Security Testing Report — Reflected Cross-Site Scripting (XSS)

Application: Damn Vulnerable Web Application (DVWA)
URL: http://localhost/dvwa/
Module Tested: Reflected XSS (xss_r)
Security Level: Low
Tester: *Risha Gupta*
Tools Used:

- OWASP ZAP 2.16 (HUD Enabled, Manual Explore, Active Scan)
- Chrome Browser via ZAP Proxy
- DVWA running on XAMPP

1. Introduction

This security assessment focuses on identifying Reflected Cross-Site Scripting (XSS) vulnerabilities in the DVWA application.
The primary objective was to:

- Test the input handling of DVWA
- Validate the presence of reflected XSS
- Capture technical evidence using OWASP ZAP
- Understand how the vulnerability behaves
- Provide remediation guidelines to fix the issue

Reflected XSS is a high-risk vulnerability where malicious JavaScript is executed immediately when a user interacts with a crafted URL or form.

2. Scope of Testing

Component	Details
Target Application	DVWA (Localhost)
Module	`/vulnerabilities/xss_r/`
Security Level	Low
Proxy	127.0.0.1:8080
Tools Used	OWASP ZAP (Manual + Active Scan), Browser HUD

Testing Techniques Used:

- Manual payload injection
- Analysis of reflected HTML response
- Observing browser execution behavior
- ZAP HUD interception
- Full-site crawling (Spider)
- Active automated vulnerability scan

3. Environment Setup

Component	Details
Operating System	Windows 10
Local Server	XAMPP (Apache + MySQL)
Application	DVWA
Browser	Chrome (proxy configured to ZAP)
ZAP Version	2.16
HUD	Enabled default

---

4. Manual XSS Testing

4.1 Target URL

```
http://localhost/dvwa/vulnerabilities/xss_r/
```

4.2 Payloads Tested

Primary payload:

```
"><script>alert(1)</script>
```

Additional payloads (for confirmation):

```
"><img src=x onerror=alert('XSS')>
"><svg onload=alert('XSS')>
```

4.3 Steps Performed

1. Launched DVWA through **OWASP ZAP – Manual Explore**
2. Opened the **Reflected XSS (xss_r)** module
3. Entered the malicious payload in the *Name* input field
4. Submitted the form
5. Observed:
    - Response in browser
    - HTML source captured in ZAP HUD
    - Active scan alerts triggered by ZAP
6. Verified if script execution occurred or was silently blocked by browser settings

4.4 Observations

- The input field **did not sanitize or encode user input**
- ZAP HUD showed the payload injected directly inside the HTML response
- Browser sometimes did **not show alert popup** (normal behavior in some browsers), but:
    - **HTML response contained the script**
    - ZAP triggered **High-Risk Reflected XSS alert**
    - Script reflected exactly as entered → vulnerability confirmed

✔ The application reflects user input without validation
✔ No filtering, encoding, or sanitization applied
✔ Page dynamically injected user input in HTML

4.5 Result of Manual Test

Vulnerability Status:
 CONFIRMED — Reflected XSS (High Severity)

Reason:
User-supplied input is returned in server response and executed in browser context.

---

5. Automated Testing (OWASP ZAP Active Scan)

ZAP Active Scan was executed after spidering the DVWA site.

Findings from Automated Scan

Alert Type	Risk	Plugin ID	Evidence
Reflected Cross-Site Scripting	High	40012	`<script>alert(1)</script>`
Suspicious Comments	Low	10027	"eval", "script" detected
Server Leaks Information	Medium	20017	Apache/2.4.54 in headers
Cross-Domain Script Include	Medium	10017	Observed external JS

ZAP Highlights for XSS:

- Attack vector detected in `name` parameter
- Script reflected in HTML body
- Unsanitized dynamic HTML content
- Missing Content Security Policy (CSP)

6. Root Cause Analysis

✔ The application does NOT:

- Validate input
- Encode output
- Filter special characters
- Implement CSP or input sanitization
- Use prepared/whitelisted input handling

✔ Application directly prints:

```
echo "Hello " . $_GET['name'];
```

This is inherently unsafe → allows JavaScript injection.

---

7. Impact Analysis

If exploited, an attacker can:

- Steal session cookies
- Hijack user accounts
- Modify page content
- Redirect users to malicious websites
- Execute unauthorized actions
- Steal sensitive data
- Perform phishing attacks

Severity: **HIGH**
Category (OWASP Top 10): **A03 – Injection**

8. Proof of Concept (PoC)

URL:

```
http://localhost/dvwa/vulnerabilities/xss_r/?name=%22%3E%3Cscript%3Ealert(1)%3C/script%3E
```

Observed HTML Response:

```
<p>Hello "><script>alert(1)</script></p>
```

✔ Confirms reflected XSS
✔ Injection successful
✔ Browser executed script or attempted execution

---

9. Mitigation Recommendations

To fix this vulnerability:

✔ 1. Output Encoding

Encode HTML special characters:

```
echo htmlspecialchars($_GET['name'], ENT_QUOTES, 'UTF-8');
```

✔ 2. Input Validation

Allow only safe characters (whitelisting)

✔ 3. Use Security Headers

Add:

```
Content-Security-Policy: default-src 'self'
X-XSS-Protection: 1; mode=block
```

✔ 4. Avoid Dynamic HTML Injection

Never print unvalidated user data

✔ 5. Use frameworks with built-in escaping

10. Conclusion

The DVWA Reflected XSS module was confirmed vulnerable using manual and automated testing techniques.
The application lacks output encoding and input validation, making it highly exploitable.

This report documents:

- Payloads used
- Manual + automated testing
- Evidence from browser + ZAP
- Root cause + impact
- Full mitigation steps

Overall Result: HIGH-SEVERITY Reflected XSS Vulnerability Identified.