Cross-Site Scripting (XSS): Attacks, Detection Techniques, and Prevention Methods – A Review

Bakul Dehariya[1] Pradeep Pandey[2]
[1]Research Scholar [2]Assistant Professor
[1,2]Department of Computer Science & Engineering
[1,2]SAM College of Engineering and Technology, Bhopal, India

Abstract — Cross-Site Scripting (XSS) is one of the most common and dangerous security vulnerabilities affecting modern web applications. It allows attackers to inject malicious client-side scripts into trusted websites, which can lead to session hijacking, data theft, phishing, and website defacement. Due to the rapid growth of dynamic and interactive web applications, the risk of XSS attacks has increased significantly. This review paper presents a comprehensive overview of Cross-Site Scripting attacks, their types, impacts, and existing defense mechanisms. The study reviews and analyzes various XSS detection and prevention techniques proposed in the literature, including client-side, server-side, and hybrid approaches. It also discusses traditional methods such as static analysis, dynamic analysis, filtering techniques, and proxy-based solutions, along with recent advancements using machine learning and artificial intelligence. Furthermore, the paper highlights the limitations of existing solutions, such as high false positives, performance overhead, and lack of complete protection against all XSS types. Finally, this review identifies key research gaps and emphasizes the need for efficient hybrid and intelligent security frameworks to effectively mitigate XSS attacks in modern web applications.

Keywords: Cross-Site Scripting (XSS), Web Application Security, XSS Detection, XSS Prevention, Client-Side Security, Server-Side Security, Hybrid Security Approach

I. Introduction

Web applications have become an essential part of daily life, providing services such as online learning, banking, shopping, healthcare, and social networking. As these applications increasingly rely on dynamic content and user interaction, security threats have also grown rapidly. Among various web security vulnerabilities, Cross-Site Scripting (XSS) remains one of the most common and dangerous attacks affecting modern web applications. Cross-Site Scripting (XSS) occurs when a web application fails to properly validate or sanitize user input and allows malicious scripts to be injected into web pages. These scripts are executed in the victim's browser, enabling attackers to steal sensitive information such as session cookies, login credentials, and personal data. XSS attacks can also lead to website defacement, phishing attacks, and loss of user trust. Due to these serious consequences, XSS has consistently appeared in the OWASP Top Ten list of critical web application vulnerabilities. XSS attacks are broadly classified into three main types: reflected XSS, stored XSS, and DOM-based XSS. Each type exploits different weaknesses in web applications and requires different detection and prevention strategies. Over the years, researchers have proposed various security mechanisms, including client-side solutions, server-side defenses, and hybrid approaches that combine both. These methods use techniques such as static analysis, dynamic analysis, filtering rules, proxy-based systems, and, more recently, machine learning and artificial intelligence. Despite significant research efforts, many existing solutions suffer from limitations such as high false positive rates, performance overhead, incomplete coverage of XSS attack types, and difficulty in securing legacy web applications.

II. Background on Cross-Site Scripting Attacks

Cross-Site Scripting (XSS) is one of the most serious security threats affecting modern web applications. It occurs when a web application accepts user input and displays it on a web page without proper validation or encoding. Attackers exploit this weakness by injecting malicious client-side scripts, most commonly written in JavaScript. When a user visits the affected page, the browser executes the malicious script, believing it to be trusted content. Although XSS attacks are executed on the client side, their root cause usually exists on the server side due to poor input handling and insecure coding practices. Insecure processing of user inputs through URLs, form fields, cookies, or HTTP headers allows attackers to manipulate web content and gain unauthorized access. Through XSS attacks, attackers can steal session cookies, hijack user accounts, modify web pages, redirect users to malicious websites, or perform phishing attacks. XSS attacks are generally classified into three main types: Reflected XSS, Stored XSS, and DOM-based XSS. Reflected XSS occurs when malicious input is sent to the server and immediately reflected back in the response without being stored. Stored XSS is more dangerous because the malicious script is permanently saved in the server's database and executed whenever users access the infected page. DOM-based XSS occurs entirely in the browser when client-side scripts process untrusted data and dynamically modify the Document Object Model (DOM). With the rapid growth of interactive and data-driven web applications, XSS vulnerabilities have become more common and harder to detect. Many applications rely heavily on client-side scripting, making them attractive targets for attackers. Due to its high impact and widespread occurrence, XSS has consistently been ranked among the top vulnerabilities in web security reports. Understanding how XSS attacks work and how they exploit web application weaknesses is essential for developing effective detection and prevention mechanisms.

A. Cross-Site Script Types

1) Cross-Site Scripting (XSS)

Attacks are commonly classified into three major types based on how and where the malicious script is injected and executed. Each type exploits different weaknesses in web applications and poses varying levels of risk to users and systems.

2) Reflected XSS

Reflected XSS is a non-persistent attack in which the malicious script is sent to the server as part of an HTTP request, usually through a URL or form input. The server processes this input and immediately reflects it back in the response without proper validation or encoding. When the victim clicks on a crafted link or submits malicious input, the script is executed in the user's browser. Since the script is not stored on the server, this attack mainly affects users who interact with the malicious link.

3) Stored XSS

Stored XSS, also known as persistent XSS, is the most dangerous form of XSS attack. In this case, the malicious script is permanently stored in the server's database through vulnerable input fields such as comment boxes, forums, or message posts. Whenever a user accesses the affected page, the stored script is automatically executed in the browser. Stored XSS can impact a large number of users and may lead to serious consequences such as large-scale data theft and session hijacking.

4) DOM-Based XSS

DOM-based XSS occurs entirely on the client side and does not involve direct interaction with the server. This type of attack happens when client-side JavaScript processes untrusted data from sources such as the URL, document location, or browser storage and updates the Document Object Model (DOM) without proper sanitization. The malicious script executes during runtime when the DOM is modified, making this attack harder to detect using traditional server-side security tools.

III. Dissection of Methods

This section presents an overview of the different methods used to detect and prevent Cross-Site Scripting (XSS) attacks in web applications. Based on existing literature, XSS defense techniques are generally categorized according to their deployment location and analysis approach. These methods aim to identify malicious scripts and prevent their execution in the user's browser. XSS mitigation techniques can broadly be divided into client-side methods, server-side methods, and hybrid (client–server) methods. Client-side methods focus on protecting the user's browser by using browser extensions, filters, or proxy-based solutions to block malicious scripts before execution. These methods are effective against some reflected and DOM-based XSS attacks but may introduce performance overhead and depend heavily on browser compatibility. Server-side methods are implemented within the web application itself. These approaches rely on techniques such as input validation, output encoding, filtering rules, and code analysis. Server-side solutions often use static analysis, which examines source code without execution, or dynamic analysis, which monitors application behavior during runtime. While static analysis can identify vulnerabilities early, it may produce false positives. Dynamic analysis, on the other hand, provides more accurate detection but is computationally expensive. Hybrid methods combine both client-side and server-side defenses to provide stronger protection. These approaches leverage the advantages of each side and reduce the limitations of single-layer security mechanisms. Many recent studies suggest that hybrid solutions are more effective in handling complex XSS attacks, including stored and DOM-based XSS.

A. Research Procedure Used for Literature Selection

To conduct a comprehensive review of Cross-Site Scripting (XSS) attacks and their mitigation techniques, a systematic literature selection process was followed. A set of specific keywords was used to retrieve relevant publications, such as Cross-Site Scripting, XSS attacks, XSS detection, XSS prevention, and web application security. The search was limited to studies published between 2014 and 2024 to capture recent advancements and current research trends. After collecting the initial set of papers, duplicates and irrelevant studies were removed through title and abstract screening. The remaining papers were then carefully reviewed based on their relevance, methodology, and contribution to XSS security research. Only studies that clearly discussed XSS attack types, detection techniques, prevention methods, or evaluation results were selected for detailed analysis. This systematic approach ensured that the reviewed literature was both relevant and comprehensive.

B. Methodology Used to Study the Literature

After selecting the relevant research papers, a structured methodology was followed to analyze and compare the existing studies on Cross-Site Scripting (XSS) attacks. The main objective of this analysis was to understand current research trends, commonly used techniques, and limitations of existing XSS detection and prevention methods. Each selected study was examined based on several key parameters, including the type of XSS attack addressed (reflected, stored, or DOM-based), the deployment location of the solution (client-side, server-side, or hybrid), and the analysis technique used.

Step 1: Define Research Scope
- Focus on Cross-Site Scripting (XSS) attacks, detection, and prevention methods
- Time period selected: 2014–2024

Step 2: Literature Search
- Academic databases searched: IEEE Xplore, Springer Link, Science Direct, Google Scholar
- Keywords used: Cross-Site Scripting, XSS detection, XSS prevention, Web Application Security

Step 3: Initial Paper Collection
- Collect all relevant journal papers, conference papers, and review articles
- Remove duplicate and irrelevant publications

Step 4: Screening and Selection
- Title and abstract screening
- Select papers directly related to XSS attack types and mitigation techniques

Step 5: Classification of Studies
- Classify studies based on:
- XSS type (Reflected, Stored, DOM-based)
- Deployment side (Client-side, Server-side, Hybrid)
- Analysis technique (Static, Dynamic, Hybrid)

Step 6: Method Analysis

- Analyze traditional methods (filtering, proxy, rule-based) Analyze intelligent methods (Machine Learning, Artific
- Identify limitations in existing methods
- Highlight need for hybrid, scalable, and efficient XSS defense solutions

Step 7: Comparative Evaluation
- Compare techniques based on detection accuracy, false positives, performance overhead
- Identify strengths and weaknesses of each approach

Step 8: Research Gap Identification

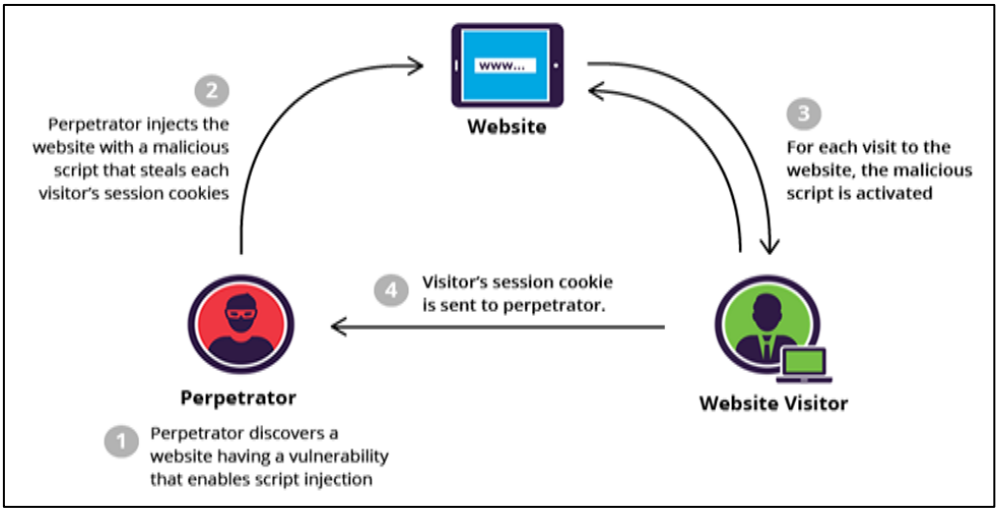

Fig. 1.a: Cross-site script attack scenario

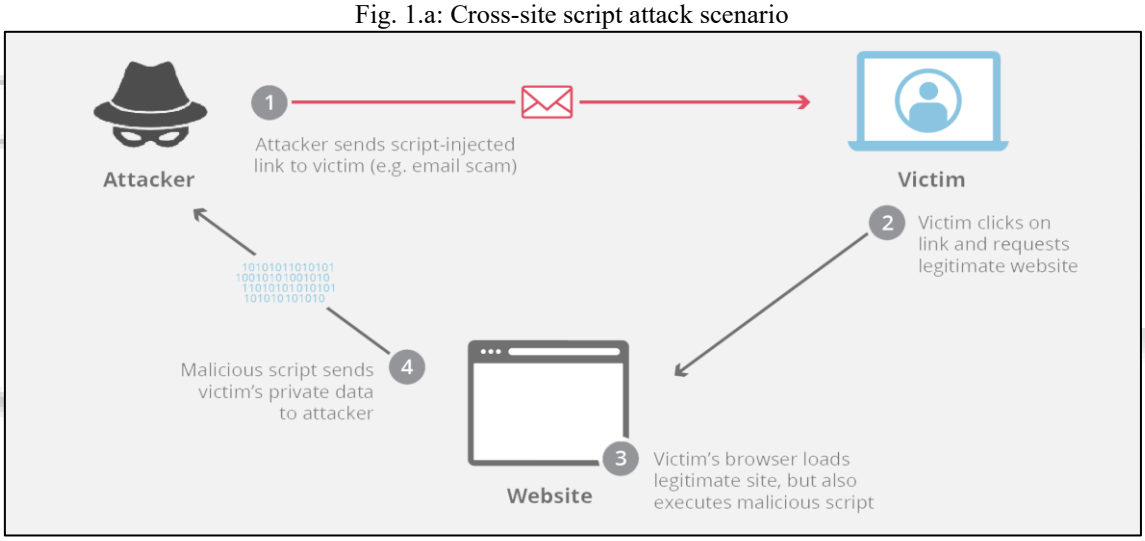

Fig. 1.b: Cross-site script attack scenario

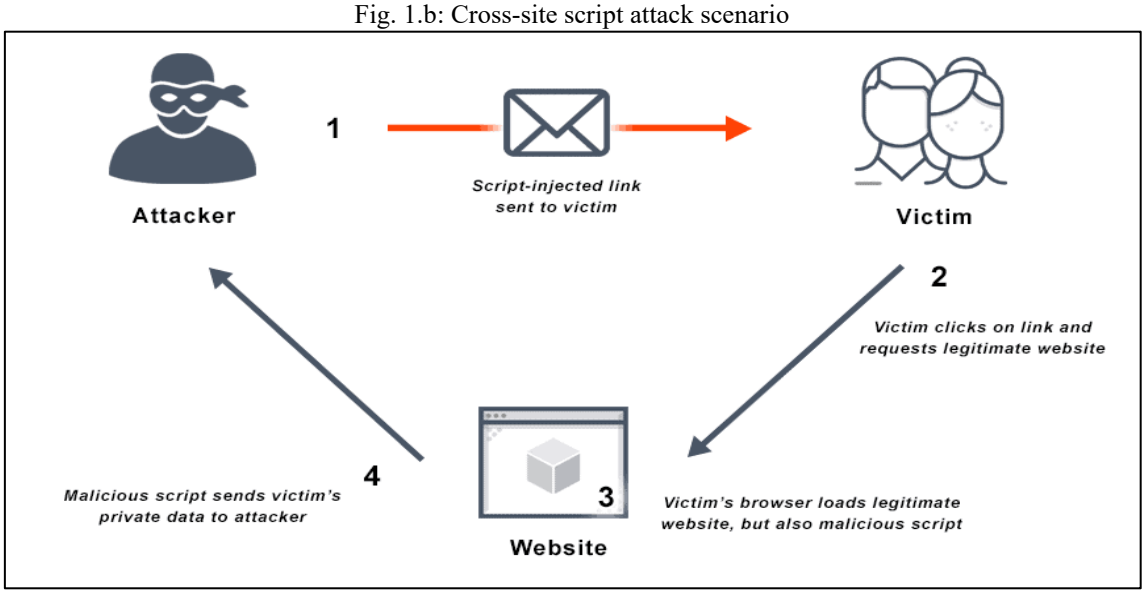

Fig. 1.c: Cross-site script attack scenario

C. Analysis of Scientific Literatures

This section presents a detailed analysis of existing research related to the detection and prevention of Cross-Site Scripting (XSS) attacks. The reviewed studies highlight that XSS remains a major web security issue due to improper input validation and insecure handling of dynamic content in web applications. Researchers have proposed a wide range of solutions using traditional security techniques as well as modern intelligent approaches. Early studies mainly focused on server-side solutions, such as input filtering, output encoding, proxy-based filtering, and static code analysis. These approaches are effective in identifying basic XSS vulnerabilities but often suffer from high false positive rates and limited coverage of complex attacks, especially DOM-based XSS. Static analysis techniques can detect vulnerabilities at an early development stage; however, they are unable to analyze runtime behavior accurately. To overcome these limitations, several researchers proposed dynamic analysis methods, which monitor application behavior during execution. Dynamic techniques are more effective in detecting real attack scenarios and zero-day vulnerabilities, but they introduce higher computational overhead and may affect system performance. Some studies combined static and dynamic techniques to form hybrid approaches, which showed improved detection accuracy and reduced false positives. In recent years, significant attention has been given to machine learning and artificial intelligence-based methods for XSS detection. These approaches use classifiers such as Support Vector Machines, Random Forests, Neural Networks, and Deep Learning models to distinguish between benign and malicious scripts. Although AI-based methods demonstrate promising results, they require large and high-quality datasets and may struggle with generalization across different web environments.

Ref.	Year	Attack Type	Method Type	Traditional / AI	Tool / Technique	Deployment Side	Purpose
[1]	2016	XSS	Static Analysis	Traditional	JSPChecker	Server-Side (Proxy)	Detection of all XSS types
[2]	2017	DOM-XSS	Dynamic Analysis	Traditional	Taint Tracking (TT-XSS)	Client-Side	Detection of DOM-based XSS
[3]	2017	Stored-XSS	Static Analysis	Traditional	Unit Testing Method	Server-Side	Detection of stored XSS vulnerabilities
[4]	2017	HTTP Attacks	Encryption-Based	Traditional	Enc-DNS-HTTP	Client-Side	Secure web communication
[5]	2018	XSS	Machine Learning	AI	SVM, k-NN, Random Forest	Server-Side	Detection of XSS attacks
[6]	2019	XSS & SQLi	Deep Learning	AI	CNN-based IDS	Server-Side	Detection of code injection attacks
[7]	2020	XSS	Dynamic Scanning	Traditional	SW-Scanner	Server-Side	Detection of Service Worker XSS
[8]	2021	XSS	Machine Learning	AI	ERDNS Framework	Client-Side	Classification of XSS attacks
[9]	2021	XSS	Dynamic Analysis	Traditional	XSS Detection Software	Hybrid	Detection of all XSS types
[10]	2022	XSS	Filtering Method	Traditional	XSSFilter	Client-Side	Detection of reflected XSS
[11]	2023	XSS	Deep Learning	AI	LSTM Model	Server-Side	Detection of XSS attacks
[12]	2023	XSS	Neural Network	AI	MLP-Based Model	Client & Server	Detection of XSS attacks
[13]	2024	XSS	Deep Learning	AI	LSTM-TFIDF	Server-Side	Early detection of XSS
[14]	2024	XSS	Ensemble Learning	AI	RF, SVM, CNN, ANN	Server-Side	Detection in cloud environments

Table 1. Discusses the Searched Proposal Approaches for XSS Detection and Prevention

Several researchers have proposed different techniques to detect and prevent Cross-Site Scripting (XSS) attacks using static, dynamic, hybrid, and intelligent approaches.

1) Steinhauser et al. [1] proposed *JSPChecker*, a static analysis-based tool designed to detect context-sensitive XSS vulnerabilities in legacy web applications. This approach tracks data flow from sanitization functions to output sinks and checks whether the applied sanitization matches the output context. JSPChecker can detect all three types of XSS attacks without requiring changes to the runtime environment. However, its applicability is limited mainly to legacy systems.
2) Wang et al. [2] introduced *TT-XSS*, a client-side dynamic detection framework for DOM-based XSS attacks using taint tracking. The approach rewrites JavaScript code and DOM APIs to trace untrusted data during execution. It automatically verifies vulnerabilities by simulating browser behavior. Although effective for DOM-XSS, the method is limited to client-side attacks and requires modification of browser internals.

3) Mohammadi et al. [3] proposed an automated unit-testing-based method to detect XSS vulnerabilities. In this approach, test cases are automatically generated using syntax-based attack patterns and executed on web pages. The method is capable of identifying zero-day XSS vulnerabilities with low false positive rates, but it mainly focuses on detection rather than prevention.
4) Hussain et al. [4] presented *Enc-DNS-HTTP*, an encryption-based mechanism to secure client–server communication and protect against web-based attacks. The approach distributes a server's public key through secure DNS communication and encrypts HTTP traffic. Although effective in securing data transmission, it does not directly address XSS code sanitization.
5) Mereani et al. [5] applied machine learning classifiers such as Support Vector Machines, k-Nearest Neighbors, and Random Forests to detect malicious JavaScript code. By combining linguistic and behavioral features, the classifiers achieved high accuracy on real-world datasets. However, the approach depends heavily on the quality of training data.
6) Abaimov et al. [6] proposed *CODDLE*, a deep learning-based intrusion detection system for SQL injection and XSS attacks. The framework uses customized preprocessing to encode attack-related symbols and employs a Convolutional Neural Network for classification. Experimental results showed strong detection performance, though computational complexity remains a concern.
7) Phakpoom et al. [7] studied Service Worker-based XSS (SW-XSS) and enhanced a scanning tool named *SW-Scanner*. The tool was used to scan popular web applications and successfully identified multiple real-world XSS vulnerabilities. The study highlights emerging attack vectors but focuses mainly on detection.
8) Niranjan et al. [8] introduced *ERDNS*, a hybrid machine learning framework that combines Random Forest, Decision Tree, and Naïve Bayes classifiers using stacking. The approach improves classification accuracy for XSS attacks, but requires careful feature selection and parameter tuning.
9) Alsaffar et al. [9] proposed a software-based dynamic detection method capable of identifying reflected, stored, and DOM-based XSS attacks. The system analyzes HTTP messages, HTML content, and POST requests to detect malicious scripts. While comprehensive, the method increases runtime overhead.
10) Khazal et al. [10] proposed the *Prevent Stored-XSS Server (PSS)* to detect and sanitize stored XSS attacks before data is saved in the database. The solution successfully removes malicious scripts from user input, but is limited to stored XSS scenarios.
11) Alenzi et al. [11] analyzed existing client-side XSS filters and proposed an improved rule-based filtering model named *XSSFilter*. The filter uses XPath-based regular expressions to detect reflected XSS attacks and showed better reliability for custom web applications.
12) Mokbal et al. [12] introduced an Artificial Neural Network-based Multilayer Perceptron (MLP) model for detecting XSS attacks. The model can be deployed on both client and server sides and demonstrates improved detection accuracy with dynamic feature extraction.
13) Younas et al. [13] proposed an AI-based early detection framework using LSTM combined with TF-IDF feature extraction. The method effectively captures both sequential and textual patterns in XSS payloads, though it requires large datasets and computational resources.
14) Alhamyani et al. [14] evaluated multiple machine learning and deep learning models, including Random Forest, SVM, CNN, ANN, and ensemble techniques, for XSS detection. Feature selection methods such as Information Gain and ANOVA were used to improve performance, making the approach suitable for cloud environments.

IV. Discussion

Table 1 summarizes various research studies that propose defense mechanisms against Cross-Site Scripting (XSS) attacks. These approaches can be broadly classified into static analysis, dynamic analysis, and hybrid analysis methods. The studies show that XSS protection techniques are implemented on the server side, client side, or on both sides together (hybrid approach). Server-side solutions mainly focus on source code analysis, input validation, and filtering mechanisms. However, in many real-world situations, server-side protection is difficult to implement because developers may not always have access to the complete source code, especially in third-party or legacy web applications. As a result, relying only on server-side methods is often insufficient to fully protect web applications. To overcome these limitations, browser developers such as Chrome, Internet Explorer, and Firefox have introduced client-side security filters to detect and block malicious scripts before execution. Client-side methods help protect users even when the server-side code is vulnerable. However, these approaches may increase browser overhead and cannot always detect complex attack patterns. The reviewed literature clearly indicates that hybrid approaches, which combine client-side and server-side defenses, provide better protection against XSS attacks. By integrating multiple analysis techniques and deployment layers, hybrid solutions reduce the weaknesses of single-layer security methods and offer more effective and reliable defense against modern XSS threats.

A. Server-Side Methods

Server-side methods focus on detecting and preventing Cross-Site Scripting (XSS) attacks within the web application before malicious content is delivered to the user's browser. These methods are implemented at the application or server level and mainly rely on secure coding practices, input validation, output encoding, and code analysis techniques. Server-side XSS defense techniques are commonly classified into static analysis, dynamic analysis, and filtering or proxy-based approaches.

– Static analysis methods examine the application source code without executing it to identify insecure data flows from user input to output points. Although effective during development, these methods may produce false positives.

– Dynamic analysis methods monitor application behavior at runtime by injecting test inputs and observing responses, providing more accurate detection but with higher computational cost.
– Filtering and proxy-based methods inspect incoming requests and outgoing responses to block known malicious scripts using predefined rules or signatures.

While server-side solutions provide centralized control and protect all users, they have limitations in real-world scenarios. In many cases, developers may not have full access to the source code, especially for third-party or legacy applications. Additionally, server-side methods alone may fail to detect client-side attacks such as DOM-based XSS. Therefore, server-side defenses are most effective when combined with client-side mechanisms as part of a hybrid security framework.

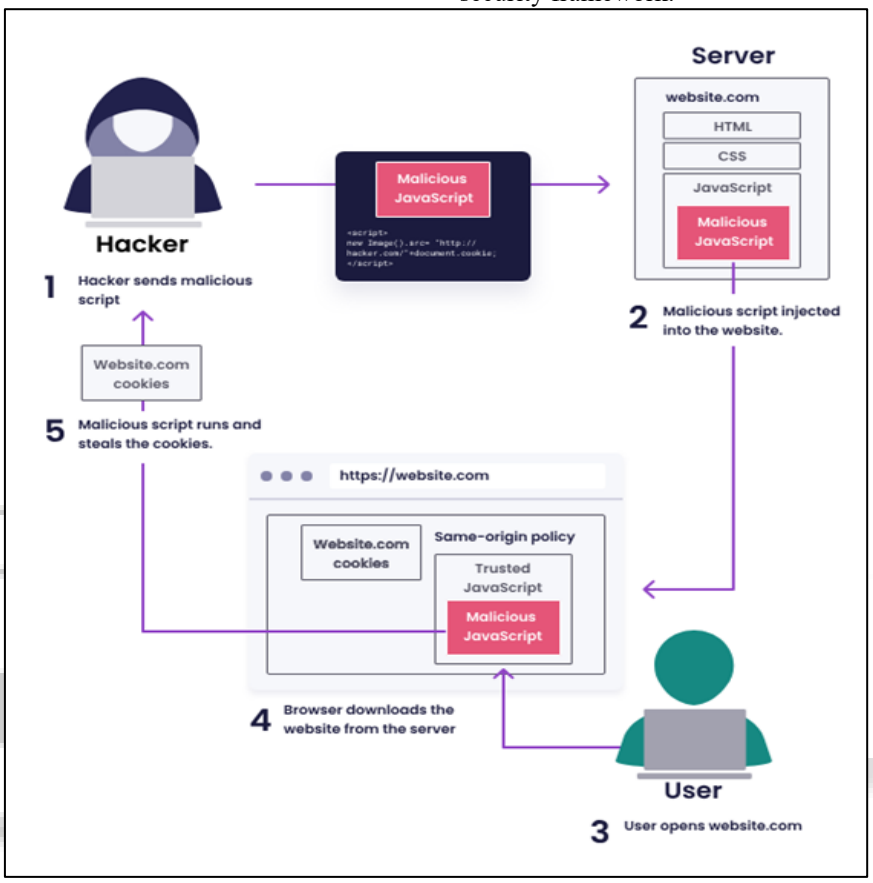

Fig. 4.1: Server-Side Method for XSS Prevention

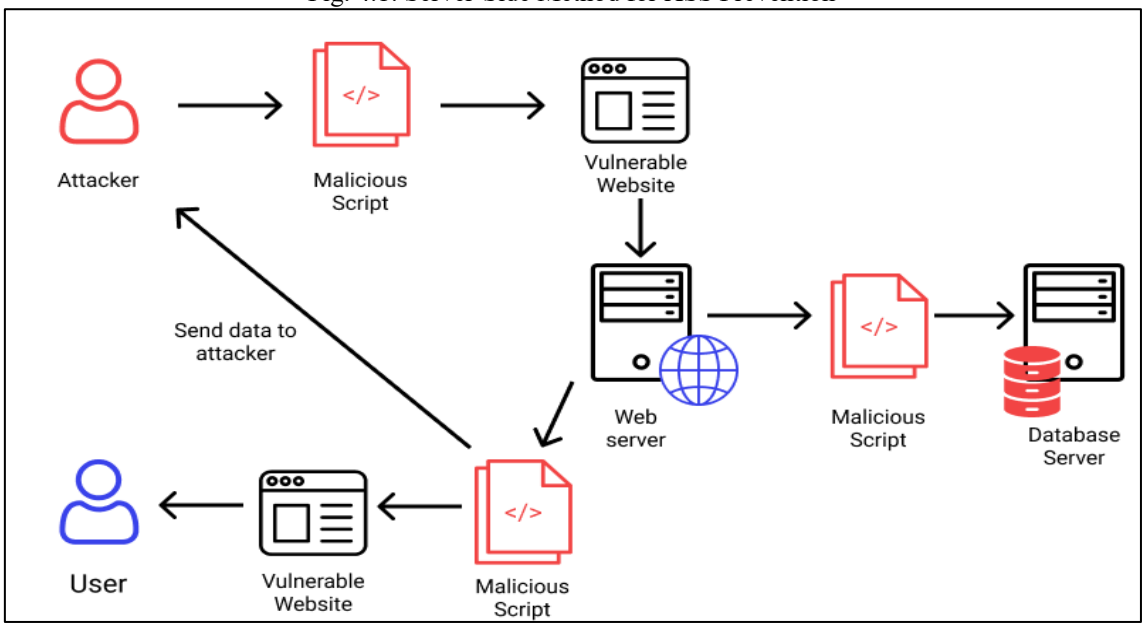

Fig. 4.2: Server-Side Method for XSS Prevention

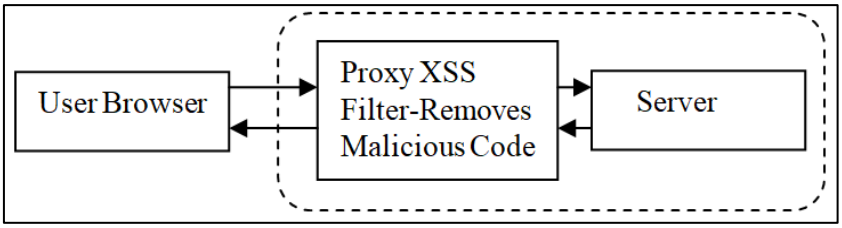

Fig. 4.3: Server-Side Method for XSS Prevention

B. Client-Side Methods

Client-side methods aim to protect users from Cross-Site Scripting (XSS) attacks by detecting and blocking malicious scripts directly within the user's browser. These techniques operate independently of the web application server and are especially useful when server-side code is vulnerable or inaccessible. Client-side defenses are commonly implemented using browser extensions, built-in browser filters, security policies, or proxy-based mechanisms. Most client-side XSS prevention approaches work by analyzing web page content, URLs, and scripts before or during execution. Filtering-based methods inspect HTML and JavaScript code to identify suspicious patterns and prevent script execution. Proxy-based approaches act as an intermediary between the browser and the web server, sanitizing responses before they are rendered. Modern browsers also use security features such as Content Security Policy (CSP) to restrict the execution of untrusted scripts and reduce the risk of XSS attacks. Client-side methods are effective against reflected and DOM-based XSS attacks and can protect users even when server-side vulnerabilities exist. However, these methods have certain limitations. They may increase browser processing overhead, depend on browser compatibility, and may fail to detect complex or obfuscated attack payloads. Therefore, client-side solutions are most effective when used alongside server-side defenses in a hybrid security framework.

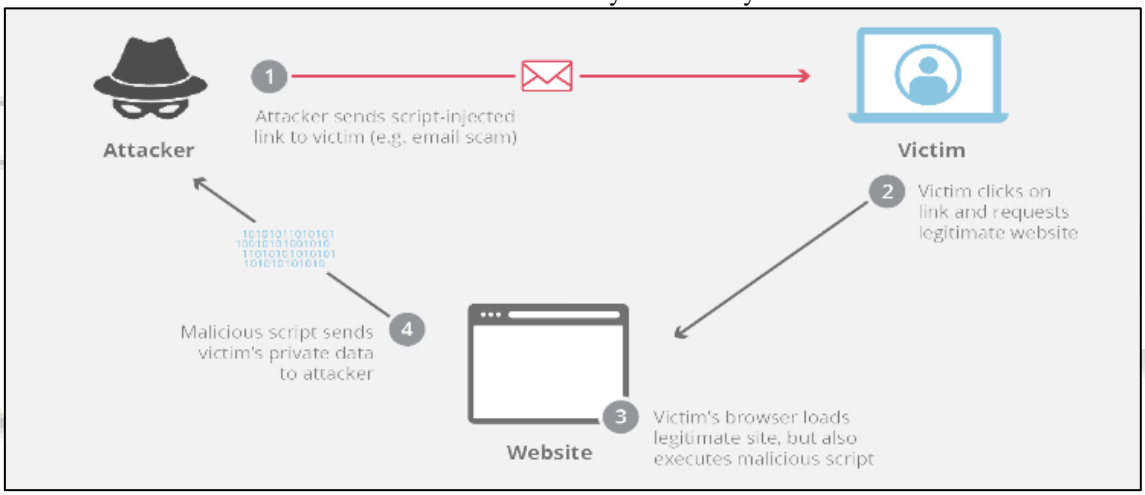

Fig. 4.4: Client-Side Method for XSS Prevention

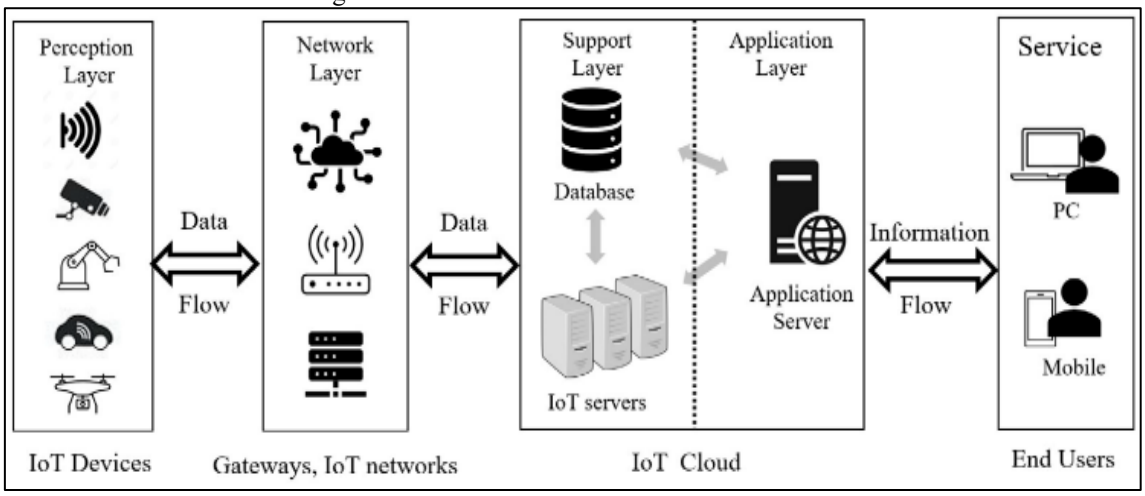

Fig. 4.5: Client-Side Method for XSS Prevention

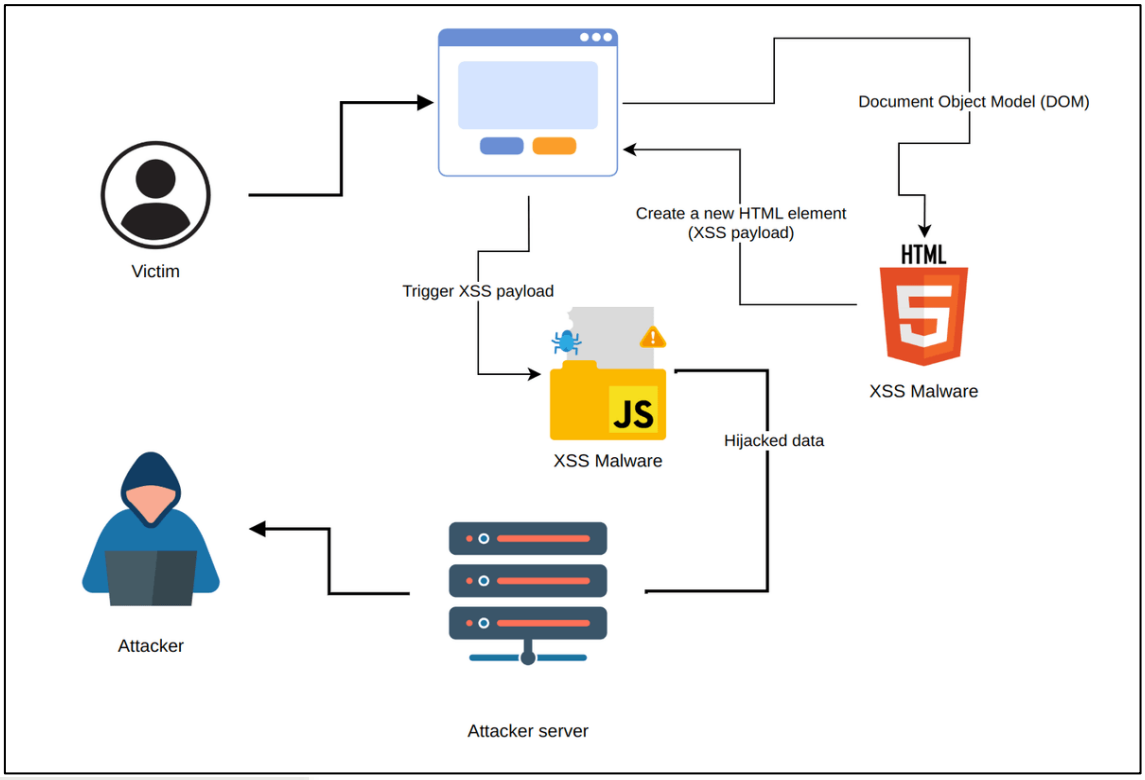

Fig. 4.6: Client-Side Method for XSS Prevention

C. *Hybrid (Client–Server) Method*

Hybrid methods combine both client-side and server-side security mechanisms to provide comprehensive protection against Cross-Site Scripting (XSS) attacks. Instead of relying on a single layer of defense, this approach distributes security checks across the web application server and the user's browser, thereby reducing the limitations of individual methods. On the server side, hybrid frameworks perform input validation, output encoding, static and dynamic code analysis, and database sanitization to prevent malicious scripts from being stored or reflected. On the client side, browser-based filters, security policies (such as Content Security Policy), and runtime script monitoring help detect and block malicious code that may bypass server-side defenses. By working together, both sides ensure that malicious scripts are identified before execution and that any residual threats are neutralized at runtime. Hybrid approaches are particularly effective against stored, reflected, and DOM-based XSS attacks, including those present in legacy web applications. Although hybrid methods may introduce additional computational overhead, their improved detection accuracy, reduced false positives, and broader attack coverage make them more reliable than single-layer solutions. Consequently, hybrid client–server frameworks are widely regarded as the most effective strategy for mitigating modern XSS threats.

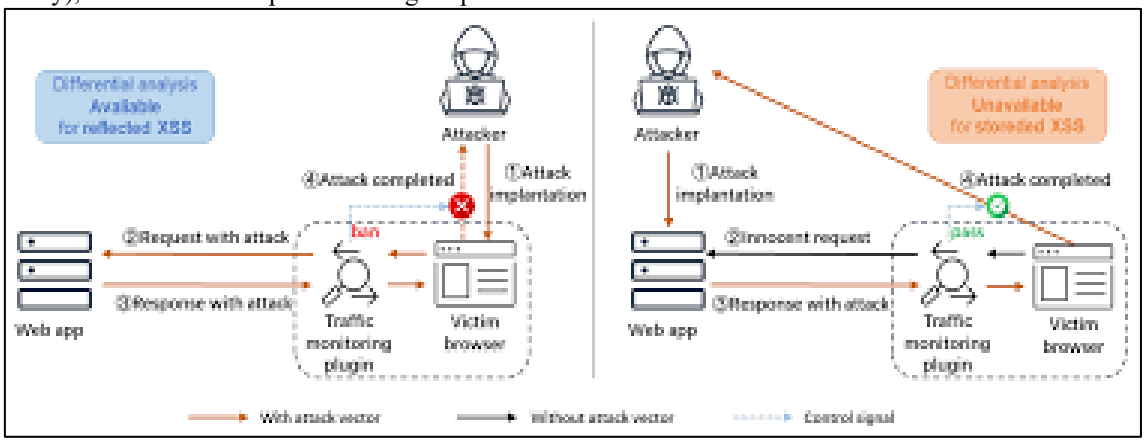

Fig. 4.7: Hybrid Client–Server Method for XSS Prevention

V. Challenges of Cross-Site Scripting and Research Gap

Cross-Site Scripting (XSS) remains one of the most challenging security issues in modern web applications due to the widespread use of dynamic content and user-generated data. Despite the availability of various detection and prevention techniques, XSS vulnerabilities continue to exist in many web applications. One major challenge is that existing security tools and scanners are often insufficient to detect all types of XSS vulnerabilities, especially DOM-

based and zero-day attacks. Many proposed solutions focus on only one type of XSS attack or operate on a single layer, either client-side or server-side, which limits their overall effectiveness. Static analysis techniques often produce high false positive rates, while dynamic analysis methods are computationally expensive and may affect application performance. Client-side solutions, such as browser filters and extensions, can introduce performance overhead and depend heavily on browser compatibility. Although machine learning and artificial intelligence-based approaches have shown promising results, they require large, high-quality datasets and often lack generalization across different web environments. Another significant challenge is that most existing methods are unable to detect or remove already embedded malicious scripts in legacy or compromised web applications. Furthermore, much of the existing research emphasizes detection rather than prevention, leaving applications vulnerable during runtime. These challenges reveal a clear research gap for the development of an efficient, scalable, and hybrid client–server security framework that can detect, prevent, and eliminate all types of XSS attacks while minimizing false positives and performance overhead. Addressing these gaps is essential for improving the security of modern and legacy web applications.

VI. Conclusion

This review analyzed a total of 39 research articles to understand current trends and methodologies used for the detection and prevention of Cross-Site Scripting (XSS) attacks. The analysis shows that most existing studies primarily focus on detecting XSS vulnerabilities, while comparatively less attention is given to preventing attacks by eliminating vulnerabilities at the source code level. Effective removal of XSS weaknesses during development can significantly reduce the chances of successful attacks. The reviewed literature from 2014 to 2024 indicates that many proposed solutions operate only on the client side or the server side, which limits their overall effectiveness. In contrast, hybrid approaches that combine both client-side and server-side mechanisms provide more reliable protection against different types of XSS attacks. Recent studies also demonstrate a growing interest in the use of machine learning and artificial intelligence techniques for XSS detection; however, traditional security methods are still more widely adopted due to their simplicity and lower implementation cost. This review highlights the need for further research on intelligent and hybrid security frameworks that integrate prevention and detection mechanisms. By organizing and analyzing existing approaches, this study provides a structured reference that can assist researchers and developers in designing more effective and robust solutions for mitigating XSS attacks in modern web applications.

References

[1] I. F. Khazal and M. A. Hussain, "Server-side method to detect and prevent stored XSS attack," *Iraqi Journal of Electrical and Electronic Engineering*, vol. 17, no. 2, pp. 58–65, 2021.

[2] A. Steinhauser and F. Gauthier, "JSPChecker: Static detection of context-sensitive cross-site scripting flaws in legacy web applications," in *Proceedings of the ACM Workshop on Programming Languages and Analysis for Security (PLAS)*, 2016, pp. 57–68.

[3] 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, 2018.

[4] M. Mohammadi, B. Chu, and H. R. Lipford, "Detecting cross-site scripting vulnerabilities through automated unit testing," in *IEEE International Conference on Software Quality, Reliability and Security (QRS)*, 2017, pp. 364–373.

[5] M. A. Hussain, H. Jin, Z. A. Hussien, Z. A. Abduljabbar, S. H. Abbdal, and A. Ibrahim, "Enc-DNS-HTTP: Utilizing DNS infrastructure to secure web browsing," *Security and Communication Networks*, vol. 2017, Article ID 9479476, 2017.

[6] F. A. Mereani and J. M. Howe, "Detecting cross-site scripting attacks using machine learning," in *Advances in Intelligent Systems and Computing*, vol. 723, Springer, 2018, pp. 200–210.

[7] S. Abaimov and G. Bianchi, "CODDLE: Code-injection detection with deep learning," *IEEE Access*, vol. 7, pp. 128617–128627, 2019.

[8] P. Chinprutthiwong, R. Vardhan, G. Yang, and G. Gu, "Security study of service worker cross-site scripting," in *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, 2020, pp. 643–654.

[9] A. Niranjan, K. M. Akshobhya, A. S. Chouhan, and P. Tumuluru, "ERDNS: Ensemble learning for efficient cross-site scripting attack classification," in *Communications in Computer and Information Science*, vol. 1483, Springer, 2021, pp. 353–365.

[10] M. Alsaffar et al., "Detection of web cross-site scripting (XSS) attacks," *Electronics*, vol. 11, no. 14, p. 2212, 2022.

[11] K. F. Alenzi and O. A. B. Abbase, "A defensive framework for reflected XSS in client-side applications," *Journal of Web Engineering*, vol. 21, no. 7, pp. 2209–2229, 2022.

[12] F. M. M. Mokbal et al., "MLPXSS: An integrated XSS-based attack detection scheme using multilayer perceptron," *IEEE Access*, vol. 7, pp. 100567–100580, 2019.

[13] F. Younas et al., "An efficient artificial intelligence approach for early detection of cross-site scripting attacks," *Decision Analytics Journal*, vol. 11, p. 100466, 2024.

[14] R. Alhamyani and M. Alshammari, "Machine learning-driven detection of cross-site scripting attacks," *Information*, vol. 15, no. 7, p. 420, 2024.

