams, "Whatare WeakLinks in the npm Supply Chain?," in *Proceedings of the 44th International Conference on So□ware Engineering: Software Engineering in Practice*, Pennsylvania, Pittsburgh, 2022.

[26] OWASP Foundation, "OWASP Top 10:2025 - The Ten Most Critical Web Application Security Risks," [Online]. Available: https://owasp.org/Top10/2025/. [Accessed 23 January 2026].

[27] First.org, "Common Vulnerability Scoring System v3.1: Specification Document," 2019. [Online]. Available: https://www.first.org/cvss/v3.1/specification-document. [Accessed 23 January 2026].

[28] NIST, "National Vulnerability Database," 2024. [Online]. Available: https://nvd.nist.gov. [Accessed 23 January 2026].

[29] S. Lipp, S. Banescu and A. P. A. f. V. Detection, "AnEmpirical Study on the Effectiveness of Static C Code," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysys*, South Korea (Virtual), 2022.