REAL-TIME TRAFFIC-BASED DETECTION OF XSS VULNERABILITIES VIA BIDIRECTIONAL HTTP TRAFFIC ANALYSIS

Anas Roubi, Muhalim Mohamed Amin*

Universiti Teknologi Malaysia, Skudai, Johor
Malaysia

* Corresponding Author, e-mail: roubi@graduate.utm.my

Abstract: Cross-site Scripting (XSS) vulnerabilities continue to compromise web application security due to delayed detection by periodic scans. This paper proposes a novel real-time, traffic-based detection system that inspects HTTP request-response flows to verify exploitability dynamically. Unlike existing solutions that rely on static rules or post-analysis, the introduced proxy-based framework passively tracks and correlates incoming requests with their reflections in outgoing responses, specifically examining executable contexts. Evaluation using established testing suites demonstrates that the system accurately identifies 66% of exploitable XSS vulnerabilities confirmed by dynamic scanners, with no false positives. The results highlight that real-time traffic analysis effectively complements existing tools, providing immediate and actionable vulnerability insights, significantly narrowing the window for attackers and accelerating the defensive response.

Key words: xss, traffic analysis, web security, web application firewall.

1. INTRODUCTION

As web applications continue to serve as critical interfaces between users and services across industries, their security remains a pressing concern. Among the most persistent threats is Cross-site Scripting (XSS) [1], a vulnerability that enables attackers to inject malicious scripts into web pages viewed by other users. Despite decades of research and numerous defensive tools, XSS remains prevalent due to the complexity of modern web application logic and the limitations of detection mechanisms.

This persistence is not merely a function of developer error but is deeply rooted in the limitations of existing detection methodologies. The dominant tools in use today—manual penetration testing, static code analysis, and dynamic application scanners—are all fundamentally periodic in nature. That is, they operate on predefined cycles, whether as part of a DevSecOps pipeline or scheduled security audits. As a result, vulnerabilities that emerge between scans can remain undetected and exploitable for extended periods, leaving applications exposed in production environments.

What is critically missing in the current landscape is a detection strategy that operates in real time, continuously monitoring live application behaviour to identify exploitable vulnerabilities as they manifest. Not days or weeks later during the next security scan. Such a capability would fundamentally shift the paradigm from speculative identification to verified, actionable vulnerability intelligence.

This research introduces a novel real-time, traffic-based system for detecting XSS vulnerabilities. This approach monitors both incoming HTTP requests and outgoing responses, dynamically correlating user-supplied input with its presence and context in rendered pages. By focusing on reflection and execution paths, the system identifies genuine vulnerabilities, those where input is reflected back into the browser in an executable form at the moment they occur. This provides immediate, high-confidence feedback to developers and security teams, enabling rapid remediation and significantly reducing the window of exposure.

The contributions are:

i. A real-time XSS detection model that continuously monitors HTTP request-response flows to identify reflected inputs indicating potential injection points.
ii. A context-aware engine that analyses the syntactic location of input to detect exploitability within executable browser contexts.

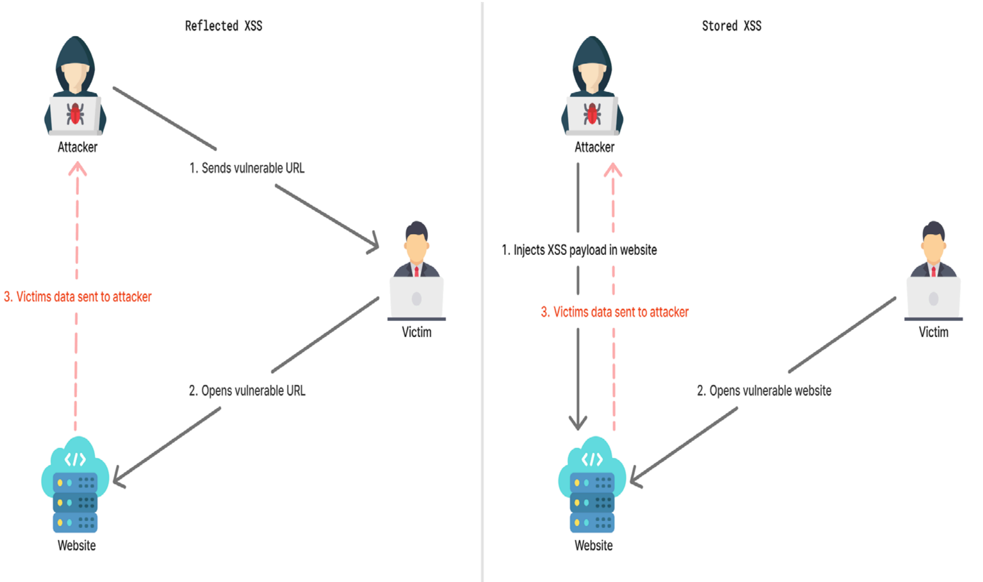
2. LITERATURE REVIEW

XSS is one of the most persistent and impactful vulnerabilities affecting web applications today. It enables attackers to inject and execute arbitrary scripts within a victim's browser, leading to data theft, session hijacking, defacement, and a wide range of client-side exploits [2]. Despite widespread awareness and ongoing research, XSS remains prevalent due to the dynamic and complex nature of modern web applications, were user input flows through multiple layers of client and server logic. A wide range of detection approaches have been developed to combat XSS, including manual penetration testing, static code analysis, dynamic application scanners, and more recently, machine learning-based classifiers. However, these methods are largely periodic and often focus on the presence of suspicious input patterns rather than confirming actual exploitability in the browser context. This results in delayed detection and remediation, leaving applications exposed between security scan cycles. To design effective detection strategies, it is essential to understand the different forms of XSS and how they manifest within web application behaviour.

2.1. Types of Cross-Site Scripting (XSS)

XSS attacks exploit unvalidated user input in browser contexts and occur in various forms—reflected, stored, and DOM-based—each presenting unique detection challenges. Many traditional tools only address subsets of these, often lacking immediate response to new threats [3]. Understanding how each type functions is critical for developing comprehensive detection systems.

As shown in Figure 1, reflected XSS involves injecting scripts via URLs that are echoed by the server and executed in the user’s browser. Stored XSS embeds malicious scripts in server-side data (e.g., comments), which are later delivered to users directly. Both forms enable attackers to steal sensitive information via script execution. [4]



(a) *(b)*

Figure 1. Attack Flow of Reflected and Stored XSS Vulnerabilities.

2.2. Detection Methods

The security community has developed a variety of strategies for identifying and mitigating XSS vulnerabilities. These approaches are typically grouped into static analysis, dynamic analysis, and traffic-based detection. Each category has unique benefits and inherent limitations, particularly with regard to detection timeliness and exploitability verification.

2.2.1. Static Analysis

Static analysis techniques inspect source code without executing it, tracing data flows from input sources to potential output sinks to identify unsanitized injection paths. These tools are especially effective during early development, enabling pre-emptive remediation. However, they often suffer from high false-positive rates due to conservative assumptions and incomplete visibility into dynamic execution flows. Recent tools have improved upon these issues through context-sensitive models. For example, SPLENDOR performs targeted detection of stored XSS vulnerabilities in PHP applications by modelling sanitization logic and analysing how input flows reach HTML output sinks [5]. Sanitizer-centric models further refine this process by focusing only on data flows that bypass known sanitization routines, substantially reducing noise [6]. Nonetheless, static approaches remain blind to runtime behaviour and can miss vulnerabilities introduced through dynamic content, third-party libraries, or client-side scripting.

2.2.2. Dynamic Analysis

Dynamic analysis aims to address the runtime visibility limitations of static analysis by observing application behaviour during execution. These methods involve automated input injection, browser instrumentation, and taint tracking to detect potentially exploitable flows. A notable example is TT-XSS, which dynamically monitors input propagation to identify data that reaches dangerous DOM APIs or script contexts [7]. Hybrid frameworks enhance this further by combining pre-runtime taint analysis with fuzzing or in-browser payload simulation, increasing the likelihood of exposing real vulnerabilities. However, dynamic analysis remains resource-intensive, often struggles with application state coverage, and lacks the immediacy required for continuous production monitoring [8].

2.2.3. Traffic-Based Detection and Limitations of WAFs

Traffic-based detection methods inspect actual HTTP request-response pairs during application operation, providing a real-time perspective on potentially malicious interactions. This strategy offers significant advantages over code-centric tools, especially in deployed environments. Systems like XSSDS perform passive correlation of inbound parameters with reflected output in responses to detect successful XSS attacks [9]. Building upon this, fusion-based frameworks such as that proposed by Lu et al. integrate traffic inspection with machine learning to achieve high accuracy across a range of obfuscated payloads [10].

Despite the advantages of traffic-aware detection, most real-world deployments continue to rely on Web Application Firewalls (WAFs) for application-layer protection. WAFs typically operate by matching HTTP traffic against a predefined set of static rules or regular expressions, such as those found in the OWASP Core Rule Set [11]. While effective at blocking known payloads, traditional WAFs are notoriously brittle: adversaries can evade them through minor encoding changes, grammar mutations, or payload segmentation [12]. Even with frequent rule updates, WAFs struggle to generalize to new attack variants.

Recent attempts to augment WAFs with machine learning have shown promise in detecting payload anomalies and behavioural deviations. Deep learning-enhanced WAFs have demonstrated increased detection accuracy and adaptability [18,19]. However, these systems still rely on periodic model retraining and curated datasets and often operate as black-box classifiers with limited explainability. More critically, WAFs—regardless of intelligence—do not confirm exploitability; they merely filter input patterns without verifying execution within a browser context.

A particularly relevant precedent in the traffic-based detection space is DIAVA, a real-time framework developed for SQL injection (SQLi) vulnerability detection and evaluation [13]. DIAVA not only detects malicious SQL queries by analysing bidirectional traffic but also verifies the success of an attack by inspecting both requests and corresponding database responses. It further provides severity assessment by analysing leaked data using GPU-accelerated dictionary attacks. This architecture presents a shift in paradigm—from detecting suspicious inputs to confirming real vulnerabilities based on runtime evidence.

3. PROPOSED SOLUTION

This section presents the design and operational logic of the real-time, traffic-based XSS vulnerability detection system. Unlike traditional approaches that rely on signature matching or post-hoc log analysis, this method introduces an inline, proxy-based architecture that performs live request-response correlation to identify exploitable reflection points. The system operates passively in production environments without altering traffic unless a confirmed vulnerability is detected, at which point it can optionally apply runtime mitigation. The core goal is to distinguish between mere injection attempts and actual, browser-executable XSS vulnerabilities—thereby enabling high-confidence, actionable reporting and proactive defence. In its current form, the system is designed specifically for reflected XSS detection and does not address stored or DOM-based XSS, which require extended session tracking or client-side instrumentation.

3.1. System Overview

The proposed architecture introduces a reverse proxy deployed between the client and the target web application server (client ⇄ proxy ⇄ server). This proxy intercepts all HTTP(S) traffic flowing in both directions, enabling full visibility into user input and its manifestation in server-generated responses. The system performs request-response correlation and inspects whether user-supplied data is reflected back in contexts that can lead to script execution.

The proxy operates inline, ensuring zero latency between observation and analysis. Unlike WAFs, which only checks the request and based on preconfigured rules, this system does not block requests unless a verified exploit is detected in the response. Instead, it evaluates each transaction in real time, flagging only those inputs that are both reflected and located in potentially executable JavaScript contexts.

The key architectural components include:

- Traffic Interceptor: Monitors inbound and outbound HTTP traffic.
- Parameter Tracker: Extracts and stores request parameters with metadata.
- Reflection Analyzer: Compares response content to prior input values.
- Contextual Verifier: Determines the execution context of reflected values.
- Static/Dynamic Endpoint Classifier: Determines whether a request targets a static or dynamic resource to optimize processing.
- Vulnerability Reporter: Logs detection events with contextual metadata for developer triage or automated ticketing.

3.2. Input and Output Handling

Effective detection of exploitable XSS vulnerabilities requires a nuanced understanding of how user inputs traverse the application and manifest in the rendered output. The system treats each HTTP transaction as a candidate for reflection analysis, but to avoid unnecessary computation, it first filters traffic to isolate dynamic endpoints likely to handle user-controllable data. Once identified, these requests undergo structured input extraction and response correlation to determine whether any submitted values are reflected unsanitized in the server's output. If so, the system further evaluates the syntactic placement of these reflections to distinguish innocuous echoes from dangerous script execution contexts. This section outlines the sequential processes of endpoint classification, input tracking, response normalization, and context analysis that form the core of the reflection detection pipeline.

3.2.1. Endpoint Classification

To avoid unnecessary analysis and conserve system resources, the system employs a classification module that filters requests into static and dynamic categories.

- **Static Endpoint Detection**: Uses URI patterns and file extensions (e.g., .css, .js, .png), and maintains history of classified static paths for caching
- **Dynamic Endpoint Identification**: Prioritizes requests with Content-Type: text/html, and flags variable-length or user-influenced responses

3.2.2. Parameter and Response Processing

For eligible dynamic endpoints, the Parameter Tracker extracts all user-provided inputs from the HTTP request, including query parameters, POST bodies, cookies, and custom headers. These are stored with metadata such as request path, method, and timestamp, indexed by transaction ID.

The corresponding HTTP response is intercepted and normalized to allow for reliable reflection matching. The normalization process includes:

- HTML entity decoding (e.g., < → <)
- URL decoding (e.g., %3Cscript%3E → <script>)

This normalized response is then scanned for the presence of any parameter values submitted in the request.

If a reflected value is identified, the Contextual Verifier examines its surrounding syntax in the response DOM. Reflections are only considered exploitable if they appear in execution-prone locations such as:

- <script> tag bodies
- Inline JavaScript event handlers (onclick=, onload=, etc.)

3.3. Architecture Diagram

As shown in Figure 2, the proposed detection system is deployed as a reverse proxy between client endpoints and the target web server, intercepting and analysing all HTTP request and response traffic in real time to illustrate the high-level system architecture.

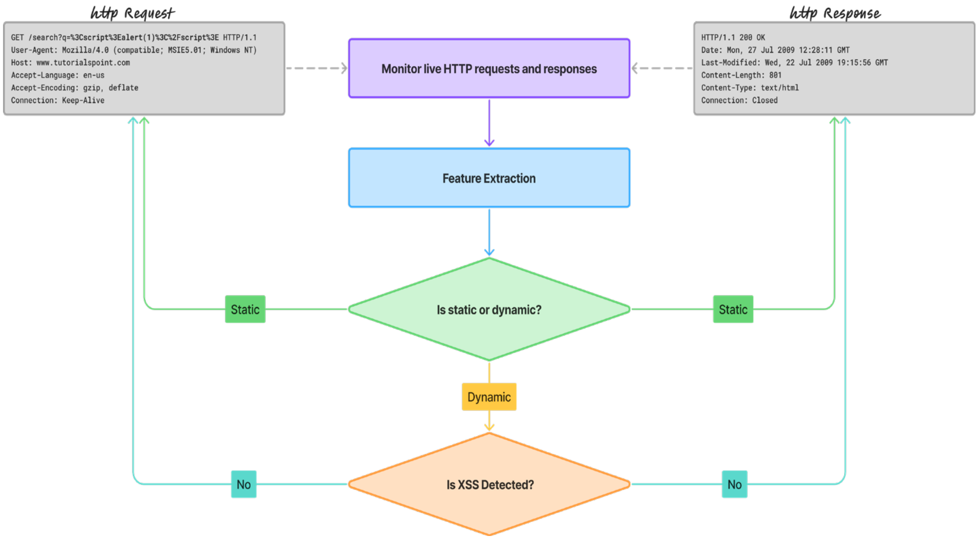


Figure 2. System Architecture – Reverse proxy intercepting bidirectional HTTP traffic.

4. RESULTS

This section presents the empirical evaluation of the proposed real-time, traffic-based XSS vulnerability detection system. Unlike previous research that focuses either on static code inspection or post-hoc traffic log analysis, the system operates live in the communication path, identifying exploitable XSS vulnerabilities as they occur in the traffic flow. To our knowledge, no prior academic work has implemented such a bidirectional HTTP traffic-based verifier capable of confirming XSS exploitability during live attack execution.

4.1. Evaluation Methodology

To validate the system's accuracy and responsiveness, we conducted an experimental study using the public-firing-range.appspot [15], a widely recognized and maintained suite of real-world XSS challenges. These labs span various XSS categories (reflected, stored, DOM-based) with different injection contexts, including script tags, attributes, and event handlers. Although our method is limited to detected reflected cross-site scripting only.

We used Burp Suite Professional [14] (v2025.5) a state-of-the-art dynamic scanner as a proxy attacker to simulate automated reconnaissance and exploitation. The detection proxy was deployed between Burp and the lab targets, intercepting and analysing all HTTP traffic generated by the scanner.

4.2. Comparative Evaluation Design

We evaluated the system against Burp Suite, assuming that any XSS vulnerability it reports is genuinely exploitable. The goal was to see if the same vulnerabilities could be detected solely by observing the HTTP traffic used to exploit them, without relying on scanner internals. Using public-firing-range.appspot.com [15], Burp Suite found 67

XSS issues, and our proxy analysed all traffic in real time. Table 1 shows the system's detection performance, with detailed results available on GitHub [16].

As shown in Figure 3, the real-time interaction between Burp Suite Scanner and the proposed traffic-based XSS detection proxy. The results confirm the analyser's effectiveness in identifying exploitable XSS vulnerabilities by passively inspecting live HTTP traffic. Operating independently from Burp Suite, it accurately verified 66% of detected vulnerabilities in real time with zero false positives. This shows that reflection-based verification can complement or even replace traditional Dynamic Application Security Testing tools. Although some stored XSS cases were missed due to multi-step flow limitations, the system was highly reliable for reflected cases, demonstrating the potential of real-time, exploitability-focused detection to streamline security response and minimize disruptions.

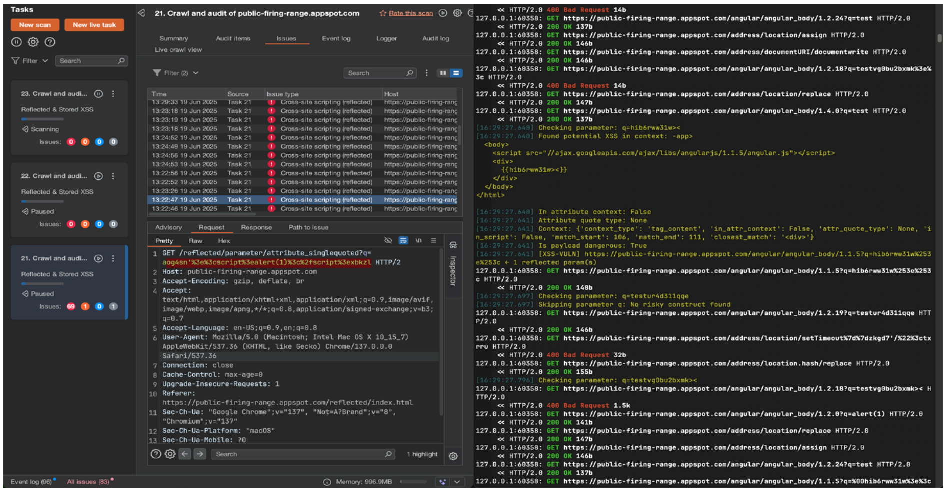


Figure 3. Traffic-based vulnerability detection with Burp Suite Scanner.

Table 1. Comparison of Burp Suite vs. The Traffic Analyser

Metric	***Traffic Analysis***	***Description***
Total XSS reported	*51*	*Raw count of detected XSS*
True Positives	*51*	*Detected XSS also found by Burp*
False Positives	*0*	*Reported by the analyser, not confirmed.*
False Negatives	*26*	*Found by Burp, missed by the analyser*
Accuracy	*66%*	*(True Positive + True Negative) / Total cases*

Latency testing with 40,000 requests with 200 threads running in parallel split evenly between proxy and direct connections showed that without XSS, the proxy's average latency was effectively the same as direct requests (±36.69 ms). With XSS, the proxy was about 364 ms slower on average. Parameters lowered latency for direct requests but increased it for proxy requests, indicating that XSS traffic introduces measurable delay when routed through the proxy. However, even in the worst case, the added latency remained well under half a second and is therefore acceptable for production use, maintaining the system's real-time detection capability.

Table 2. Latency Impact Summary for the Traffic Analyser

Condition	***Direct (Avg / Median / p95)***	***Proxy (Avg / Median / p95)***	***Proxy Delay (Avg)***
Clean	*443.68 / 289 / 1353 ms*	*406.99 / 343 / 875 ms*	*-36.69 ms*
XSS	*307.08 / 285 / 409 ms*	*671.12 / 363 / 2244 ms*	*+364.04 ms*

5. CONCLUSION

In this research, we introduced a novel approach for real-time XSS detection through bidirectional HTTP traffic analysis, shifting focus from speculative filtering to exploitability verification at runtime. Unlike WAFs, which often disrupt functionality through aggressive or misinformed blocking, this approach provides continuous, passive, context-aware detection with minimal risk of false positives or interference. It opens a new direction in web security by showing that live traffic can be used not just for attack detection, but for vulnerability confirmation—a key distinction for effective remediation. Future work can build on this to improve detection of stored XSS over longer sessions or asynchronous flows, integrate browser-side instrumentation for better DOM insight, and develop adaptive mitigation strategies that neutralize threats without affecting legitimate rendering.

REFERENCES

[1] HackerOne. *Top Ten Vulnerabilities.* URL, 2024. https://www.hackerone.com/top-ten-vulnerabilities (Visited on 06/06/2025).

[2] OWASP Foundation. *OWASP Top Ten Web Application Security Risks - A03:2021 – Injection.* URL: https://owasp.org/Top10/A03_2021-Injection/ (Visited on 06/06/2025).

[3] Rathod, J. A., Gowda, D. S., Kartik, M., Talekar, P., Daddi, N., Bhairanallikar, A. The cross-site scripting (XSS) attack: A comprehensive review. *International Journal of Advanced Research in Science, Communication and Technology*, vol.4, no.2, 2024, pp.199-205. DOI: 10.48175/ijarsct-19230.

[4] PortSwigger Ltd. PortSwigger *Web Security Academy: XSS labs. Online platform for learning and testing web vulnerabilities*. URL: https://portswigger.net/web-security/cross-site-scripting (Visited on 15.06.2025).

[5] Su, H., Li, F., Xu, L., Hu, W., Sun, Y., Sun, Q., Chao, H., Huo, W. SPLENDOR: Static Detection of Stored XSS in Modern Web Applications. *In Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '23)*, 2023, pp. 1043–1054. DOI: 10.1145/3597926.3598116

[6] Su, H., Xu, L., Chao, H., Li, F., Yuan, Z., Zhou, J., Huo, W. A sanitizer-centric analysis to detect cross-site scripting in PHP programs. *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*, vol. 33, 2022, pp. 355–365. DOI: 10.1109/ISSRE55969.2022.00042

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

[8] Gan, J.-M., Ling, H.-Y., Leau, Y.-B. A Review on Detection of Cross-Site Scripting Attacks (XSS) in Web Security. *Proceedings of the International Conference on Advances in Cyber Security*, 2021, pp. 685–709. DOI: 10.1007/978-981-33-6835-4_45.

[9] Lu, J., Wei, Z., Qin, Z., Chang, Y., Zhang, S. Resolving cross-site scripting attacks through fusion verification and machine learning. *Mathematics*, vol. 10, no. 20, 2022, pp. 3787. DOI: 10.3390/math10203787.

[10] Garn, B., Lang, D. S., Leithner, M., Kuhn, D. R., Kacker, R. N., Simos, D. E. Combinatorially XSSing web application firewalls. *2021 IEEE International Conference on Software Testing*, *Verification and Validation Workshops (ICSTW)*, 2021, pp. 85–94. DOI: 10.1109/ICSTW52544.2021.00026

[11] Annas, M., Adek, R. T., Afrillia, Y. Web Application Firewall (WAF) design to detect and anticipate hacking in web-based applications. *Journal of Advanced Computer Knowledge and Algorithms (JACKA)*, vol. 1, no. 3, 2024, pp. 52–58. DOI: 10.29103/jacka.v1i3.16315.

[12] Sa'adah, K., Hanaputra, R. A., Girinoto, Setiyo Cahyono. A deep learning approach to detect cross-site scripting attack as a web application firewall. *Proceedings of the International Conference on Information Technology and Computing (ICITCOM)*, August 2024, pp. 685–709. DOI: 10.1109/ICITCOM62788.2024.10762471.

[13] Gu, H., Zhang, J., Liu, T., Hu, M., Zhou, J., Wei, T., Chen, M. DIAVA: A traffic-based framework for detection of SQL injection attacks and vulnerability analysis of leaked data. *IEEE Transactions on Reliability*, vol.69, no.1, 2020, pp.188–203. DOI: 10.1109/TR.2019.2925415.

[14] PortSwigger Ltd. Burp Suite Professional v2024.x – Web vulnerability scanner and testing toolkit. URL: https://portswigger.net/burp (Visited on 15.06.2025).

[15] Google Security Team. Public Firing Range: A collection of web pages with common XSS vulnerabilities, *Google Open Source*, 2014, URL: https://public-firing-range.appspot.com/ (Visited on 15.06.2025).

[16] LiveRefXSS: Real-Time XSS Vulnerability Detection via Bidirectional HTTP Traffic Analysis, GitHub, 2025. URL: https://github.com/xhzeem/LiveRefXSS (Visited on 15.06.2025).

Information about the authors:

Anas Roubi – Security Researcher Working Since 2016. Top-100 Worldwide Bug Bounty Hunter for Multiple Years, Red Teamer, Penetration Tester, Researcher, Discovered and reported various security vulnerabilities to high-profile companies and vendors, and got acknowledged by companies such as Google, Meta, Microsoft, Twitter, Apple, Yahoo, and 100+

Muhalim Mohamed Amin – Dr. Muhalim Bin Mohamed Amin is a Senior Lecturer at the School of Computing, Universiti Teknologi Malaysia (UTM). He holds a Ph.D. in Computer Science from UTM, with earlier degrees in Computer Science and System Design from UTM and UMIST (UK), respectively. His research interests focus on computer networks and cyber security. He is an active member of the Information Assurance and Security Research Group (IASRG) at UTM and has published extensively in areas such as cryptography, fraud, and steganography. He also serves as a programme coordinator at Computer Science department and teaches various subjects related to network security and system software since 2000.

Manuscript received on 16 July 2025