Sri Lanka Institute of Information Technology

TryHackme Room Report

WS Assignment

IE2062 – Web Security

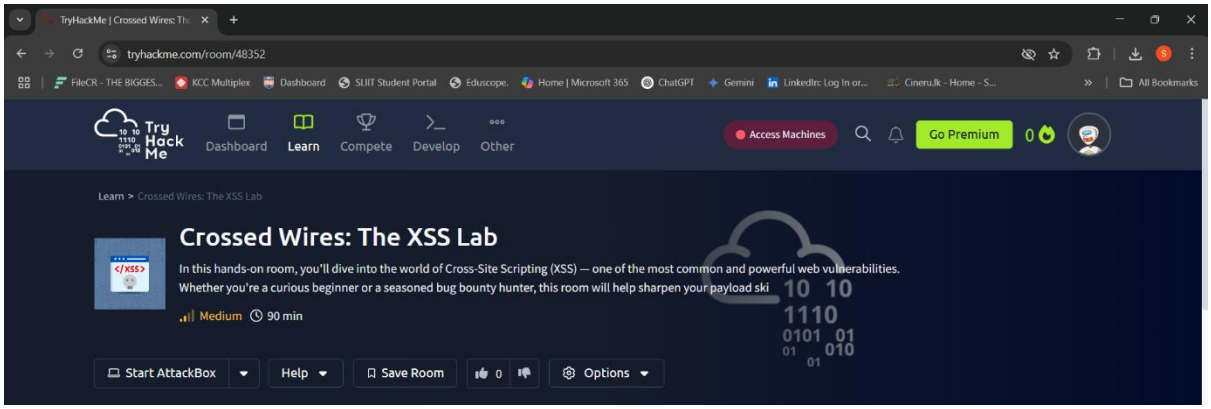
IT23159730

W.H.M.S.R.Bandara

Table of Contents

01 - Room Title:

Crossed Wires: The XSS Lab



Summary:

Crossed Wires is an entirely interactive TryHackMe room designed to guide students through the entire lifecycle of Cross-Site Scripting (XSS) from discovery and exploitation to mitigation. Blending theory, demonstration, and interactive attack simulation, it lets users get hands-on with XSS both as a threat and as a puzzle demanding precision, creativity, and context awareness.

Whether the player is a future ethical hacker or a developer wanting to secure their applications, this lab provides real-world usable knowledge through hands-on scenarios. The room finishes with a bespoke web challenge where the user must exploit a live vulnerability to disclose a concealed password using XSS.

Key Concepts Covered

- **Understanding XSS:** Definitions, types, and business risks.
- **Reflected XSS:** User input reflected immediately in responses.
- **Stored XSS:** Persistent payloads embedded in server-side storage.
- **DOM-Based XSS:** Client-side vulnerabilities in JavaScript logic.
- **Payload Crafting:** Event-based and tag less XSS vectors.
- **Bypass Techniques:** Encoding, obfuscation, nested payloads.
- **Framework Vulnerabilities:** Unsafe rendering in React, Angular, Vue.
- **Mitigation Strategies:** Secure coding, escaping, sanitization, CSP.
- **End-to-End Exploitation:** Find, craft, and execute working attacks.

02 - Learning Objectives

Participants will walk away from the room with.

Knowledge Outcomes:

- Deep knowledge of XSS varieties.
- Understanding of XSS attack cases in real life.
- Knowledge of how modern-day applications are still vulnerable.
- Knowledge of how attackers bypass security filters.

Practical Skills:

- Identify and take advantage of various XSS vulnerabilities.
- Apply developer tools and source inspection to analyze input handling.
- Craft complex payloads in JavaScript and HTML.
- Apply encoding and obfuscation to evade sanitizers.
- Inspect source code and identify insecure rendering in frameworks.
- Understand and implement XSS mitigation best practices.

03 - Room Structure

The room is divided into 8 progressive tasks, each one aimed at introducing a basic idea, followed by a hands-on component or scenario.

Task 1: What is Cross-Site Scripting (XSS)?

Cross-Site Scripting (XSS) is a web vulnerability that allows attackers to inject malicious scripts into web pages that are viewed by other users. This introduction lab acquaints students with what XSS is, why it is possible, and how it can be harmful. Students learn about the three main types of XSS: Reflected, Stored, and DOM-based, through real-world analogies and real-world attack examples. The task describes how XSS is used to steal cookies, hijack sessions, redirect victims to other malicious sites, and execute arbitrary actions on behalf of the victim. This foundation sets the stage for the rest of the hands-on exploitation exercises.

Questions & Answers

1. What does XSS stand for?

- **Answer:** Cross-Site Scripting

2. Which type of XSS occurs when malicious input is saved and shown to other users later?

- **Answer:** Stored XSS

3. In which XSS type is the attack reflected immediately in the response, without storing it?

- **Answer:** Reflected XSS

4. Which XSS type happens entirely on the client side using JavaScript in the browser?

- **Answer:** DOM-based XSS

5. Name one method attackers use XSS for.

- **Answer:** Session hijacking

6. What is a keyway to prevent XSS in web applications?

- **Answer:** Input sanitization

Task 2: Reflected XSS

Here, the attendees will learn about Reflected XSS, which is where the malicious input is directly reflected from the server in the response, typically through query parameters. The aim is to locate a vulnerable input field and develop a payload that causes the browser to execute JavaScript. Students will understand how reflected XSS most often happens in search results, error messages, and form feedback, especially when input is shown without proper output encoding. Through experimentation, they will learn context-dependent payloads, how browsers interpret HTML and JavaScript, and how user input can change a web page's behaviour unknowingly.

Questions & Answers

1. What type of XSS involves reflecting input back in the immediate HTTP response?

- **Answer:** Reflected XSS

2. What payload is commonly used to test for XSS?

- **Answer:** <script>alert("XSS")</script>

3. Where can you check if your input is being reflected?

- **Answer:** Page source

4. What part of the URL often holds reflected input?

- **Answer:** Query parameter

Task 3: Stored XSS

Stored XSS is an even more dangerous type of cross-site scripting, it stores a malicious input in a database or file served to other users. In this exercise you will simulate a blog or comment section where the application saves user input and displays it without cleaning. The injected JavaScript persists in the system and executes whenever anyone visits the compromised page. Such persistence enables attackers to focus on many victims without having to require their click on a malicious URL. The participants will also face filtering mechanisms and how with event-based payloads or with creative tag manipulation the filtering can be bypassed.

Questions & Answers

1. What is another name for Stored XSS?

- **Answer:** Reflected XSS

2. Where is the malicious input stored in Stored XSS?

- **Answer:** On the server

3. What type of form often leads to Stored XSS?

- **Answer:** Comment form

4. Why is Stored XSS more dangerous than Reflected XSS?

- **Answer:** It affects multiple users.

Task 4: DOM-Based XSS

This exercise covers DOM-Based XSS, which occurs solely on the client side when JavaScript alters the web page using untrusted user data. Instead of server-side reflection, the browser's DOM is altered using unsafe methods like innerHTML or document.write. Volunteers will review the JavaScript code run on the page to discover insecure code that reads from the URL and dynamically writes on the page. Students will manipulate the browser's address bar to inject a script that executes server-side with no server involvement, highlighting how client-side scripting errors can lead to full XSS exploitation.

Questions & Answers

1. Where does DOM XSS happen?

- **Answer:** In the browser

2. Which part of the URL is often used in DOM XSS?

- **Answer:** The hash (#)

3. Which JavaScript method is commonly exploited in DOM XSS?

- **Answer:** innerHTML

4. How can you prevent DOM XSS?

- **Answer:** Use textContent

5. Why is DOM XSS harder to detect?

- **Answer:** It’s client-side

Task 5: Payloads & Filter Bypasses

This is an advanced exercise in which the learners see how attackers circumvent input filters and web application firewalls. If tags are prohibited, it is still possible to do XSS by other means or even by encoding characters. You will explore permutations of payloads to bypass simple sanitizers and learn that blacklisting tags alone just never cuts it. It also adds more depth to understanding of XSS payloads and involves interesting topics such as attribute injection, malformed tags and context-aware escaping.

Questions & Answers

1. What encoding can you use to hide malicious characters in a URL?

- **Answer:** URL Encoding

2. What JavaScript event can you use to execute a script when an image fails to load?

- **Answer:** onerror

3. What protocol can you use to execute JavaScript in certain attributes like href?

- **Answer:** javascript

4. What is a common technique to bypass filters that only check for lowercase characters?

- **Answer:** Use case variations

Task 6: XSS in Modern Web Frameworks

New JavaScript frameworks like React, Angular, and Vue have XSS protection integrated into them, but they can still be misused. This activity has simple examples where developers use dangerous render functions like dangerouslySetInnerHTML (React) or v-html (Vue). Students will examine these examples, identify what was wrong, and offer safer alternatives. The task centers on the fact that security is not just about the technologies used, but rather how they are used, especially when programmers over-rely on user content.

Questions & Answers

1. What React feature can lead to XSS if used improperly?

- **Answer:** dangerouslySetInnerHTML

2. In Angular, how can you sanitize user input before rendering it?

- **Answer:** DomSanitizer

3. Which Vue directive can introduce XSS vulnerabilities if used with untrusted data?

- **Answer:** v-html

4. What is the best way to prevent XSS in modern frameworks?

- **Answer:** Sanitize user input

5. How can you improve security with modern frameworks to prevent XSS?

- **Answer:** Use Content Security Policy (CSP)

Task 7: XSS Mitigation Techniques

Preventing XSS is as worthwhile as exploiting it. This exercise introduces defense mechanisms that can be applied by developers in order to avoid cross-site scripting. Students learn about output encoding, safe DOM methods (textContent instead of innerHTML), and the utilization of sanitization libraries like DOMPurify. The exercise also mentions the necessity of setting HTTP security headers such as Content-Security-Policy, X-XSS-Protection, and cookie flags such as HttpOnly and Secure. Students will analyze several code examples to assess secure vs insecure practices, which will shift them into the mindset of a secure developer.

Questions & Answers

1. What is the best way to prevent dangerous input from users?

- **Answer:** Sanitize the input

2. Which JavaScript method is safer than innerHTML for inserting content?

- **Answer:** textContent

3. What header can you use to prevent unauthorized scripts from running in a browser?

- **Answer:** Content Security Policy (CSP)

4. How do you protect cookies from being accessed by JavaScript?

- **Answer:** Use HTTPOnly

Task 8: Prove Yourself – Final Practical Lab

The final challenge encompasses all the learned lessons from previous exercises. The participants are given a live, exploitable web application that features a hidden field with a secret password. They have to discover where input is reflected, construct an XSS payload capable of circumventing filters, and execute JavaScript to fetch and display the password. This exercise simulates an actual exploitation scenario, where the learner has to employ reflection analysis, JavaScript access to the DOM, filter evasion, and output manipulation. Successful completion of this lab shows a thorough grasp of XSS and readiness for real-world application.

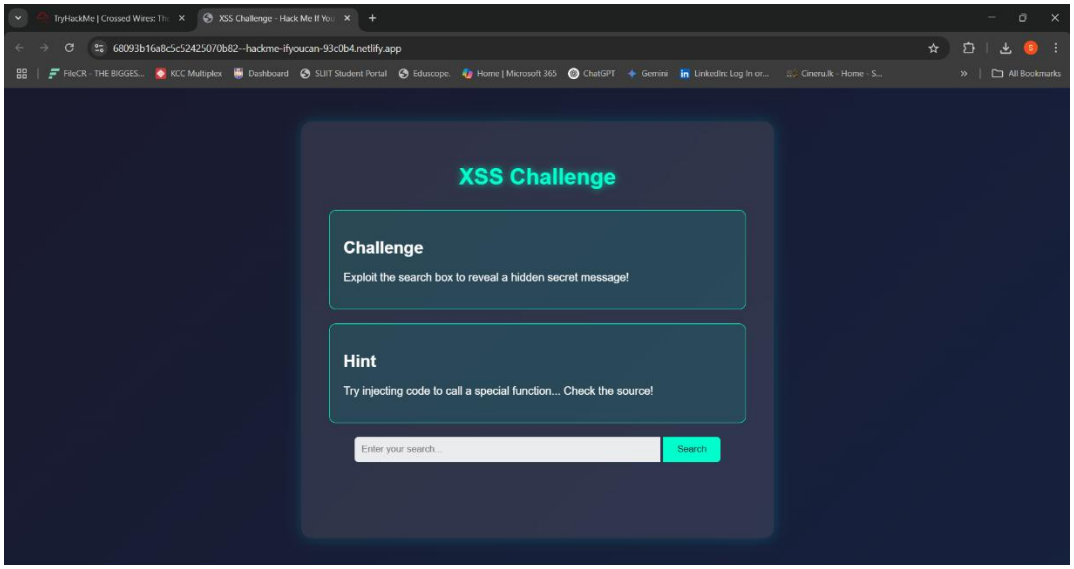
Questions & Answers

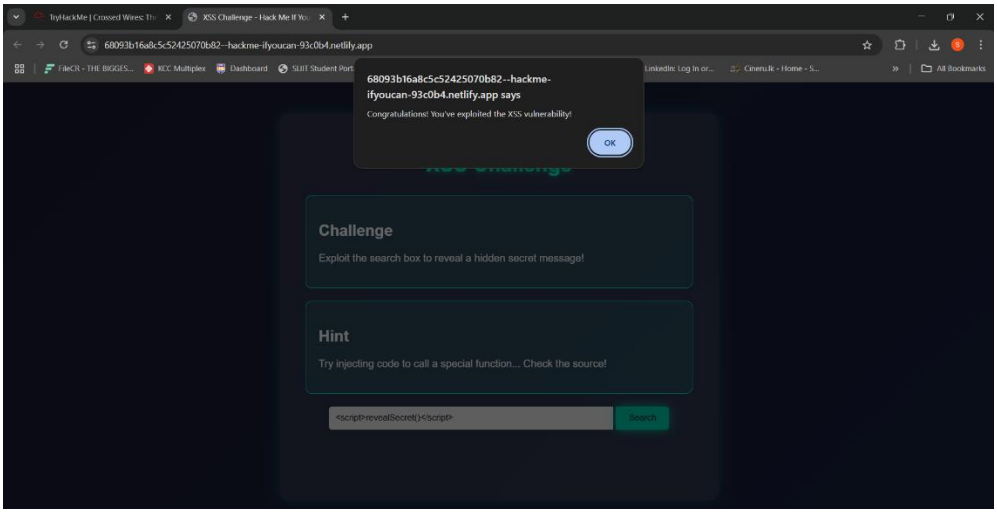
1. What is the Correct Exploit code?

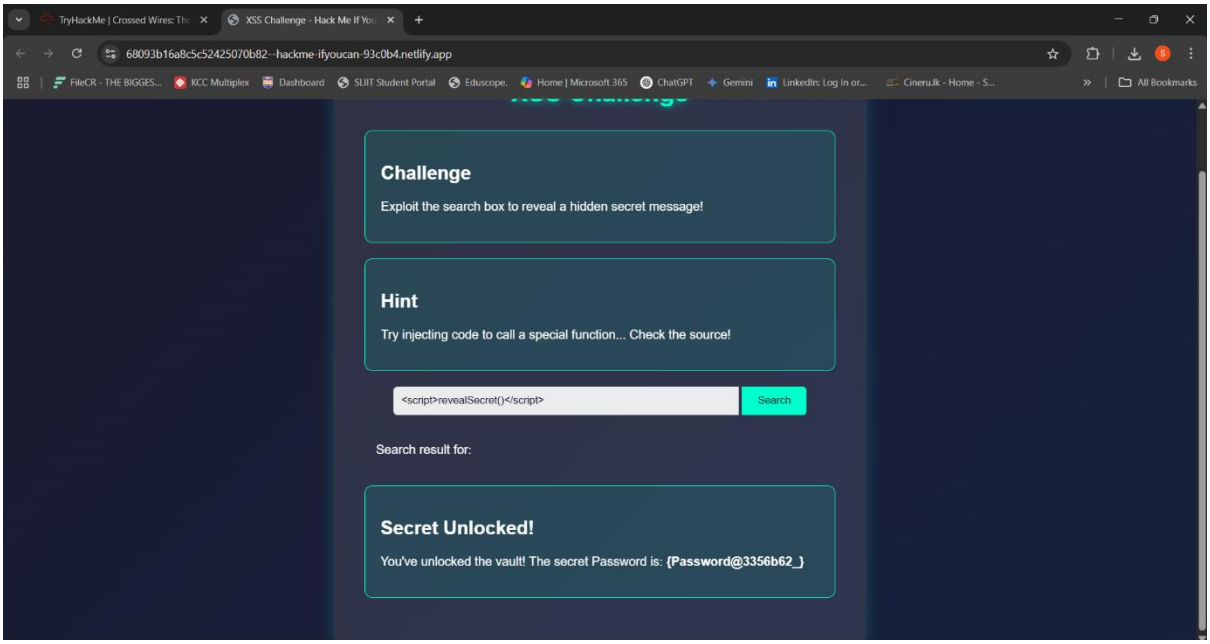
- **Answer:** <script>revealSecret()</script>

2. What is the secret password?

- **Answer:** {Password@3356b62_}







04 - Room Link

🔗 https://tryhackme.com/jr/48352

05 - Reflection

Personal Insights

Creating Crossed Wires: The XSS Lab allowed me to approach cybersecurity education from both the hacker's and developer's perspectives. Balancing readability with realism, I learned to structure progressively difficult tasks and encourage critical thinking instead of rote follow through. I also learned to create interactive material that bridges knowledge gaps, from introductory to intermediate levels.

Contribution to the Community

This room is not only a challenge, but also a learning experience. It's a safe place to learn about a dangerous attack vector and teaches exploitation and defence. The final lab encourages practical skill application, which is an excellent confidence builder for students.

By sharing this lab with the TryHackMe community, I can:

- Spread awareness of web security.
- Provide formal XSS training with real-world relevance.
- Promote secure coding practices by example.