TryHackMe Advent of Cyber 2025
Day 11 Challenge Report

Cross-Site Scripting (XSS) Vulnerabilities

1. Executive Summary

This report documents the completion of Day 11 of the TryHackMe Advent of Cyber 2025 event. The challenge focused on identifying and exploiting Cross-Site Scripting (XSS) vulnerabilities in a web application. Successfully demonstrated both Reflected XSS and Stored XSS attacks, revealing critical input validation weaknesses that allow malicious JavaScript execution in user browsers.

2. Challenge Overview

Objective: Identify and exploit XSS vulnerabilities in a gift portal web application, demonstrating both reflected and stored XSS attacks.

Target: Web application at http://{VM_IP}

3. Cross-Site Scripting Fundamentals

XSS (Cross-Site Scripting) is a web application vulnerability allowing attackers to inject malicious code, typically JavaScript, into input fields that reflect content viewed by other users. When applications fail to properly validate or escape user input, that input is interpreted as executable code rather than harmless text.

3.1 Attack Mechanism

1. Attacker injects malicious JavaScript into vulnerable input
2. Application stores or reflects input without sanitization
3. Victim's browser receives and executes the malicious code
4. Attacker gains ability to perform actions as the victim

3.2 Impact Categories

- **Credential Theft:** Steal session cookies and authentication tokens
- **Session Hijacking:** Impersonate legitimate users
- **Page Defacement:** Alter website appearance and content
- **Phishing:** Display fake login forms
- **Malware Distribution:** Redirect to malicious sites

4. XSS Attack Types

4.1 Reflected XSS

Reflected XSS occurs when malicious injection is immediately projected in a response without being stored server-side.

Example Attack Vector:

Legitimate Search URL:

```
https://trygiftme.thm/search?term=gift
```

Malicious Search URL:

```
https://trygiftme.thm/search?term=<script>alert(atob("VEhNe0V2aWxfQnVubl9"))</script>
```

Exploitation Method:

5. Attacker crafts malicious URL with JavaScript payload
6. Victim receives URL via phishing (email, social media, messaging)
7. Victim clicks link, loading page with injected code
8. Browser executes JavaScript in victim's session context

Characteristics:

- Requires social engineering to deliver malicious link
- Targets individual victims (not widespread)
- Payload not stored on server
- Temporary - affects only current request/response

4.2 Stored XSS

Stored XSS occurs when malicious script is permanently saved on the server and executed for every user viewing the affected page. Unlike Reflected XSS, this is a 'set-and-forget' attack affecting all users.

Example Attack Vector - Blog Comments:

Legitimate Comment Submission:

```
POST /post/comment HTTP/1.1 Host: tgm.review-your-gifts.thm postId=3 name=Tony Baritone
email=tony@normal-person-i-swear.net comment=This gift set my carpet on fire but my kid
loved it!
```

Malicious Comment Submission:

```
POST /post/comment HTTP/1.1 Host: tgm.review-your-gifts.thm postId=3 name=Tony Baritone
email=tony@normal-person-i-swear.net
comment=<script>alert(atob("VEhNe0V2aWxfU3RvcmVkX0VnZ30="))</script>
```

Exploitation Method:

9. Attacker submits malicious JavaScript through input field
10. Application stores payload in database without sanitization
11. Every user loading the page retrieves and executes the payload
12. Attack persists until payload is removed

Characteristics:

- No social engineering required after initial injection
- Affects all users viewing compromised content
- Payload persisted in database
- More dangerous than Reflected XSS due to widespread impact

Answer: Stored XSS requires payloads persisted on backend

5. Exploitation - Reflected XSS

5.1 Vulnerability Assessment

Identified search functionality as potential injection point. Search inputs reflected directly in results without proper encoding.

5.2 Test Payload

Utilized basic XSS payload to confirm vulnerability:

```
<script>alert('Reflected Meow Meow')</script>
```

5.3 Exploitation Process

13. Navigated to search functionality on target application
14. Injected test payload into search bar
15. Clicked 'Search Messages' button
16. Alert box displayed, confirming successful XSS execution

5.4 Attack Analysis

Vulnerability Root Cause:

- Search input reflected directly in results without encoding
- Browser interpreted HTML/JavaScript as executable code
- No input validation or output sanitization implemented

System Logs confirmed payload execution, tracking malicious behavior.

Reflected XSS Flag: THM{Evil_Bunny}

6. Exploitation - Stored XSS

6.1 Identifying Persistent Storage

Identified message submission functionality storing data server-side for later viewing by McSkidy. Unlike search (client-side temporary storage), messages persist in database.

6.2 Payload Injection

Utilized similar payload structure for stored XSS testing:

```
<script>alert('Stored Meow Meow')</script>
```

6.3 Exploitation Process

17. Navigated to message submission form
18. Entered malicious payload in message field
19. Clicked 'Send Message' button
20. Payload stored in database and executed on every page load

6.4 Persistence Verification

Confirmed persistent execution by refreshing page multiple times. Alert displayed consistently, demonstrating successful Stored XSS affecting all future visitors.

Stored XSS Flag: THM{Evil_Stored_Egg}

7. XSS Prevention & Mitigation

7.1 Input Validation & Output Encoding

Disable Dangerous Rendering:

- Avoid innerHTML property - allows direct HTML injection
- Use textContent property instead - treats input as text
- Parses HTML entities rather than executing them

7.2 Cookie Security

Cookie Attributes:

- **HttpOnly:** Prevents JavaScript access to cookies
- **Secure:** Transmits cookies only over HTTPS
- **SameSite:** Restricts cross-site cookie transmission

7.3 Sanitization & Encoding

When applications require limited HTML input (links, basic formatting), critical to sanitize and encode all user-supplied data:

- Remove or escape executable elements (scripts, event handlers, JavaScript URLs)
- Preserve safe formatting while preventing code execution
- Implement whitelist approach for allowed HTML tags
- Use established sanitization libraries

8. Key Skills Developed

- Understanding XSS attack mechanisms
- Reflected XSS identification and exploitation
- Stored XSS identification and exploitation
- Payload crafting for web vulnerability testing
- Input validation weakness identification
- Web application security assessment
- XSS prevention and mitigation strategies

9. Conclusion

Day 11 of the TryHackMe Advent of Cyber 2025 provided comprehensive training in Cross-Site Scripting vulnerabilities. Successfully demonstrated both Reflected and Stored XSS attacks, revealing critical input validation weaknesses in the target web application.

The challenge emphasized the significant security impact of improper input handling. Reflected XSS enables targeted attacks via social engineering, while Stored XSS creates persistent threats affecting all users. Proper input validation, output encoding, and cookie security configurations are essential for XSS prevention.

Challenge Status: COMPLETED ✓