FUDMA Journal of Sciences (FJS)
ISSN online: 2616-1370
ISSN print: 2645 - 2944
Vol. 10 No. 14, July, 2026, pp 65 – 69
DOI: https://doi.org/10.33003/fjs-2026-1014-4467

Client-Side HTML Encoding for the Mitigation of DOM-Based Cross-Site Scripting

[1]Zerah Abba, *[1]Fatimah Adamu-Fika, [2]Jude Enenche Ameh, [1]Dauda Sule, [1]Usman Adedayo Adeniyi, [1]Samaila Musa Abdullahi and [1]Maryam Safiyanu Masari

[1]Department of Cyber Security, Air Force Institute of Technology, Kaduna State, Nigeria.
[2]Department of Computing, Sheffield Hallam University, Sheffield, UK.

*Corresponding authors' email: faateemahfika@gmail.com

Received: 17 Jan. 2026
Accepted: 19 Feb. 2026
Published: 14 Aug. 2026

ABSTRACT
Cross-Site Scripting (XSS) remains a pervasive threat to web application security, with DOM-based XSS attacks posing particular challenges due to their execution within the client-side environment. This study proposes a client-side defence mechanism based on HTML encoding to mitigate DOM-based XSS vulnerabilities by sanitising user inputs prior to rendering in the Document Object Model (DOM). The mechanism was evaluated in a controlled environment using representative attack scenarios, including script injection, event-based exploitation, and attribute manipulation. Results demonstrated that attack payloads which executed successfully in the unprotected configuration were rendered as inert text in the protected system, indicating effective mitigation of the evaluated DOM-based XSS scenarios. The proposed approach is lightweight, introduces minimal additional computational overhead, and is straightforward to integrate into existing client-side workflows. This study contributes a practical and targeted solution for addressing client-side XSS vulnerabilities and outlines directions for future work, including extending protection to other XSS types, improving compatibility with legacy browsers, and assessing scalability through large-scale performance evaluations.

Keywords: Polyherbal Formulation, Antidiabetic, Syrup, LD50, GC-MS, Diabetes Mellitus

INTRODUCTION

Web applications play a crucial role in modern technology ecosystems, enabling dynamic content delivery and interactive user experiences. Despite their advantages, these systems are often vulnerable to exploitation, with Cross-Site Scripting (XSS) ranking among the most persistent and critical threats. XSS attacks occur when an attacker injects malicious scripts into web applications, targeting the end user's browser and compromising sensitive data, application integrity, or session controls. The Open Web Application Security Project (OWASP) consistently identifies XSS as a top threat, underscoring its prevalence in web application vulnerabilities (OWASP, 2025). Among the three primary categories of XSS — stored, reflected, and Document Object Model (DOM)-based XSS poses a unique challenge due to its execution entirely within the client-side environment (Shar & Tan, 2012).

DOM-based XSS attacks exploit vulnerabilities in client-side scripts that dynamically manipulate the Document Object Model (DOM) using untrusted data during browser-side execution. Unlike server-side XSS, where malicious input is processed on the server before being sent to the user's browser, DOM-based XSS occurs when unsafe client-side scripts dynamically update the DOM with unsanitised input. This bypasses traditional server-side defences such as input validation and output encoding (Stock et al., 2014). For example, when data returned from backend components, such as stored procedures, is directly inserted into the DOM without appropriate client-side sanitisation, attackers may exploit this rendering stage to inject and execute malicious scripts. Such vulnerabilities can result in significant consequences, including data theft, session hijacking, and unauthorised account access (Gupta & Gupta, 2017; Galán et al., 2010).

Previous studies have proposed various approaches for detecting and mitigating web application vulnerabilities, including SQL injection and XSS attacks (Maina, 2024). Server-side mechanisms, including input sanitisation and output encoding, are widely adopted and effective for stored and reflected XSS attacks (Abikoye et al., 2020; Weamie, 2022). However, these solutions often overlook client-side vulnerabilities, particularly in dynamic web applications where the DOM is frequently updated using user-generated content (Hydara et al., 2015). Runtime monitors and machine learning-based tools have also been explored as potential solutions, offering broader detection capabilities but introducing computational overhead and implementation complexities (Shar et al., 2015). In contrast, lightweight mechanisms targeting specific vulnerabilities in client-side environments remain underexplored, creating a gap in addressing DOM-based XSS.

This study addresses this gap by introducing a client-side defence mechanism that applies HTML encoding to user-supplied data immediately before DOM rendering in the browser, thereby preventing script execution during client-side runtime. Unlike existing server-side approaches, the proposed method focuses explicitly on securing DOM manipulations associated with stored procedures, a critical yet underexamined vector for DOM-based XSS attacks. The mechanism is evaluated in controlled environments, comparing its performance and effectiveness in unprotected and protected systems. The results demonstrate its ability to prevent DOM-based XSS vulnerabilities while maintaining system performance, offering a practical solution for developers and organisations seeking lightweight and efficient defences.

The findings contribute to the growing body of web application security research by presenting an accessible and effective client-side defence against DOM-based XSS, complementing rather than replacing existing server-side protections.

MATERIALS AND METHODS

The proposed approach addresses DOM-based XSS vulnerabilities by implementing a client-side defence mechanism that leverages HTML encoding. This method focuses on sanitising user inputs dynamically before rendering them in the DOM, ensuring that malicious scripts cannot execute in the client-side environment. The methodology involves two key components: the proposed mechanism and its evaluation using unprotected and protected system configurations. Figure 1 illustrates the client-side evaluation workflow, highlighting the contrast between unprotected DOM insertions and protected rendering using HTML encoding.

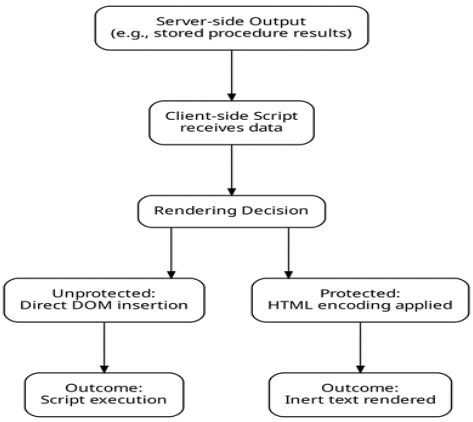

Figure 1: Client-side evaluation workflow for DOM-based XSS mitigation using HTML encoding

The mechanism utilises HTML encoding to convert potentially malicious characters in user inputs into their encoded equivalents. For instance:

i. < is encoded as <.
ii. is encoded as >.
iii. " is encoded as ".
iv. ' is encoded as '.

This transformation ensures that user inputs are displayed as plain text in the DOM rather than being executed as code. Unlike server-side validation, which occurs before data is sent to the browser, this client-side mechanism targets vulnerabilities that arise from unsafe DOM manipulations during runtime, making it suitable for mitigating DOM-based XSS attacks that originate from dynamic client-side rendering rather than server-side processing. The use of a lightweight HTML encoding library ensures minimal performance impact while maintaining high security standards, as the approach requires no runtime monitoring, server-side modification, or complex instrumentation (Dalai et al., 2017).

Example Implementation:

```
Function encodeHTML (input) {
const map = {
'<': '&lt;',
'>': '&gt;',
"": '&quot;',
"'": '&#39;',
'&': '&amp;',};
Return input.replace(/[<>"'&]/g, (char) => map[char]);}
// Usage example
document.getElementById('output').innerHTML=encodeHTML(userInput);
```

The mechanism was implemented and tested in a controlled web application environment, focusing on two key scenarios: an unprotected system and a protected system. These environments were designed to simulate common DOM-based XSS scenarios involving server-side outputs (e.g., data returned from stored procedures).

The unprotected system displayed stored procedure outputs directly in the DOM without sanitisation. For example:

```
document.getElementById('output').innerHTML= userInput;
```

This allowed attackers to inject malicious scripts, such as:

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

which executed upon rendering in the browser.

In the protected system, all user inputs were passed through the HTML encoding mechanism before rendering in the DOM. This transformation neutralised any potentially harmful characters, ensuring that the input was displayed as plain text rather than executable code.

Effectiveness was assessed through execution-based behavioural validation, based on whether DOM-based XSS payloads that executed successfully in the unprotected configuration were rendered as inert text in the protected configuration.

The evaluation procedure followed a structured sequence of steps, as illustrated in Figure 2:

i. Set up an unprotected environment with server-side outputs rendered directly in the DOM.
ii. Simulate representative DOM-based XSS attacks using various payloads, including script injection and event-based exploitation.
iii. Deploy the HTML encoding mechanism to sanitise user inputs dynamically.
iv. Retest the system with the same attack payloads to evaluate the effectiveness of the protection.

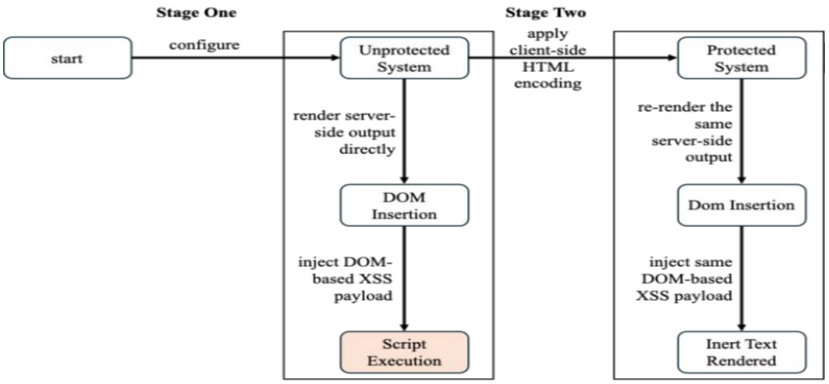

Figure 2: Output-driven process flow for execution-based behavioural validation of DOM-based XSS mitigation

RESULTS AND DISCUSSION

The HTML encoding mechanism was implemented in a controlled environment and tested against common DOM-based XSS attack scenarios. The findings are summarised in Table 1.

Table 1: Summary of DOM-Based XSS Prevention Outcomes

Attack Scenario	Status (Unprotected)	Status (Protected)	Technique Used
Script injection (<script> tag)	Successful	Prevented	HTML Encoding
Event-based attack (onload)	Successful	Prevented	HTML Encoding
Attribute manipulation (src)	Successful	Prevented	HTML Encoding

The results presented in Table 1 show that the evaluated DOM-based XSS attacks were successful in the unprotected system but prevented in the protected system. Figure 3 illustrates the result of the script injection attack in the unprotected system. When the injected script was processed, it executed in the browser and generated the displayed alert, confirming that the user input was rendered without sanitisation. In contrast, Figure 4 shows the behaviour of the protected system before and after the same script injection payload was processed. Rather than executing the script, the system rendered the payload as text on the page. This demonstrates that the HTML encoding mechanism neutralised the executable content before DOM rendering.

The proposed mechanism was evaluated in relation to existing XSS prevention approaches, highlighting its unique advantages. Unlike server-side validation and output encoding, which primarily focus on sanitising data before it reaches the browser, the HTML encoding mechanism specifically targets vulnerabilities that arise during client-side DOM manipulations. The mechanism successfully prevented all simulated attack scenarios, demonstrating complete protection against the evaluated DOM-based XSS attack cases.

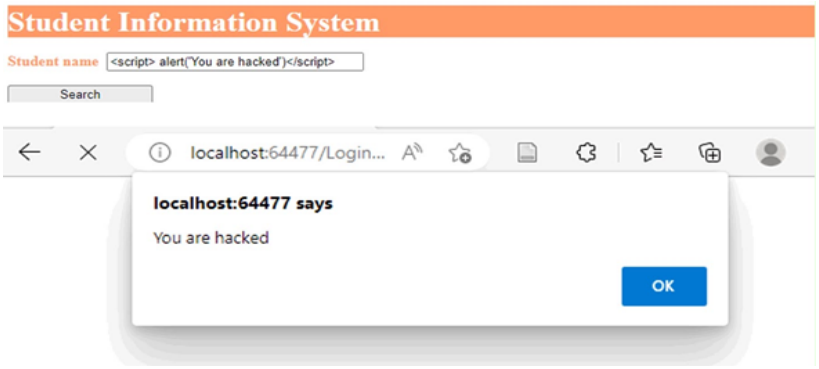

Figure 3: Script injection result in the unprotected system

In terms of implementation overhead, the mechanism introduces minimal performance impact. During controlled testing, the observed average response time increase was less than 2 ms, indicating that the encoding process does not introduce perceptible latency into client-side rendering workflows. Its lightweight design and reliance on simple HTML encoding libraries also make it straightforward to integrate into existing client-side applications.

Figure 4: Script injection result in the protected system before and after processing

Existing approaches, such as server-side defences, remain effective for reflected and stored XSS but do not directly address DOM-based attacks that originate during client-side execution (Stock et al., 2014). Machine learning-based techniques have also been explored for XSS detection (Weamie, 2022); however, their computational requirements and deployment complexity may limit their suitability for lightweight client-side environments (Stock et al., 2014). In this context, the proposed mechanism provides a practical and low-cost client-side defence that complements existing server-side protections.

The study's findings offer several practical benefits for developers and organisations:

Improved Security

The mechanism effectively prevents DOM-based XSS attacks, reducing exposure to client-side threats such as session hijacking and unauthorised script execution.

Developer Accessibility

Its lightweight nature and reliance on standard HTML encoding functions allow straightforward adoption without requiring advanced expertise or infrastructure changes.

Cost Efficiency

The approach avoids the need for specialised security tools or hardware investments, making it suitable for small- and medium-scale applications.

Despite these advantages, the proposed mechanism has certain limitations. First, its focus on DOM-based XSS means that complementary server-side defences are still required to address reflected and stored XSS vulnerabilities. Second, effective protection depends on consistent application of the encoding mechanism across all relevant DOM update points; inconsistent usage could leave residual attack surfaces. Finally, compatibility with legacy browsers may vary, potentially limiting effectiveness in environments that lack support for modern client-side security practices.

CONCLUSION

This study addressed the challenge of mitigating DOM-based XSS vulnerabilities in web applications that render server-side outputs in the DOM by introducing a client-side defence mechanism based on HTML encoding. The mechanism dynamically sanitises user inputs before rendering them in the DOM, effectively neutralising attack vectors that exploit unsanitised client-side scripts. Evaluation in a controlled environment demonstrated the mechanism's ability to prevent DOM-based XSS attacks, including script injection, event-based exploitation, and attribute manipulation. These findings highlight its potential as a lightweight and practical solution for addressing client-side vulnerabilities in web applications. This research makes the following contributions:

Proposed Mechanism

Introduced a client-side defence mechanism that leverages HTML encoding to mitigate DOM-based XSS vulnerabilities effectively.

Targeted Mitigation

Demonstrated complete prevention of the evaluated DOM-based XSS attack scenarios, offering a specific solution for vulnerabilities arising during dynamic DOM manipulations.

Practical Application

Provided a developer-friendly, lightweight solution that requires minimal additional computational resources and is compatible with existing workflows.

Several avenues for future research could extend the applicability and effectiveness of the proposed mechanism:

i. Development of complementary mechanisms to address other XSS types, such as reflected and stored XSS, ensuring a comprehensive defence strategy.
ii. Exploration of browser-based enhancements, including built-in sanitisation libraries, to further automate and standardise DOM-based XSS prevention.
iii. Adaptation of the proposed mechanism for legacy browser environments, improving compatibility across diverse user systems.
iv. Large-scale performance evaluations in high-traffic applications to assess scalability and resilience under real-world conditions.

The findings of this study contribute to the growing body of research on web application security by offering a targeted, cost-effective solution to a critical vulnerability. The proposed mechanism provides a pathway for developers to enhance the security of dynamic web applications, particularly those that render server-side data, including outputs from stored procedures.

REFERENCES

Abikoye, O. C., Abubakar, A., Dokoro, A. H., Akande, O. N., & Kayode, A. A. (2020). A novel technique to prevent SQL injection and cross-site scripting attacks using Knuth-Morris-Pratt string match algorithm. EURASIP Journal on Information Security, 2020(1), 14. https://doi.org/10.1186/s13635-020-00113-y

Dalai, A. K., Ankush, S. D., & Jena, S. K. (2017). XSS attack prevention using DOM-based filter. In S. C. Satapathy, V. Bhateja, K. Raju, & B. Janakiramaiah (Eds.), Data engineering and intelligent computing: Proceedings of IC3T 2016 (pp. 227–234). Springer. https://doi.org/10.1007/978-981-10-3376-6_25

Galán, E., Alcaide, A., Orfila, A., & Blasco, J. (2010). A multi-agent scanner to detect stored-XSS vulnerabilities. In 2010 International Conference for Internet Technology and

Secured Transactions (pp. 1–6). IEEE. https://doi.org/10.1109/ICITST.2010.5608846

Gupta, S., & Gupta, B. B. (2017). Detection, avoidance, and attack pattern mechanisms in modern web application vulnerabilities: Present and future challenges. International Journal of Cloud Applications and Computing (IJCAC), 7(1), 1–43. https://doi.org/10.4018/IJCAC.2017010101

Hydara, I., Sultan, A. B. M., Zulzalil, H., & Admodisastro, N. (2015). Current state of research on cross-site scripting (XSS)—A systematic literature review. Information and Software Technology, 58, 170–186. https://doi.org/10.1016/j.infsof.2014.11.001

Maina, Y. B. (2024). A critical evaluation of security approaches for detection and prevention of SQL injection attacks in web-based applications. *FUDMA Journal of Sciences*, *8*(2), 241–246. https://doi.org/10.33003/fjs-2024-0802-2308

OWASP. (2025). OWASP Top 10 - 2025: The ten most critical web application security risks. Retrieved from https://owasp.org/Top10/

Shar, L. K., & Tan, H. B. K. (2012). Automated removal of cross site scripting vulnerabilities in web applications. Information and Software Technology, 54(5), 467–478. https://doi.org/10.1016/j.infsof.2011.11.006

Shar, L. K., Briand, L. C., & Tan, H. B. K. (2015). Web application vulnerability prediction using hybrid program analysis and machine learning. IEEE Transactions on Dependable and Secure Computing, 12(6), 688–707. https://doi.org/10.1109/TDSC.2014.2373377

Stock, B., Lekies, S., Mueller, T., Spiegel, P., & Johns, M. (2014). Precise client-side protection against DOM-based cross-site scripting. In 23rd USENIX Security Symposium (USENIX Security 14) (pp. 655–670). USENIX Association. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/stock

Weamie, S. J. Y. (2022). Cross-site scripting attacks and defensive techniques: A comprehensive survey. International Journal of Communications, Network and System Sciences, 15(5), 73–94. https://doi.org/10.4236/ijcns.2022.155006