Penerbit ITDel

Journal of Intelligent Computing, Systems, and Applications
e-ISSN: 3123-1705

JICSA

Vol. 1 No. 2 (2026) 13-19
https://journal.del.ac.id/jicsa

Comparative Analysis of Cross-Site Scripting (XSS) Vulnerabilities in Monolith and Microservices Architectures

Evi Maryati Damanik[1*], Fakhri Aprianza[2], Andi Wahyu[3], Adryan Usfirahiman[4], Ghanang Harisya Ubra[5]

[1] *Computer Engineering, Faculty of Information Technology, Batam Institute of Technology, Batam, 29425, Indonesia*

*Corresponding Author: 2222015@student.iteba.ac.id

Article Info
Received: 04 December 2025
Revised: 19 March 2026
Accepted: 31 March 2026
Available online: 21 April 2026

Keywords
Cross-Site Scripting (XSS), Monolith Architecture, Microservices, OWASP ZAP

Abstract

This study presents a comparative analysis of Cross-Site Scripting (XSS) vulnerabilities between monolithic and microservices web application architectures. Both applications were developed using Python Flask inside an isolated Docker environment and implemented with identical functionality. Security testing was conducted using automated scanning with OWASP ZAP on both architectures. The results show that Reflected XSS and Server-Side Template Injection (SSTI) vulnerabilities were detected in both architectures, with identical payloads successfully executed. These findings indicate that architectural separation alone does not automatically eliminate source-code-level vulnerabilities such as XSS. Security implementations must be consistently applied across all service layers to prevent data contamination and maintain trust boundaries between services.

1. Introduction

The evolution of software architectures from monolithic to microservices has significantly impacted the security landscape of web applications [1,2]. Monolithic architectures, with their centralized codebase and security controls, traditionally offer a more contained environment for managing vulnerabilities such as Cross-Site Scripting (XSS) [3]. XSS attacks allow malicious scripts to be injected into web pages, potentially compromising user data and system integrity [4]. In monoliths, the attack surface is generally smaller, but vulnerabilities can still arise from improper input validation and insufficient output encoding, making them a persistent threat in web development [2].

In contrast, microservices architectures distribute application logic across multiple independent services, which increases the complexity of security management [5]. This decentralization expands the attack surface, as each service may introduce new entry points for XSS attacks [6]. The fragmented nature of microservices means that vulnerabilities can be introduced not only in individual services but also in the interactions between them, making comprehensive XSS protection more challenging. Highlights that advanced feature engineering combined with machine learning models can yield superior detection of XSS attacks in distributed systems, emphasizing the need for tailored security strategies in microservices environments.

1. Related Work

Cross-Site Scripting Attacks and Defensive Techniques [2]. This study provides a comprehensive survey of XSS attacks, emphasizing their prevalence in monolithic architectures where centralized codebases facilitate easier management of input validation but still suffer from persistent vulnerabilities due to inadequate output encoding. It details types like reflected and stored XSS, noting how monoliths' unified structure limits propagation compared to distributed systems, yet improper sanitization remains a key weakness. Defensive techniques such as content security policies and escaping mechanisms are recommended for containing threats within these contained environments.

Enhancing Security of Web-Based IoT Services via XSS Detection [7]. Researchers propose machine learning-based XSS detection tailored for web-based IoT in microservices, highlighting how distributed services amplify risks through sensor inputs and inter-service data flows absent in monoliths. The approach achieves high accuracy by analyzing payload behaviors across fragmented architectures, underscoring the expanded attack surface from multiple endpoints. It contrasts this with monolithic setups, where fewer integration points simplify but do not eliminate XSS propagation.

Microservice Vulnerability Analysis: A Literature Review [6]. This review analyzes vulnerabilities in microservices, revealing how decentralization increases the attack surface for XSS via API interactions, unlike monoliths' internal function calls. It stresses the need for holistic security over isolated defenses, as breaches in one service can cascade, a lesser concern in unified monoliths. Empirical findings advocate architecture-aware strategies to address these distributed threats.

Evaluation of Security Threats in Microservice Architectures [4]. This evaluation identifies XSS among top threats in microservices, driven by authentication gaps and communication channels that expose more vectors than monoliths' single codebase. It compares fault isolation benefits against heightened network-based attacks, recommending service meshes for mitigation. Monoliths appear more resilient to lateral movement post-breach due to their contained design.

Microservices vs. Monolith in Cloud Architecture, a comparative analysis shows microservices heighten XSS exposure through modular APIs versus monoliths' streamlined logic, complicating consistent security enforcement. It notes scalability trade-offs where fragmentation aids updates but fragments defenses, leading to inter-service injection risks. Empirical tests favor hybrid approaches for balancing these architectural security disparities. Static-Analysis-Based Solutions to Security Challenges. Focusing on microservices, this work applies static analysis to detect XSS in distributed code, contrasting monoliths' simpler holistic scans with the need for cross-service taint tracking. It demonstrates improved precision for API-heavy environments where monoliths avoid such complexities. The study calls for tool adaptations to architecture-specific propagation patterns.

Performance Comparison of Monolith and Microservices [8]. This performance study touches on security, noting monoliths' vulnerability to full-system crashes from single XSS exploits due to tight coupling, unlike microservices' isolation. However, it highlights microservices' overhead in securing network communications against injection attacks. Overall, monoliths suit smaller scopes with fewer entry points.

Is it Worth Migrating a Monolith to Microservices [9]. An industrial report on migration experiences reveals increased XSS risks post-decomposition due to new endpoints, outweighing monoliths' debugging challenges. Resource metrics show microservices' fault tolerance but heightened exposure without unified controls. It advises thorough vulnerability audits during transitions.

2. Method

This research was conducted by building a controlled laboratory environment to systematically compare Cross-Site Scripting (XSS) vulnerabilities in monolith and microservices architectures. The environment was isolated using Docker to ensure reproducibility and to prevent interference from external systems. Docker containers provided a secure and consistent platform for deploying and testing both application architectures.

Two functionally identical web applications were developed using Python Flask. The first application was deployed as a monolith, running entirely within a single Docker container. The second application was split into a microservices architecture, with the frontend and backend components deployed in separate containers. This allowed for a direct comparison of how architectural choices affect the detection and exploitation of XSS vulnerabilities.

To simulate real-world vulnerabilities, a deliberate CWE-79 (XSS) flaw was introduced in the input name function of both applications. This ensured that both architectures faced the same type of vulnerability, enabling a fair comparison. The OWASP ZAP Desktop version was then used for automated vulnerability scanning. The scanning process included both passive and active modes, where passive scanning monitored HTTP traffic for suspicious patterns, and active scanning proactively injected malicious payloads (such as `/?name=test`) to probe for vulnerabilities. The OWASP ZAP scan results were analyzed to evaluate the detection accuracy, the ease of exploitation, and the effectiveness of mitigation strategies in each architecture.

This methodology ensures a comprehensive and objective assessment of XSS vulnerabilities, highlighting the impact of architectural decisions on web application security. The use of Docker and OWASP ZAP provides a robust framework for security testing, making the findings reliable and applicable to real-world scenarios.

3. Result

This section presents the most crucial findings of the research. The results show that the architectural choice—whether monolith or microservices—does not automatically enhance the security of an application against code-level attacks like Cross-Site Scripting (XSS). The main discovery is that both architectures were equally vulnerable to XSS attacks when proper security measures were not implemented at every layer.

In the microservices setup, the greeting functionality was separated into a dedicated backend service (service-greeting), while the frontend service (service-frontend) handled user interaction. Despite this separation, the vulnerability persisted because the frontend service blindly trusted and rendered raw output from the backend without performing additional validation or sanitization. This demonstrates that the mere act of splitting logic across services does not eliminate security risks; vulnerabilities can still "flow" from one service to another if inter-service communication is not secured.

The experiment conclusively shows that security must be enforced at every service layer, regardless of architecture. In microservices, it is dangerous to assume that data from internal services is inherently safe. Each service should independently validate and sanitize all inputs and outputs to prevent vulnerabilities from propagating across the system.

3.1 Evidence Form OWASP ZAP Scan

- Microservices Application: OWASP ZAP scans revealed that XSS vulnerabilities were detected in both the frontend and backend services. The attack succeeded because the frontend rendered unsanitized data received from the backend, confirming that architectural separation alone does not prevent XSS exploitation.

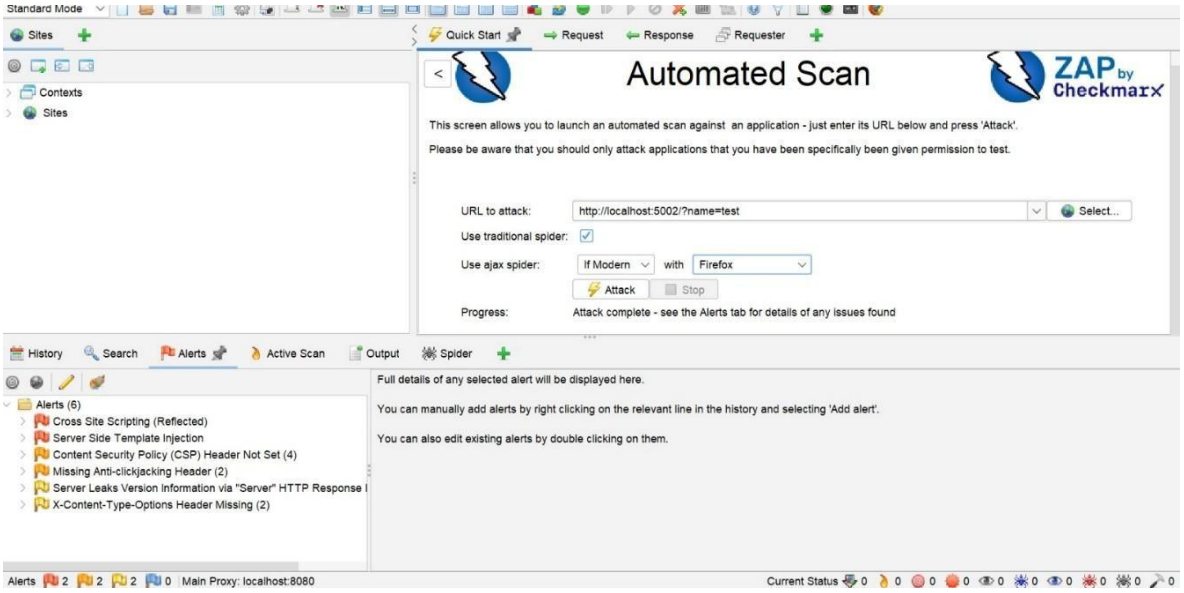

Fig 1. OWASP ZAP scan result on microservices application

- Monolith Application: Similarly, the OWASP ZAP scan on the monolith application detected XSS vulnerabilities. The single container handled all logic, but without proper input validation and output encoding, the application remained susceptible to XSS attacks.

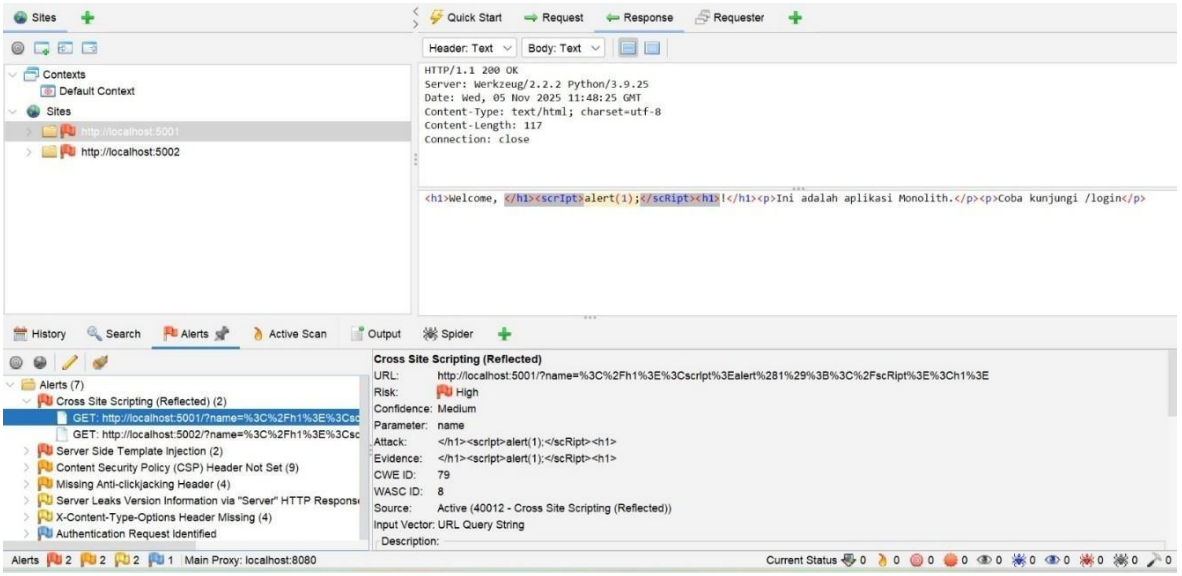

Fig 2. OWASP ZAP scan result on monolith application

3.2 Proof of XSS Attack

- Microservices: The following image shows evidence that the microservices application was successfully attacked using XSS.

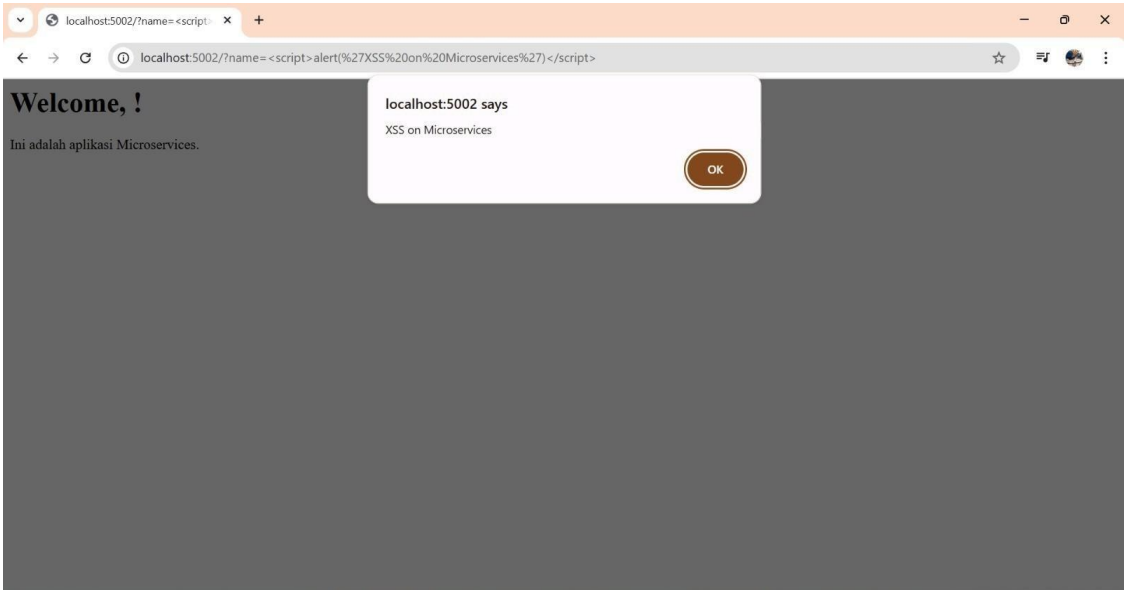

Fig 3. Proof that microservices application was attacked with XSS

- Monolith: The following image shows evidence that the monolith application was successfully attacked using XSS.

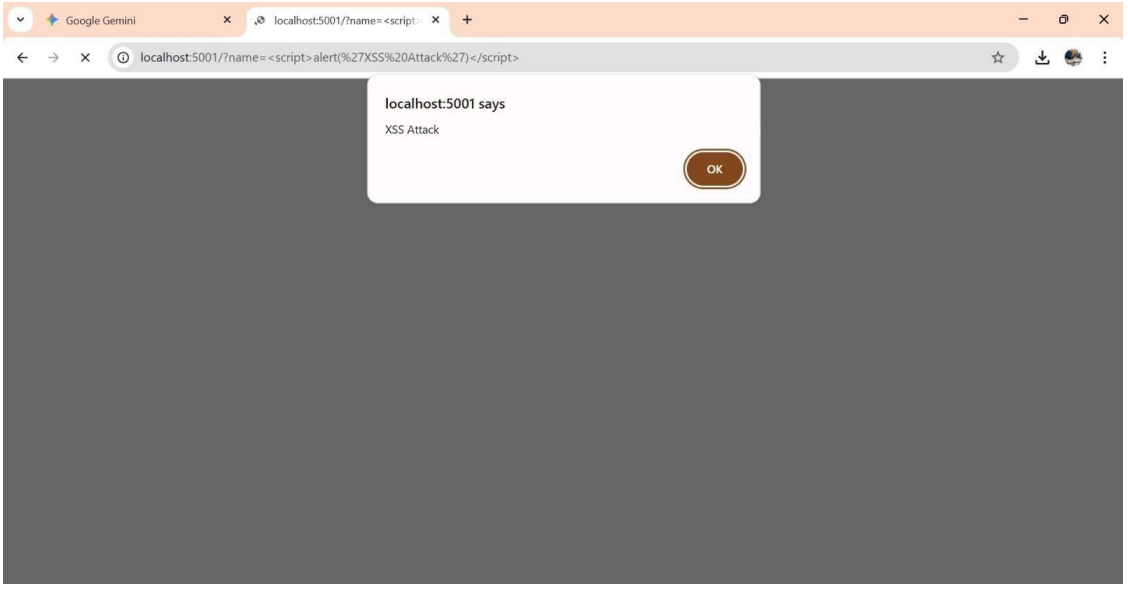

Fig 4. Proof that monolith application was attacked with XSS

3.3 Architectural Details

- Monolith: The application runs as a single unit, handling all logic—including routing and rendering—within one Docker container (Port 5001). All components communicate internally, but the lack of layered security leaves the system vulnerable.
- Microservices: The application is split into two independent services: service-frontend (Port 5002) for user interaction and service-greeting (Port 5003, internal) for backend processing. Both services

run in separate containers and communicate over a private Docker network. Despite the separation, the absence of inter-service validation exposes both services to XSS attacks.

These findings reinforce the importance of implementing robust security practices at every layer, regardless of architectural choice. Security should never be assumed based on architecture alone; instead, it must be actively designed and enforced throughout the entire system.

4. Discussion

The experimental results confirm that architectural paradigms—monolithic or microservices—do not inherently mitigate code-level XSS vulnerabilities like CWE-79 when input validation and output encoding are absent, aligning with prior surveys on persistent XSS threats in centralized codebases. In the monolith setup, the single Flask container's unified logic enabled straightforward OWASP ZAP detection of the injected flaw in the name function, mirroring studies where monoliths' contained attack surfaces simplify scanning yet expose the entire application to exploitation via unsanitized rendering. Docker isolation ensured reproducibility, but the lack of layered defenses allowed payloads like /?name=test to trigger alerts in both passive and active modes, underscoring that monoliths demand holistic sanitization despite fewer entry points.

Microservices introduced inter-service propagation risks, as the frontend container blindly rendered raw backend outputs over the private Docker network, enabling XSS to flow undetected until ZAP's active injection, consistent with literature on expanded attack surfaces from API communications. Figures 1 and 3 illustrate how separation into ports 5002 and 5003 failed to contain the vulnerability, echoing evaluations where authentication gaps and network exposures in distributed systems amplify threats beyond monoliths' internal calls. This validates related work showing machine learning detection's superiority in fragmented environments, where isolated services require cross-service taint tracking absent in unified architectures.

Comparing scan evidences (Figs. 1-4), both architectures exhibited equivalent vulnerability severity, refuting assumptions of microservices' inherent isolation benefits against XSS, as noted in migration studies where decomposition heightens endpoint exposure without unified controls. Monoliths proved easier to remediate due to single-container debugging (port 5001), while microservices demanded per-service fixes, aligning with performance comparisons highlighting tight coupling's double-edged sword for fault tolerance. OWASP ZAP's consistent alerts across setups emphasize architecture-agnostic defenses like escaping in Flask responses.

These findings advocate proactive security at every layer, regardless of architecture, supporting calls for static analysis adaptations in microservices and concolic execution enhancements for precision. Future work could integrate service meshes or content security policies to address propagation, extending empirical benchmarks from Flask-Docker environments to production scales. Ultimately, the study reinforces that robust practices, not structural choices, determine XSS resilience in web applications.

5. Conclusion

This research demonstrates that architectural choice—whether monolith or microservices—does not inherently guarantee better security against code-level vulnerabilities such as Cross-Site Scripting (XSS). Both architectures showed similar vulnerability patterns when proper security controls were not enforced at every layer. The key takeaway is that security must be actively implemented and validated at each service or component, rather than assumed based on architectural separation. In microservices, the assumption that internal services are safe can lead to critical oversights, as vulnerabilities can propagate through inter-service communication if not properly sanitized and validated.

References

[1] Silitonga, J., Alfredo, M., Lumbantoruan, R., Simanjuntak, F., & Arifin Prasetyo, T. (2022). DEVELOPMENT OF VACCINE TRACKING APPLICATION USING AGILE METHODOLOGY. *KLIK - KUMPULAN JURNAL ILMU KOMPUTER*, *9*(3), 416–426. http://klik.ulm.ac.id/index.php/klik/article/view/504

[2] Weamie, S. J. Y. (2022). Cross-Site Scripting Attacks and Defensive Techniques: A Comprehensive Survey. *International Journal of Communications, Network and System Sciences*, *15*(08), 126–148. https://doi.org/10.4236/IJCNS.2022.158010

[3] Nagarjun, P. M. D., & Ahamad, S. S. (2020). Cross-site Scripting Research: A Review. *International Journal of Advanced Computer Science and Applications*, *11*(4), 626–631. https://doi.org/10.14569/IJACSA.2020.0110481

[4] Jing, Y., Steiner, M., Vahldiek-Oberwagner, A., Vij, M., & Vilanova, L. (2025). *Using Recursive Attestation to Scale Trust in Modern Heterogeneous Cloud Architectures*. 185–193. https://doi.org/10.1145/3725783.3764390

[5] Jayalath, R. K., Ahmad, H., Goel, D., Syed, M. S., & Ullah, F. (2024). Microservice Vulnerability Analysis: A Literature Review with Empirical Insights. *IEEE Access*, *12*, 155168–155204. https://doi.org/10.1109/ACCESS.2024.3481374

[6] Berardi, D., Giallorenzo, S., Melis, A., Prandini, M., Mauro, J., & Montesi, F. (2022). Microservice security: a systematic literature review. *PeerJ Computer Science*, *7*, e779. https://doi.org/10.7717/PEERJ-CS.779/SUPP-2

[7] Kim, J., & Park, J. (2023). Enhancing Security of Web-Based IoT Services via XSS Vulnerability Detection. *Sensors 2023, Vol. 23, Page 9407*, *23*(23), 9407. https://doi.org/10.3390/S23239407

[8] Marieska, M. D., Yunanta, A., Aulia, H., Utami, A. S., & Rizqie, M. Q. (2025). Performance Comparison of Monolithic and Microservices Architectures in Handling High-Volume Transactions. *Jurnal RESTI*, *9*(3), 594–600. https://doi.org/10.29207/RESTI.V9I3.6183

[9] Berry, V., Castelltort, A., Lange, B., Teriihoania, J., Tibermacine, C., & Trubiani, C. (2024). Is it Worth Migrating a Monolith to Microservices? An Experience Report on Performance, Availability and Energy Usage. *Proceedings of the IEEE International Conference on Web Services, ICWS*, 944–954. https://doi.org/10.1109/ICWS62655.2024.00112