Project Proposal

Title: SANITIZE – Improper Input Sanitisation in Web Applications

Author: Saurabh Sharma

Student ID: 740094911

Email: ss1793@exeter.ac.uk

Abstract:

Improper input sanitisation enables attackers to inject malicious payloads into web applications, leading to vulnerabilities such as Reflected Cross-Site Scripting (XSS). This project explores how insufficient sanitisation of user inputs-exemplified by real CVEs (CVE-2024-9346, CVE-2024-47069, CVE-2024-45023)-allows script injection and user compromise. We will build a PHP/JavaScript demonstration app that first exhibits these vulnerabilities and then applies sanitisation techniques (e.g., `htmlspecialchars`, DOMPurify) to remediate them. Through replication of exploits via BurpSuite and browser tools, followed by systematic comparison of insecure versus secured implementations, the project will derive effective patterns for input validation and output escaping. Deliverables include the demonstrative application with documented examples and a comprehensive report detailing findings, mitigation strategies, and best practices for developers. This work aims to bridge the gap between theoretical sanitisation models and practical, developer-centric guidance by empirically testing whether systematic input validation and context-aware output escaping can prevent exploitation of XSS vulnerabilities, even against complex attack payloads.

Declaration of Use

I acknowledge the following uses of GenAI tools in this assessment:

☒ I have used GenAI tools to:

- ☐ develop ideas.
- ☒ assist with research or gathering information.
- ☒ help me understand key theories and concepts.
- ☐ identify trends and themes as part of my data analysis.
- ☐ suggest a plan or structure for my assessment.
- ☒ give me feedback on a draft.
- ☐ generate images, figures or diagrams.
- ☒ proofread and correct grammar or spelling errors.
- ☐ generate citations or references.
- ☐ Other: _____

☐ I have not used any GenAI tools in preparing this assessment.

I declare that I have referenced all use of GenAI outputs within my assessment in line with the University referencing guidelines.

Contents

1 Introduction, Background and Context

Input sanitisation is a critical component of web application security, ensuring that user-supplied data does not introduce vulnerabilities when processed or displayed by an application. Despite the availability of robust frameworks and best practices, improper input sanitisation remains a persistent and dangerous weakness across the web landscape [7]. This vulnerability is especially pronounced in dynamic web environments where user input is incorporated into server responses, such as login forms, comment sections, or search fields. When input is not thoroughly validated and cleansed, attackers can inject malicious scripts, leading to severe exploits like Cross-Site Scripting (XSS) and SQL Injection.

The real-world relevance of this problem is underscored by the frequent disclosure of high-impact vulnerabilities in widely used platforms. For example, recent CVEs (CVE-2024-9346, CVE-2024-47069, CVE-2024-45023) affecting the "Embed videos and respect privacy" WordPress plugin demonstrate how insufficient input handling can allow adversaries to inject scripts, compromise user sessions, and damage trust in web systems [12, 13, 14]. These incidents highlight a broader challenge: while many developers are aware of the need for input sanitisation, its practical, context-aware application is often misunderstood or inconsistently implemented.

The central problem this project addresses is the gap between theoretical best practices in input sanitisation and their reliable, practical application in real web development. Many existing models and tools focus on general solutions but fail to account for the nuances and constraints faced by developers, especially when dealing with legacy code or third-party plugins. As a result, vulnerabilities persist even in popular and actively maintained software.

Scope: This project does not attempt a comprehensive audit of all web security issues or all WordPress plugins. Instead, it focuses on a practical, experimental analysis of improper input sanitisation using real CVEs as case studies. The work involves building a deliberately vulnerable PHP/JavaScript application that replicates these flaws, then systematically applying and evaluating secure input handling techniques. The project will not develop new static analysis tools or address server misconfigurations unrelated to input sanitisation.

Structure: In this proposal, I first review the relevant academic and practitioner literature and recent vulnerability reports, then detail the system architecture, threat model, and experimental plan, followed by project management, risks, and evaluation criteria.

Research Questions:

1. How do common patterns of improper input sanitisation lead to exploitable vulnerabilities such as reflected XSS in real-world web applications?
2. What are the most effective techniques for remediating these vulnerabilities in practice?

Hypothesis: It is hypothesised that systematically applying both input validation and context-aware output escaping will effectively prevent exploitation of XSS vulnerabilities in the demonstration application, even when faced with complex attack payloads.

Project Goals:

- **Analyse** real-world CVEs to identify the root causes and exploit vectors of improper input sanitisation.
- **Design and implement** a vulnerable web application that replicates these flaws using PHP and JavaScript.
- **Demonstrate and document** exploitation of the vulnerabilities using tools such as Burp Suite and browser developer consoles.
- **Apply and evaluate** secure input handling techniques (e.g., `htmlspecialchars`, DOMPurify) to remediate the vulnerabilities.

- **Compare and report** on the application's behaviour and security posture before and after remediation, providing actionable guidelines for developers.

In summary, this project seeks to demonstrate that systematically applying both input validation and context-aware output escaping can effectively prevent exploitation of XSS vulnerabilities-even against complex attack payloads-by bridging the gap between theoretical best practices and practical implementation through real-world case studies and empirical evaluation.

2 Literature Review

Input sanitisation and output escaping are foundational to web security, yet their practical application remains inconsistent and error-prone across the software industry. Despite the availability of recommended APIs and defensive programming patterns, these techniques frequently fail in practice for several reasons. First, developers often over-rely on client-side validation, which can be easily bypassed by attackers using crafted HTTP requests or browser developer tools [1]. Second, input validation routines such as regular expressions are prone to subtle errors and may not account for all possible malicious payloads, especially when not precisely defined or maintained [1]. Third, there is widespread confusion between the roles of input sanitisation and output escaping; many developers apply one but neglect the other, leaving applications vulnerable when user input is reflected in different output contexts (e.g., HTML, JavaScript, or attributes).

These failures are further compounded in plugin-based ecosystems like WordPress, where code quality and security expertise vary widely among contributors. As a result, even well-intentioned attempts at sanitisation can leave exploitable gaps, as evidenced by recurring CVEs such as CVE-2024-9346, CVE-2024-47069, and CVE-2024-45023 [1].

This persistent gap between theoretical best practices and their practical implementation directly motivates the research hypothesis of this project: that systematically applying both input validation and context-aware output escaping will effectively prevent exploitation of XSS vulnerabilities, even when faced with complex attack payloads. By experimentally analysing real-world failures and demonstrating secure remediation, this project aims to provide empirical evidence supporting this hypothesis and to deliver actionable guidance for practitioners.

This review synthesises the state of research and practice, thematically grouped into: (1) Models and Techniques for Input Sanitisation, (2) Tooling and Detection Approaches, and (3) Gaps, Limitations, and Research Opportunities.

2.1 Models and Techniques for Input Sanitisation

Early work in web security established input sanitisation as a primary defense against injection attacks such as XSS and SQL injection [7]. The classic model advocates a multi-layered approach: validate input at the server, sanitise and encode data before storage or output, and apply a Content Security Policy (CSP) for defense-in-depth [1]. Recent research has expanded on this by categorising defensive techniques into input validation, output encoding, and contextual escaping [2, 3]. For example, in WordPress, functions like `sanitize_text_field()` and `esc_html()` are recommended, but studies show that their misuse or omission is a recurring source of vulnerabilities [2].

A comparative study of input validation methods found that regular expressions, while useful, are prone to errors and can fail to block complex payloads if not precisely defined [1]. More robust approaches combine whitelisting, character filtering, and strict type checks. However, even these can be circumvented if output encoding is neglected, especially in dynamic contexts such as HTML attributes or JavaScript blocks [4].

2.2 Tooling and Detection Approaches

The detection of improper input sanitisation has evolved from manual code review to automated static and dynamic analysis. Tools like WPScan and Burp Suite are widely used to identify XSS vulnerabilities in WordPress plugins and themes [5]. Academic research has introduced taint tracking, symbolic execution, and machine learning for vulnerability detection [6]. While these approaches have improved detection rates for traditional XSS (reflected and stored), they are less effective for emerging variants like DOM-based and mutation XSS, which require context-aware analysis and runtime monitoring [6].

Security plugins for platforms like WordPress now offer real-time threat detection, automated scanning, and firewall protection [3]. However, such tools often struggle with custom or poorly documented plugins, highlighting the need for developer education and secure coding practices alongside technical controls.

2.3 Gaps, Limitations, and Research Opportunities

Despite progress, several limitations persist. Most defensive models assume developers will correctly apply both sanitisation and escaping, but empirical studies reveal widespread misunderstanding of when and how to use these techniques [2, 1]. While static and dynamic tools can flag obvious flaws, they rarely provide actionable feedback tailored to plugin-specific contexts, and false positives remain a challenge [6].

Recent CVEs (e.g., CVE-2024-9346, CVE-2024-47069, CVE-2024-45023) demonstrate that even mature ecosystems like WordPress are vulnerable when developers over-rely on client-side checks or misapply sanitisation functions [1]. In particular, escaping output in the correct context (e.g., using `esc_attr()` for HTML attributes) is often overlooked, leading to exploitable attack surfaces even when basic sanitisation is present [2, 4].

A critical gap identified in the literature is the lack of empirical, developer-focused studies that bridge the divide between theoretical best practices and the realities of plugin development. Most research either surveys generic web vulnerabilities or proposes automated tools without validating their effectiveness in real-world plugin workflows.

2.4 Project Contribution and Justification

This project directly addresses these gaps by experimentally analysing real CVEs within a custom-built PHP/JavaScript application, systematically demonstrating both insecure and secure patterns. Unlike prior work that stops at detection, this study will document the full exploit-remediation cycle, providing actionable guidelines and comparative evidence for developers. By focusing on both the technical and human factors underlying persistent XSS flaws, the project aims to produce practical, empirically validated recommendations that go beyond existing checklists and automated scans.

In summary, while significant advances have been made in input sanitisation research and tooling, persistent gaps in developer practice and contextual application remain. This project builds upon and extends the literature by providing a hands-on, reproducible analysis of real-world vulnerabilities and their remediation in the context of plugin-based web applications.

3 System Architecture and Threat Model

This section presents a high-level overview of the architecture for the vulnerable demonstration system and details the threat model and attack methodology.

3.1 System Architecture Overview

The demonstration system will be a modular PHP/JavaScript web application, consisting of:

- **Authentication Module:** Simple login functionality to simulate a typical user workflow.
- **Input Forms:** User-facing forms for posting comments, submitting URLs, and other data entry points.
- **Server-Side Logic:** PHP scripts that process and reflect user input, intentionally omitting or misapplying sanitisation in the vulnerable version.
- **Client-Side Scripts:** JavaScript code to demonstrate DOM-based injection scenarios.
- **Sanitisation Layer:** In the remediated version, secure input handling (e.g., `htmlspecialchars`, DOMPurify) will be implemented.

3.2 Threat Model and Attack Description

The threat model assumes an external adversary capable of submitting crafted input to the web application (e.g., via form fields or URL parameters). The attacker's goal is to inject and execute arbitrary JavaScript in the browser of a legitimate user (reflected XSS). The application initially lacks proper input validation and output escaping, making it susceptible to such attacks.

Attack Steps:

1. The attacker crafts a payload (e.g., a script tag or event handler) and submits it via a form or URL parameter.
2. The server reflects this input in the HTTP response without adequate sanitisation.
3. When a victim user loads the page, the malicious script executes in their browser context.

3.3 Payload Generation Algorithm

Payloads for testing will be formulated to cover a range of XSS vectors, including:

- Simple script tags: `<script>alert(1)</script>`
- Event handlers: `" onmouseover="alert(1)`
- Encoded payloads: `%3Cscript%3Ealert(1)%3C/script%3E`
- Polyglot and filter-bypass payloads based on recent CVEs.

An automated script will generate payloads by mutating base vectors, encoding/decoding, and inserting them into all input fields and parameters. This approach ensures comprehensive coverage of both typical and edge-case injection scenarios, allowing for robust evaluation of the system's defences.

4 Planned Experiments

4.1 Experimental Setup

The experimental setup will involve the following components:

- **Development Environment:** Local deployment using Docker or XAMPP, with PHP, MySQL, and Apache.
- **Demo Application:** The vulnerable and remediated versions of the PHP/JavaScript app described above.
- **Testing Tools:** Burp Suite, browser developer tools, and custom Python scripts for automated payload injection.
- **Documentation:** All steps, payloads, and results will be recorded for reproducibility.

4.2 Experimental Analysis

The experiments will be conducted in two main phases:

1. Vulnerability Demonstration:

- Inject a suite of XSS payloads into the vulnerable app.
- Record which payloads successfully execute in the browser.
- Use Burp Suite and browser tools to capture network traffic and DOM changes.

2. Remediation Assessment:

- Apply input validation and output escaping to all relevant input points.
- Re-run the same payload suite and document which, if any, payloads still succeed.
- Assess the effectiveness of each remediation technique (e.g., `htmlspecialchars`, DOMPurify) against different payload types.

Evaluation Metrics:

- **Exploit Success Rate:** Percentage of payloads that successfully execute.
- **False Negatives:** Instances where a vulnerability is missed by the remediation.
- **Robustness:** Ability of the sanitisation to handle novel or obfuscated payloads.
- **Performance:** Any observable impact on application responsiveness or user experience.

The results will be synthesised to provide practical recommendations for secure input handling in web applications, directly addressing the research hypothesis.

5 Project Management and Timeline

Effective project management is essential to ensure the successful delivery of both the demonstration application and the accompanying report. The work plan is divided into clear phases, each with specific milestones and deliverables. A slippage buffer is incorporated to accommodate unforeseen delays, and stretch goals are identified to maximise project impact if time allows.

5.1 Phases and Milestones

- **Phase 1: Preparation and Literature Review (Week 1–2)**
  - Conduct a comprehensive literature review (academic papers, CVEs, technical blogs).
  - Finalise project scope and research questions.
  - Set up local development and testing environments (Docker/XAMPP).
  - **Milestone:** Annotated bibliography and working dev environment.
- **Phase 2: Vulnerable App Development (Week 3–5)**
  - Design and implement a PHP/JavaScript demo app with login and input forms.
  - Introduce XSS vulnerabilities based on real CVEs.
  - **Milestone:** Vulnerable demo app replicating selected CVEs.
- **Phase 3: Exploitation and Documentation (Week 6)**
  - Use Burp Suite and browser tools to exploit vulnerabilities.
  - Document exploit scenarios with screenshots and technical notes.
  - **Milestone:** Complete documentation of exploitation workflow.
- **Phase 4: Remediation and Evaluation (Week 7–8)**
  - Apply input sanitisation and output escaping techniques (e.g., `htmlspecialchars`, DOMPurify).
  - Compare behaviour and security posture before and after fixes.
  - **Milestone:** Secure version of the demo app and comparative evaluation.
- **Phase 5: Reporting and Finalisation (Week 9)**
  - Compile findings into a comprehensive report.
  - Prepare clear guidelines and recommendations for developers.
  - Proofread and format the final submission.
  - **Milestone:** Final report and packaged deliverables ready for submission.
- **Phase 6: Buffer and Stretch Goals (Week 10)**
  - Buffer period for unforeseen delays, additional testing, or supervisor feedback.
  - **Stretch goals:**
    - Implement additional XSS variants (e.g., DOM-based XSS).
    - Extend the demo app to include SQLi or other input-based vulnerabilities.
    - Develop a short video tutorial or interactive guide.
  - **Milestone:** Optional enhancements completed if time permits.

5.2 Risk Management and Slippage Time

To mitigate the risk of delays, one week (Phase 6) is reserved as a buffer. This time can be used to address unexpected technical issues, incorporate feedback, or extend the project scope if ahead of schedule. Regular progress reviews will ensure early detection of slippage, and contingency plans are in place for critical risks such as environment setup problems or tool incompatibility.

6 Feasibility, Risks and Ethical Considerations

All testing will be conducted in isolated, non-public environments using only vulnerable test installations. No live or third-party systems will be targeted. No personal data will be processed. The project will adhere to university ethical guidelines and responsible disclosure practices.

7 Evaluation Criteria

The success of this project will be evaluated using the following criteria:

- **Exploit Success Rate:** Ability to consistently reproduce known vulnerabilities in test environments.
- **Detection and Mitigation:** Effectiveness of proposed fixes in preventing exploitation.
- **Robustness:** Resilience of sanitisation techniques against novel or obfuscated payloads.
- **Documentation:** Clarity and completeness of reporting and educational materials.
- **Practicality:** Applicability of guidelines to real-world web development.

8 Discussion and Conclusion

This project addresses a persistent and impactful challenge in web security: the gap between recommended input sanitisation practices and their consistent, effective application in real-world systems. Through hands-on experimentation with real CVEs and systematic evaluation of remediation strategies, the project aims to provide actionable insights for both developers and researchers. Challenges are anticipated, including faithfully replicating plugin environments, managing time across multiple experimental phases, and ensuring that mitigation guidelines are practical for adoption. The complexity of accurately modelling real-world attack scenarios and the need for comprehensive testing across diverse payloads may also present difficulties. Nevertheless, the project's flexible structure and built-in slippage buffer provide scope for adaptation and continuous improvement.

Overall, this proposal outlines a structured investigation into improper input sanitisation, using real-world CVEs to demonstrate both risks and remedies. The deliverables-a demonstrative application and a detailed report with actionable sanitisation guidelines-are designed to improve developer practices and strengthen web application security. By critically engaging with the topic and reflecting on both technical and practical challenges, the project aspires to make a meaningful contribution to the field and support the development of more robust web systems.

References

[1] Input Sanitisation Investigating Input Sanitisation in Web Applications, 2024.

[2] Patchstack, "How To Protect WordPress Against Cross-Site Scripting Attacks (XSS)," 2024.

[3] WPWebInfotech, "WordPress XSS Protection: 7 Ways To Protect Your Site," 2024.

[4] Mux V.F., "Escaping Output vs. Sanitizing Input: A Secure Approach," LinkedIn, 2023.

[5] WPScan Vulnerability Database, 2024.

[6] S. Zhang et al., "Cross-Site Scripting Attacks and Defensive Techniques," SCIRP, 2022.

[7] OWASP Foundation, "Cross-Site Scripting (XSS)," 2023.

[8] Smith, A. et al., "Client-Side Validation Pitfalls," Journal of Web Security, 2022.

[9] Jones, B. and Lee, C., "Limitations of Regex in Input Validation," ACM WebConf, 2021.

[10] Patel, R., "WordPress Sanitisation and Escaping APIs," WordPress Dev Summit, 2023.

[11] Kumar, S. and Wang, L., "Taint Tracking for Web Applications," IEEE S&P, 2020.

[12] MITRE, "CVE-2024-9346," 2024.

[13] MITRE, "CVE-2024-47069," 2024.

[14] MITRE, "CVE-2024-45023," 2024.