A Paradigm shift in XSS-DOM mitigation via dynamic content security policy

Seethalakshmi Perumal*, Dr. P Kola Sujatha*, Krishnaa S† and Thamizharasi M†
Department of Information Technology, Madras Institute of Technology Campus, Anna University Chennai, Tamil Nadu, India
* seethalaxmiperumal@gmail.com; pkolasujatha@annauniv.edu † krishnaa2902@gmail.com

***Abstract*—This paper introduces an innovative approach to mitigate Cross-Site Scripting (XSS) attacks, particularly those exploiting Document Object Model vulnerabilities. Our solution involves the dynamic generation of Content Security Policy (CSP) files coupled with advanced pattern matching techniques. By adapting CSP in real-time, based on the web application's context and employing pattern matching algorithms for early detection of malicious patterns in user inputs, our system provides robust protection against XSS-DOM threats. Through rigorous evaluation against diverse attack scenarios and comparative analysis with existing mitigation strategies, we demonstrate the effectiveness and practicality of our approach. This research aims to revolutionize XSS-DOM security, offering developers, security professionals, and policy makers valuable insights to bolster client-side security in web applications amidst evolving cyber threats.**

***Index Terms*—Web application security, XSS, DOM-based XSS, Prevention strategies, Content Security Policy, Pattern matching techniques, Attack mitigation, Input validation, Taint analysis, Script injection, Risk assessment, Pattern library, Malicious script detection**

I. Introduction

In the ever-evolving landscape of web applications, securing against XSS attacks targeting Document Object Model (DOM) vulnerabilities is paramount. These vulnerabilities not only compromise data but also threaten system integrity, raising concerns for developers and end-users alike. As web applications evolve, defending against XSS-DOM exploits becomes increasingly complex due to dynamic content and nuanced interactions. Balancing robust security with user experience presents a profound challenge, highlighting the necessity of comprehending and addressing XSS-DOM vulnerabilities within the fabric of modern web application security.

Maintaining the security of dynamic online applications presents challenges in mitigating XSS-DOM assaults. Recognizing attack patterns, responding swiftly to threats, and achieving a balance between security and user friendliness are key difficulties. Creative solutions that merge knowledge from dynamic policy adaption, pattern recognition, and online security are required to strengthen security against XSS-DOM attacks in the ever-changing landscape of contemporary online applications.

The primary objective of this project is to develop a proactive, adaptive system capable of identifying and preventing XSS-DOM assaults in web applications. This involves creating a framework for generating dynamic Content Security Policy and implementing sophisticated pattern matching algorithms to efficiently neutralize harmful patterns in user inputs. The aim is to enhance client-side security by offering a dynamic, context-aware, and pattern-sensitive approach, ultimately leading to a paradigm shift in XSS-DOM security. Thorough testing against various XSS-DOM attack scenarios, comparative analysis with existing mitigation techniques, and practical insights for security and web development professionals will be included.

This project, focused on addressing XSS-DOM vulnerabilities, holds significant potential for advancing online application security. It aims to create a system that generates dynamic CSP and utilizes sophisticated pattern matching algorithms to detect and prevent XSS-DOM assaults. The study's outcomes will contribute to a paradigm shift in security tactics by emphasizing pattern sensitivity and flexibility, offering valuable perspectives for web developers, security experts, and legislators. Improving client-side security in web applications through real-time threat adaptation and efficient pattern matching is the study's primary goal.

The project significantly advances online application security, particularly in the realm of XSS DOM mitigation, through the incorporation of sophisticated pattern matching techniques and the creation of a dynamic CSP generation system. By dynamically adapting CSP according to the application's environment and implementing efficient pattern matching techniques, the system enhances defense against XSS-DOM assaults, promoting a more dynamic, context-aware, and pattern-sensitive approach to combat evolving threats.

The parts below are used to organise the entire work done on detection and mitigation of XSS-DOM attacks using CSP. As preparatory work for building this system, in section II a series of concepts, techniques are provided along with their explaination. In section III a literature survey of the various methods and approaches taken to combat XSS based attacks are discussed. Section IV elborates on how the algorithm and techniques proposed addresses the challenges posed by XSS Vulnerabilities. In section V, details about the systems architecture and the working of the algoritm are provided. The system's outcomes are laid out in Section VI followed by the metrics used to evaluate the system in section VII. Section VIII gives the conclusion of the suggested system and mentions the scope for future research as well.

II. Background Work

A. Web attacks

In the rapidly evolving landscape of web applications, the proliferation of cyberattacks poses significant threats to their integrity and security. These attacks, ranging from simple vulnerabilities to sophisticated strategies, compromise the availability, confidentiality, and integrity of online systems. Among the myriad of attack vectors, injection assaults, Cross-Site Scripting (XSS), and other malicious tactics exploit vulnerabilities within web applications, targeting weaknesses in security protocols and human input.

B. Injection

Web applications are susceptible to injection attacks, a category that includes SQL injection, operating system command injection, and the increasingly pervasive XSS. These attacks allow adversaries to insert malicious code into input fields, facilitating unauthorized execution of commands, unauthorized database access, and potentially compromising entire systems. Injection vulnerabilities are recognized as severe security issues by the Open Web Application Security Project (OWASP), advocating for measures such as parameterized SQL queries and input sanitization to mitigate these risks effectively.

C. Cross-Site Scripting

XSS represents a critical security vulnerability that enables attackers to execute malicious code in a victim's web browser, circumventing the same-origin policy and granting unauthorized access to data and actions. Reflected XSS, Stored XSS, and DOM-based XSS are common manifestations of this vulnerability, posing significant risks to dynamic websites that rely on JavaScript for interactivity and functionality .

A particularly serious security risk is posed by DOM-based XSS, which enables attackers to manipulate the client's browser environment to insert harmful payloads into websites. Unlike other forms of XSS, DOM-based XSS alters client-side scripts post-page load, making detection and mitigation challenging. Our project focuses on mitigating XSS vulnerabilities stemming from DOM-based assaults, addressing the specific challenges posed by this type of vulnerability.

D. Pattern Matching

Pattern matching, a fundamental computational method utilized in various disciplines including cybersecurity, involves identifying recurrent structures or patterns in data. This method is essential for effective threat detection and prevention, allowing systems to recognize and respond to potential security threats by establishing patterns associated with malicious actions.

E. Content Security Policy

Content Security Policy is a web security standard designed to mitigate the risks of XSS attacks by allowing web developers to declare content sources, reducing the likelihood of unauthorized script execution. By defining content policies, CSP enhances the protection of web applications against XSS vulnerabilities, serving as a critical layer of defense in web security architecture.

III. Related Work

Web security researchers have introduced innovative approaches to combat XSS attacks, a significant threat to online security. Various methodologies have been proposed, each addressing specific aspects of XSS vulnerability detection and mitigation.

A. Content Security Policy Approach

Guangquan Xu et al. (2022) [1] propose JSCSP, an innovative defence against XSS attacks. Addressing the challenges of low CSP adoption, JSCSP supports real-world browsers and automatically generates security policies. Its self-defined policy restricts JavaScript functions, DOM elements, and data access, offering finer granularity than traditional CSP. Implemented as a Chrome extension, JSCSP demonstrates compatibility with popular JavaScript libraries and outperforms other XSS defence solutions in performance.

B. Enhanced Taint Analysis for Input Validation

Abdalla Wasef Marashdih et al. (2023) [2] introduce an enhanced static taint analysis approach aimed at improving the detection precision of input validation vulnerabilities by minimizing false positives. Traditional static taint analysis often suffers from high false positive rates as it fails to consider the feasibility of program paths, thereby flagging harmless code as potentially vulnerable. The proposed method enhances the analysis by examining the source code to evaluate the feasibility of each path, ensuring that only likely-to-be-executed paths are considered vulnerabilities. This approach tracks tainted variables from their source to the sink statement, implementing an algorithm tailored for PHP to handle variable-handling functions. Evaluated using SARD datasets and large-scale PHP programs, this method demonstrated significant improvements in precision: approximately 44% better for XSS vulnerabilities and 10% better for SQL injection compared to existing tools like WAP and RIPS. Additionally, it outperformed previous symbolic execution studies in detecting vulnerabilities, thus offering a more precise and efficient solution for input validation vulnerability detection.

C. Taint Tracking for Dynamic DOM XSS Detection

Wang et al. (2017) [3] propose TT-XSS, an innovative dynamic detection framework for identifying DOM Cross-Site Scripting (DOM XSS) vulnerabilities through taint tracking. Traditional methods such as black-box fuzzing and static analysis often suffer from high false negative and false positive rates, respectively, while current dynamic analysis approaches, though more effective, are typically complex and resource-intensive. TT-XSS addresses these challenges by dynamically rewriting JavaScript features and DOM APIs to track tainted data throughout the rendering process in the browser. This framework introduces new data types and methods to extend the original data structure's semantic capabilities, enabling

comprehensive analysis of taint traces across all sources, sinks, and transfer processes during page parsing. By doing so, TT-XSS can automatically derive and verify attack vectors, enhancing the detection accuracy. In comparative evaluations, TT-XSS outperformed AWVS 10.0 by detecting 1.8% more vulnerabilities and verifying 9.1% of them automatically. This approach not only improves detection efficiency but also facilitates the generation of attack vectors, streamlining the security testing and vulnerability assessment process, thereby offering a robust solution to safeguard web applications from DOM XSS attacks.

D. Subsequence Matching Approach

Jingyu Zhang et al. (2021) [4] introduce a method for detecting XSS vulnerabilities using a subsequence matching algorithm. By identifying common subsequences between input parameters and generated data, the approach successfully detects XSS vulnerabilities. Its effectiveness in mitigating XSS attacks showcases its potential as a robust solution for enhancing web security against this prevalent threat.

E. Recorded CVE Database Approach

J. C. Pazos et al. (2021) [5] present , a Firefox extension offering client-side protection against XSS attacks. Leveraging prior knowledge of web application HTML templates and rich DOM context, XSnare utilizes an exploit database crafted from recorded CVEs to identify and sanitize injection points, effectively preventing malicious code from appearing in the DOM. XSnare displays a secured version of the site even under attack and defends against 93.8% of tested XSS exploits.

F. Anti-DOM XSS Framework Approach

K. Ali et al. (2014) [6] introduce , a prototype tool demonstrating the practical implementation of an anti-DOM XSS framework. Exhibiting a high success rate in detecting and mitigating various forms of XSS attacks, the framework maintains compatibility with existing web applications. The paper outlines best practices for developers to seamlessly integrate the framework into their projects, enhancing the security posture of web applications and ensuring a safer browsing experience for users.

G. Deep Learning Approach for Securing Web Applications

Tadhani et al. (2024) [7] present a novel deep learning-based method to safeguard web applications from Cross-Site Scripting (XSS) and SQL Injection (SQLi) attacks. Recognizing the limitations of traditional detection methods that rely heavily on signature-based approaches and manual analysis, this study introduces a hybrid model combining Convolutional Neural Networks (CNNs) for feature extraction and Long Short-Term Memory (LSTM) networks for capturing sequential patterns in HTTP requests and payloads. The proposed method leverages the strengths of deep learning to detect subtle attack signatures that traditional techniques might overlook. Data preprocessing involved standardizing and decoding HTTP, SQL, and XSS payloads to create a consistent input for the model. Evaluated on multiple datasets, including a custom testbed and benchmark datasets like HTTP CSIC 2010, the hybrid model demonstrated remarkable performance, achieving accuracy rates of 99.84%, 99.23%, and 99.77% respectively. This approach not only significantly reduces false positives but also enhances the detection of novel attack variants, proving more effective than traditional machine learning methods. This innovative method has potential applications in various network security areas, such as intrusion detection systems and web application firewalls, offering a robust defense against prevalent web application vulnerabilities.

H. Prototype Approach

The importance of understanding different types of XSS attacks and implementing effective preventive measures is emphasized by Ankit Shrivatsava et al. (2016) [8]. Employing input validation, output encoding, secure coding practices, and the adoption of CSP can significantly mitigate the risk of XSS attacks. Regular security audits, code reviews, and employee training are essential for maintaining a robust security posture against evolving XSS threats.

I. Cookie Monitor Approach

Ankit Shrivatsava et al. (2016) [9] introduced which offers a novel approach to detect and prevent XSS attacks by intelligently monitoring cookies and sessions using distinct operators. Employing smart agents and server-side specialists, XTrap scrutinizes and counteracts abuses on both the client and server sides, offering a comprehensive defense strategy against diverse XSS threats.

J. Machine Learning Approach

W. Melicher et al. (2021) [10] propose a hybrid method for detecting DOM XSS vulnerabilities by combining machine learning with traditional taint tracking. The authors address the computational challenges of taint tracking by employing a Deep Neural Network (DNN) classifier as a pre-filter. They collected over 18 billion JavaScript functions through a large-scale web crawl and labeled 180,000 as potentially vulnerable using taint tracking. This data trained the DNN to predict DOM XSS vulnerabilities. The hybrid approach integrates the DNN to classify 97.5% of functions as non-vulnerable, applying taint tracking only to the flagged remainder, which reduces computational overhead by 3.43 times while maintaining a high recall rate of 94.5%. This efficient method offers a promising solution for real-world DOM XSS detection, enhancing web application security.

Gulit Habibi and Nico Surantha et al., (2020) [11] evaluate machine learning methods, including Support Vector Machine (SVM), K-Nearest Neighbour, and Naïve Bayes, enhanced with the n-gram technique for script feature analysis, to address XSS attacks. The simulation demonstrates that combining SVM with the n-gram method yields the highest accuracy, contributing to advancing XSS attack detection through the integration of machine learning and feature analysis methodologies.

The paper by Chen et al. (2019) [12] addresses the escalating seriousness of XSS attacks in web applications, emphasizing the inadequacy of traditional defense methods against evolving and complex threats. It proposes a machine learning-based approach for XSS recognition, categorizes algorithms by recognition strategy, evaluates their pros and cons, and provides insights for the future of XSS defense research, aiming to guide subsequent researchers.

K. XSS Attacks on Web Security

The paper by Ninawe et al. (2019) [13] addresses the significant threat of Cross-Site Scripting (XSS) in web applications, highlighting the ease of attack implementation and the potential consequences, including compromised user data and slow web surfing. It reviews existing research on XSS prevention and introduces a detection framework, offering a viable solution to mitigate these attacks.

The literature survey outlines various innovative approaches to combat XSS vulnerabilities, with a particular focus on DOM-Based XSS. Researchers propose frameworks leveraging CSP, application-specific solutions like XSnare, and machine learning methods for detection. These diverse contributions collectively advance strategies to mitigate XSS threats, demonstrating a range of methodologies and applications

IV. Proposed Solution

Our proposed solution aims to address the challenges posed by XSS vulnerabilities, particularly focusing on DOM-Based XSS attacks. Building upon the insights gained from existing literature and research, our solution offers a comprehensive approach to mitigate XSS threats effectively.

A. Customized CSP Implementation for Dynamic Web Applications

At the heart of our solution lies the deliberate integration of a customized CSP framework into the web application architecture. This bespoke CSP is intricately designed to adapt to the dynamic behaviors inherent in the web application, thereby fortifying its defenses against XSS attacks that specifically target vulnerabilities within the Document Object Model. Through this tailored approach, our solution not only enhances security measures but also fosters adaptability to the evolving threat landscape, ensuring a resilient shield against malicious exploits.

B. Advanced Pattern Matching Algorithms

In addition to dynamic CSP generation, our solution integrates advanced pattern matching algorithms to enhance detection capabilities. These algorithms analyze user inputs in real-time, identifying malicious patterns indicative of XSS attacks. By efficiently detecting and neutralizing harmful patterns, we can proactively mitigate XSS threats before they escalate.

C. Context-Aware XSS Mitigation

Our solution emphasizes a context-aware approach to XSS mitigation, recognizing that the nature of XSS attacks can vary depending on the context of the web application. By considering factors such as user interactions, application behavior, and data flow, we can implement targeted mitigation strategies tailored to specific scenarios, enhancing the effectiveness of our defense mechanisms.

Our proposed solution offers a proactive and context-aware approach for mitigating XSS vulnerabilities, particularly focusing on DOM-Based XSS attacks. By leveraging dynamic CSP generation, advanced pattern matching algorithms, and context-aware mitigation strategies, we aim to enhance the security posture of web applications and protect against evolving XSS threats.

V. Proposed System Architecture and Design

The proposed system architecture for enhancing web application security is meticulously designed to proactively defend against Cross-Site Scripting (XSS) and other injection-based attacks, as illustrated in Figure 1. In this architecture, the user interacts with the web application by providing various inputs, which are subjected to rigorous scrutiny using carefully crafted regular expression patterns. These patterns are designed to detect and mitigate potential security threats, thereby enhancing the overall robustness of the system.

Upon receiving user input, the system initiates an attack detection process. This process employs advanced pattern-matching techniques to analyze the input for any signs of malicious intent. If the input is deemed safe, the user is granted seamless access to the web application. However, if the detection mechanism identifies a potential attack, a dynamic response mechanism is triggered to ensure immediate mitigation of the threat.

A unique aspect of this architecture is the integration of a separate Admin page, which plays a crucial role in maintaining and enhancing the system's security capabilities. When an attack pattern is detected, the system provides detailed information about the threat, allowing administrators to update the pattern library with the new attack pattern through the Admin page. This continuous updating process ensures that the system remains resilient against evolving threats and enhances its ability to detect and prevent future attacks.

Following the detection of a potential attack, a carefully curated Content Security Policy (CSP) is implemented to mitigate the identified threat. The CSP is meticulously designed to restrict the sources from which content can be loaded, thereby preventing the execution of malicious scripts. This policy includes directives that specify trusted sources for images, stylesheets, and scripts, effectively blocking any content from untrusted or suspicious domains. By integrating this well-defined CSP into the web page, the system fortifies its defenses against XSS DOM attacks, ensuring that only safe and verified content is executed.

Moreover, the dynamic and adaptive nature of this architecture not only addresses immediate threats but also contributes

to the continuous improvement of the security framework. By updating the pattern library with newly identified threats and integrating a robust CSP, the system evolves to counteract emerging vulnerabilities, thereby maintaining a high level of security integrity. This sophisticated approach to web application security exemplifies a proactive defense mechanism that is both resilient and adaptive, offering a comprehensive solution to protect against a wide range of security threats.

In conclusion, the proposed system architecture leverages advanced detection mechanisms, continuous pattern library updates via an Admin interface, and a meticulously curated CSP to provide a robust and dynamic defense against XSS and other injection attacks. This comprehensive approach ensures the web application's resilience and security, safeguarding user interactions and maintaining the integrity of the web environment.

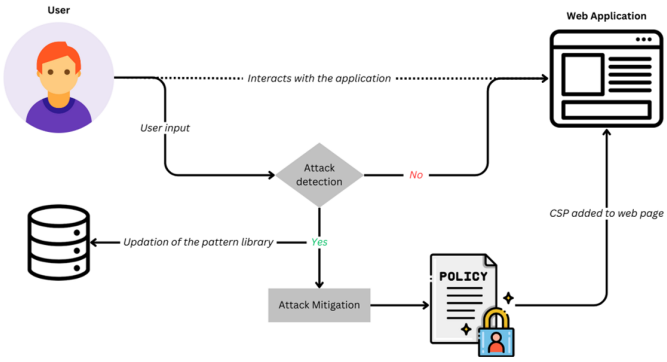

Fig. 1. System architecture

The figure 2 illustrates the critical role of a Content Security Policy (CSP) in fortifying web application security by meticulously regulating the sources from which various types of content can be loaded. In this scenario, "My Website" employs a CSP to rigorously control and restrict the origins of essential resources, such as images, stylesheets, and scripts. The CSP functions as a robust security layer, effectively acting as a gatekeeper that enforces strict directives to permit resources from trusted and verified sources (e.g., https://goodwebsite.com) while blocking potentially harmful content from untrusted or suspicious domains (e.g., https://attacker.com). By specifying these allowed and disallowed sources, the CSP mitigates the risk of Cross-Site Scripting (XSS) attacks. In such attacks, malicious actors inject harmful scripts into webpages, which are then executed in the context of other users' browsers. These scripts can perform unauthorized actions, such as stealing sensitive information, manipulating webpage content, or redirecting users to phishing sites.

Implementing a CSP significantly reduces the likelihood of these attacks by ensuring that only predefined, trusted content sources are executed. This proactive approach not only prevents the execution of malicious scripts but also enhances the overall security posture of the web application by creating a safer browsing environment for users. Furthermore, the CSP's ability to report violations allows administrators to monitor and respond to security incidents effectively, providing an additional layer of defense against evolving threats. By incorporating a well-defined CSP, web developers can create resilient applications that are robust against a wide range of security vulnerabilities, ensuring the integrity and security of their web services.

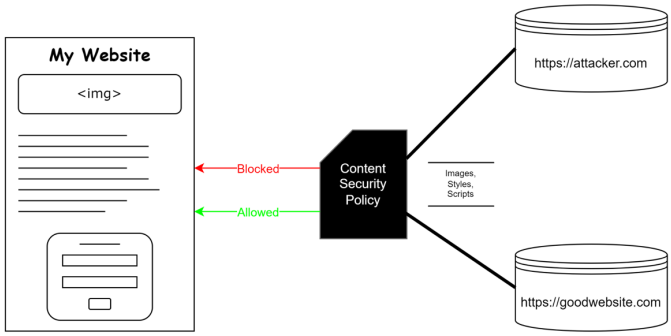

Fig. 2. Content Security Policy

A. Dynamic Response Mechanism for Pattern Matching

The dynamic response mechanism for pattern matching plays a crucial role in the system's security. When a threat is detected, the system triggers dynamic responses, such as updating the pattern library in real-time and generating a tailored Attack Prevention CSP. This CSP serves as a proactive measure against identified threats, adding an additional layer of defense to the web application. The dynamic response mechanism ensures that the security posture of the web application is not only reactive but also predictive, adapting in real-time to evolving cyber threats for comprehensive protection.

B. XSS DOM Attack detection and mitigation

The proposed system architecture for web application security meticulously addresses XSS vulnerabilities, with a particular focus on mitigating DOM-Based XSS attacks. The architecture is meticulously crafted to encompass an intuitive Content Security Policy (CSP) mechanism and advanced pattern matching algorithms, fortifying security measures against potential threats. A pivotal aspect of this architecture lies in the integration of a dynamic response mechanism, facilitating continuous updates to the pattern library and the generation of tailored Attack Prevention CSPs. This proactive approach ensures real-time threat detection and mitigation, enhancing the overall security posture of the web application. By prioritizing user experience alongside robust security measures, the system design strikes an optimal balance, fostering a resilient and secure web environment conducive to safe user interactions. Through diligent implementation of these strategies, web applications can effectively mitigate the risks associated with XSS-DOM attacks, thereby safeguarding sensitive user data and maintaining the trust of their user base.

C. Pattern Matching Techniques

Pattern matching techniques serve as a linchpin in identifying and mitigating XSS-DOM vulnerabilities prevalent in

web applications. These techniques systematically scrutinize input data, code snippets, or discernible patterns to flag potential threats associated with XSS-DOM attacks. Leveraging sophisticated algorithms, security systems adeptly recognize and intercept patterns commonly linked with such attacks, thereby enabling proactive defense measures. By harnessing the power of pattern matching, web applications can preemptively thwart malicious attempts at code injection, bolstering their defenses against evolving cybersecurity threats and ensuring the integrity and security of user data and interactions. The continuous refinement and adaptation of pattern matching algorithms contribute to the resilience of web applications, enabling them to stay ahead of emerging threats and maintain a robust security posture in the ever-evolving digital landscape.

D. Algorithm

Algorithm 1 presents a robust and intricate mechanism tailored for the detection and prevention of Cross-Site Scripting (XSS) and Document Object Model (DOM) attacks within the confines of a web application. The algorithm orchestrates its operations with a meticulous orchestration, commencing with the establishment of a dynamic pattern library meticulously curated from a repository of predefined malicious patterns. This library serves as the cornerstone for subsequent operations, enabling the algorithm to discern and react to potential threats effectively.

As the algorithm traverses the digital landscape of user inputs stemming from various sources within the web application, it meticulously matches each input against the dynamic pattern library, employing a systematic approach that leaves no room for oversight. Upon encountering a match indicative of a potential security breach, the algorithm swiftly activates its defensive mechanisms, triggering an immediate alert and imposing a stringent access blockade to thwart the execution of malicious scripts and prevent any ensuing damage. Furthermore, the algorithm integrates a sophisticated layer of taint analysis into its framework, allowing it to meticulously scrutinize and log any inputs tainted by malicious intent for further analysis and investigation. In instances where the user input defies conventional patterns and eludes detection, the algorithm exhibits its adaptability by augmenting the dynamic pattern library with the newfound input, ensuring its arsenal remains fortified and up-to-date against emerging threats in real-time.

The culmination of the algorithm's endeavors manifests in the meticulous application of a comprehensive Content Security Policy (CSP), meticulously curated to fortify the web application against potential security breaches. By imposing strict directives on content sources, the CSP serves as a formidable bulwark, bolstering the application's defenses and instilling confidence in its ability to repel malicious incursions, thus safeguarding the integrity and security of the digital ecosystem it inhabits.

Algorithm 1 Malicious Script Detection

Require: User input from various sources in the web application.

Ensure: Detection of potential malicious scripts and application of CSP for mitigation.

$DynamicPatternLibrary \Leftarrow PredefinedPatterns$
$UserInput \Leftarrow MultipleSourceInput$
for all $pattern \in \mathcal{D}ynamicPatternLibrary$ **do**
    if $UserInput$ matches $pattern$ **then**
        Alert("Potential malicious script")
        Block("Deny access")
    end if
end for
Perform taint analysis on UserInputs
Identify and log any TaintedInput
if $UserInput \notin \mathcal{D}ynamicPatternLibrary$ **then**
    $DynamicPatternLibrary$ += $[UserInput]$
    Log("New pattern added")
end if
AddCSP(default-src, script-src, ...)

Algorithm 2 embarks on a meticulous journey of fortifying a web application against the pernicious threat of Cross-Site Scripting (XSS) attacks through the meticulous configuration of a Content Security Policy (CSP). Commencing its odyssey with the establishment of CSP reporting, the algorithm lays the groundwork for effective monitoring and management of policy violations by designating a dedicated endpoint tasked with capturing and reporting any breaches. This proactive measure sets the stage for a proactive defense strategy, ensuring that any transgressions against the established policy are promptly detected and addressed.

With the foundations laid, the algorithm proceeds to erect the edifice of the CSP policy, meticulously defining each directive with surgical precision to curtail the avenues of exploitation available to would-be attackers. The policy's architecture is built upon a framework of stringent restrictions, with default sources tightly constrained to permit resources exclusively from the same origin, thus mitigating the risk of unauthorized access and data exfiltration. Furthermore, the algorithm allocates careful consideration to the delineation of permissible sources for images, scripts, and styles, affording them refuge solely from HTTPS and vetted origins, thereby sealing off potential vectors of compromise.

As the policy takes shape, the algorithm enlists the aid of a stalwart guardian in the form of a security middleware, such as Talisman, entrusting it with the duty of enforcing the CSP policy within the application. This symbiotic partnership ensures seamless integration and compliance, empowering the application to assert its authority and repel incursions with unwavering resolve. Through this meticulous configuration, the algorithm erects a formidable bulwark against XSS attacks, fortifying the application's defenses and imbuing it with resilience in the face of adversity, thus safeguarding its sanctity and preserving the trust of its denizens.

Algorithm 2 CSP Configuration for XSS Mitigation

Require: Web application framework instance `app`
Ensure: XSS attacks are mitigated through CSP

Configure CSP Reporting
Set `CSP_REPORT_URI` to a designated endpoint for reporting policy violations.
`app.config['CSP_REPORT_URI']` ← `'/csp-report'`
Initialize CSP Policy
Initialize a dictionary to define the CSP directives.
`csp` ← {}
Define Default Source
Set `'default-src'` to `"'self'"` to restrict resources to the same origin.
`csp['default-src']` ← `"'self'"`
Define Image Sources
Allow images from HTTPS sources and data URIs.
`csp['img-src']` ← `['https:', 'data:']`
Define Script Sources
Allow scripts from the same origin and other trusted sources.
`csp['script-src']` ← `["'self'", '<trusted_script_sources>']`
Define Style Sources
Allow styles from the same origin and other trusted sources.
`csp['style-src']` ← `["'self'", '<trusted_style_sources>']`
Apply CSP Policy
Use a security middleware to enforce the CSP policy in the application.
Apply CSP using `Talisman(app, content_security_policy=csp)`

E. Dynamic Pattern Updation

Our innovative Dynamic Pattern Updating algorithm employs advanced taint analysis to track data flow and assess inputs for suspicious keywords. This dynamic approach ensures the algorithm adapts to evolving threats by learning from real-time interactions. By autonomously updating the pattern library based on observed behaviors, our algorithm enhances the system's resilience against emerging XSS-DOM attacks.

F. Risk Assessment

Risk assessment involves assigning scores to inputs, allowing the system to differentiate between normal and potentially harmful inputs. By setting a threshold, the system prioritizes proactive measures against high-risk patterns, enhancing its ability to preemptively address emerging threats.

G. Pattern Addition

The Pattern Addition mechanism integrates newly identified attack patterns into the system's library in real-time, ensuring it remains responsive to emerging threats. This feature enhances threat detection precision and facilitates agile mitigation against evolving XSS-DOM attack methods.

H. Proposed System Implementation

The proposed system architecture adopts a user-centric approach, analyzing user input using regular expression patterns to enhance security. Detection and mitigation modules incorporate CSP and a whitelist framework to effectively identify and respond to XSS-DOM vulnerabilities, ensuring robust security measures are in place.

VI. Results and Discussions

A. Implementation Environment

Our web application security system utilizes a versatile tech stack comprising Python, Flask, JavaScript, and Sqlite. This choice ensures not only platform independence but also seamless functionality, allowing for robust security measures across different operating systems and browsers. By leveraging these technologies, we establish a solid foundation for effective security implementation while prioritizing compatibility and optimal performance.

B. Injecting Script in Various Ways

Through our comprehensive security assessment, we meticulously scrutinized web applications to identify and evaluate various avenues susceptible to malicious script injection. Our analysis revealed several entry points vulnerable to exploitation, notably including the catalogue page, notes page, and additional potential vectors.

These findings illuminate the pervasive nature of security vulnerabilities inherent in web application development, underscoring the critical imperative for proactive measures to fortify defenses. By recognizing and acknowledging these vulnerabilities, we emphasize the paramount importance of implementing robust input validation mechanisms and adopting secure coding practices throughout the development lifecycle.

Through diligent attention to detail and adherence to industry best practices, we fortify the resilience of web applications against a myriad of injection attacks. Such measures serve as a bulwark against unauthorized access, data breaches, and compromise of user privacy, ensuring the integrity and security of sensitive information within the digital landscape.

Figure 3 vividly portrays a sinister scenario where a malicious script infiltrates a web application through the seemingly innocuous search bar. In this depiction, attackers capitalize on the search functionality as an entry point, cunningly disguising JavaScript code within search queries. Exploiting vulnerabilities in the application's input handling mechanism, these nefarious actors maneuver past the defenses, thereby initiating a cascade of potentially devastating consequences. Such injection techniques pose an imminent threat to the application's security posture, as the execution of arbitrary code within the user's browser environment can lead to widespread compromise and exploitation of sensitive data.

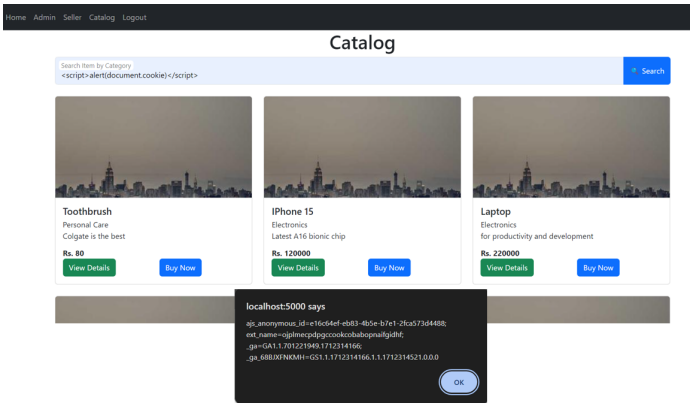

Fig. 3. Inject script via Search bar

Figure 4 presents a chilling depiction of cyber sabotage through URL manipulation. Here, attackers wield the power of URL parameters as a clandestine conduit for injecting malicious scripts into the web application's domain. With meticulous precision, they embed JavaScript code within the URL structure, exploiting inherent weaknesses to evade detection and enforcement measures. Once triggered, these embedded scripts unleash a barrage of malevolent actions, ranging from data exfiltration to session hijacking, thus compromising the integrity and confidentiality of user interactions within the digital realm.

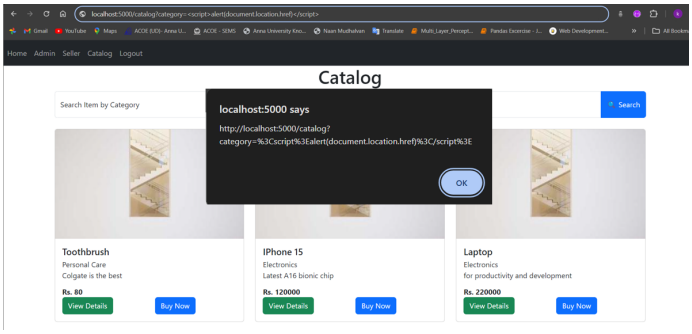

Fig. 4. Inject script via URL

Figure 5 demonstrates the exploitation of a vulnerability in the notes page of the web application. Designed for users to jot down reminders or notes related to purchasing products, this feature inadvertently becomes a vector for malicious script injection. Attackers capitalize on lax input validation by inserting JavaScript code disguised as innocuous notes. Once executed within the application's context, these injected scripts can compromise user data, manipulate page content, or facilitate further exploitation, underscoring the imperative for stringent input sanitization and security measures.

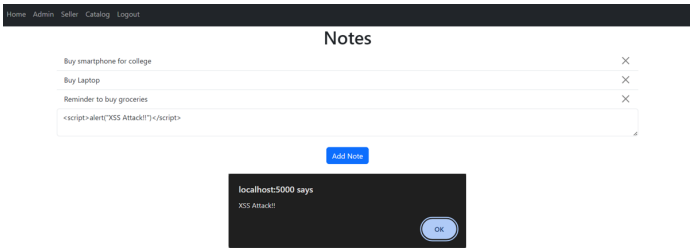

Fig. 5. Inject script via Notes Page

Figure 6 illustrates the injection of malicious scripts via the seller page of the web application. Attackers exploit vulnerabilities in the product information input fields to insert JavaScript code, which can be executed when the catalog page is accessed by unsuspecting users. This type of injection poses a threat not only to the integrity of the application but also to the trust and safety of its users, as it can lead to the compromise of sensitive data or the dissemination of malware.

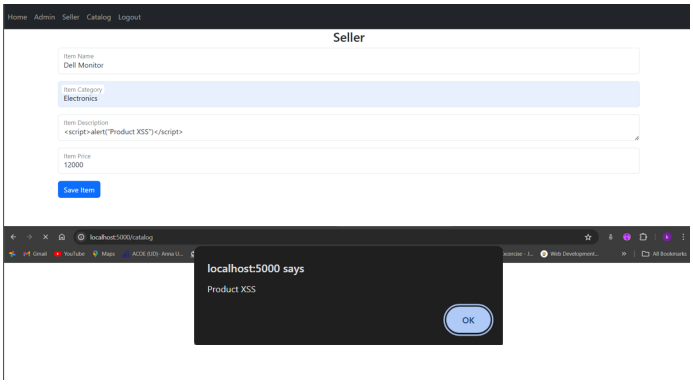

Fig. 6. Inject script via Seller Page

C. Detection of XSS-DOM Attack

Our system employs sophisticated pattern matching techniques to detect XSS-DOM attacks, systematically analyzing user inputs for suspicious sequences indicative of malicious scripts. This advanced analysis ensures that even the most subtle and obfuscated attack patterns are identified with high precision. Upon detection, the system promptly triggers alerts to notify administrators of potential breaches, as illustrated in Figure 7. The system's dynamic response mechanism then updates its pattern library in real-time via a dedicated administrative interface, incorporating newly identified attack patterns to enhance future detection capabilities. This real-time update process ensures that the detection framework remains robust and adaptive to emerging threats.

The proactive approach of our system not only blocks the detected malicious input but also logs the incident for comprehensive analysis. This logging facilitates continuous refinement of detection algorithms, strengthening the system's resilience against sophisticated attack vectors. By continuously monitoring user inputs and adapting to new threat patterns, our system maintains a fortified security posture, effectively

mitigating the risk of XSS-DOM attacks. This continuous monitoring and adaptation are critical in the ever-evolving landscape of cybersecurity, where attackers are perpetually developing new methods to bypass traditional security measures.

Figure 7 demonstrates a scenario in which a potential XSS-DOM attack is detected and subsequently mitigated, showcasing the efficacy of our system's detection and prevention mechanisms. Through this vigilant and adaptive approach, we reinforce the security of web applications, significantly reducing the risk of XSS-DOM attacks and ensuring a secure user experience.

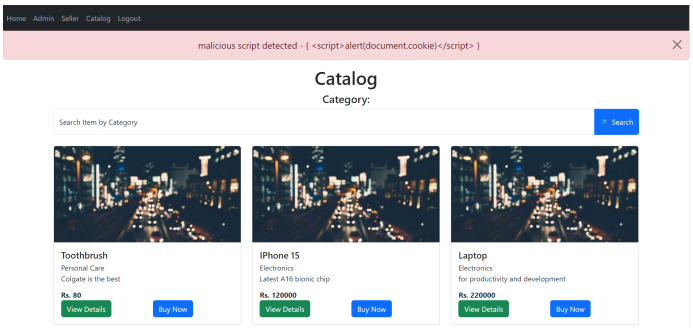

Fig. 7. Detection of attack

D. Pattern Updation

To adapt to emerging threats, our system implements a robust and iterative approach to pattern library updates, as shown in Figure 8. This process is pivotal in maintaining an up-to-date defense mechanism against sophisticated and evolving script-based attacks. The system leverages both observed malicious behaviors and admin-defined keywords to continuously refine its detection capabilities. When a new potentially malicious attack pattern is identified, administrators can access a dedicated Admin page to seamlessly update the pattern library with the new attack pattern. This dynamic updating process ensures that the system's defenses are always one step ahead of attackers, enhancing its ability to detect and mitigate threats in real-time.

The proactive strategy of incorporating new attack patterns into the detection framework is crucial for maintaining an agile and resilient security posture. By continuously monitoring for new threats and updating the pattern library accordingly, our system can quickly adapt to the changing landscape of cybersecurity threats. This approach not only strengthens the system's ability to identify and block novel attack vectors but also reduces the likelihood of false positives, thereby ensuring a more reliable security mechanism. Through continuous refinement and adaptation, we reinforce the system's capability to detect and mitigate XSS-DOM vulnerabilities effectively, providing robust protection for web applications.

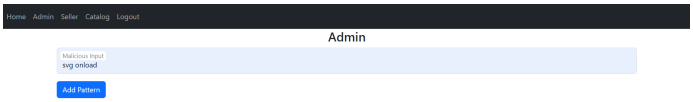

Fig. 8. Pattern Updation

Figure 9 provides a detailed view of the pattern library utilized by our system to detect XSS-DOM attacks. This library comprises a collection of predefined patterns and keywords that are indicative of malicious script behavior. Each entry in the pattern library is meticulously curated based on observed attack vectors and admin-defined keywords, ensuring comprehensive coverage of potential threats. The continuous updating and expansion of this library enable our system to effectively detect and mitigate emerging threats, maintaining robust security for the web application.

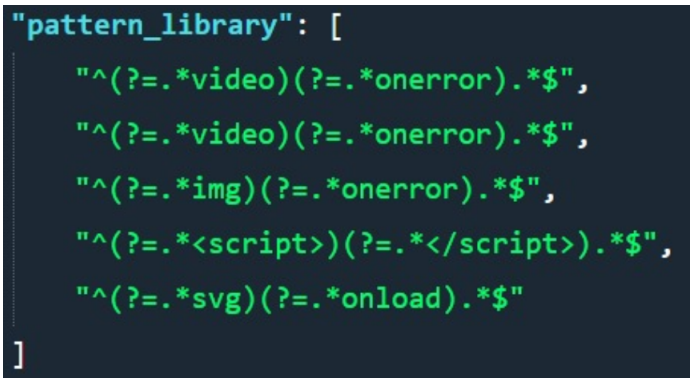

Fig. 9. Pattern Library

E. Mitigation via CSP

Mitigating XSS-DOM risks is a cornerstone of our security strategy, effectively achieved through the integration of a robust Content Security Policy (CSP). The CSP serves as a critical defense mechanism, leveraging a whitelist framework to regulate the execution of scripts and other potentially dangerous content. By specifying trusted sources for various types of content, such as scripts, styles, and images, the CSP restricts the execution of unauthorized scripts, thereby preventing malicious script injections. This proactive approach ensures that only pre-approved and safe resources are executed within the web application environment.

In the event of an attempted attack, the CSP framework promptly triggers alerts and initiates mitigation measures, as depicted in Figure 10. This figure illustrates how the CSP logs warnings and blocks unauthorized scripts in the console, providing real-time feedback on potential security threats. The enforcement of CSP not only prevents the execution of malicious scripts but also offers detailed reporting capabilities that help in identifying and analyzing attack vectors. This dual functionality of prevention and reporting ensures comprehensive defense against client-side scripting and DOM manipulation attacks.

Through stringent enforcement and real-time response mechanisms, the CSP significantly enhances the security of web applications. By continuously monitoring and regulating the content that can be executed, the CSP minimizes the risk of XSS-DOM vulnerabilities, safeguarding sensitive user data and maintaining the integrity of the web application. Additionally, the CSP can be fine-tuned and updated as new threats emerge, ensuring that the security measures remain effective and up-to-date. This adaptability is crucial for maintaining a resilient security posture in the face of evolving cyber threats.

Incorporating CSP into our security framework demonstrates our commitment to proactive threat mitigation and robust application security. The comprehensive nature of CSP, coupled with its ability to enforce strict content policies and provide real-time alerts, makes it an indispensable tool in defending against sophisticated script-based attacks. By leveraging CSP, we not only protect the web application from immediate threats but also establish a foundation for long-term security and user trust.

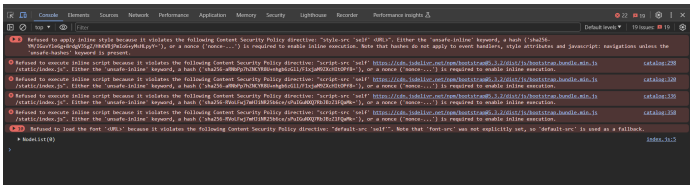

Fig. 10. Warning on attack mitigation

VII. Evaluation Metrics

In our evaluation, various types of injected scripts were detected, highlighting the effectiveness of our security measures. Additionally, we assessed website performance metrics, including search engine optimisation scores as shown in Figure 11, to ensure optimal user experience. The satisfactory performance results, coupled with minimal blocking time and layout shifts, underscore the robustness of our security implementation. Also the system was tested and evaluated on a variety of scripts, few of which are shown in Table I. Thus, by prioritizing both security and user experience, we strive to create a secure and seamless browsing environment for users.

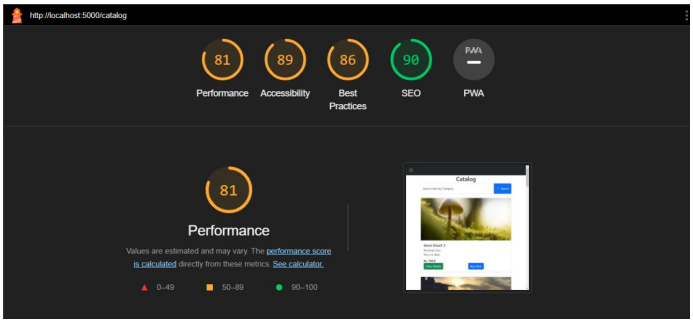

Fig. 11. Performance metrics

TABLE I
Type of scripts detected

S.No	Example Script
1	`<script>alert('XSS')</script>`
2	`<img src="x" onerror="alert('XSS')">`
3	`<a href="javascript:alert('XSS')"> ClickMe</a>`
4	`<input type="text" value="x" onfocus="alert('XSS')">`
5	`<svg onload="alert('XSS')">`
6	`<div onmouseover="alert('XSS')" >HoverMe</div>`
7	`<iframe src="javascript:alert('XSS')">`
8	`<textarea autofocus onfocus=coŏ06efirŏ06d(1) >`
9	`<details ontoggle=coŏ06efirŏ06d`1` >clickMe</details>`
10	`<h1 onclick=coŏ06efirŏ06d(1)>Clickme</h1>`
11	`<h1 ondrag=coŏ06efirŏ06d`1`)>DragMe</h1>`

VIII. Conclusion and Future Work

In conclusion, this paper presented a robust web application security framework designed to effectively counteract XSS-DOM manipulation attack through the implementation of regular expression patterns, heuristic analysis, and a resilient dynamic CSP. Proactive threat detection and dynamic pattern updates contribute to the creation of a secure web environment, emphasizing the importance of ongoing security assessments and user awareness programs. Looking ahead, our future work will explore the integration of machine learning techniques, specifically supervised learning algorithms, to enhance threat detection accuracy. Additionally, emphasis will be placed on feature engineering and unsupervised learning approaches, such as anomaly detection, to further refine detection capabilities. Collaborative efforts with threat intelligence feeds and the implementation of user feedback mechanisms aim to empower end-users in actively contributing to application security, fostering the development of a more adaptive and user-centric security framework capable of addressing evolving web-based security threats effectively.

Declarations

A. *Ethics Approval*

Not Applicable

B. *Data Availability*

Not Applicable

C. *Funding*

Not Applicable

D. *Consent to publish*

Not Applicable

REFERENCES

[1] G. Xu et al., "JSCSP: A Novel Policy-Based XSS Defense Mechanism for Browsers," in IEEE Transactions on Dependable and Secure Computing, vol. 19, no. 2, pp. 862-878, 1 March-April 2022, doi: 10.1109/TDSC.2020.3009472.

[2] Abdalla Wasef Marashdih, Zarul Fitri Zaaba, Khaled Suwais, An Enhanced Static Taint Analysis Approach to Detect Input Validation Vulnerability, Journal of King Saud University - Computer and Information Sciences, Volume 35, Issue 2, 2023, Pages 682-701, ISSN 1319-1578, https://doi.org/10.1016/j.jksuci.2023.01.009.

[3] R. Wang, G. Xu, X. Zeng, X. Li, Z. Feng, TT-XSS: A novel taint tracking based dynamic detection framework for DOM Cross-Site Scripting, J. Parallel Distrib. Comput. (2017), http://dx.doi.org/10.1016/j.jpdc.2017.07.006

[4] Z. Jingyu, H. Hongchao, H. Shumin and L. Huanruo, "A XSS Attack Detection Method Based on Subsequence Matching Algorithm," 2021 IEEE International Conference on Artificial Intelligence and Industrial Design (AIID), Guangzhou, China, 2021, pp. 83-86, doi: 10.1109/AIID51893.2021.9456515.

[5] J. C. Pazos, J. -S. Legare and I. Beschastnikh, "XSnare: Application-specific client-side cross-site scripting protection," 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), Honolulu, HI, USA, 2021, pp. 154-165, doi: 10.1109/SANER50967.2021.00023.

[6] K. Ali, A. Abdel-Hamid and M. Kholief, "Prevention Of DOM Based XSS Attacks Using A White List Framework," 2014 24th International Conference on Computer Theory and Applications (ICCTA), Alexandria, Egypt, 2014, pp. 68-75, doi: 10.1109/ICCTA35431.2014.9521633.

[7] Tadhani, J.R., Vekariya, V., Sorathiya, V. et al. Securing web applications against XSS and SQLi attacks using a novel deep learning approach. Sci Rep 14, 1803 (2024). https://doi.org/10.1038/s41598-023-48845-4

[8] A. Shrivastava, S. Choudhary and A. Kumar, "XSS vulnerability assessment and prevention in web application," 2016 2nd International Conference on Next Generation Computing Technologies (NGCT), Dehradun, India, 2016, pp. 850-853, doi: 10.1109/NGCT.2016.7877529.

[9] A. Shrivastava, V. K. Verma and V. G. Shankar, "XTrap: Trapping client and server side XSS vulnerability," 2016 Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC), Waknaghat, India, 2016, pp. 394-398, doi: 10.1109/PDGC.2016.7913227.

[10] W. Melicher, C. Fung, L. Bauer and L. Jia, "Towards a lightweight hybrid approach for detecting DOM XSS vulnerabilities with machine learning", Proc. Web Conf., pp. 2684-2695, Apr. 2021.

[11] G. Habibi and N. Surantha, "XSS Attack Detection With Machine Learning and n-Gram Methods," 2020 International Conference on Information Management and Technology (ICIMTech), Bandung, Indonesia, 2020, pp. 516-520, doi:10.1109/ICIMTech50083.2020.9210946.

[12] Chen, X., Li, M., Jiang, Y., Sun, Y. (2019). A Comparison of Machine Learning Algorithms for Detecting XSS Attacks. In: Sun, X., Pan, Z., Bertino, E. (eds) Artificial Intelligence and Security. ICAIS 2019. Lecture Notes in Computer Science(), vol 11635. Springer, Cham.

[13] Ninawe, S., Wajgi, R. (2020). Detection of DOM-Based XSS Attack on Web Application. In: Balaji, S., Rocha, Á., Chung, YN. (eds) Intelligent Communication Technologies and Virtual Mobile Networks. ICICV 2019. Lecture Notes on Data Engineering and Communications Technologies, vol 33. Springer, Cham.

[14] I. Dolnák, "Content Security Policy (CSP) as countermeasure to Cross Site Scripting (XSS) attacks," 2017 15th International Conference on Emerging eLearning Technologies and Applications (ICETA), Stary Smokovec, Slovakia, 2017, pp. 1-4, doi: 10.1109/ICETA.2017.8102476.

[15] J. Harish Kumar and J. J Godwin Ponsam, "Cross Site Scripting (XSS) vulnerability detection using Machine Learning and Statistical Analysis," 2023 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 2023, pp. 1-9, doi: 10.1109/ICCCI56745.2023.10128470.

[16] J. Pan and X. Mao, "Detecting DOM-Sourced Cross-Site Scripting in Browser Extensions," 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), Shanghai, China, 2017, pp. 24-34, doi: 10.1109/ICSME.2017.11.

[17] J. Pan and X. Mao, "DomXssMicro: A Micro Benchmark for Evaluating DOM-Based Cross-Site Scripting Detection," 2016 IEEE Trustcom/BigDataSE/ISPA, Tianjin, China, 2016, pp. 208-215, doi: 10.1109/TrustCom.2016.0065.

[18] K. Ali, A. Abdel-Hamid and M. Kholief, "Prevention Of DOM Based XSS Attacks Using A White List Framework," 2014 24th International Conference on Computer Theory and Applications (ICCTA), Alexandria, Egypt, 2014, pp. 68-75, doi: 10.1109/ICCTA35431.2014.9521633.

[19] K. Santithanmanan, "The Detection Method for XSS Attacks on NFV by Using Machine Learning Models," 2022 International Conference on Decision Aid Sciences and Applications (DASA), Chiangrai, Thailand, 2022, pp. 620-623, doi: 10.1109/DASA54658.2022.9765122.

[20] P. Wang, J. Bangert and C. Kern, "If It's Not Secure, It Should Not Compile: Preventing DOM-Based XSS in Large-Scale Web Development with API Hardening," 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), Madrid, ES, 2021, pp. 1360-1372, doi: 10.1109/ICSE43902.2021.00123.

[21] A. F. Maskur and Y. Dwi Wardhana Asnar, "Static Code Analysis Tools with the Taint Analysis Method for Detecting Web Application Vulnerability," 2019 International Conference on Data and Software Engineering (ICoDSE), Pontianak, Indonesia, 2019, pp. 1-6, doi: 10.1109/ICoDSE48700.2019.9092614.