Penetration Testing Report

Full Name: Rishabh Shetye
Program: HCPT
Date:16/02/2025

Introduction

This report document hereby describes the proceedings and results of a Black Box security assessment conducted against the **Week {1} Labs**. The report hereby lists the findings and corresponding best practice mitigation actions and recommendations.

1. Objective

The objective of the assessment was to uncover vulnerabilities in the **Week {1} Labs** and provide a final security assessment report comprising vulnerabilities, remediation strategy and recommendation guidelines to help mitigate the identified vulnerabilities and risks during the activity.

2. Scope

This section defines the scope and boundaries of the project.

Application Name	{Lab 1 - HTML Injection }, {Lab 2 - Cross Site Scripting }

3. Summary

Outlined is a Black Box Application Security assessment for the **Week {1} Labs**.

Total number of Sub-labs: {count} Sub-labs

High	Medium	Low
8	5	4

High - **4 Sub-labs with hard difficulty level**

Medium - **5 Sub-labs with Medium difficulty level**

Low - **8 Sub-labs with Easy difficulty level**

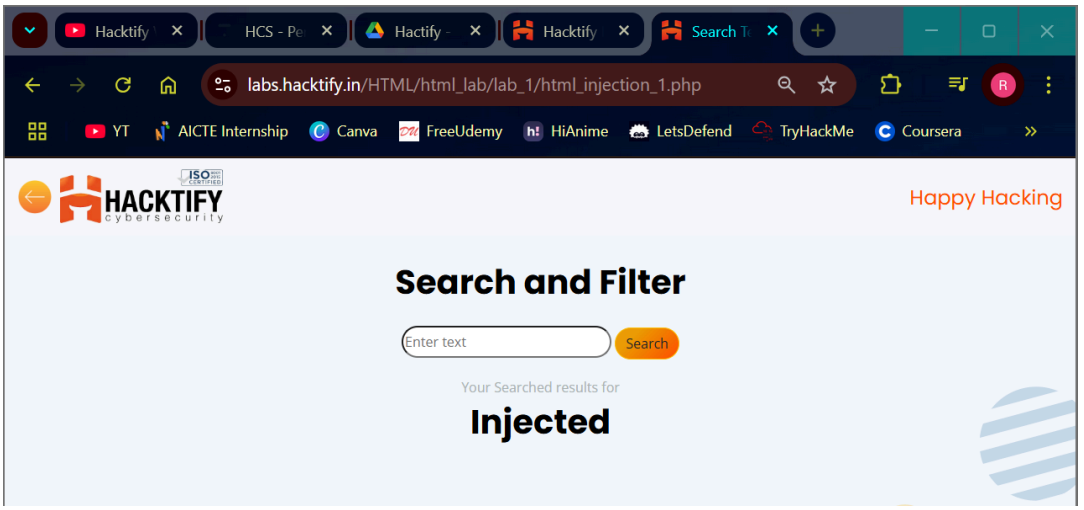
1. HTML Injection

1.1. HTML's are easy!

Reference	Risk Rating
HTML's are easy!	**Low**
Tools Used	
Web Browser	
Vulnerability Description	
HTML Injection occurs when user input is directly inserted into a webpage's HTML structure without proper sanitization. The payload Injected exploits this by injecting a malicious HTML tag that gets rendered by the browser, modifying the webpage's content.	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_1/html_injection_1.php	
Consequences of not Fixing the Issue	
If an attacker successfully injects Injected into a vulnerable website, they can manipulate the page's content, misleading users with false information, fake announcements, or phishing forms. This can damage the website's credibility, confuse visitors, and potentially trick users into performing unintended actions. In severe cases, attackers may use this technique to embed harmful links, redirect users to malicious websites, or conduct social engineering attacks.	
Suggested Countermeasures	
To prevent HTML injection, sanitize user input by encoding special characters like <, >, and ". Use secure output methods such as textContent instead of innerHTML to render user-generated content safely. Implement server-side validation to reject inputs containing unwanted HTML elements. Additionally, enabling Content Security Policy (CSP) can help block unauthorized content from being executed or embedded into the site.	
References	
https://www.imperva.com/learn/application-security/html-injection/	

Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab.

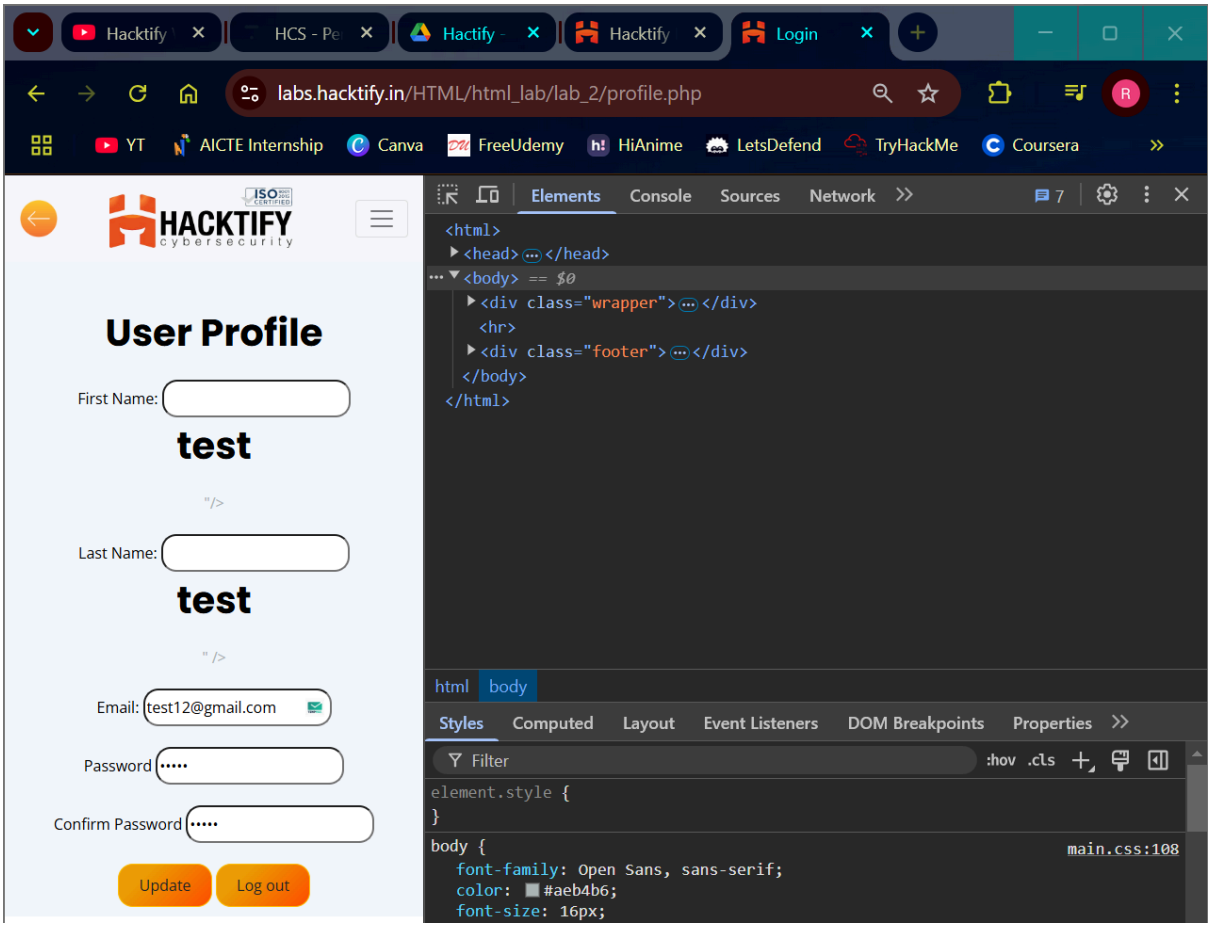


1.2. Let me Store them!

Reference	Risk Rating
Let me Store them	**Low**
Tools Used	
Web Browser	
Vulnerability Description	
This HTML Injection vulnerability occurs when user input is inserted directly into the webpage's HTML structure without proper sanitization. The payload "/><h1>test</h1> breaks an existing HTML attribute or tag, allowing the injected <h1> tag to be interpreted as part of the page content. This results in content manipulation, where an attacker can modify the page's appearance or insert misleading text. Payload: "/><h1>test</h1>	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_2/html_injection_2.php	
Consequences of not Fixing the Issue	
Content Alteration: Attackers can inject unauthorized headings, messages, or fake notifications. Phishing Attacks: Users may be tricked into interacting with deceptive content. Defacement: Attackers can alter how a website looks, affecting credibility. Further Exploits: If combined with JavaScript injection, it can lead to stored or reflected XSS attacks.	
Suggested Countermeasures	
Proper Input Validation: Reject inputs containing unwanted HTML tags. Escape Special Characters: Encode <, >, and " to prevent HTML parsing. Use Secure Rendering Methods: Prefer textContent over innerHTML. Apply a Content Security Policy (CSP): Restrict unauthorized content execution.	
References	
https://www.imperva.com/learn/application-security/html-injection/	

Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab.



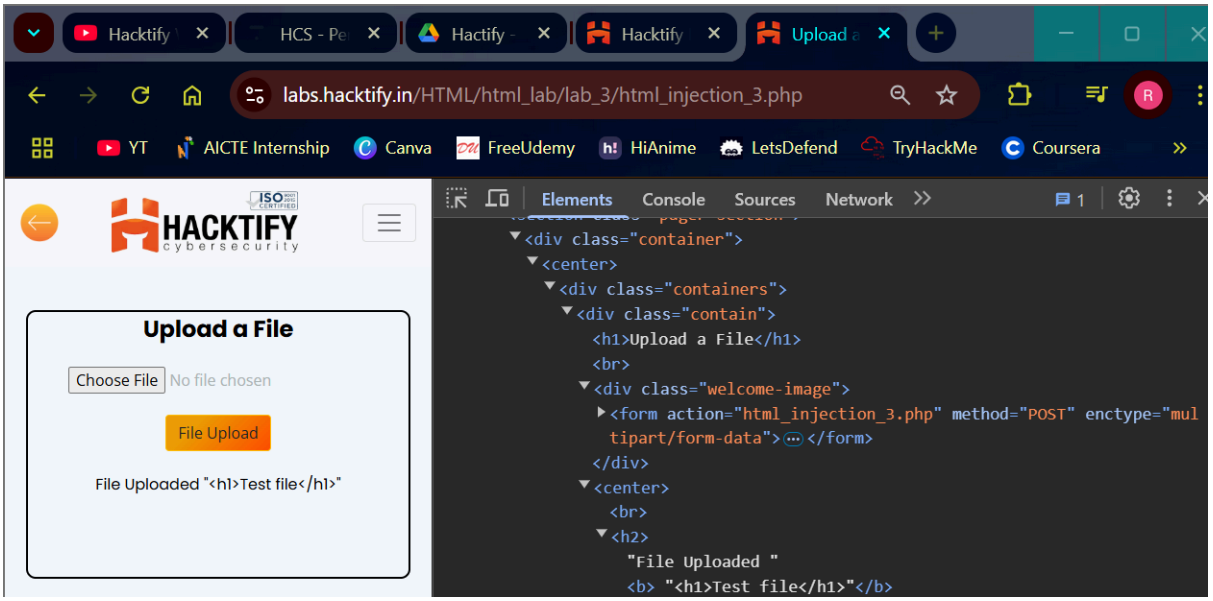
1.3. File names are also vulnerable!

Reference	Risk Rating
File names are also vulnerable	Low
Tools Used	
Web Browser	
Vulnerability Description	
HTML injection via file names occurs when an application does not properly sanitize user-supplied file names before embedding them into the page source. If the uploaded file name is directly inserted into the HTML structure without encoding, an attacker can craft a file name containing HTML or JavaScript code, leading to content manipulation or cross-site scripting (XSS) attacks. Payload: "/> Test file	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_3/html_injection_3.php	

Consequences of not Fixing the Issue
Fake Content Display: Injecting misleading messages into the page. XSS Attacks: If JavaScript execution is possible, attackers can steal cookies or session data. Defacement: Altering the webpage appearance by inserting custom elements. Phishing Attempts: Creating deceptive links or login prompts.
Suggested Countermeasures
Sanitize and Encode File Names: Strip or encode special characters like <, >, ", and '. Use Secure Output Methods: Avoid inserting raw user input into HTML. Restrict Allowed File Names and Extensions: Allow only alphanumeric characters. Validate Input on Both Client and Server Side: Prevent injection attempts before storing or displaying file names.
References
https://www.imperva.com/learn/application-security/html-injection/

Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab.



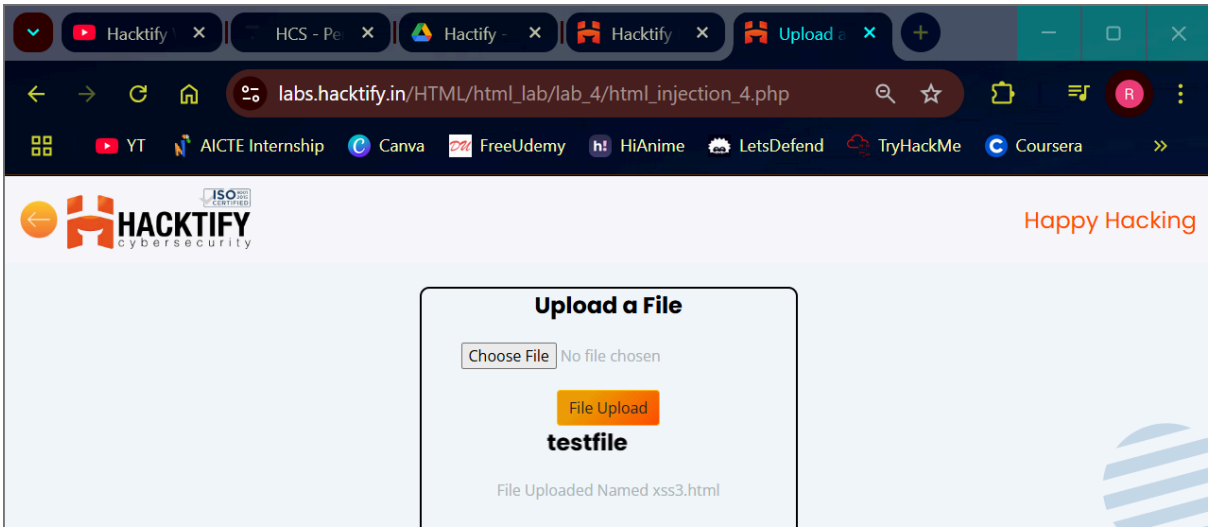
1.4. File content and HTML injection are a perfect pair!

Reference	Risk Rating
File content and HTML injection a perfect pair	**medium**
Tools Used	
Web Browser	
Vulnerability Description	
HTML injection via file content occurs when an application allows users to upload files that contain malicious HTML or JavaScript code, and then renders the file's content directly in the browser without sanitization. This can lead to content manipulation, phishing attacks, or even stored XSS, depending on	

how the file is processed and displayed. Payload: <marquee> testfile </marquee>
How It Was Discovered
Manual Analysis
Vulnerable URLs
https://labs.hacktify.in/HTML/html_lab/lab_4/html_injection_4.php
Consequences of not Fixing the Issue
Stored XSS Attacks: Malicious scripts persist on the site and execute when viewed. Phishing Attacks: Attackers can display fake login pages to steal credentials. Defacement: The attacker can modify how the page appears to users. Session Hijacking: If cookies or tokens are accessible, they can be stolen.
Suggested Countermeasures
Restrict Uploadable File Types: Block .html, .js, and other executable files. Serve Files as Downloads Instead of Rendering: Use Content-Disposition: attachment to prevent execution. Sanitize and Encode File Content: Strip or escape harmful HTML tags before rendering. Use a Content Security Policy (CSP): Prevent the execution of injected scripts.
References
https://www.imperva.com/learn/application-security/html-injection/

Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab.



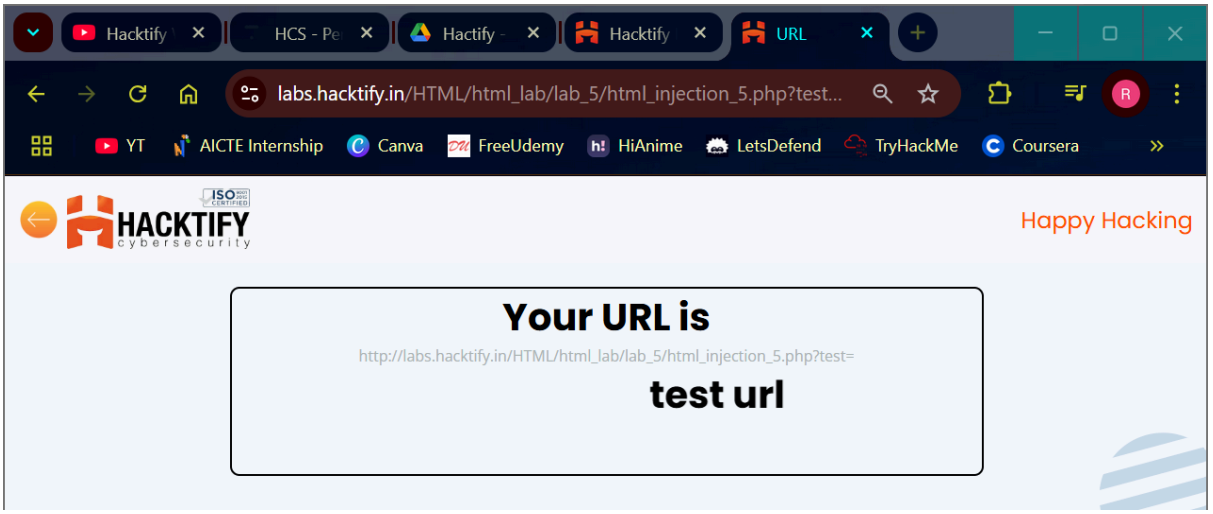
1.5. Injecting HTML using URL

Reference	**Risk Rating**
Injecting HTML using URL	**Medium**
Tools Used	
Web Browser	
Vulnerability Description	

HTML injection via **URL parameters** occurs when a web application **reflects user-supplied input from the URL into the page's HTML source** without proper sanitization or encoding. If the application dynamically inserts the URL parameter's value into the webpage, an attacker can manipulate the structure or inject malicious content. Payload: <marquee> test%20url </marquee>
How It Was Discovered
Manual Analysis
Vulnerable URLs
https://labs.hacktify.in/HTML/html_lab/lab_5/html_injection_5.php
Consequences of not Fixing the Issue
Content Manipulation: Attackers can modify page appearance or insert misleading content. Defacement Attacks: Injecting large headings, images, or fake warnings. Phishing Attempts: Creating deceptive login forms or messages. Stored XSS (if saved in the database): Persistent attacks affecting multiple users.
Suggested Countermeasures
Properly Encode User Input: Use htmlspecialchars() in PHP or equivalent functions in other languages. Validate and Sanitize URL Parameters: Remove <, >, and other special characters before rendering. Use a Whitelist Approach: Only allow expected alphanumeric input for URL parameters. Implement a Content Security Policy (CSP): Restrict inline scripts to prevent script execution.
References
https://www.imperva.com/learn/application-security/html-injection/

Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab.



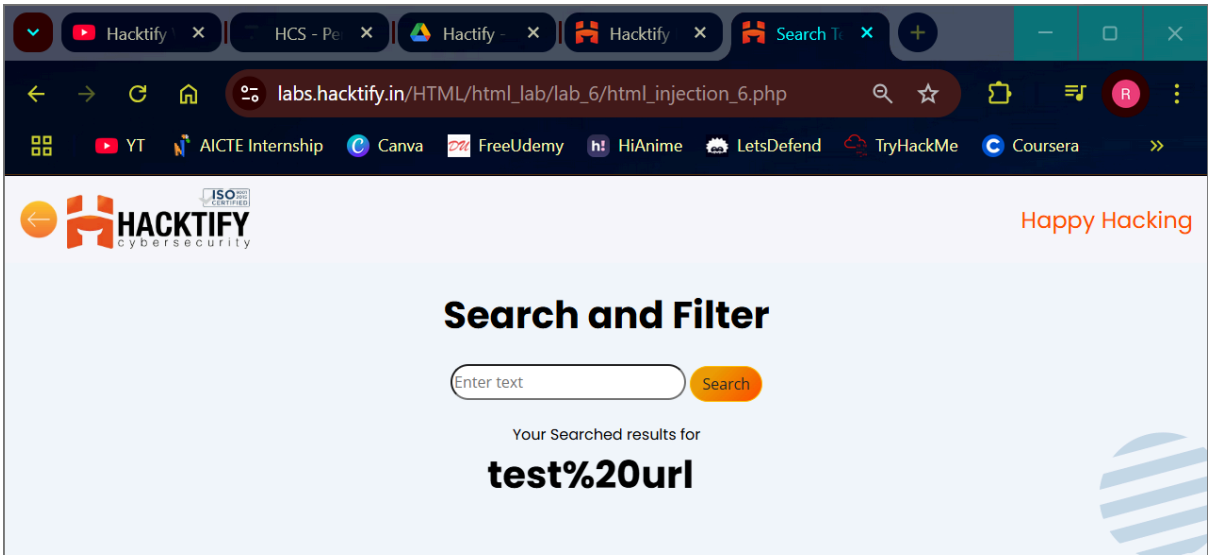
1.6. Encode it

Reference	Risk Rating
Let me Store them	**High**
Tools Used	

Web Browser
Vulnerability Description
HTML injection through **encoded payloads** occurs when an application **fails to properly decode and sanitize user input**, allowing attackers to inject encoded HTML elements that get executed once decoded by the browser. Encoding techniques such as **URL encoding** (%3C for <, %3E for >), **Hex encoding**, and **Base64 encoding** can bypass simple filtering mechanisms that block direct HTML tags. Payload: %3Cmarquee%3E%3Ch1%3Etest%3C%2Fh1%3E%3C%2Fmarquee%3E
How It Was Discovered
Manual Analysis
Vulnerable URLs
https://labs.hacktify.in/HTML/html_lab/lab_6/html_injection_6.php
Consequences of not Fixing the Issue
Defacement Attacks: Attackers modify page layout and content. **Phishing Attempts:** Fake messages or pop-ups can be inserted. **Stored XSS (if saved in the database):** Persistent attack affecting multiple users. **Bypassing Basic Filters:** Encoding can evade weak security measures.
Suggested Countermeasures
Decode and Sanitize Input Before Rendering: Convert encoded characters and filter dangerous tags. Use htmlspecialchars() or htmlentities(): Properly escape special characters before outputting. Validate Input Against a Strict Whitelist: Allow only expected characters and formats. Implement a Content Security Policy (CSP): Restrict inline script execution to mitigate risks.
References
https://www.imperva.com/learn/application-security/html-injection/

Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab.



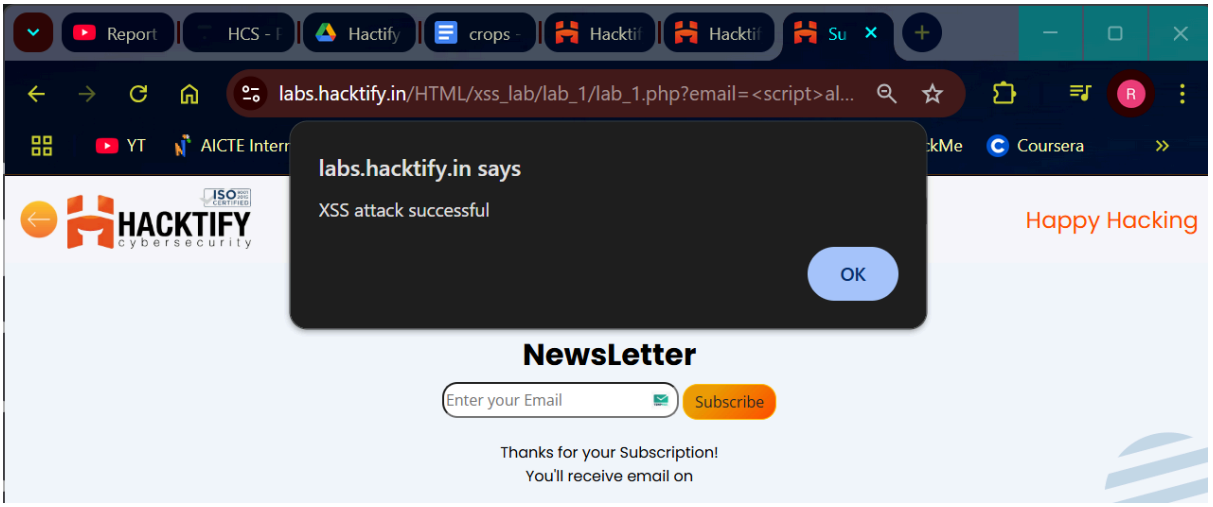
2. Cross Site Scripting

2.1. Let's Do IT!

Reference	Risk Rating
Let's Do IT!	**Low**
Tools Used	
Web Browser	
Vulnerability Description	
The first lab is vulnerable to Reflected Cross-Site Scripting (XSS) within the search parameter functionality. User supplied input is reflected in the URL after pressing the subscribe button. The URL parameter lacks proper sanitation or encoding to prevent such vulnerabilities. By modifying such parameters to include a malicious JavaScript payload, the web-application executes the injected script in the victim's browser. Such vulnerability can lead to Phishing Attacks, defacement of the web application or hijacking of the session. Payloads: `<script>alert('XSS')</script>`	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_1/lab_1.php	
Consequences of not Fixing the Issue	
Failure to remediate this vulnerability could lead to: **Account takeover:** Attackers can steal login credentials. **Session hijacking:** Attackers can take over a user's current session. **Malware distribution:** Attackers can inject malicious scripts. **Website defacement:** Attackers can alter the website's appearance. **Data theft:** Attackers can steal sensitive user data.	
Suggested Countermeasures	
Input validation: Sanitize and validate all user inputs. **Output encoding:** Encode data before displaying it on the page. **Use a Content Security Policy (CSP):** Restrict the sources from which resources can be loaded. **HTTPOnly cookies:** Prevent client-side scripts from accessing cookies. **Regular security testing:** Conduct penetration testing and vulnerability scanning.	
References	
https://portswigger.net/web-security/cross-site-scripting	

Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab.



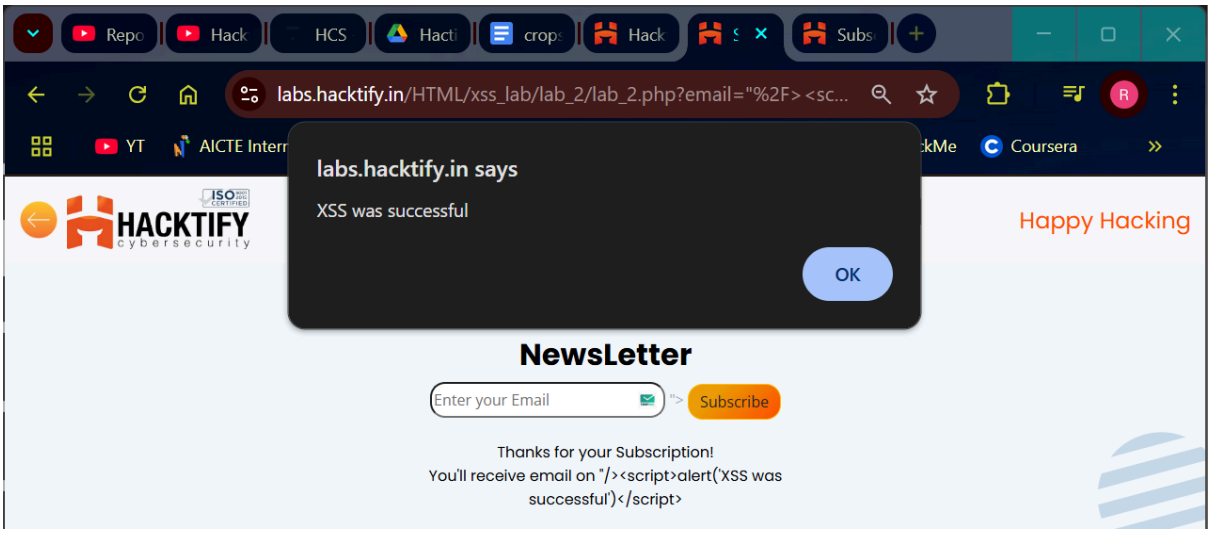
2.2. Balancing is Important in Life!

Reference	Risk Rating
Balancing is Important in Life!	**Low**
Tools Used	
Web Browser	
Vulnerability Description	
This payload exploits **Reflected XSS** by injecting a `<script>` tag into the webpage. If user input is not sanitized, the script executes when the page loads, triggering an alert box. An attacker can use this to steal session data, deface the page, or inject malicious code. Payload: "/><script>alert('XSS was successful')</script>	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_2/lab_2.php	
Consequences of not Fixing the Issue	
Consequences of not preventing this XSS vulnerability: **Account Takeover:** Attackers can steal user credentials (cookies, session tokens) and hijack accounts. **Data Theft:** Sensitive user data, including personal information, financial details, or other confidential data, can be stolen. **Malware Distribution:** The attacker can inject malicious scripts to distribute malware to unsuspecting users. **Website Defacement:** The attacker can alter the website's content, look, or functionality, damaging the site's reputation. **Session Hijacking:** Attackers can steal a user's active session and perform actions on their behalf without their knowledge.	
Suggested Countermeasures	
Regular Security Audits: Conduct frequent vulnerability scanning and penetration testing to identify and address weaknesses. **Web Application Firewall (WAF):** Implement a WAF to filter malicious traffic and block XSS attempts. **Intrusion Detection/Prevention System (IDS/IPS):** Deploy IDS/IPS to monitor network traffic for suspicious activity and block or alert on potential attacks. **Security Information and Event Management (SIEM):** Use SIEM to collect and analyze security logs from various sources to detect and respond to attacks.	

Incident Response Plan: Have a well-defined incident response plan in place to handle security breaches effectively and minimize damage. This includes steps for containment, eradication, recovery, and post-incident analysis.
References
https://portswigger.net/web-security/cross-site-scripting

Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab.



2.3. XSS is everywhere!

Reference	Risk Rating
XSS is everywhere!	**Medium**
Tools Used	
Web Browser	
Vulnerability Description	
This targets **Stored XSS**, where an email field does not properly escape user input before storing and displaying it. When the page loads the stored input, the script executes, affecting every user who views the page. Payload: qwerty@gmail.com<script>alert('XSS')</script>	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_3/lab_3.php?	
Consequences of not Fixing the Issue	
Consequences of not preventing this XSS vulnerability: **Account Takeover:** Attackers can steal user credentials (cookies, session tokens) and hijack accounts. **Data Theft:** Sensitive user data, including personal information, financial details, or other confidential data, can be stolen.	

Malware Distribution: The attacker can inject malicious scripts to distribute malware to unsuspecting users. **Website Defacement:** The attacker can alter the website's content, look, or functionality, damaging the site's reputation. **Session Hijacking:** Attackers can steal a user's active session and perform actions on their behalf without their knowledge.
Suggested Countermeasures
Regular Security Audits: Conduct frequent vulnerability scanning and penetration testing to identify and address weaknesses. **Web Application Firewall (WAF):** Implement a WAF to filter malicious traffic and block XSS attempts. **Intrusion Detection/Prevention System (IDS/IPS):** Deploy IDS/IPS to monitor network traffic for suspicious activity and block or alert on potential attacks. **Security Information and Event Management (SIEM):** Use SIEM to collect and analyze security logs from various sources to detect and respond to attacks. **Incident Response Plan:** Have a well-defined incident response plan in place to handle security breaches effectively and minimize damage. This includes steps for containment, eradication, recovery, and post-incident analysis.
References
https://portswigger.net/web-security/cross-site-scripting

Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab.

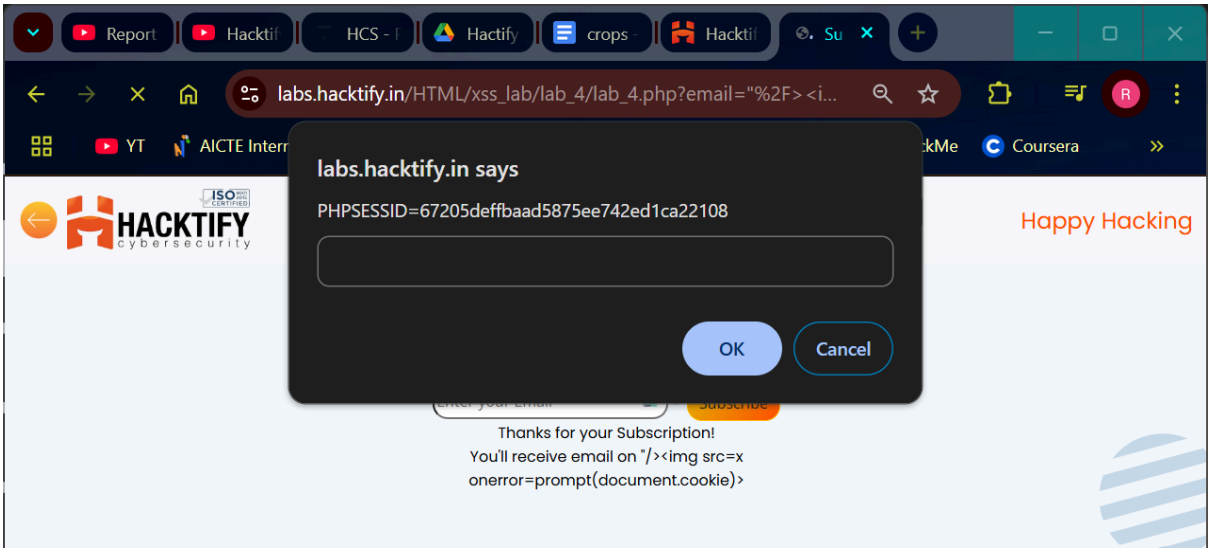
2.4. Alternatives are must!

Reference	Risk Rating
Alternatives are must!	**Medium**
Tools Used	
Web Browser	
Vulnerability Description	

This is an Event-based XSS attack, where an invalid image source (x) triggers the onerror event, executing prompt(document.cookie). If document.cookie is accessible, it can be used to steal session cookies. Payload: "/><img src=x onerror=prompt(document.cookie)>
How It Was Discovered
Manual Analysis
Vulnerable URLs
https://labs.hacktify.in/HTML/xss_lab/lab_4/lab_4.php?
Consequences of not Fixing the Issue
What will be the consequences if the vulnerability is not patched?
Suggested Countermeasures
Give some Suggestions to stand against this vulnerability
References
https://portswigger.net/web-security/cross-site-scripting

Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab.



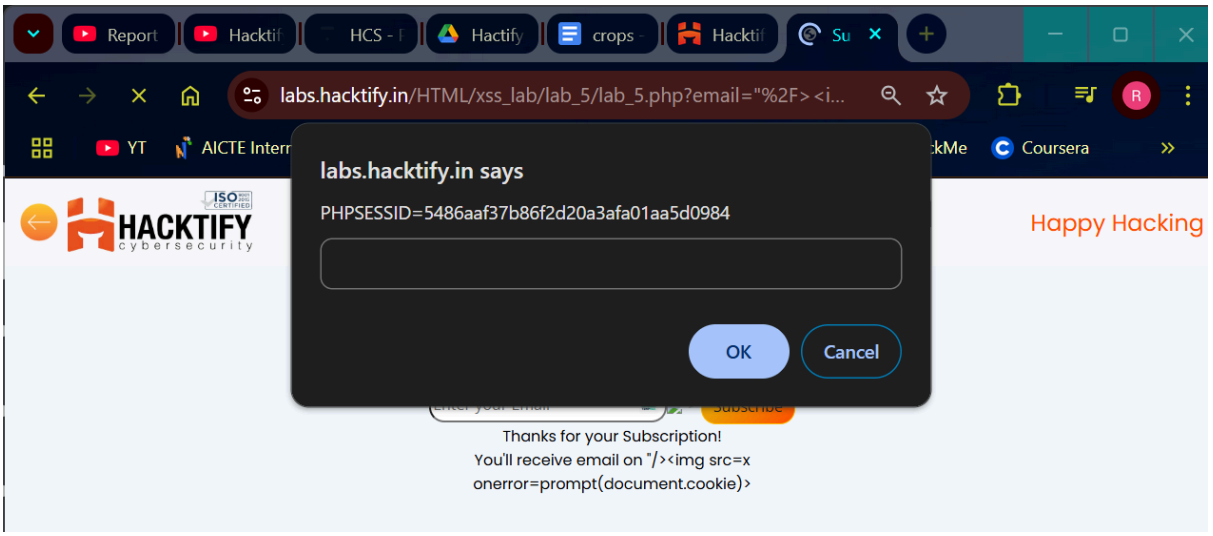
2.5. Developer hates scripts!

Reference	Risk Rating
Developer hates scripts!	**High**
Tools Used	
Web Browser	
Vulnerability Description	
This is an Event-based XSS attack, where an invalid image source (x) triggers the onerror event, executing prompt(document.cookie). If document.cookie is accessible, it can be used to steal session cookies.	

Payload: "/><img src=x onerror=prompt(document.cookie)>
How It Was Discovered
Manual Analysis
Vulnerable URLs
https://labs.hacktify.in/HTML/xss_lab/lab_5/lab_5.php?
Consequences of not Fixing the Issue
What will be the consequences if the vulnerability is not patched?
Suggested Countermeasures
Give some Suggestions to stand against this vulnerability
References
https://portswigger.net/web-security/cross-site-scripting

Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab.



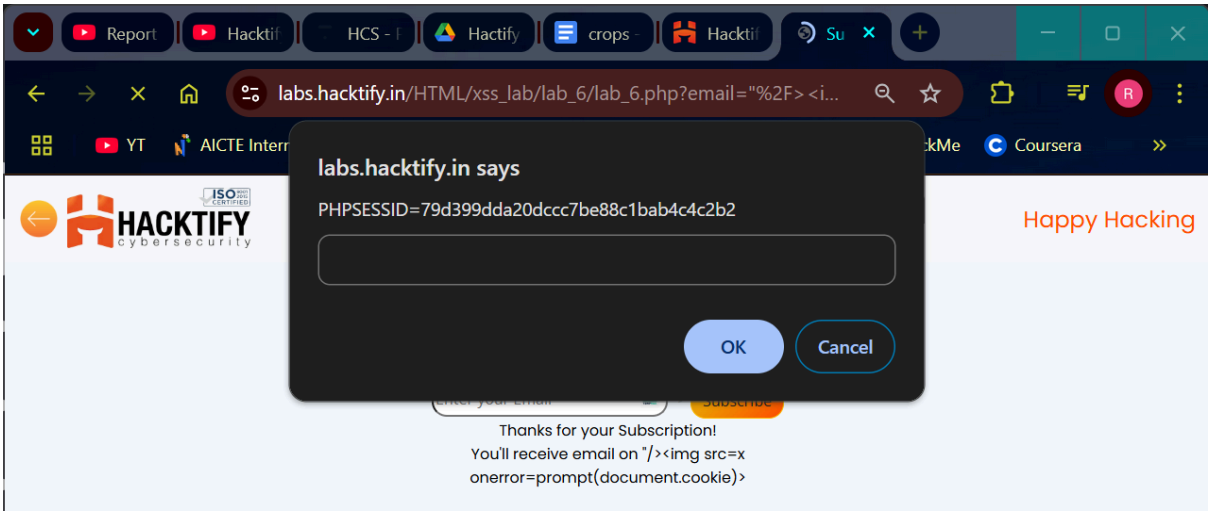
2.6. Change the Variation!

Reference	Risk Rating
Change the Variation!	**Medium**
Tools Used	
Web Browser	
Vulnerability Description	
This is an Event-based XSS attack, where an invalid image source (x) triggers the onerror event, executing prompt(document.cookie). If document.cookie is accessible, it can be used to steal session cookies. Payload: "/><img src=x onerror=prompt(document.cookie)>	
How It Was Discovered	
Manual Analysis	

Vulnerable URLs
https://labs.hacktify.in/HTML/xss_lab/lab_6/lab_6.php?
Consequences of not Fixing the Issue
What will be the consequences if the vulnerability is not patched?
Suggested Countermeasures
Give some Suggestions to stand against this vulnerability
References
https://portswigger.net/web-security/cross-site-scripting

Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab.



2.7. Encoding is the key?

Reference	Risk Rating
Encoding is the key?	**Medium**
Tools Used	
Tools that you have used to find the vulnerability.	
Vulnerability Description	
This is a URL-encoded XSS attack, where the <script> tag is encoded to %3Cscript%3E, attempting to bypass simple input filters. If the page decodes and executes it, the attack functions like a regular <script> injection. Payload: %3Cscript%3Ealert%28document.cookies%29%3C%2Fscript%3E	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_7/lab_7.php?	
Consequences of not Fixing the Issue	
What will be the consequences if the vulnerability is not patched?	

Suggested Countermeasures
Give some Suggestions to stand against this vulnerability
References
https://portswigger.net/web-security/cross-site-scripting

Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab.

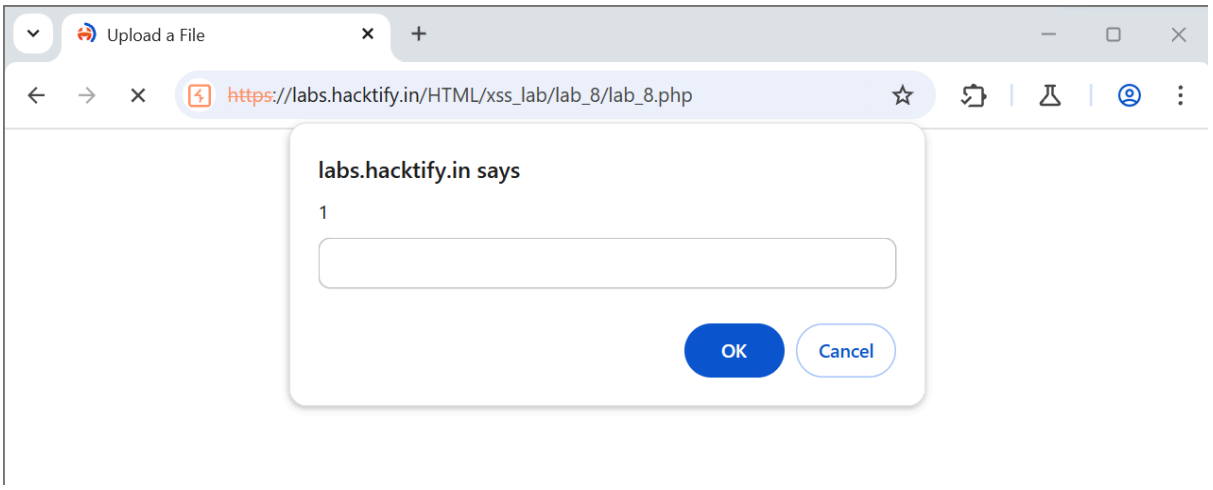
2.8. XSS with File Upload (file name)

Reference	Risk Rating
XSS with file upload (file name)	**Low**
Tools Used	
Web Browser and Burp Suite Community Edition	
Vulnerability Description	
In a **Burp Suite Interceptor payload attack**, an attacker manipulates the **file name** during an **HTTP file upload request** to inject malicious content. This attack is commonly used in **Stored XSS (Cross-Site Scripting) or Remote Code Execution (RCE)** scenarios, depending on how the application processes file names. Payload: "><img src=q onerror=prompt(1)">	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_8/lab_8.php	
Consequences of not Fixing the Issue	
- **Stored XSS:** Malicious scripts in file names (`"><script>alert('XSS')</script>.jpg`) can execute when displayed, leading to session hijacking or phishing.	

- **Remote Code Execution (RCE):** Changing file names (`shell.php.jpg`) can bypass validation, allowing execution of arbitrary code on the server. - **Phishing Attacks:** Renaming a file to `login.html` can trick users into entering credentials on a fake page hosted within the application. - **Session Hijacking:** If file names are reflected in session-related features, injected JavaScript can steal cookies, leading to account takeovers. - **Defacement & Data Theft:** Injected scripts in manipulated file names can alter UI, redirect users, or exfiltrate data via AJAX requests.
Suggested Countermeasures
- **Sanitize Inputs:** Remove special characters (<, >, ", ', &) from file names to prevent script injection. - **Restrict File Types:** Validate file MIME types instead of extensions and enforce strict upload policies. - **Disable Direct Execution:** Store uploaded files in non-executable directories to prevent unauthorized execution. - **Implement Content Security Policy (CSP):** Restrict inline JavaScript execution to prevent session hijacking. - **Secure File Storage Paths:** Avoid directly reflecting file names in pages; use hashed names and controlled access methods.
References
https://portswigger.net/web-security/cross-site-scripting https://portswigger.net/burp/documentation/desktop/getting-started/running-your-first-scan

Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab.



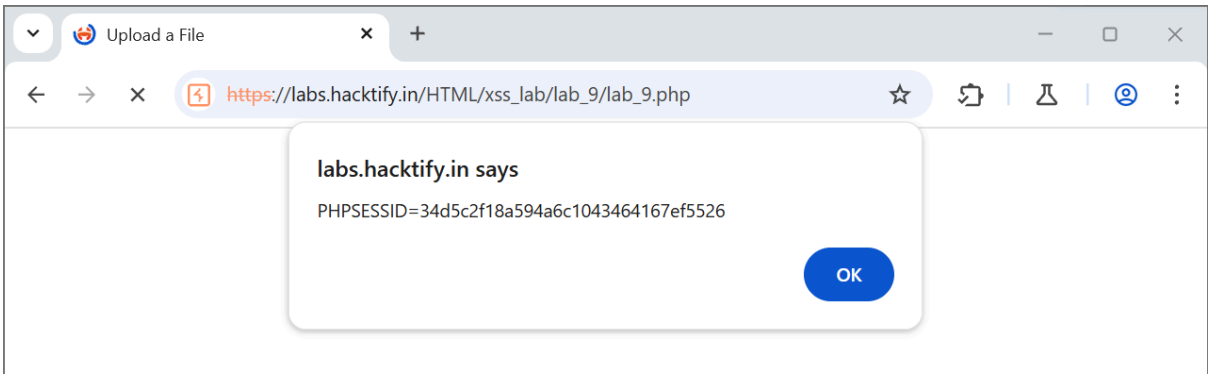
2.9. XSS with File Upload (file content)

Reference	Risk Rating
XSS with file upload (file content)	**High**
Tools Used	
Web Browser	

Vulnerability Description
A file upload attack occurs when an attacker uploads a malicious file containing executable code, such as a script, instead of a legitimate file. In this case, the attacker uploads a file containing the payload. If the application does not properly validate or restrict file uploads, this file may be stored on the server and executed when accessed, leading to Cross-Site Scripting (XSS). When a victim opens the file or a vulnerable page displays its contents, the script executes, stealing cookies or performing unauthorized actions. Payload:<script>alert(document.cookie)</script>
How It Was Discovered
Manual Analysis
Vulnerable URLs
https://labs.hacktify.in/HTML/xss_lab/lab_9/lab_9.php
Consequences of not Fixing the Issue
- **Stored XSS:** The uploaded file may be executed later when accessed, stealing session cookies and allowing account takeovers. - **Remote Code Execution (RCE):** If improperly validated, an attacker could upload and execute scripts or shell files to gain full server control. - **Malware Distribution:** Malicious files can be uploaded and shared, leading to the spread of viruses or trojans among users. - **Defacement & Data Theft:** The attacker can modify website content or extract sensitive information stored in the system. - **Phishing Attacks:** Uploaded files could contain fake login pages or other social engineering content to steal user credentials.
Suggested Countermeasures
- **Restrict File Types:** Allow only safe file types (e.g., `.jpg`, `.png`, `.pdf`) and validate MIME types to prevent execution of scripts. - **Sanitize File Names & Content:** Strip harmful content and block scripts within uploaded files to prevent execution. - **Store in Non-Executable Directories:** Save uploaded files in directories that do not allow execution (e.g., disable PHP execution in upload folders). - **Implement Content Security Policy (CSP):** Prevent execution of scripts from untrusted sources, reducing XSS risks. - **Perform Server-Side Scanning:** Use antivirus and sandboxing techniques to detect and block malicious file uploads.
References
https://portswigger.net/web-security/cross-site-scripting

Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab.



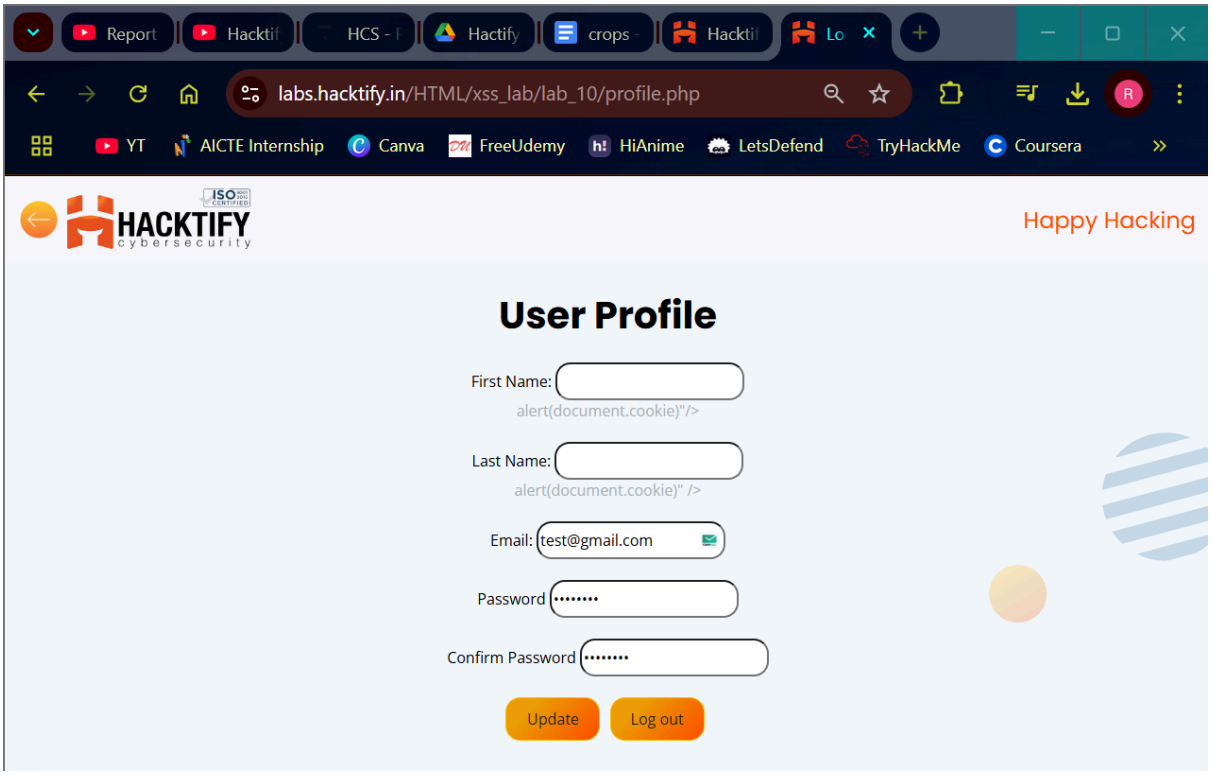
2.10.Stored Everywhere!

Reference	Risk Rating
Stored Everywhere!	**High**
Tools Used	
Web Browser	
Vulnerability Description	
DOM-Based XSS occurs when a web application dynamically updates the Document Object Model (DOM) using unvalidated user input, leading to script execution within the victim's browser. In this case, the attacker injects the payload "/<script>alert(document.cookie)</script> into the First Name and Last Name fields of a registration form. If the application improperly handles user input (e.g., inserting it directly into the webpage without sanitization), the malicious script will execute when the page loads or updates, allowing attackers to steal cookies, hijack sessions, or perform unauthorized actions on behalf of the victim. Payload: "/<script>alert(document.cookie)</script>	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_10/lab_10.php	
Consequences of not Fixing the Issue	
- **Session Hijacking:** The injected script steals session cookies, enabling attackers to impersonate users and gain unauthorized access. - **Phishing Attacks:** Attackers can modify webpage content to trick users into entering credentials or other sensitive information. - **Defacement & UI Manipulation:** Injected scripts can alter website appearance, replacing legitimate content with attacker-controlled elements. - **Malicious Redirections:** Users may be redirected to attacker-controlled phishing sites, leading to credential theft or malware downloads. - **Data Theft:** Sensitive user data stored in forms or local storage can be extracted and sent to an attacker's server.	
Suggested Countermeasures	
- **Sanitize User Input:** Remove or escape special characters (<, >, ", ') before inserting user data into the DOM. - **Use Secure JavaScript Methods:** Avoid `innerHTML` and `document.write`; use `textContent` or `createElement` instead to prevent script execution.	

- **Implement Content Security Policy (CSP):** Restrict execution of inline scripts and only allow trusted sources. - **Validate & Encode Data on the Client & Server:** Ensure proper input validation before reflecting user input on the page. - **Use JavaScript Security Libraries:** Utilize libraries like DOMPurify to sanitize user-generated content before displaying it.
References
https://portswigger.net/web-security/cross-site-scripting

Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab.



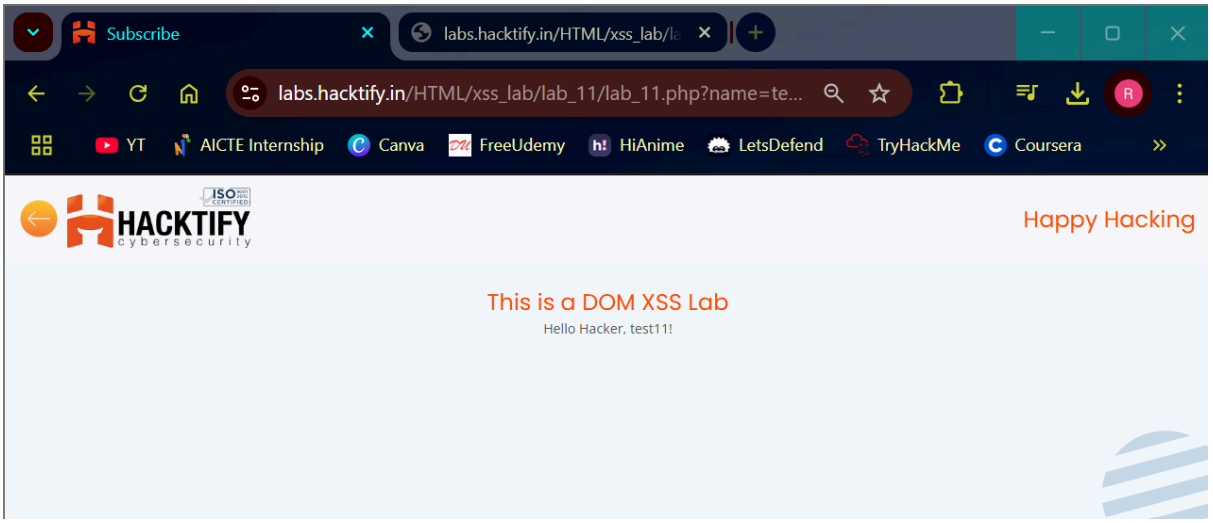
2.11. DOM's are love!

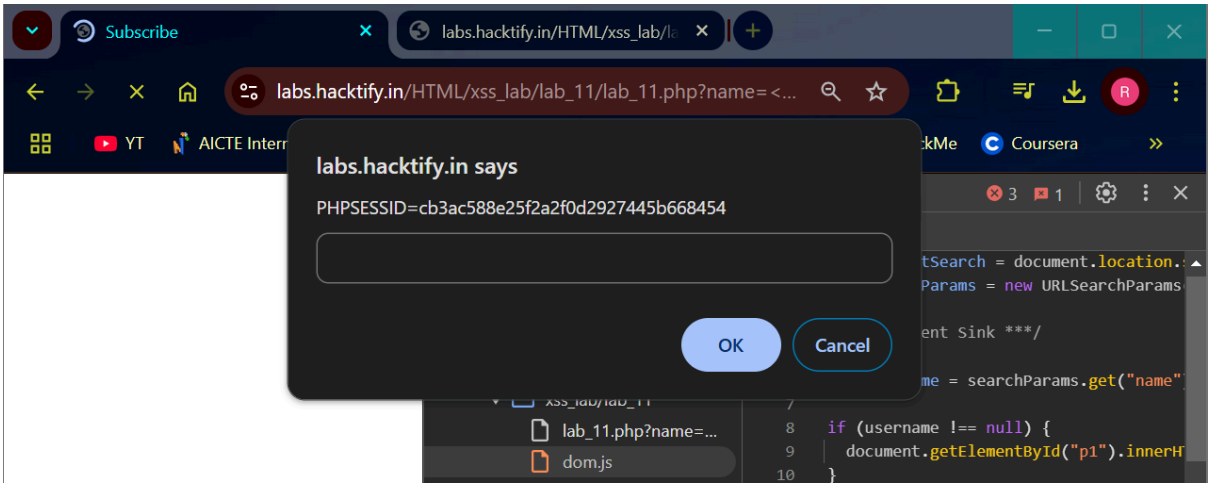
Reference	Risk Rating
DOM'S are love!	**High**
Tools Used	
Web Browser	
Vulnerability Description	
DOM-Based XSS occurs when user input is dynamically inserted into the DOM without proper sanitization. This payload is used to exploit an input field (e.g., a name field in a form) where the application directly reflects user input onto the webpage. Since <image> is an HTML tag, the browser attempts to load an image from the invalid src=q. This failure triggers the onerror event, which then	

executes prompt(document.cookie), allowing the attacker to steal session cookies. If the page does not properly sanitize input, this can lead to data theft, session hijacking, and phishing attacks. Payload: name=<image src =q onerror=prompt(document.cookie>)
How It Was Discovered
Manual Analysis
Vulnerable URLs
https://labs.hacktify.in/HTML/xss_lab/lab_11/lab_11.php
Consequences of not Fixing the Issue
- Session Hijacking: Attackers can steal authentication cookies, allowing them to take over user sessions. - Credential Theft: Users may be tricked into entering login details on fake forms injected into the page. - Website Defacement: Malicious scripts can alter webpage content, displaying misleading or harmful information. - Automated Malware Injection: Attackers can modify the page to serve malware-infected files to users. - Persistent XSS Attack: If the input is stored and reflected back on a future page, it could continuously execute for all visitor
Suggested Countermeasures
- Input Sanitization: Remove or escape special characters like <, >, ', and " to prevent script execution. - Use Secure JavaScript Methods: Replace innerHTML with textContent or createElement to prevent raw input execution. - Restrict Inline JavaScript Execution: Implement Content Security Policy (CSP) to block inline scripts. - Encode User Input Before Rendering: Convert user input into a safe format using encoding techniques (e.g., HTML entities). - Use Security Libraries: Implement DOMPurify or other sanitization libraries to filter harmful user inputs.
References
https://portswigger.net/web-security/cross-site-scripting

Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab.


Subscribe
labs.hacktify.in/HTML/xss_lab/la
labs.hacktify.in/HTML/xss_lab/lab_11/lab_11.php?name=te...
YT
AICTE Internship
Canva
FreeUdemy
HiAnime
LetsDefend
TryHackMe
Coursera
HACKTIFY
cybersecurity
Happy Hacking
This is a DOM XSS Lab
Hello Hacker, test11!


Subscribe
labs.hacktify.in/HTML/xss_lab/la
labs.hacktify.in/HTML/xss_lab/lab_11/lab_11.php?name=<...
YT
AICTE Intern
Coursera
labs.hacktify.in says
PHPSESSID=cb3ac588e25f2a2f0d2927445b668454
OK
Cancel
lab_11.php?name=...
dom.js
tSearch = document.location.
Params = new URLSearchParams
ent Sink ***/
me = searchParams.get("name"
if (username !== null) {
document.getElementById("p1").innerH
}