PLOMS Journal of Artificial Intelligence (PLOMS AI)

PLOMS AI

Review

Survey on Web Application Security Testing Methods

Amman Alamri[1], Hamad Albahri[1], and Rabie A. Ramadan[1]

[1]Department of Information Systems, College of Economics, Management, and Information Systems, University of Nizwa, Nizwa, Sultanate of Oman
*Correspondence: 21339191@uofn.edu.om

Received: January 1st, 2025; Accepted: March 3rd, 2025; Published: May 1st, 2025

Abstract: This research study delivers comprehensive coverage of tools, techniques, and processes for ensuring security within web applications. The analysis encompasses both automated and manual approaches, including code reviews, penetration testing, and tools addressing common vulnerabilities listed in the OWASP Top 10. As web applications have become critical infrastructure for modern organizations managing sensitive data and performing significant transactions, the need for robust security testing has grown exponentially. The research presents case studies and results of testing performed in real environments to illustrate the strengths and weaknesses of various security testing methodologies. The findings demonstrate that while automated tools provide efficiency and scalability, manual testing remains essential for detecting complex logical vulnerabilities and context-sensitive security issues. Additionally, the paper emphasizes the importance of integrating multiple testing approaches into a unified web application security strategy to address the evolving threat landscape effectively.

Keywords: Web Application Security; Penetration Testing; OWASP Top 10; Code Review; Security Testing Tools; Burp Suite; OWASP ZAP

I. Introduction

The purpose of this paper is to analyze the tools, methodologies, and procedures for evaluating the security of web applications that are becoming very critical parts of modern organizations managing sensitive data and performing significant transactions. Web application attacks are rising as applications become increasingly complex, making them prime targets for malicious actors seeking to exploit vulnerabilities that could facilitate unauthorized access, data breaches, or system disruption [2]. This justifies the survey that aims at showing both automated and manual testing methodologies in addressing this developing concern.

Manual techniques include code reviews and penetration testing which rely on security expertise to detect and evaluate vulnerabilities [6]. In turn, automated technologies provide scalability and efficiency by allowing the quicker identification of general vulnerabilities over large-scale applications [9]. Additionally, this study reviews some of the most popular tools for finding threats, such as OWASP ZAP and Burp Suite, which have become essential instruments in the security tester's toolkit [18].

The primary focus of the paper will deal with the OWASP Top 10 vulnerabilities that still pose serious threats to web applications [15]. Major targets for these threats include SQL injection, cross-site scripting (XSS), and broken authentication, which continue to plague web applications despite widespread awareness of their existence [1]. The practical pros and cons of the various methods of remedying such threats are demonstrated using real-life case studies and security testing reports in this study.

The conclusions want, ultimately, to provide some important insight on the efficiency of existing security testing methodologies as well as emphasizing the importance of integrating automated and manual strategies into a more unified web application security strategy [3]. This integrated approach is essential for organizations seeking to establish comprehensive security postures in an increasingly hostile digital environment.

II. Security Tests on Web Applications

Security testing processes are critical components in protecting web applications from vulnerabilities that could lead to confidential information theft or unauthorized systems access by attackers [12]. These testing methodologies can be broadly categorized into manual and automated approaches, each with distinct advantages. However, research indicates that the combination of both methodologies typically yields superior results, particularly when tailored to the specific security requirements of the application under test [16]. The focus of these testing efforts is primarily directed toward the most hazardous threats, which are often outlined in the OWASP Top 10 guidelines [15].

II.1 Manual Testing Methods

Manual testing methodologies rely extensively on the expertise and critical thinking abilities of security professionals who possess in-depth knowledge of security threats and vulnerabilities [9]. These specialists leverage their technical expertise and insight to identify logical flaws, subtle vulnerabilities, and misconfigurations that automated tools might overlook due to their inherent limitations in contextual understanding [1].

II.1.1 Penetration Testing

Penetration testing, often abbreviated as "pen testing," involves simulating realistic attacks to identify vulnerabilities in web applications [11]. Cybersecurity professionals methodically attempt to exploit discovered weaknesses and thoroughly document their findings to facilitate remediation efforts. As noted by Chen et al. [7], the primary objective of penetration testing is to understand and replicate the methodologies that might be employed by actual cybercriminals attempting to compromise the system.

Penetration testing is particularly effective for identifying complex security issues such as misconfigurations, chained vulnerabilities, and business logic failures [5]. These types of vulnerabilities often require solutions that differ significantly from standard approaches—such as identifying weaknesses in authentication mechanisms or exploiting vulnerable API endpoints that might not be detected through conventional automated scanning [10].

II.1.2 Code Review

Code reviewing is a meticulous process that involves the inspection of application source code to identify security vulnerabilities and weaknesses [14]. Security experts conduct line-by-line code reviews to identify logical errors, risky coding practices, and potential security risks such as hard-coded credentials, insufficient input validation, or improper error handling mechanisms.

This technique has proven particularly effective in identifying subtle vulnerabilities, including minor logic errors or security misconfigurations that automated techniques might fail to detect [3]. To enhance efficiency and coverage, code reviews can be conducted either manually by experienced security professionals or with the assistance of static analysis tools that can quickly scan large codebases for common security patterns and anti-patterns [8].

II.2 Automated Testing Methods

Automated testing methods have become essential components of modern web application security assessment frameworks due to their ability to rapidly identify common vulnerabilities across extensive codebases and complex systems [9]. These tools facilitate faster detection of security vulnerabilities, improve scalability of testing processes, and significantly reduce the manual effort required for comprehensive security assessments [16]. However, it's important to note that automated tools may sometimes fail to identify complex logic-related issues that typically require manual validation, and they may occasionally generate false positives that necessitate human verification [10].

II.2.1 Static Application Security Testing (SAST)

Static Application Security Testing (SAST) represents a white-box testing approach that evaluates the source code, bytecode, or binaries of an application without executing it [8]. SAST tools are designed to identify potential vulnerabilities during the early stages of the development lifecycle by analyzing code for security issues such as SQL injection vulnerabilities, unsafe function calls, and embedded credentials [14].

The primary advantage of SAST is that it allows developers to identify and address security issues during the coding phase, which substantially reduces the overall cost of remediation compared to addressing vulnerabilities discovered in production environments [12]. Popular SAST tools include SonarQube, Checkmarx, and Fortify, each offering specialized capabilities for different programming languages and development environments [4].

II.2.2 Dynamic Application Security Testing (DAST)

Dynamic Application Security Testing (DAST) is a black-box testing methodology that identifies vulnerabilities during the execution of an application [10]. These tools interact with web applications externally, simulating real-time attacks to identify security issues such as SQL injection, cross-site scripting (XSS), and authentication vulnerabilities [5].

DAST is particularly effective at identifying runtime vulnerabilities that may result from improper configurations, inadequate input validation, or insecure session management practices [1]. Unlike SAST, DAST does not typically require access to the application's source code, making it suitable for testing third-party applications or systems where source code access is restricted [3]. Widely used DAST tools include Nessus, Burp Suite, and OWASP ZAP, each offering varying levels of automation and customization capabilities [18].

It's worth noting that modern security testing strategies often incorporate both SAST

and DAST as complementary approaches, allowing organizations to identify a broader range of potential vulnerabilities throughout the software development lifecycle [16]. Additionally, these tools are increasingly being integrated into continuous integration and deployment (CI/CD) pipelines to ensure that security testing becomes an integral part of the development process rather than a separate, isolated activity [7].

II.3 OWASP Top 10 Vulnerabilities

The OWASP Top 10 is a widely recognized framework for identifying and categorizing the most critical web application security risks [15]. Developed and maintained by the Open Web Application Security Project (OWASP), this list serves as an essential reference for developers, security professionals, and organizations seeking to strengthen their web application security posture. The OWASP Top 10 is regularly updated to reflect evolving threat landscapes and emerging attack vectors in the web application security domain [11].

II.3.1 SQL Injection

SQL injection vulnerabilities occur when user-supplied input is inadequately validated before being incorporated into database queries [13]. This allows attackers to manipulate query syntax and execute unauthorized database operations, potentially leading to unauthorized access, data theft, or data corruption [1]. The prevalence of SQL injection attacks highlights the critical importance of implementing proper input validation and parameterized queries in web application development [10].

II.3.2 Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) vulnerabilities enable attackers to inject malicious scripts into web pages that are subsequently viewed by unsuspecting users [21]. These injected scripts can perform various malicious actions, including session hijacking, credential theft, and unauthorized actions performed on behalf of the compromised user [3]. The persistent nature of some XSS vulnerabilities makes them particularly dangerous, as malicious scripts may remain embedded in web applications, affecting multiple users over extended periods [5].

II.3.3 Broken Authentication

Broken authentication vulnerabilities arise from flawed implementation of authentication mechanisms in web applications [15]. Attackers can exploit these weaknesses to capture

legitimate user credentials or bypass authentication entirely, potentially gaining unauthorized access to sensitive systems or user accounts [16]. Common manifestations of broken authentication include weak password policies, improper session management, and insecure credential storage practices, all of which can significantly compromise application security [6].

II.3.4 Sensitive Data Exposure

This vulnerability category encompasses failures to adequately protect sensitive information such as financial data, personal identification information, and authentication credentials during transmission or storage [2]. These vulnerabilities often arise from weak encryption implementations, improper certificate validation, or inadequate data handling practices [1]. The growing volume of sensitive data processed by modern web applications makes addressing these vulnerabilities increasingly critical for maintaining user trust and regulatory compliance [7].

II.3.5 Insecure Deserialization

Insecure deserialization vulnerabilities arise when applications deserialize untrusted data without sufficient validation, potentially allowing attackers to execute arbitrary code or launch denial-of-service attacks [15]. These vulnerabilities are particularly dangerous because they can lead to complete system compromise in many cases, and they can be difficult to detect through automated scanning tools [14]. Proper input validation, type checking, and integrity verification are essential for mitigating these risks in web applications that process serialized data [5].

III. Tools for Web Application Security Testing

Web application security testing tools play a crucial role in evaluating, identifying, and mitigating vulnerabilities in web applications that have become essential components of modern digital infrastructure [18]. These tools enable developers and security professionals to identify critical threats such as SQL injection, cross-site scripting (XSS), broken authentication mechanisms, and various other security weaknesses that could potentially compromise application integrity and data confidentiality [4]. Among the most widely adopted security testing platforms, Burp Suite and OWASP ZAP stand out for their comprehensive capabilities that support both automated scanning and manual security testing methodologies [10].

III.1 Burp Suite

Burp Suite is a comprehensive and widely-used web application security testing platform developed by PortSwigger [15]. Its versatility, user-friendly interface, and effectiveness in identifying and exploiting vulnerabilities have made it a preferred tool among security professionals, penetration testers, and developers engaged in web application security testing [11]. The platform offers a sophisticated suite of tools for testing common vulnerabilities, intercepting network traffic, and analyzing HTTP/HTTPS requests in detail [5].

III.1.1 Key Features

Intercepting Proxy: Burp Suite functions as an intercepting proxy, positioning itself between the web browser and the target application. This capability allows security testers to capture, inspect, and modify HTTP/HTTPS communications in real-time, facilitating the identification of vulnerabilities such as improper redirections, inadequate input validation, and authentication flaws [6].

Scanner: The platform includes a powerful vulnerability scanner that automatically identifies security issues such as cross-site scripting (XSS), SQL injection, and insecure session management [16]. This functionality enables rapid vulnerability discovery and provides detailed recommendations for remediation [10].

Intruder: Burp Suite's Intruder module provides capabilities for parameter manipulation, fuzzing, and brute-force attack simulation [11]. This feature allows security testers to send multiple payloads to target endpoints to evaluate their resilience against various attack vectors, making it particularly valuable for testing authentication systems and input validation mechanisms [1].

Repeater: The Repeater functionality enables testers to manually craft and resend HTTP/HTTPS requests to observe application responses and behavior [5]. This capability is instrumental in understanding application behavior and testing specific parameters for vulnerabilities [3].

Extensibility: Burp Suite supports extension through plugins and custom scripts, allowing testers to integrate external tools and create specialized testing workflows tailored to specific application requirements [7]. This extensibility makes Burp Suite adaptable to various testing scenarios and enhances its utility across different application environments [18].

III.2 OWASP ZAP

The OWASP Zed Attack Proxy (ZAP) is an open-source web application security testing tool developed and maintained by the Open Web Application Security Project (OWASP) community [15]. Its user-friendly interface makes it accessible to both novice and experienced security professionals, while its comprehensive functionality supports various testing methodologies [2]. As a versatile security testing platform, ZAP can perform both automated vulnerability scanning and manual testing of web applications [16].

III.2.1 Key Features

Intercepting Proxy: Similar to Burp Suite, ZAP functions as an intercepting proxy that captures HTTP/HTTPS traffic between the browser and the target application [6]. This functionality allows security testers to analyze and manipulate request and response messages in real-time, facilitating the identification of potential vulnerabilities [5].

Automated Scanning: ZAP includes an automated scanner that detects common web application vulnerabilities such as cross-site scripting (XSS), SQL injection, authentication failures, and security misconfigurations [10]. This feature plays a critical role in comprehensive web application security assessment by identifying potential entry points for attackers [11].

Spidering: The spider tool in ZAP automatically crawls through the target web application to discover all accessible endpoints and pages [7]. This ensures comprehensive coverage during security assessments by identifying previously unknown or hidden application components [1].

Active and Passive Scanning: ZAP supports both active scanning, which involves sending test payloads to the application, and passive scanning, which analyzes application responses without active interaction [5]. This dual approach enables thorough vulnerability detection while minimizing potential impact on the target application [14].

Fuzzing: ZAP includes fuzzing capabilities that allow testers to send multiple varied inputs to specific parameters, facilitating the identification of abnormal behaviors, buffer overflows, and input validation weaknesses [16]. This functionality is particularly valuable for identifying edge cases and unexpected application behaviors that might indicate security vulnerabilities [18].

Extensibility: ZAP supports customization through scripts and extensions, enabling testers to extend its functionality to address specific testing requirements [4]. The platform supports scripting in various languages, including Python, JavaScript, and Groovy, allowing for sophisticated automation and customization of testing workflows [3].

IV. Real-World Analysis of Security Testing

Testing web applications for security vulnerabilities is essential for identifying weaknesses that could potentially be exploited by malicious actors [12]. This section examines the practical efficacy of various security testing methodologies through case studies, security reports, and documented test results from real-world implementations [2]. The analysis demonstrates how both automated tools and manual penetration testing techniques are utilized to identify and mitigate security risks in web applications, with particular emphasis on addressing the OWASP Top 10 vulnerabilities such as SQL injection and cross-site scripting (XSS) [15].

IV.1 Effectiveness of Penetration Testing

Penetration testing has consistently demonstrated its value as a methodology for identifying and verifying complex vulnerabilities that might remain undetected through automated scanning approaches [6]. The contextual understanding and adaptive problem-solving capabilities of skilled penetration testers enable them to uncover sophisticated security weaknesses that have eluded traditional security assessment methods [11].

IV.1.1 Case Study: SQL Injection in an Online Store

As part of a comprehensive security assessment, a large e-commerce platform underwent penetration testing that revealed a critical SQL injection vulnerability in its product search functionality [1]. The penetration testers demonstrated that this input validation vulnerability could potentially allow attackers to extract sensitive customer information, including payment details, from the underlying database [13]. This case illustrates why manual testing remains essential for identifying vulnerabilities in widely used web applications, particularly those processing sensitive data [7].

IV.1.2 Case Study: Financial Application with Business Logic Error

Another noteworthy case involved an online banking application where penetration testing uncovered a significant business logic vulnerability that allowed users to circumvent transaction limits by manipulating request parameters [5]. This type of vulnerability

typically evades detection by automated tools because it involves flawed application logic rather than standard security misconfigurations [10]. The case highlights the importance of human expertise in penetration testing, as security professionals can understand complex application workflows and identify logical inconsistencies that might be exploited by attackers [14].

IV.2 Effectiveness of Automated Tools

The implementation of automated security testing tools such as Burp Suite and OWASP ZAP has significantly enhanced the efficiency and coverage of vulnerability assessments for web applications [16]. These tools enable standardized testing procedures that can be consistently applied across different application versions and deployments, facilitating more comprehensive security monitoring throughout the development lifecycle [18].

IV.2.1 Case Study: Cross-Site Scripting (XSS) Detection in Public Portal

Automated scanning with OWASP ZAP identified multiple reflected cross-site scripting (XSS) vulnerabilities in a government web portal that accepted public input data [21]. These vulnerabilities could potentially allow attackers to inject malicious scripts that would execute in users' browsers, potentially leading to session hijacking or credential theft [3]. The security team implemented remediation measures efficiently thanks to the detailed vulnerability reports generated by the automated scanning tool, demonstrating the value of automation in identifying and addressing common web application security issues [7].

IV.2.2 Case Study: Credentials Exposure on an Online Store

During an automated security assessment of a major online retailer, scanning tools identified that sensitive data, including payment card information and user credentials, was being transmitted without proper encryption [1]. This Sensitive Data Exposure vulnerability, which features prominently in the OWASP Top 10 list, was flagged by the automated tools, leading to recommendations for implementing TLS/SSL encryption for sensitive endpoints [15]. This case exemplifies how automated tools can efficiently identify configuration weaknesses and data handling issues that might otherwise remain undetected [12].

IV.3 Comparative Insights: Manual vs. Automated Testing

While manual testing has proven highly effective at identifying sophisticated vulnerabilities, practical security testing typically involves a combination of both manual and

automated approaches [6]. Manual penetration testing excels at uncovering complex vulnerabilities such as chained attack sequences, business logic flaws, and application-specific weaknesses that might not conform to standard vulnerability patterns [5].

Human testers leverage their knowledge, experience, and intuition to evaluate application behavior and identify potential security risks that automated tools might overlook due to their reliance on predefined vulnerability signatures [10]. Conversely, automated tools excel at identifying common vulnerabilities across large applications, including SQL injection, cross-site scripting, and configuration errors, with remarkable efficiency and consistency [16].

These tools are particularly valuable in continuous integration/continuous deployment (CI/CD) environments, rapid scanning scenarios, and situations requiring regular reassessment of application security posture [7]. The real-world cases discussed in this section illustrate the complementary nature of both approaches: while penetration testers provide depth through detailed analysis of specific application components, automated tools offer breadth by quickly identifying common vulnerabilities across the application surface [12].

IV.4 Importance of Addressing OWASP Top 10 Vulnerabilities

The vulnerabilities cataloged in the OWASP Top 10 consistently appear in security breach incidents, highlighting the critical importance of addressing these common security weaknesses [15]. Poor programming practices, inadequate security configurations, and insufficient security testing during development are frequently identified as root causes for these vulnerabilities [2].

Cross-Site Scripting (XSS) and SQL Injection remain among the most frequently exploited vulnerabilities in web applications [21][13]. Case studies have repeatedly demonstrated that insufficient input validation and inadequate data sanitization often create opportunities for attackers to execute malicious code or gain unauthorized access to sensitive data [1].

Numerous high-profile security incidents have involved compromised user accounts and massive data breaches resulting from broken authentication mechanisms and sensitive data exposure [6]. By prioritizing the detection and remediation of OWASP Top 10 vulnerabilities, organizations can significantly reduce their security risk profile and enhance the overall resilience of their web applications against common attack vectors [12].

V. Conclusion

This paper has emphasized the critical importance of web application security testing in identifying and addressing vulnerabilities that could potentially compromise sensitive systems and data [2]. Through a comprehensive examination of both manual and automated security testing methodologies, including code reviews, penetration testing, and specialized security tools such as OWASP ZAP and Burp Suite, we have demonstrated the relative strengths and limitations of different approaches to security assessment [6].

The findings from this study strongly suggest that optimal security testing outcomes are achieved through the integration of multiple complementary methodologies rather than reliance on any single approach [12]. While automated tools provide efficiency, consistency, and broad coverage, manual testing contributes depth, contextual understanding, and the ability to identify complex, logic-based vulnerabilities that might elude automated detection [10].

Moreover, the persistent prevalence of OWASP Top 10 vulnerabilities in production web applications underscores the continuing need for comprehensive security testing throughout the development lifecycle [15]. Future research directions may explore the application of machine learning and artificial intelligence to enhance vulnerability detection capabilities, potentially bridging the gap between automated efficiency and human analytical capabilities [7].

Author Contributions

Conceptualization, A.A., H.A. and R.A.R.; methodology, A.A.; validation, R.A.R.; formal analysis, H.A.; investigation, A.A. and H.A.; resources, R.A.R.; data curation, A.A.; writing—original draft preparation, A.A. and H.A.; writing—review and editing, R.A.R.; visualization, H.A.; supervision, R.A.R.; project administration, A.A. All authors have read and agreed to the published version of the manuscript.

Funding

The APC was funded by University of Nizwa.

Acknowledgments

The authors thank University of Nizwa for their support.

Conflicts of Interest

The authors declare no conflict of interest.

References

References

[1] Akrout, R., Alata, E., Kaaniche, M., Nicomette, V. (2014). An automated black box approach for web vulnerability identification and attack scenario generation. Journal of the Brazilian Computer Society, 20(1), 1-16.

[2] Alden, J. (2020). A Survey on Web Application Security. IEEE Security & Privacy, 18(1), 78-82.

[3] Alonso, J. M., Guzman, A., Beltrán, M. (2018). A practical approach for web security testing. International Journal of Computer Network and Information Security, 10(2), 1-10.

[4] Antunes, N., Vieira, M. (2009). Comparing the effectiveness of penetration testing and static code analysis on the detection of SQL injection vulnerabilities in web services. In 2009 15th IEEE Pacific Rim International Symposium on Dependable Computing (pp. 301-306).

[5] Bau, J., Bursztein, E., Gupta, D., Mitchell, J. (2010). State of the art: Automated black-box web application vulnerability testing. In 2010 IEEE Symposium on Security and Privacy (pp. 332-345).

[6] Bertoglio, D. D., Zorzo, A. F. (2017). Overview and open issues on penetration test. Journal of the Brazilian Computer Society, 23(1), 1-16.

[7] Chen, J., Kudjo, P. K., Mensah, S., Amankwah, R., Towey, D. (2018). An empirical comparison of commercial and open-source web vulnerability scanners. In 2018 IEEE 42nd Annual Computer Software and Applications Conference (pp. 192-201).

[8] Chess, B., McGraw, G. (2007). Secure programming with static analysis. Pearson Education.

[9] Deepa, G., Thilagam, P. S. (2014). A research study on web application security. In 2014 International Conference on Advances in Computing, Communications and Informatics (pp. 1016-1022).

[10] Doupé, A., Cova, M., Vigna, G. (2010). Why Johnny can't pentest: An analysis of black-box web vulnerability scanners. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (pp. 111-131). Springer.

[11] Engebretson, P. (2013). The basics of hacking and penetration testing: ethical hacking and penetration testing made easy. Elsevier.

[12] Felderer, M., Büchler, M., Johns, M., Brucker, A. D., Breu, R., Pretschner, A. (2016). Security testing: A survey. Advances in Computers, 101, 1-51.

[13] Halfond, W. G., Viegas, J., Orso, A. (2006). A classification of SQL-injection attacks and countermeasures. In Proceedings of the IEEE International Symposium on Secure Software Engineering (Vol. 1, pp. 13-15).

[14] McCormac, A., Zwaans, T., Parsons, K., Calic, D., Butavicius, M., Pattinson, M. (2016). Security awareness and training programs in organizations: A review of their effectiveness. In International Conference on Human Aspects of Information Security, Privacy, and Trust (pp. 71-82). Springer.

[15] OWASP. (2021). OWASP Top Ten Web Application Security Risks. Retrieved from https://owasp.org/www-project-top-ten/

[16] Sagar, R., Singh, D., Kumar, V. (2017). A survey on web application security: Attacks and prevention. In 2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS) (pp. 1-7).

[17] Sampaio, L. M., Silva, J. C. M. (2008). A survey of web application security testing tools. In Brazilian Symposium on Computer Networks (SBRC) (pp. 217-230).

[18] Shinde, P., Ardhapurkar, S. (2018). Application of security scanners for web vulnerabilities identification. In 2018 International Conference on Current Trends towards Converging Technologies (ICCTCT) (pp. 1-6).

[19] Stuttard, D., Pinto, M. (2011). The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws. John Wiley Sons.

[20] Stuttard, D., Pinto, M. (2017). The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws (2nd Edition). John Wiley Sons.

[21] Wassermann, G., Su, Z. (2008). Static detection of cross-site scripting vulnerabilities. In Proceedings of the 30th international conference on Software engineering (pp. 171-180).

[22] Howard, M., Lipner, S. (2001). The Security Development Lifecycle. IEEE Software, 18(4), 13-17.

[23] Muzaki, F., Fauzan, A., Hariyono, B. (2018). Web application firewall using mod security and Reverse proxy. In 2018 International Conference on Applied Information Technology and Innovation (pp. 105-109). IEEE.