Vol. 8, No. 2, April 2026

ISI Journal of Information Systems and Informatics

Published By Asosiasi Doktor Sistem Informasi Indonesia

Security Analysis of Indonesian Region Government Web Applications Based on NIST SP 800-115 and WSTG v4.2

Arizal[1], Muhammad Hilal[2], Dimas Febriyan Priambodo[3]

[1,2,3]Cybersecurity Department, National Cyber and Crypto Polytechnic, Bogor, Indonesia

Received:
October 3, 2025
Revised:
March 11, 2026
Accepted:
March 27 2026
Published:
April 12, 2026

Corresponding Author:
Author Name*:
Dimas Febriyan Priambodo
Email*:
dimas.febriyan@poltekssn.ac.id

DOI:
10.63158/journalisi.v8i2.1558

Abstract. The rapid adoption of e-government systems has increased the exposure of government web applications to cybersecurity threats with the lack of security-focused implementation. Previous studies on web application security assessment commonly using automated vulnerability scanners or validated with another tools, which may produce false positives and fail to provide comprehensive insights. This research addresses this limitation by conducting a structured and multi-target security assessment of regional government web applications. The assessment integrates a systematic penetration testing process with comprehensive web application security testing guidelines. Automated scanning using OWASP ZAP and Arachni was combined with manual validation to ensure the accuracy of findings. The results identified nine validated vulnerabilities in the government portal and public service applications, and ten vulnerabilities in the legal documentation system. A significant portion of initial findings were confirmed as false positives after manual verification, highlighting the limitations of automated tools. The most common vulnerabilities were related to security misconfigurations, including missing security headers, outdated JavaScript libraries, and insecure cookie settings that highlight on weak in configuration hygiene and dependency management in this regional government. This study also demonstrates that combining structured penetration testing with detailed validation provides a more accurate and reliable assessment of government web application security.

Keywords: Hybrid Pentesting, Penetration Testing, Vulnerability Assessment, Vulnerability Scoring, Web Aplication Security

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

1. INTRODUCTION

The adoption of e-government systems has substantially transformed the delivery of public services by enhancing accessibility, efficiency, and transparency [1]. Through the implementation of web-based platforms, government institutions are able to provide information and services in a more timely, organized, and responsive manner. This transformation supports the broader objective of digital governance, in which information technology is utilized to improve institutional performance and strengthen engagement between the government and the public. As the reliance on online platforms continues to increase, government web applications have become essential components in ensuring the continuity and quality of public service delivery.

Notwithstanding these advantages, the rapid development of government web applications is frequently not accompanied by the implementation of adequate security controls. This condition increases the susceptibility of such applications to a wide range of cyber threats. Government web applications often manage sensitive citizen information and support critical administrative functions, thereby making them attractive targets for malicious activities. Security incidents such as web defacement, data leakage, and injection-based attacks may not only disrupt service availability but also undermine public trust, damage institutional reputation, and compromise the integrity and confidentiality of government-managed information. Accordingly, information security must be regarded as a fundamental requirement in the development and operation of e-government systems.

The urgency of this issue is reflected in the 2022 Indonesian Cybersecurity Landscape Report [2], which indicates that web defacement constituted the highest number of incident notifications, amounting to 933 cases out of 1,433 total incidents. This figure demonstrates that web-based systems remain among the most frequently targeted components in Indonesia's cybersecurity landscape. In parallel, BSSN Regulation [3] stipulates that every Electronic-Based Government System (SPBE) must satisfy the principles of confidentiality, authenticity, integrity, non-repudiation, and availability. However, the occurrence of a web defacement incident on the JDIH page, as identified through interviews with the Dinas Komunikasi dan Informatika (Diskominfo) Office in a

Regency, suggests that the system has not yet fully complied with the required SPBE security standards and therefore requires further evaluation and improvement.

Previous studies on web application security assessment have predominantly relied on automated vulnerability scanning tools [4], [5]. These tools are valuable for initial identification because they enable rapid detection of common security weaknesses across web systems. Nevertheless, their effectiveness is limited by the potential generation of false positives and inconsistent findings, irrespective of whether the tools are commercial or open-source based [6], [7]. In addition, prior research has shown that different scanners possess heterogeneous detection capabilities, which may result in variations in the vulnerabilities identified across the same target systems [8]–[10]. Such inconsistencies indicate that automated scanning alone may not be sufficient to support a reliable and comprehensive security assessment.

Existing literature further emphasizes that the use of a single testing method is inadequate, as each approach tends to identify different classes of vulnerabilities. Therefore, the integration of automated and manual testing techniques is necessary to obtain findings that are more comprehensive, accurate, and dependable [11], [12]. A hybrid approach combining automated tools with manual validation has previously been implemented by Priambodo et al. [13]; however, the study was limited to a single target. In response to this limitation, the present study conducts a comprehensive security analysis of multiple government web applications based on the NIST SP 800-115 standard and incorporates detailed validation techniques derived from the WSTG v4.2 guide. By applying this combined framework, the study seeks to produce more reliable and actionable findings for improving the security posture of government web applications.

2. METHOD

This study employed a structured penetration testing methodology to assess the security posture of government web applications. The overall assessment process was designed in accordance with NIST SP 800-115 [14], which provides a technical framework for information security testing and evaluation. In this study, the assessment was organized into four main phases, namely planning, discovery, attack, and reporting. To strengthen

the technical depth of testing, the procedure also incorporated OWASP Web Security Testing Guide (WSTG) v4.2 as the primary testing catalog and manual validation reference [15]. This combination enabled the research to follow a standardized testing process while ensuring that vulnerability findings were examined using detailed and recognized web security test cases.

The study focused on three production e-government web applications managed by the Dinas Komunikasi dan Informatika (Diskominfo) in a Regency. These applications consisted of the Regency Government Portal (hereafter referred to as Target A), the Population Administration Service Web Application (Target B), and the JDIH Web Application (Target C). The selection of these three targets was intended to represent different categories of government digital services with varying business functions, user interactions, and levels of exposure to the public internet. Since each application serves a different operational purpose, the selected targets were considered suitable for obtaining a broader understanding of the security conditions of web-based government systems within the same institutional environment. The overall research design applied in this study is illustrated in Figure 1.

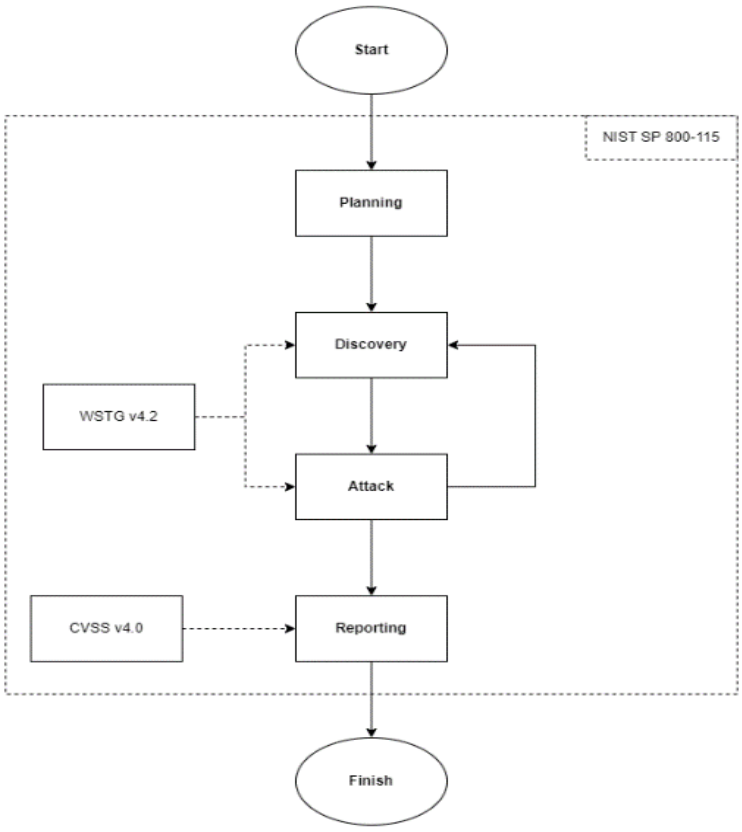

Figure 1. Research Design

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

2.1. Planning

The planning phase was conducted to establish the legal, technical, and operational boundaries of the security assessment before any testing activity was performed. At this stage, a formal rules of engagement agreement was established with Diskominfo to ensure that all testing activities were properly authorized and aligned with institutional requirements. The agreement defined the testing scope to include only Target A, Target B, Target C, and their respective subdomains. In addition, it specified the allowed testing period, particularly by scheduling activities outside peak service hours in order to minimize disruption to public services. Operational constraints were also clearly documented, including the application of rate limits for automated scans and the prohibition of destructive or service-disruptive testing techniques. These restrictions were important to ensure that the assessment remained controlled, ethical, and safe for production environments.

The planning stage also included the selection of tools and testing strategies appropriate for each phase of the assessment. Because previous studies have shown that different security tools often identify different sets of vulnerabilities, the toolset was selected to provide complementary coverage across multiple layers of the web application environment. The selected tools supported network discovery, technology fingerprinting, web crawling, vulnerability detection, and manual verification. This stage also involved identifying the types of evidence to be collected during testing, such as HTTP requests and responses, screenshots, proof-of-concept results, and system behavior observations. By defining the scope, limitations, and supporting tools in advance, the planning phase established a clear and consistent foundation for the subsequent discovery and validation activities.

2.2. Discovery

The discovery phase focused on identifying the attack surface and collecting technical information that could support subsequent security testing. This phase combined passive and active reconnaissance techniques to produce a more comprehensive understanding of the target applications. Passive information gathering was first conducted through publicly accessible sources, including search engines such as Google and the use of search operators to identify potential information leakage. This activity aimed to detect publicly indexed files, exposed directories, cached content, administrative pages, or other

digital traces that could reveal sensitive implementation details. Additional publicly available artifacts, including robots.txt, sitemap.xml, and accessible metadata from downloadable files, were examined to identify hidden paths, system structure, and functionality that might not be directly visible through normal browsing behavior.

Active discovery activities were then performed to identify the technologies, services, and exposed components used by each target application. Web server type and version information were examined using tools such as Nmap and Netcraft, while limited DNS-based enumeration techniques, including checks related to DNS Zone Transfer misconfiguration, were used to identify hidden assets or subdomains where permitted within the approved scope. At the web application level, OWASP ZAP and Arachni were used to perform crawling and automated vulnerability scanning in order to identify potential weaknesses in configuration, input handling, authentication mechanisms, and client-side behavior. Manual review of web page content and source code was also performed to detect information disclosure issues, hardcoded references, exposed comments, script dependencies, and implementation clues. Through this phase, an initial security profile was developed for each target, including its visible architecture, reachable components, and potential entry points for deeper verification.

2.3. Attack and Validation

The attack phase was designed not only to identify vulnerabilities but also to verify whether the issues reported by automated tools represented actual and exploitable security weaknesses. As shown in Figure 2, the findings generated during the discovery phase were not accepted directly as final results. Instead, each vulnerability identified by OWASP ZAP and Arachni was subjected to a structured triage and manual validation process. This procedure was necessary because prior studies have shown that automated vulnerability scanners may generate false positives and exhibit heterogeneous detection capabilities across different systems [8]. Accordingly, manual validation was used to improve the reliability of the findings and to ensure that only confirmed vulnerabilities were included in the final analysis.

The validation process was conducted in three main steps. First, each issue identified by automated scanning was reproduced manually using tools such as Burp Suite, browser developer tools, and command-line utilities, including SQLMap where relevant and

permitted. Manual retesting was guided by the applicable categories and procedures in WSTG v4.2 [15], so that each finding could be examined according to recognized web security testing practices. Second, controlled proof-of-concept (PoC) testing was carried out using safe and non-destructive payloads to determine whether the reported vulnerability could be exploited under real conditions without causing service interruption or data corruption. Third, the resulting application response was analyzed to determine the actual security impact of the issue, such as unauthorized access, information disclosure, injection behavior, session weakness, or the bypass of implemented security controls. This step was essential because a technically reproducible behavior does not always imply meaningful security impact, especially when compensating controls are present.

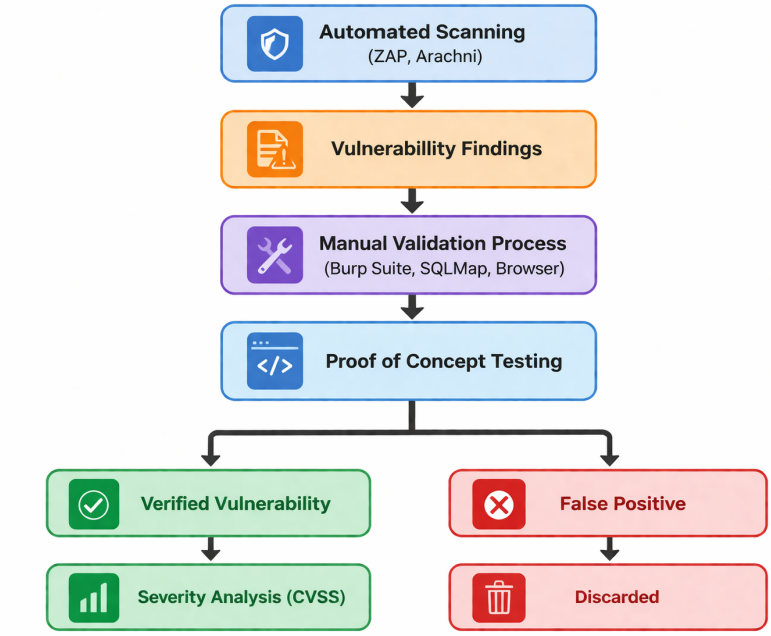

Figure 2. Attack and Validation Method

Based on this procedure, a vulnerability was classified as verified when manual testing successfully demonstrated exploitability or confirmed the absence of a required security control. In contrast, a finding was classified as a false positive when it could not be reproduced, did not lead to an observable security impact, or was effectively mitigated by existing protections in the target environment. This validation approach reduced the possibility of risk overestimation and ensured that the reported results reflected actual security conditions rather than scanner assumptions alone. By combining automated

detection with manual verification, the attack phase produced findings that were more accurate, defensible, and suitable for use in practical remediation planning.

2.4. Reporting

The reporting phase was conducted to consolidate all validated findings into a systematic and actionable security assessment report for Diskominfo in a Regency. Only vulnerabilities that had passed the manual verification process were included in the final report. Each verified issue was documented together with the affected target, vulnerable component, vulnerability category, technical description, proof-of-concept evidence, and observed impact on confidentiality, integrity, or availability. Wherever applicable, the report also included supporting evidence such as screenshots, HTTP request-response records, and concise reproduction steps to facilitate technical review by system administrators and developers. To maintain research ethics and institutional confidentiality, the targets were anonymized in the research narrative as Target A, Target B, and Target C.

To support risk prioritization, all verified vulnerabilities were assessed using the CVSS v4.0 scoring model, which was used to estimate severity by considering both exploitability and impact characteristics. The vulnerabilities were then categorized according to their severity levels in order to assist stakeholders in determining remediation priorities. In addition to severity scoring, remediation recommendations were provided for each issue by mapping the findings to the relevant WSTG category and proposing corrective actions such as configuration hardening, code-level fixes, input validation improvements, access control strengthening, or broader architectural controls. The final report therefore served not only as a record of identified weaknesses, but also as a practical decision-support document for improving the security posture of the evaluated government web applications.

3. RESULTS AND DISCUSSION

3.1. Discovery Phase

The discovery phase combined information gathering based on the WSTG v4.2 guidelines with automated vulnerability scanning to identify the attack surface of the three target applications. Search engine reconnaissance was performed using Google operators,

including the keyword "cache:", to determine whether sensitive content or previous snapshots of the applications had been indexed. This process revealed only one cached snapshot for Target C, dated 8 March 2024, 18:36:36 GMT, indicating limited historical exposure through search engine cache. In parallel, web server fingerprinting was conducted using Nmap, WhoisDomainTools, and Netcraft to identify the main domains, subdomains, and exposed services associated with Targets A, B, and C. This activity provided a baseline understanding of the external exposure of each application and supported subsequent validation activities.

A review of web server metafiles was also performed to detect potential information leakage. The results showed that Targets A and B exposed a robots.txt file that allowed web crawlers to access site content, while Target B additionally provided a sitemap.xml file to facilitate content indexing. In contrast, Target C did not expose a robots.txt file. None of the three targets provided security.txt or humans.txt, indicating the absence of publicly declared security contact information and developer metadata. Enumeration of applications on the web server was further attempted through DNS-related techniques. The DNS zone transfer test against Target A did not succeed, while Targets B and C did not expose usable NS records for the same procedure. Reverse DNS lookup showed that only Target A had a PTR record identifying a domain name. Reverse IP analysis using Myip.ms indicated that none of the three targets shared their IP address with other active websites, although the IP address of Target A was previously associated with another domain on 6 September 2021. Manual review of webpage content and client-side source code did not reveal sensitive information leakage across the three targets.

The architecture mapping stage showed that all three applications were deployed on Nginx-based infrastructure and relied on modern PHP/Laravel-related technologies as well as multiple third-party JavaScript libraries, as summarized in Table 1. Target B showed the richest framework composition, including Laravel, Livewire, and Alpine.js, whereas Target C also exposed OpenResty. Across the three targets, several external dependencies were identified, including jQuery, DataTables, Select2, SweetAlert, OWL Carousel, and other client-side components. No Web Application Firewall (WAF) was detected during fingerprinting. From a security perspective, these findings suggest that the primary attack surface of the evaluated systems was concentrated not at the network-service layer, but rather at the application layer, particularly in relation to

security headers, session management, and client-side dependency handling. Following the information-gathering process, automated vulnerability scanning was performed using OWASP ZAP and Arachni. OWASP ZAP was executed in automatic scanning mode using both the traditional spider and AJAX spider to follow static and JavaScript-generated links, followed by active scanning with a high alert threshold and maximum attack strength. Arachni was executed in its default mode, including dedicated checks for Cross-Site Scripting (XSS) and SQL Injection.

Table 1. Web Application Architecture Inspection Results

	Target A	**Target B**	**Target C**
Miscellaneous	PWA, Open Graph, dan Popper	Livewire	Popper
Security	-	-	HSTS
Font scripts	Google Font API, Font Awesome	Google Font API	Google Font API dan Font Awesome
CDN	Cloudflare dan cdnjs	-	Unpkg
Live chat	-	-	WhatsApp Business Chat
JavaScript libraries	Lodash, Axios, Selectize, SweetAlert, Select2, OWL Carousel, jQuery UI, jQuery, dan DataTables	Moment.js, Lozad.js, Highlight.js, Hammer.js, FilePond, Dropzone, core-js, Choices, jQuery, dan DataTables	Select2, Isotope, SweetAlert, OWL Carousel, jQuery, dan DataTables
Performance	-	Lozad.js	-
JavaScript frameworks	Vue.js	Alpine.js dan AlertifyJS	-
Web servers	Nginx	Nginx	Nginx dan OpenResty
Video players	-	Plyr	-
Web frameworks	-	Laravel dan Livewire	Laravel
Reverse proxies	Nginx	Nginx	Nginx
UI frameworks	Bootstrap	Bulma	Bootstrap
JavaScript graphics	Raphael dan Chart.js	D3 dan ApexCharts.js	Highcharts

	Target A	Target B	Target C
Programming languages	-	PHP	PHP
WAF	-	-	-

3.2. Attack and Validation Phase

All findings reported during automated scanning were subjected to manual verification to determine whether they represented actual exploitable vulnerabilities or merely scanner-generated false positives. Validation was performed by referring to the relevant WSTG v4.2 test cases and by using supporting tools such as SQLmap, Burp Suite, browser developer tools, Wappalyzer, and direct browser interaction. As summarized in Table 2, the validation results show that automated tools produced a meaningful number of false positives, particularly for findings initially categorized as high impact, such as SQL injection, application error disclosure, and several header-related weaknesses. This outcome reinforces the need for hybrid assessment, in which automated detection is followed by manual confirmation before a finding is accepted as evidence of security risk.

Table 2. Validation Results

Target	Vulnerability	Validation
Government Portal (Target A)	SQL Injection (CWE-89)	Not Verified
	Content Security Policy (CSP) Header Not Set (CWE-693)	Verified
	Vulnerable JS Library (CWE-829)	Verified
	Application Error Disclosure (CWE-200)	Not Verified
	Cookie No HttpOnly Flag (CWE-1004)	Verified
	Cookie Without Secure Flag (CWE-614)	Verified
	Cookie without SameSite Attribute (CWE-1275)	Verified
	Cross-Domain JavaScript Source File Inclusion (CWE-829)	Verified
	X-Content-Type-Options Header Missing (CWE-693)	Not Verified
	Absence of Anti-CSRF Tokens (CWE-352)	Verified
	Strict-Transport-Security Header Not Set (CWE-319)	Verified
	Common Directory (CWE-538)	Verified
	Password Field With Auto-Complete (CWE-200)	Not Verified
	Common Sensitive File (CWE-200)	Not Verified
Public Administration	SQL Injection (CWE-89)	Not Verified
	SQL Injection - Oracle - Time Based (CWE-89)	Not Verified

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

Target	Vulnerability	Validation
Service (Target B)	Content Security Policy (CSP) Header Not Set (CWE-693)	Verified
	Missing Anti-clickjacking Header (CWE-1021)	Verified
	Vulnerable JS Library (CWE-829)	Verified
	Big Redirect Detected (Potential Sensitive Information Leak) (CWE-201)	Not Verified
	Cookie No HttpOnly Flag (CWE-1004)	Verified
	Cookie Without Secure Flag (CWE-614)	Verified
	Private IP Disclosure (CWE-200)	Not Verified
	Server Leaks Version Information via "Server" HTTP Response Header Field (CWE-200)	Verified
	X-Content-Type-Options Header Missing (CWE-693)	Verified
	Strict-Transport-Security Header Not Set (CWE-319)	Verified
	Absence of Anti-CSRF Tokens (CWE-352)	Not Verified
	Common Sensitive File (CWE-200)	Not Verified
	Missing 'X-Frame-Options' header (CWE-693)	Verified
	Password Field With Auto-Complete (CWE-200)	Not Verified
JDIH (Target C)	Absence of Anti-CSRF Tokens (CWE-352)	Not Verified
	Content Security Policy (CSP) Header Not Set (CWE-693)	Verified
	Missing Anti-clickjacking Header (CWE-1021)	Verified
	Vulnerable JS Library (CWE-829)	Verified
	Application Error Disclosure (CWE-200)	Not Verified
	Big Redirect Detected (Potential Sensitive Information Leak) (CWE-201)	Not Verified
	Cookie No HttpOnly Flag (CWE-1004)	Verified
	Cookie Without Secure Flag (CWE-614)	Verified
	Cookie without SameSite Attribute (CWE-1275)	Verified
	Cross-Domain JavaScript Source File Inclusion (CWE-829)	Not Verified
	Timestamp Disclosure – Unix (CWE-200)	Not Verified
	X-Content-Type-Options Header Missing (CWE-693)	Verified
	Strict-Transport-Security Header Not Set (CWE-319)	Verified
	Password Field With Auto-Complete (CWE-200)	Not Verified
	Common Directory (CWE-538)	Verified
	Common Sensitive File (CWE-200)	Not Verified
	Missing 'X-Frame-Options' header (CWE-693)	Verified

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

The SQL Injection candidates reported for the personnel login endpoint of Target A were not validated. Verification referred to WSTG-INPV-05 and was performed using both SQLmap and manual payload testing against https://xyzkab.go.id/portal/loginpegawai, as illustrated in Figure 3 and Figure 4. Manual input using the payload admin' # and automated testing through SQLmap did not result in successful injection, unauthorized authentication, database interaction, or data exfiltration. The responses consistently returned 403 Forbidden, indicating that the endpoint was protected by access control or request filtering rather than being directly injectable. Consequently, the SQL injection alerts reported by OWASP ZAP and Arachni for this endpoint were classified as false positives. Similar results were also observed for Target B, including the Oracle time-based SQL injection alert, which could not be reproduced under controlled testing conditions. These findings indicate that automated SQL injection alerts are useful for triage, but they should not be interpreted as confirmed vulnerabilities in the absence of reproducible exploitability.

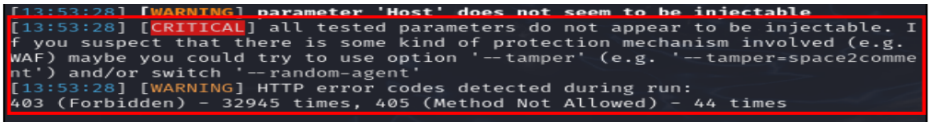

Figure 3. Validation Attack using SQLmap

Figure 4. Validation Attack using escape character

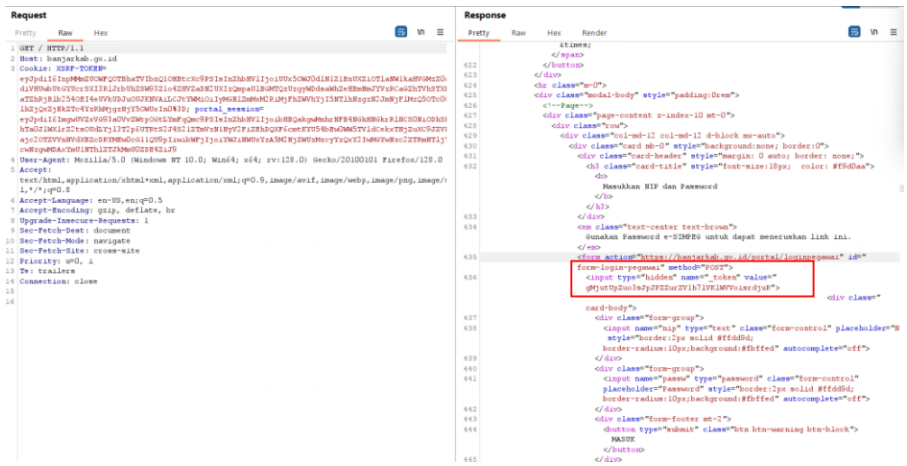

Figure 5. Validation Attack using Burbsuite for Anti-CSRF

Several configuration weaknesses and client-side security issues were, however, successfully validated. Verification using Burp Suite confirmed the absence of a Content-Security-Policy (CSP) header on the affected targets, thereby validating the corresponding scanner findings. In addition, the validation of vulnerable JavaScript libraries referred to WSTG-CLNT-02 and WSTG-INFO-08. Wappalyzer, as shown in Figure 6, identified jQuery v3.2.1 on Target A. According to the official jQuery website, a newer major version is available [16]. The Snyk Vulnerability Database also reports that jQuery v3.2.1 is associated with Cross-Site Scripting (XSS) and prototype pollution vulnerabilities [17], [18]. Further inspection showed that Target A loaded the external JavaScript file https://cdnjs.cloudflare.com/ajax/libs/jspdf/1.3.4/jspdf.min.js, as shown in Figure 10. According to Snyk, jsPDF v1.3.4 is affected by XSS and Regular Expression Denial of Service (ReDoS) vulnerabilities [19], [20]. These observations support the validation of both the Vulnerable JS Library finding and, for Target A, the Cross-Domain JavaScript Source File Inclusion finding.

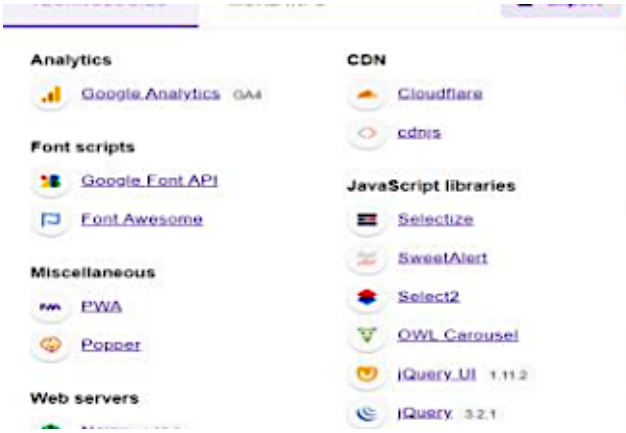

Figure 6. WAPanalyzer found jQuery version

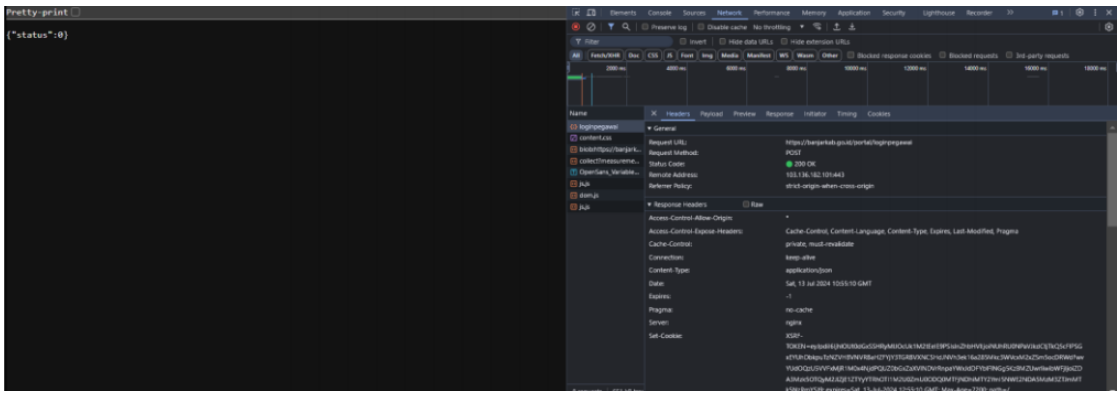

Figure 7. Target A inspect element

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

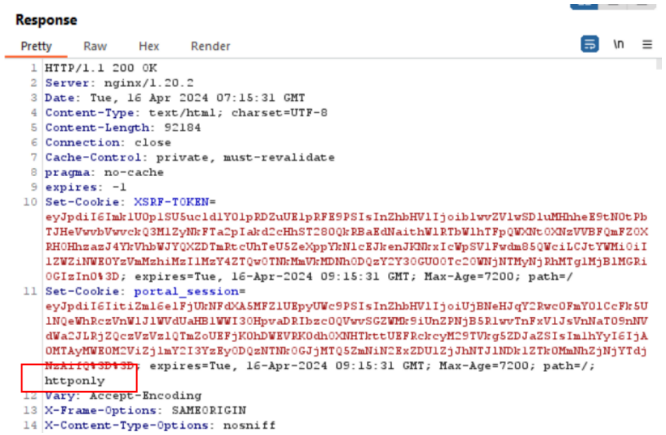

Figure 8. Burbsuite response for HttpOnly Flag on Target A

The validation process also confirmed a number of session-management and security-header weaknesses. Inspection using Burp Suite and browser storage analysis showed that several cookies were not consistently configured with the HttpOnly, Secure, and SameSite attributes, as illustrated in Figure 8 and Figure 9. These findings were therefore validated for the affected targets. Similarly, missing X-Frame-Options and anti-clickjacking protections were validated where the corresponding response headers were absent. For X-Content-Type-Options, the finding was not validated on Target A, because the intercepted response showed that the header had already been set to nosniff (Figure 11). However, the same issue remained verified for Targets B and C, where the header was not present. With respect to Strict-Transport-Security (HSTS), the issue remained verified whenever the HSTS header itself was absent, even if the application redirected HTTP requests to HTTPS. Redirection alone does not provide the browser-level persistence and downgrade protection required by HSTS. The validation of common directory exposure also confirmed that several publicly reachable paths, such as /db, /login, and /register, remained accessible on Target A, as shown in Figure 12, and were therefore retained as validated findings.

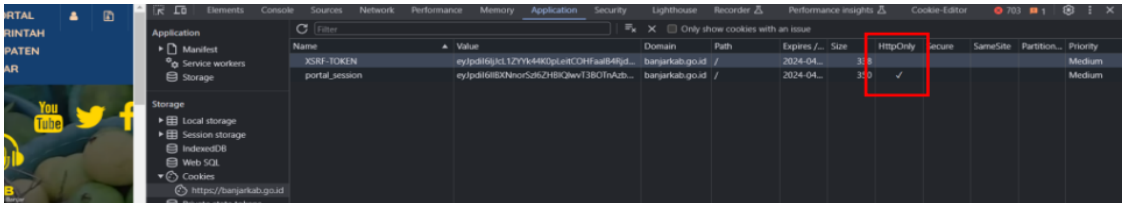

Figure 9. Inspect element for HttpOnly flag on Target A

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

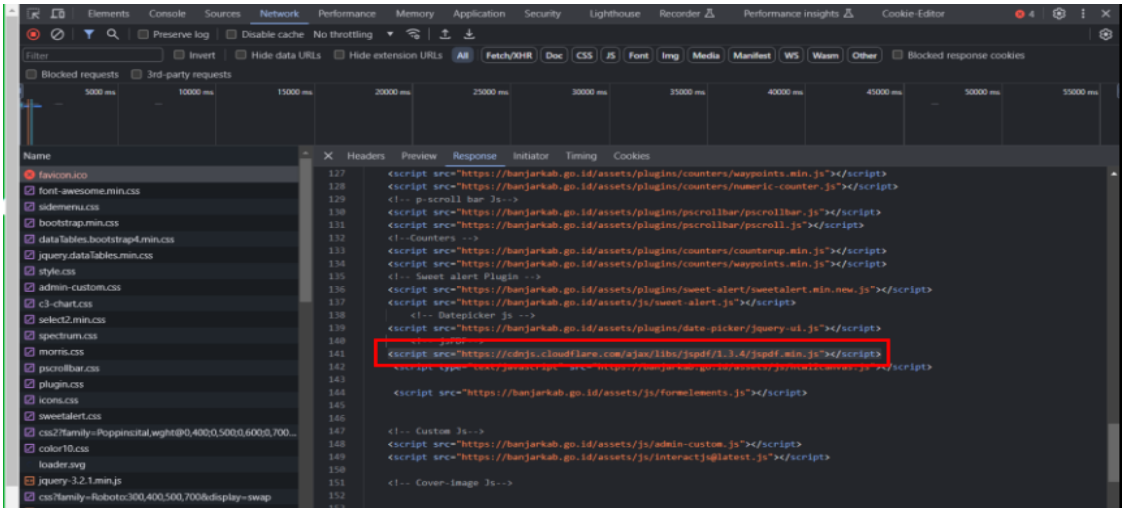

Figure 10. Inspect element for cross-domain JavaScript on Target A

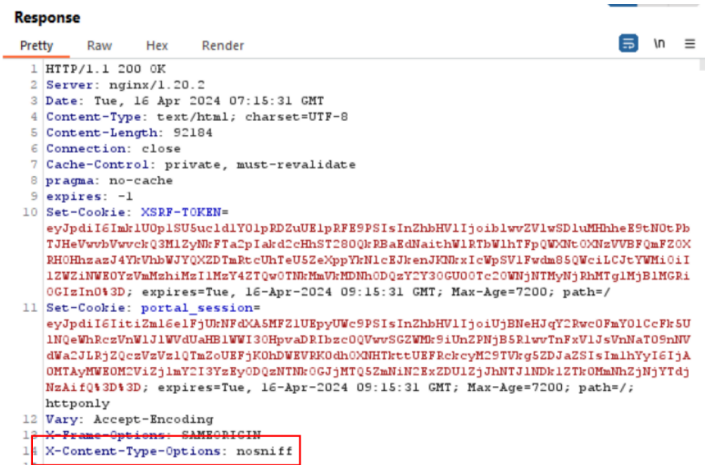

Figure 11. Burb suite responds for X-Content-Type-Options on Target A

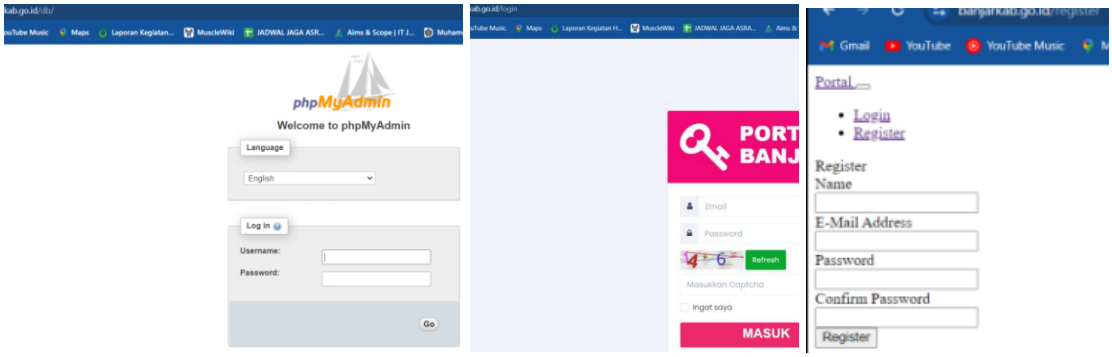

Figure 12. Common Directory is an active page.

By contrast, several scanner alerts were not supported by manual evidence. The Application Error Disclosure findings were not validated, because POST-based testing and

browser inspection did not reveal stack traces, server-side warnings, or other forms of sensitive internal diagnostic information, as shown in Figure 7. Likewise, the Big Redirect Detected, Private IP Disclosure, and Common Sensitive File alerts did not result in demonstrable security exposure and were therefore classified as false positives. For CSRF testing, inspection of the login form on Target A using Burp Suite identified a hidden token field (Figure 5), indicating that CSRF protection was present at least on the examined endpoint. Accordingly, CSRF findings should be interpreted at the endpoint level, rather than being generalized across the entire application from a single request. Overall, the validation results indicate that the dominant confirmed weaknesses across the three applications were not direct high-impact exploitation vectors, but rather security misconfigurations, incomplete browser-side protections, and outdated third-party components.

3.3. Reporting and Discussion

In the reporting phase, all validated vulnerabilities were scored using the CVSS v4.0 calculator to estimate severity based on exploitability and impact, as illustrated in Figure 13. The results, presented in Table 3, show a relatively consistent pattern across the three evaluated applications. For Target A, eight validated vulnerabilities were categorized as Medium, while one finding, namely Common Directory (CWE-538), received a score of 0 and was classified as None. For Target B, nine validated vulnerabilities were identified, all of which fell within the Medium severity range. For Target C, nine validated vulnerabilities were rated as Medium, while one Common Directory finding was again categorized as None. Across all targets, the most frequently recurring validated issues were the absence of security headers, incomplete cookie attributes, the use of outdated JavaScript libraries, and insufficient protection against clickjacking and content sniffing.

These results indicate that the security profile of the evaluated applications is dominated by systematic security misconfiguration rather than by confirmed critical vulnerabilities such as exploitable SQL injection or severe application error disclosure. From an operational standpoint, this pattern is significant. Configuration weaknesses such as missing CSP, HSTS, X-Frame-Options, and X-Content-Type-Options may individually be classified as medium-severity findings; however, when these weaknesses appear repeatedly across multiple public-facing government applications, they indicate broader weaknesses in secure deployment practice, configuration governance, and routine

maintenance. A similar observation applies to cookie hardening. Although the absence of HttpOnly, Secure, and SameSite attributes may not immediately result in compromise on their own, their repeated occurrence across applications increases the overall attack surface and may facilitate session abuse or client-side exploitation when combined with other vulnerabilities.

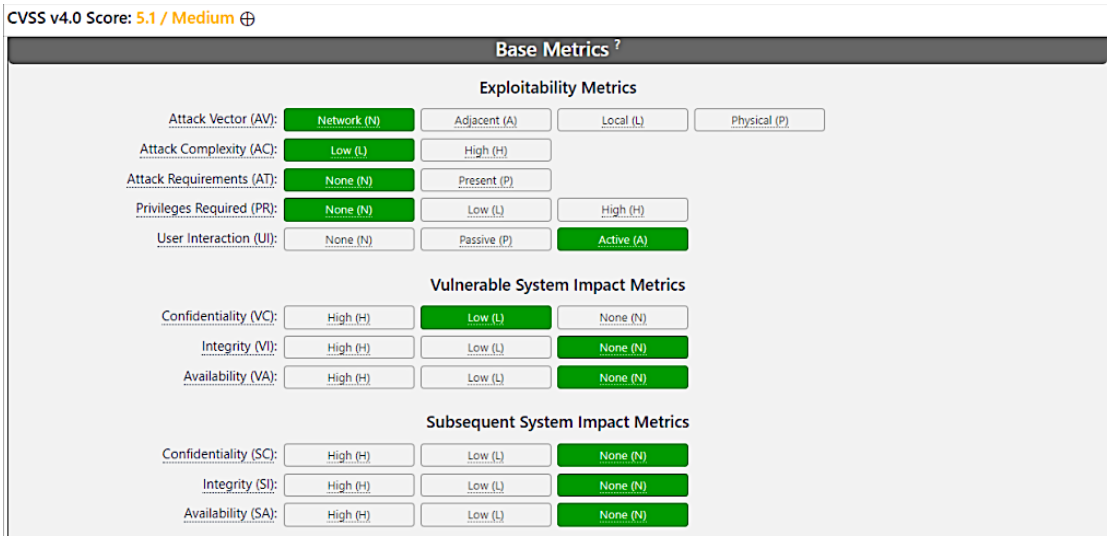

Figure 13. CVSS capture for CSP header not set

Another important result of this study is the gap between automated scanner output and manually validated findings. Several alerts that initially appeared serious were ultimately reclassified as false positives after targeted verification. This was particularly evident for SQL injection, application error disclosure, and selected header-related findings. This outcome directly supports the methodological rationale of the study, namely that automated scanners are valuable for broad coverage and rapid enumeration, but they are insufficient as standalone evidence for risk assessment in government environments. Manual verification using the WSTG v4.2 guide was necessary to distinguish actual vulnerabilities from non-exploitable behavior or scanner assumptions. Accordingly, the combination of NIST SP 800-115 as the overall assessment framework and WSTG v4.2 as the manual validation reference proved effective in producing findings that are both technically reliable and practically actionable.

Based on the validated results, remediation should prioritize low-complexity changes capable of producing broad security improvement across all three applications. First,

security headers should be standardized at the reverse proxy or web server level, including Content-Security-Policy, Strict-Transport-Security, X-Frame-Options, and X-Content-Type-Options, in order to strengthen browser-side security enforcement. Second, session cookies should be consistently configured with the HttpOnly, Secure, and SameSite attributes through the application framework or session configuration, rather than being managed inconsistently at the page level. Third, all third-party JavaScript libraries should be inventoried, updated, and periodically reviewed against vulnerability databases, especially where older versions of jQuery or externally hosted dependencies are still in use. Fourth, unnecessary public directories, default paths, and overly informative response headers such as server version disclosure should be removed or minimized. Fifth, anti-CSRF protection should be reviewed comprehensively across all state-changing endpoints rather than assumed from a framework default or a single protected form. Taken together, these measures are relatively low-cost compared with application redesign, yet they would mitigate the majority of the confirmed findings identified in this study and materially improve the security posture of the evaluated e-government applications.

Table 3. Vulnerability Severity Level

Target	Vulnerability	CVSS v4.0 Score	Severity
Government Portal (Target A)	Content Security Policy (CSP) Header Not Set (CWE-693)	5.1	Medium
	Vulnerable JS Library (CWE-829)	5.1	Medium
	Cookie No HttpOnly Flag (CWE-1004)	6.9	Medium
	Cookie Without Secure Flag (CWE-614)	5.1	Medium
	Cookie without SameSite Attribute (CWE-1275)	5.1	Medium
	Cross-Domain JavaScript Source File Inclusion (CWE-829)	7.1	Medium
	Absence of Anti-CSRF Tokens (CWE-352)	5.1	Medium
	Strict-Transport-Security Header Not Set (CWE-319)	5.1	Medium
	Common Directory (CWE-538)	0	None
Public Administration Service (Target B)	Content Security Policy (CSP) Header Not Set (CWE-693)	5.1	Medium
	Missing Anti-clickjacking Header (CWE-1021)	5.1	Medium
	Vulnerable JS Library (CWE-829)	5.1	Medium
	Cookie No HttpOnly Flag (CWE-1004)	6.9	Medium

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

Target	Vulnerability	CVSS v4.0 Score	Severity
	Cookie Without Secure Flag (CWE-614)	5.1	Medium
	Server Leaks Version Information via "Server" HTTP Response Header Field (CWE-200)	5.1	Medium
	X-Content-Type-Options Header Missing (CWE-693)	5.1	Medium
	Strict-Transport-Security Header Not Set (CWE-319)	5.1	Medium
	Missing 'X-Frame-Options' header (CWE-693)	5.1	Medium
JDIH (Target C)	Content Security Policy (CSP) Header Not Set (CWE-693)	5.1	Medium
	Missing Anti-clickjacking Header (CWE-1021)	5.1	Medium
	Vulnerable JS Library (CWE-829)	5.1	Medium
	Cookie No HttpOnly Flag (CWE-1004)	6.9	Medium
	Cookie Without Secure Flag (CWE-614)	5.1	Medium
	Cookie without SameSite Attribute (CWE-1275)	5.1	Medium
	X-Content-Type-Options Header Missing (CWE-693)	5.1	Medium
	Strict-Transport-Security Header Not Set (CWE-319)	5.1	Medium
	Common Directory (CWE-538)	0	None
	Missing 'X-Frame-Options' header (CWE-693)	5.1	Medium

3.4. Remediation

The recommended corrective actions derived from the validated findings are as follows:

1) Content Security Policy (CSP) Header Not Set: Configure an appropriate CSP policy at the web server or reverse-proxy level to restrict trusted sources for scripts, styles, frames, and other active content.
2) Strict-Transport-Security Header Not Set: Enable HSTS by returning the Strict-Transport-Security header over HTTPS responses to enforce secure transport and reduce downgrade risk.
3) Absence of Anti-CSRF Tokens: Ensure that all state-changing requests implement anti-CSRF protection and that cookie policy is aligned with SameSite requirements.
4) Vulnerable JavaScript Libraries: Update outdated client-side libraries to supported versions and establish a routine dependency-audit process.

5) Cookies Without HttpOnly, Secure, or SameSite Attributes: Configure cookie security attributes in the application/session layer so that sensitive cookies are protected consistently across all modules.
6) Cross-Domain JavaScript Source File Inclusion: Restrict untrusted external script sources, validate dependency integrity, and strengthen CSP to control permitted origins.
7) Common Directories and Unnecessary Public Paths: Remove or restrict unused public directories and ensure that directory exposure does not reveal internal application structure.
8) Missing Anti-clickjacking Protection / X-Frame-Options: Implement X-Frame-Options and/or the frame-ancestors directive in CSP to prevent unauthorized framing.
9) Server Leaks Version Information: Minimize server disclosure by suppressing unnecessary version information in HTTP response headers.
10) Missing X-Content-Type-Options Header: Enable X-Content-Type-Options: nosniff to ensure that browsers do not infer content types beyond the server declaration.

4. CONCLUSION

This study demonstrated that the integration of the NIST SP 800-115 penetration testing framework with the detailed test cases provided in WSTG v4.2, supported by multiple assessment tools such as OWASP ZAP, Arachni, Burp Suite, and SQLmap, provides a more reliable approach for evaluating the security of government web applications. The combined use of automated scanning and manual validation was effective in reducing false positives and narrowing the results to a smaller set of confirmed vulnerabilities. This finding confirms that a hybrid methodology is more suitable than relying solely on automated scanners, particularly in production e-government environments where accuracy, safety, and actionable results are essential. The security assessment conducted on three government web applications managed by Diskominfo in a Regency identified 9 validated vulnerabilities in Target A, 9 in Target B, and 10 in Target C. The results show that the most recurrent weaknesses across the evaluated systems were missing or weak security headers, including Content Security Policy (CSP), HTTP Strict Transport Security (HSTS), X-Frame-Options, and X-Content-Type-Options, as well as outdated JavaScript

libraries and insecure cookie configurations. These findings indicate that the dominant security risks in the assessed applications do not primarily arise from complex business logic flaws or confirmed injection attacks, but rather from systematic security misconfiguration, insufficient hardening, and limited dependency maintenance. The recurrence of similar weaknesses across multiple applications also suggests the existence of shared gaps in deployment practices and security governance. From a practical perspective, the findings indicate that improving the security posture of these e-government systems does not necessarily require major architectural redesign. A substantial portion of the identified weaknesses can be mitigated through standardized secure configuration, routine patch and dependency management, and consistent implementation of browser-side security controls across all applications. Therefore, the study highlights the importance of establishing secure deployment baselines and periodic maintenance procedures for government-managed web systems. For future research, it is recommended to extend the assessment through white-box testing, which would allow access to source code, application logic, and system architecture, thereby enabling the identification of vulnerabilities that may not be observable through black-box testing alone.

ACKNOWLEDGMENT

The authors gratefully acknowledge the National Cyber and Crypto Polytechnic for its valuable support in the completion of this research. The institutional support, academic guidance, and research facilities provided by the Polytechnic were instrumental in facilitating the conduct of this study and the preparation of this manuscript.

REFERENCES

[1] Presiden Republik Indonesia, *Instruksi Presiden Republik Indonesia Nomor 3 Tahun 2003 tentang Kebijakan dan Strategi Nasional Pengembangan E-Government*. Jakarta, Indonesia: Sekretariat Kabinet Republik Indonesia, Jun. 9, 2003.

[2] Direktorat Operasi Keamanan Siber, Badan Siber dan Sandi Negara, *Lanskap Keamanan Siber Indonesia 2022*. Jakarta, Indonesia: Badan Siber dan Sandi Negara, 2022.

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

[3] Badan Siber dan Sandi Negara, *Peraturan Badan Siber dan Sandi Negara Nomor 4 Tahun 2021 tentang Pedoman Manajemen Keamanan Informasi Sistem Pemerintahan Berbasis Elektronik dan Standar Teknis dan Prosedur Keamanan Sistem Pemerintahan Berbasis Elektronik*. Jakarta, Indonesia, May 19, 2021.

[4] E. Z. Darojat, E. Sediyono, and I. Sembiring, "Vulnerability assessment website e-government dengan NIST SP 800-115 dan OWASP menggunakan web vulnerability scanner," *Jurnal Sistem Informasi Bisnis*, vol. 12, no. 1, pp. 36–44, Sep. 2022, doi: 10.21456/vol12iss1pp36-44.

[5] W. Wardana, A. Almaarif, and A. Widjajarto, "Vulnerability assessment and penetration testing on the XYZ website using NIST 800-115 standard," *Syntax Literate: Jurnal Ilmiah Indonesia*, vol. 7, Special Issue no. 1, Jan. 2022, doi: 10.36418/syntax-literate.v7i1.5800.

[6] R. Amankwah, J. Chen, P. K. Kudjo, and D. Towey, "An empirical comparison of commercial and open-source web vulnerability scanners," *Software: Practice and Experience*, vol. 50, no. 9, pp. 1842–1857, Sep. 2020, doi: 10.1002/spe.2870.

[7] L. Cui, J. Cui, Z. Hao, L. Li, Z. Ding, and Y. Liu, "An empirical study of vulnerability discovery methods over the past ten years," *Computers & Security*, vol. 120, Art. no. 102817, 2022, doi: 10.1016/j.cose.2022.102817.

[8] K. Abdulghaffar, N. Elmrabit, and M. Yousefi, "Enhancing web application security through automated penetration testing with multiple vulnerability scanners," *Computers*, vol. 12, no. 11, Art. no. 235, 2023, doi: 10.3390/computers12110235.

[9] E. A. Altulaihan, A. Alismail, and M. Frikha, "A survey on web application penetration testing," *Electronics*, vol. 12, no. 5, Art. no. 1229, Mar. 2023, doi: 10.3390/electronics12051229.

[10] S. Qadir, E. Waheed, A. Khanum, and S. Jehan, "Comparative evaluation of approaches & tools for effective security testing of web applications," *PeerJ Computer Science*, vol. 11, Art. no. e2821, 2025, doi: 10.7717/peerj-cs.2821.

[11] K. U. Sarker, F. Yunus, and A. Deraman, "Penetration taxonomy: A systematic review on the penetration process, framework, standards, tools, and scoring methods," *Sustainability*, vol. 15, no. 13, Art. no. 10471, Jul. 2023, doi: 10.3390/su151310471.

[12] M. Alhamed and M. M. H. Rahman, "A systematic literature review on penetration testing in networks: Future research directions," *Applied Sciences*, vol. 13, no. 12, Art. no. 6986, Jun. 2023, doi: 10.3390/app13126986.

[13] D. F. Priambodo, A. D. Rifansyah, and M. Hasbi, "Penetration testing web XYZ berdasarkan OWASP risk rating," *Teknika*, vol. 12, no. 1, pp. 33–46, Feb. 2023, doi: 10.34148/teknika.v12i1.571.

[14] NIST, "Security assessment," *Computer Security Resource Center*, National Institute of Standards and Technology. [Online]. Available: https://csrc.nist.gov/glossary/term/security_assessment. [Accessed: Jan. 15, 2026].

[15] F. Hilario, D. Chang, C. Zafra, Y. Vasquez, and L. Chipana, "Application of the OWASP framework to identify and remediate vulnerabilities in Java web applications," *Journal of System and Management Sciences*, vol. 14, no. 7, pp. 406–425, 2024, doi: 10.33168/JSMS.2024.0722.

[16] M. Kluban, M. Mannan, and A. M. Youssef, "On detecting and measuring exploitable JavaScript functions in real-world applications," *ACM Transactions on Privacy and Security*, vol. 27, no. 1, pp. 1–37, 2023, doi: 10.1145/3630253.

[17] Z. Kang, S. Li, and Y. Cao, "Probe the Proto: Measuring client-side prototype pollution vulnerabilities of one million real-world websites," in *Proc. Network and Distributed System Security Symp. (NDSS)*, 2022, doi: 10.14722/ndss.2022.24308.

[18] M. Shcherbakov, M. Balliu, and C.-A. Staicu, "Silent Spring: Prototype pollution leads to remote code execution in Node.js," *arXiv preprint* arXiv:2207.11171, 2022, doi: 10.48550/arXiv.2207.11171.

[19] J. C. Davis, C. A. Coghlan, F. Servant, and D. Lee, "The impact of regular expression denial of service (ReDoS) in practice: An empirical study at the ecosystem scale," in *Proc. 26th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering (ESEC/FSE)*, 2018, pp. 246–256, doi: 10.1145/3236024.3236027.

[20] M. Bhuiyan, B. Çakar, E. H. Burmane, J. C. Davis, and C.-A. Staicu, "SoK: A literature and engineering review of regular expression denial of service (ReDoS)," in *Proc. ACM Asia Conf. on Computer and Communications Security (ASIA CCS)*, 2024, pp. 1659–1675, doi: 10.1145/3708821.3733912.