ORIGINAL SCIENTIFIC PAPER OPEN ACCESS

Detection of cross-site scripting (XSS) attacks: Analysis, detection approaches, and mitigation strategies

Vesna Ružičić[1*]

[1] University of Kragujevac, Faculty of Technical Sciences / Department of Information Technologies, Čačak, Serbia

ARTICLE INFO

[*] **Correspondence:** vesna.ruzicic@ftn.kg.ac.rs

DOI: https://doi.org/10.5937/engtoday2600016R

UDC: 621(497.11) **ISSN:** 2812 – 9474

Article history: Received 11 August 2026; Revised 7 October 2026; Accepted 9 October 2026

ABSTRACT

Cross-Site Scripting (XSS) remains one of the most persistent security vulnerabilities affecting modern web applications. The increasing complexity of web technologies, extensive use of client-side scripting, and processing of user-generated content provide opportunities for attackers to inject malicious scripts into trusted web environments. Successful XSS attacks can lead to session hijacking, credential theft, unauthorized access, and disclosure of sensitive information. This paper analyzes XSS vulnerabilities with a focus on attack characteristics, detection approaches, and mitigation strategies. The research includes a statistical analysis of Common Weakness Enumeration (CWE) vulnerability data collected between 2014 and 2022. The dataset was processed using Python in a Jupyter Notebook to examine the frequency, distribution, and long-term trends of XSS vulnerabilities in comparison with other major software weaknesses. The results indicate that XSS remains among the most frequently reported software weaknesses despite advances in secure development practices and web security mechanisms. The study also examines static, dynamic, hybrid, and machine learning-based detection approaches through an experimental evaluation of selected XSS detection tools. The findings demonstrate that no single technique provides complete protection against all XSS variants, while combined approaches offer more effective vulnerability detection and mitigation. The study emphasizes the importance of secure coding, input validation, output encoding, Content Security Policy (CSP), and automated security testing throughout the software development lifecycle.

ONLINE FIRST

KEYWORDS

Cross-Site Scripting, XSS, Web security, Vulnerability detection, Static analysis, Mitigation strategies, CWE, Software engineering.

1. INTRODUCTION

The widespread adoption of web-based applications has fundamentally changed the way organizations deliver services, process information, and communicate with users. The increasing complexity of digital systems, together with the growing use of interconnected software, Internet-based technologies, and data-driven applications, has created new challenges for system reliability and security [1]. As the number and complexity of web applications continue to increase, so does the number of security threats targeting these systems. Among the most significant security vulnerabilities affecting modern web applications is Cross-Site Scripting (XSS), which enables attackers to inject malicious scripts into trusted web pages that are subsequently executed by users' browsers [2]. Successful XSS attacks can result in session hijacking, credential theft, unauthorized access, phishing, website defacement, and disclosure of sensitive information [3]. XSS has remained a frequently reported software weakness in the Common Weakness Enumeration (CWE) database and has been extensively discussed in the context of web application security [3], [4]. The vulnerability

© Faculty of Mechanical and Civil Engineering in Kraljevo. All rights reserved

generally occurs when an application processes untrusted input and subsequently incorporates it into web content without appropriate validation, sanitization, or context-sensitive output encoding. Despite the availability of secure coding practices, modern development frameworks, and browser security mechanisms, XSS vulnerabilities continue to occur in software applications.

Numerous approaches have been proposed for XSS vulnerability detection and mitigation, including static analysis, dynamic application testing, hybrid analysis, taint analysis, input sanitization, output encoding, and Content Security Policy (CSP) [4], [5], [6]. Static analysis examines source code without executing the application, whereas dynamic approaches evaluate application behavior during execution. Hybrid approaches combine characteristics of both methods in an attempt to improve detection coverage and reduce their individual limitations. Research has also investigated automated testing and machine learning-based techniques for identifying XSS vulnerabilities. Although these approaches can significantly improve web security, no single technique can provide complete protection against all XSS attack variants. The diversity of web technologies, programming practices, application architectures, and attack vectors makes comprehensive vulnerability detection a challenging task. Therefore, effective mitigation strategies require the integration of multiple security mechanisms throughout the software development lifecycle.

The objective of this paper is to analyze Cross-Site Scripting vulnerabilities through a review of existing detection and mitigation approaches and a statistical analysis of Common Weakness Enumeration (CWE) data collected between 2014 and 2022. The study compares XSS with other major software weaknesses, examines the frequency and distribution of reported vulnerabilities, and analyzes long-term trends using Python in a Jupyter Notebook. In addition, existing approaches and software solutions for XSS vulnerability detection and mitigation are reviewed to identify their main characteristics and limitations. The remainder of this paper is organized as follows. Section 2 presents related work and existing approaches for detecting and mitigating XSS attacks and provides a comparative analysis of major software weaknesses based on CWE data. The following sections describe the research methodology, present and discuss the obtained results, and conclude with the main findings and recommendations for improving the security of modern web applications.

2. RELATED WORK AND OTHER SOFTWARE SOLUTIONS IN THE FIELD

Cross-Site Scripting (XSS) is a type of injection vulnerability in which an attacker causes malicious scripts to be executed within the context of a trusted web application. XSS vulnerabilities generally occur when an application processes untrusted input and incorporates it into dynamically generated web content without appropriate validation, sanitization, or output encoding. Depending on the way the malicious payload is delivered and stored, XSS attacks are commonly classified as stored, reflected, and DOM-based XSS [6], [7]. Different detection and mitigation approaches have been developed to address these vulnerabilities, including static and dynamic analysis, input validation, output encoding, security testing, and browser-based security mechanisms.

2.1. Analysis of XSS vulnerability trends

Figure 1 presents a comparative analysis of yearly XSS vulnerability trends based on CWE data for the period from 2014 to 2022. As shown in Figure 1, XSS remains among the most frequently reported software weaknesses during the analyzed period.

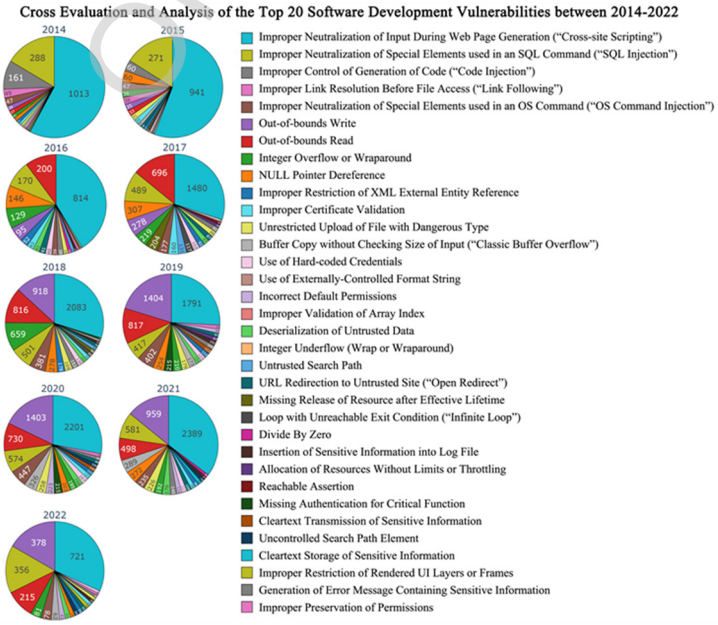


Figure 1: Comparative analysis of yearly XSS vulnerability trends (2014–2022)

ONLINE FIRST

The analysis was performed to examine the frequency and distribution of reported XSS vulnerabilities in comparison with other major software weaknesses. The results indicate that XSS has remained a significant and frequently reported weakness throughout the analyzed period. We performed the statistical analysis in Python using a Jupyter Notebook. The dataset was processed to identify the annual frequency of reported weaknesses and to examine changes in the relative occurrence of XSS over the nine years. The obtained results provide a quantitative basis for evaluating the persistence of XSS vulnerabilities and their position among other commonly reported software weaknesses.

2.2. Stored cross-site scripting (XSS) attack

Stored XSS, also known as persistent XSS, occurs when a malicious payload is stored by the web application and subsequently retrieved and rendered for other users [8]. The payload may be stored in a database, comment system, user profile, message, or other persistent data storage [9]. When another user accesses the affected content, the application retrieves the stored data and includes it in a web page without appropriate output encoding or sanitization. The victim's browser then interprets the malicious content as executable code within the security context of the vulnerable application.

Figure 2 illustrates a simplified stored XSS attack scenario. An attacker submits malicious input to a web application, where the input is stored on the server. When another user requests the affected resource, the stored payload is returned as part of the web content and may be executed by the victim's browser [6]. The persistence of the payload makes stored XSS particularly important in applications that accept and display user-generated content.

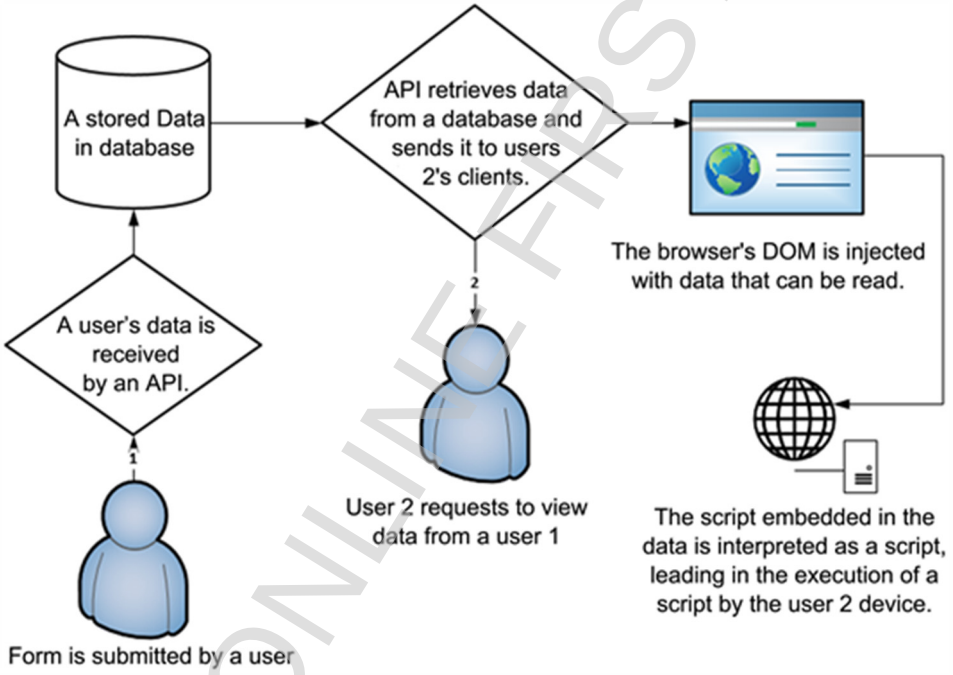


Figure 2: Stored XSS attack scenario

2.3. Reflected cross-site scripting (XSS) attack

Reflected XSS, also known as non-persistent XSS, is a common type of cross-site scripting vulnerability in which an attacker injects a malicious script through a user-controlled request [10]. The malicious payload is subsequently reflected by the vulnerable web application in its response without appropriate validation or output encoding [11]. Unlike stored XSS, the malicious payload is not permanently stored by the application. Instead, it is typically delivered to the victim through a specially crafted URL, form submission, or other HTTP request.

A typical reflected XSS scenario begins when an attacker constructs a malicious request containing an XSS payload. The victim is then induced to access the malicious URL or submit the crafted request. The vulnerable server processes the request and incorporates the untrusted input into the generated response. When the response is rendered by the victim's browser, the injected script may be executed in the context of the vulnerable web application.

Figure 3 illustrates the reflected XSS attack process. In contrast to stored XSS, the payload is transmitted through the request and reflected in the server response rather than being persistently stored in a database [6]. Consequently, reflected XSS depends on the interaction between the attacker, the vulnerable web application, and the victim's browser.

ONLINE FIRST

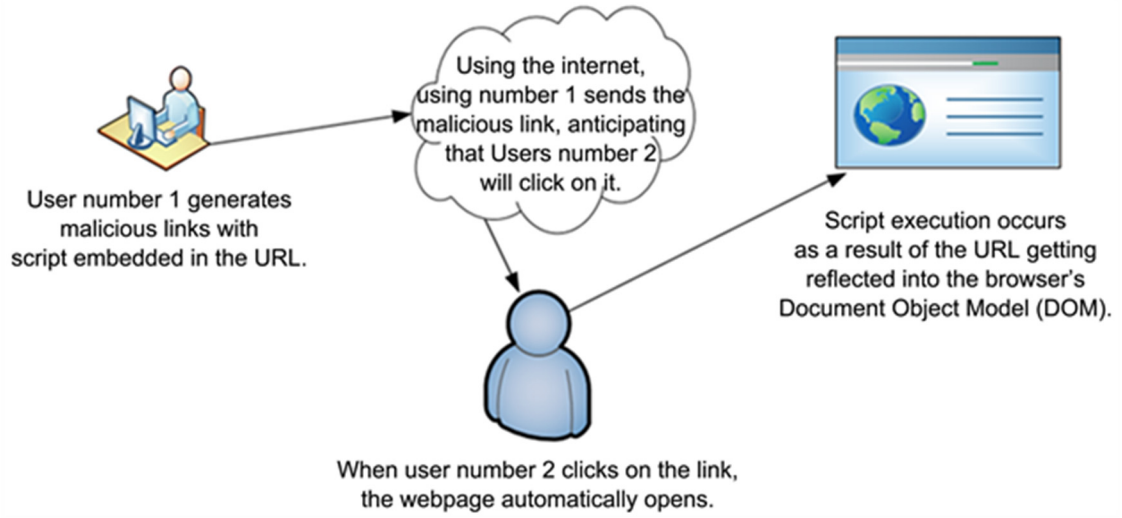


Figure 3: Reflected cross-site scripting attack scenario

3. METHODOLOGY

This research applies a combined quantitative and experimental methodology to analyze Cross-Site Scripting (XSS) vulnerabilities and evaluate the capabilities of selected XSS detection tools. The methodology consists of two complementary phases: statistical analysis of publicly available vulnerability data and experimental evaluation in a controlled web application environment. The first phase focuses on Common Weakness Enumeration (CWE) vulnerability data collected between 2014 and 2022. The objective is to examine the frequency, distribution, and long-term trends of XSS vulnerabilities in comparison with other major software weaknesses. The dataset was processed using Python in a Jupyter Notebook, including data filtering, statistical analysis, and graphical visualization.

The second phase involves the experimental evaluation of three commonly used security testing tools: OWASP ZAP, Burp Suite, and Acunetix [12], [13], [14]. The experiments were conducted using an intentionally vulnerable web application containing Stored XSS, Reflected XSS, and DOM-based XSS vulnerabilities. The same application environment and vulnerability scenarios were used for all tools to ensure comparable and repeatable testing conditions.

Each tool was configured using recommended default settings. The obtained results were evaluated using three metrics: Detection Rate (DR), False Positive Rate (FPR), and Execution Time. Detection Rate measures the proportion of existing XSS vulnerabilities correctly identified by a tool, while False Positive Rate measures the proportion of actual non-vulnerable cases that are incorrectly reported as vulnerable. Execution Time represents the time required to complete the security scan. The evaluation framework is also consistent with established approaches to cybersecurity vulnerability assessment [15].

The research workflow consists of the following steps:

1. CWE vulnerability dataset collection for the period 2014–2022;
2. Data preprocessing and statistical analysis using Python and Jupyter Notebook;
3. Selection of XSS detection tools;
4. Deployment of an intentionally vulnerable web application;
5. Execution of XSS security scans;
6. Evaluation using DR, FPR, and Execution Time;
7. Comparative analysis and discussion of the results.

The proposed methodology combines large-scale vulnerability trend analysis with controlled experimental testing, providing a comprehensive basis for evaluating the capabilities and limitations of current XSS detection approaches [16].

4. CROSS-SITE SCRIPTING (XSS)

Cross-Site Scripting (XSS) is a security vulnerability that allows an attacker to inject malicious JavaScript code into a web application, which can subsequently be executed in another user's browser. XSS typically occurs when an application processes untrusted input without appropriate validation, sanitization, or output encoding [3]. Based on the way the malicious payload is delivered and executed, XSS attacks are commonly classified as Stored, Reflected, and DOM-based XSS [17].

ONLINE FIRST

- **Stored XSS** occurs when a malicious payload is persistently stored by the application, usually in a database, and is executed when the affected content is later displayed to a user.
- **Reflected XSS** occurs when a malicious payload is included in an HTTP request and reflected by the vulnerable application in its response.
- **DOM-based XSS** occurs on the client side when insecure JavaScript code processes untrusted data and modifies the Document Object Model (DOM) in an unsafe manner.

Example of Reflected XSS.

The following simplified example illustrates a reflected XSS vulnerability:

```
// Vulnerable server-side code
app.get("/search", (req, res) => {
  const query = req.query.q;
  res.send(`<p>Results for: ${query}</p>`);
});
```

If the application directly inserts the value of the q parameter into the generated HTML without appropriate output encoding, an attacker may construct a request containing a malicious script. When the vulnerable response is rendered by the victim's browser, the injected code may be executed in the context of the web application.

4.1. Database input and XSS risk

User-provided data stored in a database can become a source of Stored XSS when it is subsequently displayed without appropriate protection. The database itself is not the vulnerability; the security risk arises when untrusted data is rendered as executable content.

For example, an application may store a value such as:

```
<script>alert('XSS')</script>
```

If this value is later displayed without output encoding, the browser may interpret it as executable JavaScript. Appropriate protection includes input validation and context-sensitive output encoding:

```
echo htmlspecialchars($username, ENT_QUOTES, 'UTF-8');
```

Prepared statements should also be used when processing database input. Although prepared statements primarily protect against SQL injection, they represent an important secure coding practice for handling untrusted data.

4.2. XSS Attack Detection and Prevention

XSS detection involves identifying malicious or suspicious input and application behavior that may indicate an injection attempt. Potential indicators include <script> elements, javascript: URLs, event handlers such as onerror and onclick, and unexpected HTML characters.

Several mechanisms can be used to improve XSS detection and prevention:

- input validation and sanitization;
- context-sensitive output encoding;
- logging of suspicious or repeated input attempts;
- automated security testing;
- Web Application Firewalls (WAFs);
- security testing tools such as OWASP ZAP and Burp Suite;
- Content Security Policy (CSP) as an additional browser-based mitigation mechanism.

Effective XSS protection requires multiple complementary mechanisms rather than reliance on a single security control. Input validation, appropriate output encoding, secure coding practices, automated testing, and continuous security assessment should therefore be integrated throughout the software development lifecycle.

5. RESEARCH RESULTS AND DISCUSSION

The analysis was based on Common Weakness Enumeration (CWE) vulnerability data collected for the period 2014–2022. The data were processed using Python in a Jupyter Notebook to examine the frequency and distribution of major software weaknesses and to evaluate the prevalence and persistence of Cross-Site Scripting (XSS). The results were complemented by a comparison of XSS detection approaches and an experimental evaluation of selected security testing tools.

ONLINE FIRST

5.1. Statistical analysis of CWE-based software vulnerability trends (2014–2022)

The CWE dataset was analyzed to identify the most frequently reported software weaknesses. Table 1 presents the ten most prevalent categories in the analyzed dataset.

Table 1: Distribution of the Most Frequently Reported Software Vulnerabilities (2014–2022)

Rank	CWE ID	Vulnerability Type	Number of Reports	Percentage (%)
1	CWE-79	Cross-Site Scripting (XSS)	8,542	18.6
2	CWE-89	SQL Injection	6,321	13.8
3	CWE-119	Improper Restriction of Operations within Memory Buffer	5,784	12.6
4	CWE-20	Improper Input Validation	4,965	10.8
5	CWE-200	Exposure of Sensitive Information	4,112	9.0
6	CWE-352	Cross-Site Request Forgery (CSRF)	3,876	8.4
7	CWE-125	Out-of-bounds Read	3,245	7.1
8	CWE-22	Path Traversal	2,918	6.4
9	CWE-78	OS Command Injection	2,441	5.3
10	CWE-287	Improper Authentication	1,982	4.3

XSS (CWE-79) represents the most frequently reported category in the analyzed dataset, accounting for 18.6% of all reported cases. SQL Injection (CWE-89) follows with 13.8%, while memory-related vulnerabilities and improper input validation also represent significant categories. These results confirm the continued relevance of XSS in modern web application security.

5.2. Yearly trend analysis of XSS vulnerabilities

The yearly distribution of reported XSS vulnerabilities was analyzed to identify changes in their frequency over the period 2014–2022. Table 2 presents the annual number of reported XSS vulnerabilities and the corresponding percentage change compared with the previous year.

Table 2: Yearly Trend of Reported XSS Vulnerabilities (2014–2022)

Year	Number of XSS Reports	Annual Change (%)
2014	1,020	-
2015	1,145	+12.2
2016	1,238	+8.1
2017	1,174	-5.2
2018	1,086	-7.5
2019	1,012	-6.8
2020	954	-5.7
2021	891	-6.6
2022	1,022	+14.7

The highest number of reported XSS vulnerabilities occurred in 2016, with 1,238 cases. A general decrease followed until 2021, while 2022 showed an increase to 1,022 cases. This variation indicates that improvements in development practices and security mechanisms have not eliminated XSS vulnerabilities. The persistence of XSS highlights the need for continuous vulnerability assessment and improved detection techniques.

5.3. Comparison of XSS Detection Techniques

Table 3 summarizes the main characteristics of commonly used XSS detection approaches.

Table 3: Comparison of XSS Detection Approaches

Detection Method	Principle	Advantages	Limitations
Static Analysis	Analysis of source code without execution	Early vulnerability detection, low cost	High false-positive rate
Dynamic Analysis	Testing running applications with malicious inputs	Detects real exploitation scenarios	Requires application execution
Hybrid Analysis	Combination of static and dynamic methods	Higher detection accuracy	More complex implementation

ONLINE FIRST

Detection Method	Principle	Advantages	Limitations
Machine Learning	Classification of malicious patterns	Adaptable to new attacks	Requires training datasets
Taint Analysis	Tracking untrusted input flow	Effective for injection detection	Computationally demanding

The comparison shows that no single technique can provide complete protection against all XSS variants [18]. Static analysis is useful during development, while dynamic analysis provides a more realistic assessment of running applications. Hybrid approaches combine the advantages of both methods but require greater implementation complexity. Machine learning-based approaches have also been investigated for XSS detection by identifying patterns associated with malicious inputs [8]. They can identify complex patterns but depend on representative training data.

5.4. Analysis of XSS attack types

The three principal XSS categories were also compared according to their characteristics and detection challenges, as presented in Table 4.

Table 4: Classification and Characteristics of XSS Attack Types

XSS Type	Description	Primary Detection Challenge
Reflected XSS	Malicious script delivered through URL or request parameters	Identification of malicious input in HTTP requests and parameters
Stored XSS	Malicious script stored permanently on the server	Detection of persistent malicious content stored on the server
DOM-based XSS	Client-side JavaScript manipulation	Analysis of unsafe client-side data flows and JavaScript execution

Reflected XSS is relatively simple to deliver through crafted requests, whereas Stored XSS can affect multiple users because the malicious payload remains in the application. DOM-based XSS is particularly challenging because the vulnerability occurs in client-side JavaScript processing. These differences demonstrate the need for detection mechanisms capable of addressing multiple XSS variants.

5.5. Experimental evaluation of XSS detection tools

The experimental evaluation was conducted using OWASP ZAP, Burp Suite, and Acunetix on the same intentionally vulnerable web application containing Stored XSS, Reflected XSS, and DOM-based XSS vulnerabilities. The tools were evaluated using Detection Rate (DR), False Positive Rate (FPR), and Execution Time. The comparative results are presented in Table 5.

Table 5: Experimental Results of XSS Detection Tools

Tool	Detection Rate (DR) (%)	False Positive Rate (FPR) (%)	Execution Time (min)
OWASP ZAP	76.7	17.1	18
Burp Suite	93.3	9.4	24
Acunetix	96.7	6.8	15

The results show that Acunetix achieved the highest Detection Rate (96.7%) and the lowest False Positive Rate (6.8%), while also requiring the shortest execution time (15 min). Burp Suite achieved a Detection Rate of 93.3% and a False Positive Rate of 9.4%, with an execution time of 24 min. OWASP ZAP achieved a Detection Rate of 76.7% and a False Positive Rate of 17.1%, with an execution time of 18 min. Under the conditions of the conducted experiment, Acunetix therefore demonstrated the best overall performance among the evaluated tools.

5.6. Evaluation metrics

The selected XSS detection tools were evaluated using Detection Rate (DR), False Positive Rate (FPR), and Execution Time. Detection Rate measures the proportion of existing vulnerabilities correctly identified:

$$DR = \frac{TP}{TP + FN} \times 100 \quad (1)$$

where TP represents correctly detected vulnerabilities and FN represents vulnerabilities that were not detected. False Positive Rate measures the proportion of secure cases incorrectly reported as vulnerable:

$$FPR = \frac{FP}{FP + TN} \times 100 \quad (2)$$

where FP represents incorrectly reported vulnerabilities and TN represents correctly identified secure cases. Execution Time represents the total time required to complete the scanning process. These three metrics provide a combined measure of detection effectiveness, reporting accuracy, and practical scanning efficiency.

ONLINE FIRST

5.7. Discussion

The results indicate that XSS remains a significant and persistent software weakness. Table 1 shows that XSS (CWE-79) represents the most frequently reported vulnerability category in the analyzed dataset. Although the number of reported vulnerabilities generally decreased between 2016 and 2021, the increase observed in 2022 demonstrates that the problem has not been eliminated. This trend is presented in Table 2, which shows the annual changes in reported XSS vulnerabilities between 2014 and 2022. The comparative analysis presented in Figure 1 further illustrates the position of XSS among the most frequently reported software weaknesses during the analyzed period [16]. The continued presence of XSS can be associated with the complexity of modern web applications, extensive client-side JavaScript, third-party libraries, and processing of user-generated content. The comparison of detection approaches confirms that each method has specific strengths and limitations. Table 3 summarizes these differences, highlighting the advantages and limitations of static, dynamic, hybrid, machine learning, and taint analysis approaches. Static analysis enables early identification of potentially vulnerable code but may produce false positives. These findings also emphasize the importance of systematic risk assessment and continuous monitoring when integrating security controls into organizational processes [19]. Dynamic analysis provides more realistic testing of running applications but depends on application coverage and payload generation. Hybrid approaches can provide broader coverage, while machine learning-based methods offer potential for detecting complex patterns but require high-quality datasets. Table 4 further illustrates the distribution of Reflected, Stored, and DOM-based XSS, emphasizing the need for detection mechanisms that address different attack variants.

The experimental evaluation of the selected security tools further demonstrates that detection performance depends on the tool and testing approach. The Detection Rate and False Positive Rate, calculated according to (1) and (2), respectively, were used to compare the effectiveness and accuracy of the selected XSS detection tools. Commercial solutions may provide higher detection coverage and optimized scanning, while open-source solutions such as OWASP ZAP provide accessible and flexible security testing capabilities. No single tool can eliminate the possibility of undetected vulnerabilities. Overall, the findings support a defense-in-depth approach combining secure coding, input validation, context-sensitive output encoding, automated vulnerability scanning, Content Security Policy, and continuous security testing. XSS mitigation should therefore be integrated throughout the software development lifecycle rather than treated as a single testing activity.

The study has several limitations. The statistical analysis is based on publicly available CWE data and therefore may not represent vulnerabilities in private or commercial software. In addition, the experimental results depend on the selected test application, vulnerability scenarios, and tool configurations. Future research should use larger datasets, standardized vulnerable applications, and broader comparative testing of XSS detection solutions. These findings are consistent with broader research showing that XSS detection and prevention require the combination of complementary approaches to address the diversity and evolving nature of XSS attacks [19]. The distinction between stored and reflected XSS and their respective attack scenarios is also consistent with established approaches to XSS analysis and prevention [20].

6. CONCLUSIONS

The experimental evaluation demonstrated that the selected XSS detection tools provide different levels of effectiveness, as summarized in Table 5. Acunetix achieved the highest Detection Rate (96.7%), followed by Burp Suite (93.3%) and OWASP ZAP (76.7%). Acunetix also achieved the lowest False Positive Rate (6.8%), compared with 9.4% for Burp Suite and 17.1% for OWASP ZAP. In terms of execution time, Acunetix completed the scan in 15 minutes, while OWASP ZAP required 18 minutes and Burp Suite 24 minutes. These results indicate that Acunetix provided the best overall performance in the controlled experimental environment. However, OWASP ZAP and Burp Suite provide valuable alternatives depending on specific security testing requirements, available resources, and the need for open-source or manual penetration testing capabilities. The CWE-based analysis confirms that XSS remains one of the most frequently reported software weaknesses. Although a general decrease in reported XSS vulnerabilities was observed between 2016 and 2021, the increase recorded in 2022 demonstrates that the problem has not been eliminated. The persistence of XSS can be associated with the increasing complexity of modern web applications, extensive use of client-side JavaScript, third-party libraries, dynamic content generation, and processing of user-generated data.

The comparison of detection approaches further demonstrates that no individual technique can provide complete protection against all XSS variants. Static analysis is useful for identifying potentially vulnerable code during development, while dynamic analysis provides a more realistic assessment of running applications. Hybrid approaches combine the advantages of both techniques, whereas machine learning-based methods offer potential for identifying complex and previously unseen attack patterns. Their effectiveness, however, depends on application characteristics, testing coverage, and the quality of available datasets.

The analysis also highlights the importance of addressing different XSS variants, including Reflected, Stored, and DOM-based XSS. Effective protection therefore requires a defense-in-depth strategy combining secure coding prac-

ONLINE FIRST

tices, input validation, context-sensitive output encoding, Content Security Policy (CSP), regular vulnerability scanning, and continuous security testing throughout the software development lifecycle. This study has several limitations. The statistical analysis relies on publicly available CWE data, which may not represent all vulnerabilities discovered in private or commercial software. In addition, the experimental evaluation was conducted in a controlled environment and may not fully reflect the complexity of large-scale production applications. Future research should include larger benchmark datasets, real-world web applications, and advanced machine learning and artificial intelligence approaches integrated into automated DevSecOps pipelines.

Overall, the results demonstrate that XSS remains a significant security challenge despite continued advances in web security technologies. The combination of automated vulnerability detection, secure programming practices, and continuous security assessment represents an effective strategy for reducing XSS-related risks in modern web applications.

ACKNOWLEDGEMENTS

The research presented in this article was conducted as part of the project Grant No. 451-03-34/2026-03/200132 at the University of Kragujevac, Faculty of Technical Sciences Čačak, funded by the Ministry of Science, Technological Development and Innovation of the Republic of Serbia.

REFERENCES

[1] G. A. Di Lucca, A. R. Fasolino, M. Mastroianni, and P. Tramontana, "Identifying cross site scripting vulnerabilities in web applications", Proceedings. Sixth IEEE International Workshop on Web Site Evolution, 11 September 2004, Chicago (USA), pp. 71–80, https://doi.org/10.1109/WSE.2004.10013, (2004)

[2] OWASP Foundation, "Cross site scripting (XSS)", Open Worldwide Application Security Project, https://owasp.org/www-community/attacks/xss/, (n.d.)

[3] MITRE, "CWE-79: Improper neutralization of input during web page generation ('cross-site scripting')", Common Weakness Enumeration, https://cwe.mitre.org/data/definitions/79.html, (n.d.)

[4] OWASP Foundation, "Cross site scripting prevention cheat sheet", OWASP Cheat Sheet Series, https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html, (n.d.)

[5] J. Grossman, R. Hansen, P. D. Petkov, A. Rager, and S. Fogie, "XSS attacks: Cross site scripting exploits and defense", 1st ed., Syngress, Burlington (USA), https://doi.org/10.1016/B978-1-59749-154-9.X5000-8, (2007)

[6] D. Das, U. Sharma, and D. K. Bhattacharyya, "Detection of cross-site scripting attack under multiple scenarios", The Computer Journal, Vol. 58(4), pp. 808–822, https://doi.org/10.1093/comjnl/bxt133, (2015)

[7] R. Wang, G. Xu, X. Zeng, X. Li, and Z. Feng, "TT-XSS: A novel taint tracking based dynamic detection framework for DOM cross-site scripting", Journal of Parallel and Distributed Computing, Vol. 118, pp. 100–106, https://doi.org/10.1016/j.jpdc.2017.07.006, (2018)

[8] F. A. Mereani and J. M. Howe, "Detecting cross-site scripting attacks using machine learning", in The International Conference on Advanced Machine Learning Technologies and Applications (AMLTA2018), A. E. Hassanien, M. F. Tolba, M. Elhoseny, and M. Mostafa, Eds., in Advances in Intelligent Systems and Computing, vol. 723, Springer International Publishing, Cham (Switzerland), pp. 200–210, https://doi.org/10.1007/978-3-319-74690-6_20, (2018)

[9] R. Alhamyani and M. Alshammari, "Machine learning-driven detection of cross-site scripting attacks", Information, Vol. 15(7), Art. no. 420, https://doi.org/10.3390/info15070420, (2024)

[10] Q. Abu Al-Haija, "Cost-effective detection system of cross-site scripting attacks using hybrid learning approach", Results in Engineering, Vol. 19, Art. no. 101266, https://doi.org/10.1016/j.rineng.2023.101266, (2023)

[11] ZAP by Checkmarx, "Documentation", https://www.zaproxy.org/docs/, (n.d.)

[12] PortSwigger, "Burp Suite documentation", https://portswigger.net/burp/documentation, (2026)

[13] Invicti, "Acunetix overview", https://www.acunetix.com/support/docs/wvs/overview/, (n.d.)

[14] OWASP Foundation, "OWASP Web Security Testing Guide", Open Worldwide Application Security Project, https://owasp.org/www-project-web-security-testing-guide/, (n.d.)

[15] G. Bošković, Ž. Kužet, B. Zoraja, V. Mučenski, and Z. Čepić, "Risks and safety management in photovoltaic systems", Engineering TODAY, Vol. 5(1), pp. 43–48, https://doi.org/10.5937/engtoday2600014B, (2026)

[16] B. R. K. Mantha, Y. Jung, and B. Garcia de Soto, "Implementation of the Common Vulnerability Scoring System to assess the cyber vulnerability in construction projects", Proceedings of the Creative Construction e-Conference 2020, 28 June–1 July 2020, Online, pp. 117–124, https://doi.org/10.3311/CCC2020-030, (2020)

ONLINE FIRST

[17] I. Hydara, A. B. M. Sultan, H. Zulzalil, and N. Admodisastro, "Current state of research on cross-site scripting (XSS) – A systematic literature review", Information and Software Technology, Vol. 58, pp. 170–186, https://doi.org/10.1016/j.infsof.2014.07.010, (2015)

[18] A. Panović and V. Ružičić, "Implementation of the ISO/IEC 42001 standard in organizational environment and modern information systems", Proceedings of the 13th International Conference on Electrical, Electronics and Computer Engineering (IcETRAN), 8–11 June 2026, Silver Lake (Serbia), pp. 214–219, https://doi.org/10.69994/13Ic26041, (2026)

[19] A. Hannousse, S. Yahiouche, and M. C. Nait-Hamoud, "Twenty-two years since revealing cross-site scripting attacks: A systematic mapping and a comprehensive survey", Computer Science Review, Vol. 52, Art. no. 100634, https://doi.org/10.1016/j.cosrev.2024.100634, (2024)

[20] I. F. Khazal and M. A. Hussain, "Server side method to detect and prevent stored XSS attack", Iraqi Journal for Electrical and Electronic Engineering, Vol. 17(2), pp. 58–65, https://doi.org/10.37917/ijeee.17.2.8, (2021)

ONLINE FIRST