Penetration Testing Report

Full Name: Sahil Naik
Program: HCPT

Date: 12/02/2025

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
4	4	9

High - **Number of Sub-labs with hard difficulty level**

Medium - **Number of Sub-labs with Medium difficulty level**

Low - **Number of 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
Manual analysis, by using html payloads.
Vulnerability Description
HTML Injection is a vulnerability which occurs in web applications that allows users to insert HTML code via a specific parameter or an entry point. It is generally exploited using social engineering in order to trick valid users of the application to open malicious websites or to insert the credentials in a fake login form that will redirect the users to a page that captures cookies or credentials.
How It Was Discovered
Manual Analysis, that is by entering a HTML payload, eg root , in the input field.
Vulnerable URLs
https://labs.hacktify.in/HTML/html_lab/lab_1/html_injection_1.php
Consequences of not Fixing the Issue
HTML injection lets attackers insert malicious content into a webpage, leading to defacement, phishing, clickjacking, or unwanted redirects. It can trick users, damage a site's reputation, and, if combined with JavaScript, even steal sessions or credentials.
Suggested Countermeasures
1. Sanitize & Validate Input – Remove or escape harmful HTML before storing or displaying user input. 2. Use Content Security Policy (CSP) – Restrict execution of injected scripts and enforce secure handling of content.

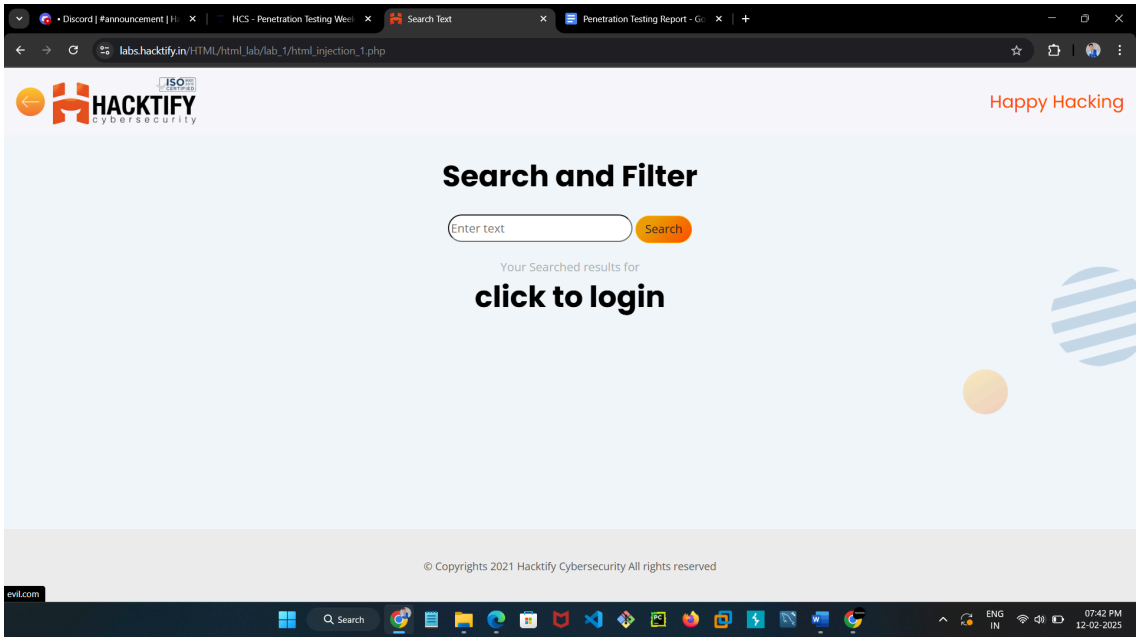
3. Escape User Output – Convert special characters (<, >, ") into safe HTML entities to prevent unintended rendering.
References
https://www.acunetix.com/vulnerabilities/web/html-injection/ https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html

Proof of Concept

1. After navigating to lab 1, I encountered a search field that takes a user input, I added this html payload in the input field :

   <h1>click to login</h1>

   At the left bottom we can see that when hovered on the text, it directs to evil.com

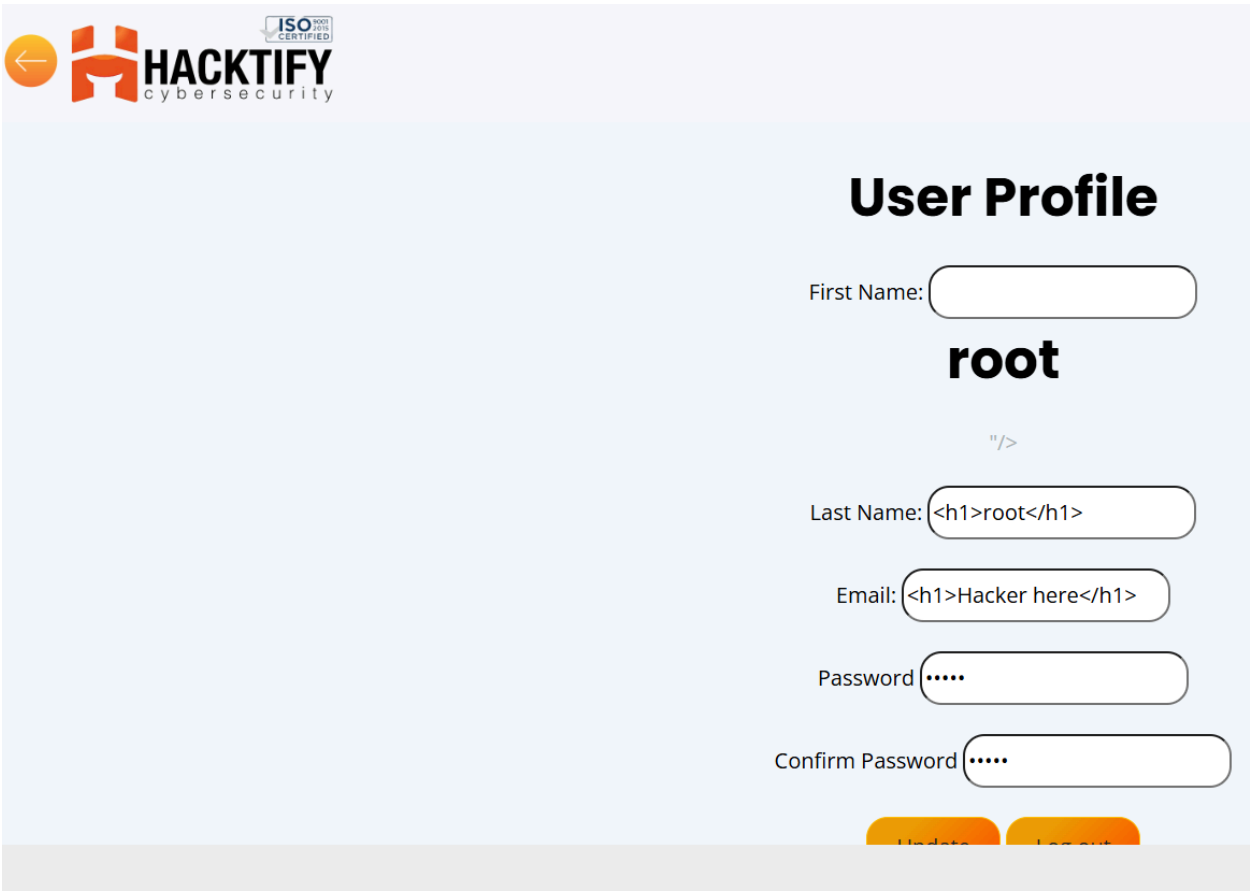


1.2. Let me Store them!

Reference	**Risk Rating**
Let me Store them!	**Low**
Tools Used	
Manual analysis, by using html payloads.	
Vulnerability Description	
Stored HTML Injection (Persistent Injection) is a web security vulnerability where malicious HTML code is permanently stored in a web application's database via user input fields and later rendered unsanitized on web pages.	
How It Was Discovered	
Manual Analysis, by adding html payloads and checking if they are stored in the database.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_2/html_injection_2.php	
Consequences of not Fixing the Issue	
Stored HTML Injection can cause data theft by manipulating web pages to steal user credentials. It enables malware distribution through injected scripts.	

Suggested Countermeasures
1. Input Validation & Sanitization – Restrict input using whitelisting, regex, and libraries like DOMPurify. 2. Output Encoding – Convert special characters into HTML entities to prevent execution. 3. Content Security Policy (CSP) – Restrict script execution and enforce secure content sources.
References
https://www.imperva.com/learn/application-security/html-injection/

Proof of Concept



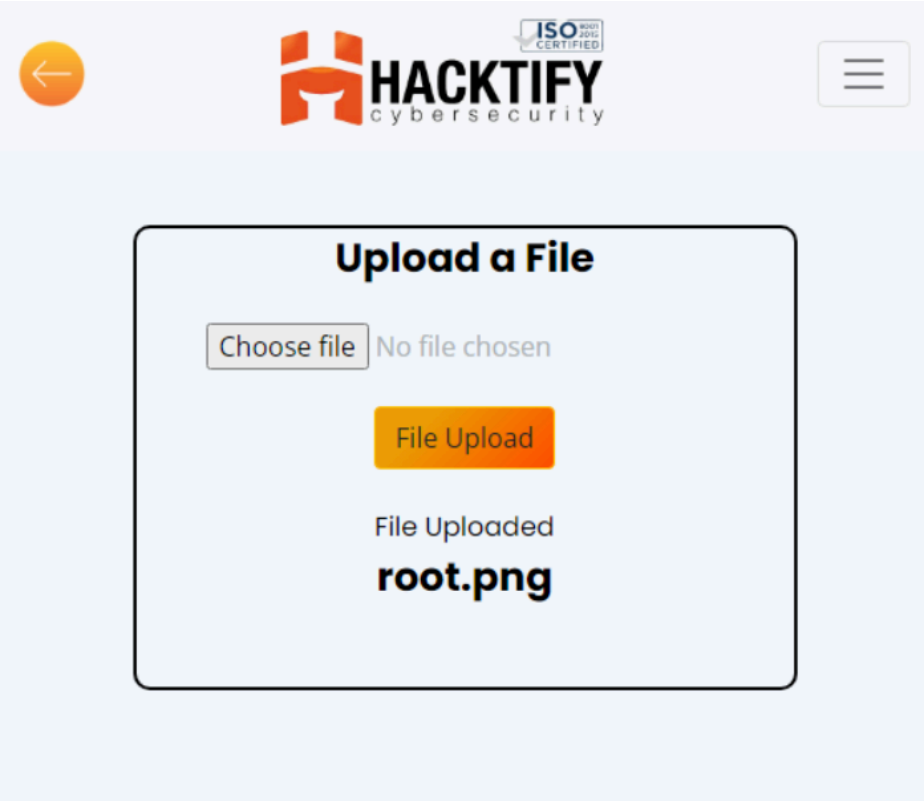
1.3. File Names are also vulnerable!

Reference	**Risk Rating**
File Names are also vulnerable!	**Low**
Tools Used	
Burp suite and Manual analysis using html payloads.	
Vulnerability Description	
A vulnerable file upload feature is one that does not properly sanitize or validate user-supplied filenames, allowing attackers to upload files with names containing HTML tags or special characters.	
How It Was Discovered	
I found the HTML injection by uploading a file with a name containing <h1> tags. When the filename was displayed on the page, it rendered as HTML instead of plain text, confirming the vulnerability.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_3/html_injection_3.php	
Consequences of not Fixing the Issue	
This vulnerability can distort the website's UI, trick users into interacting with fake elements, or even inject malicious links for phishing.	

Suggested Countermeasures
1. Sanitize filenames by stripping or encoding HTML tags before storing or displaying them. 2. Use Content-Disposition headers to force file downloads instead of rendering filenames in HTML. 3. Apply strict input validation to allow only alphanumeric characters and safe extensions for filenames.
References
https://www.imperva.com/learn/application-security/html-injection/ https://www.invicti.com/learn/html-injection/

Proof of Concept

Payload : <h1>root</h1>.png



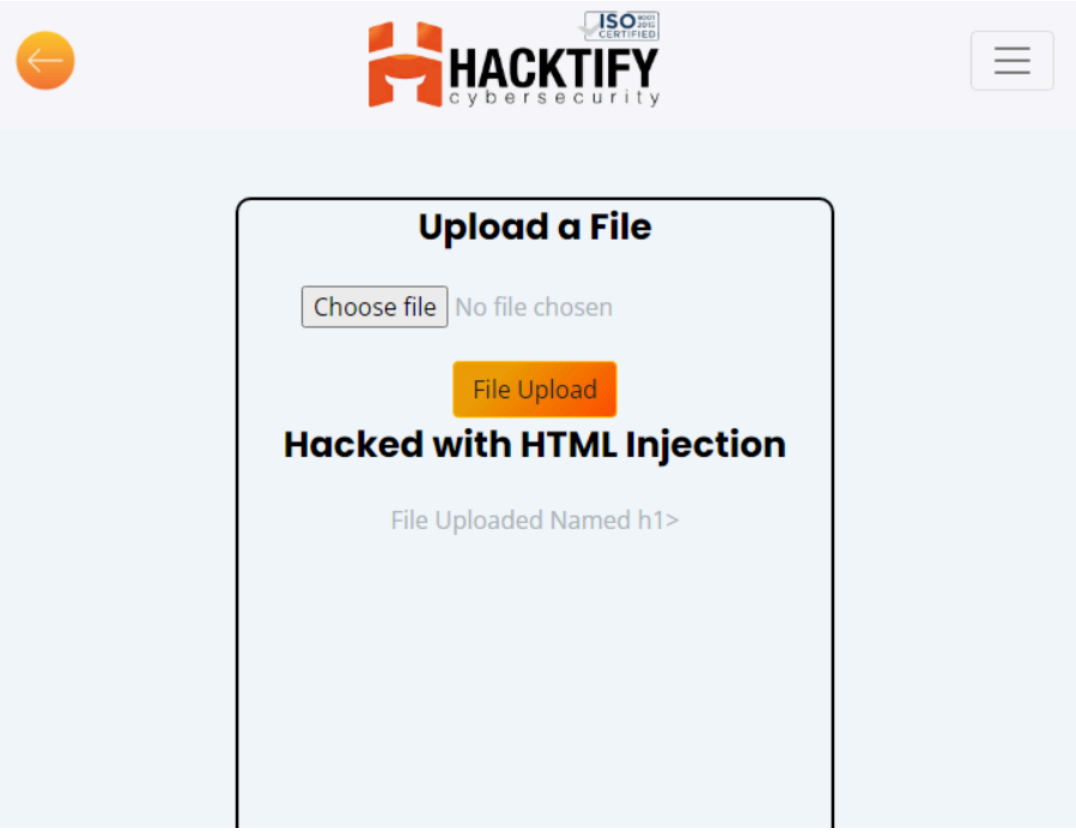
1.4. File Content and HTML Injection a perfect pair!

Reference	**Risk Rating**
File Content and HTML Injection a perfect pair!	**Low**
Tools Used	
Manual analysis using html payloads.	
Vulnerability Description	
Unrestricted File Upload vulnerability where the uploaded file's content is rendered directly in the website's UI without proper sanitization.	
How It Was Discovered	
I uploaded a file containing HTML tags like Test and noticed that the content was displayed as formatted HTML instead of plain text.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_4/html_injection_4.php	
Consequences of not Fixing the Issue	
1. UI Manipulation & Defacement : Attackers can alter the website’s appearance, insert fake links 2. Phishing & Social Engineering : Malicious HTML can trick users into entering credentials or interacting with fake elements.	

Suggested Countermeasures
1. Sanitize file content before rendering by encoding HTML characters to prevent execution. 2. Enforce content type validation to reject non-allowed formats like HTML files. 3. Serve uploaded files with proper headers e.g., `Content-Disposition: attachment`, to prevent in-browser rendering.
References
https://www.imperva.com/learn/application-security/html-injection/

Proof of Concept

Payload : text file containing <h1>Hacked with HTML Injection</h1>

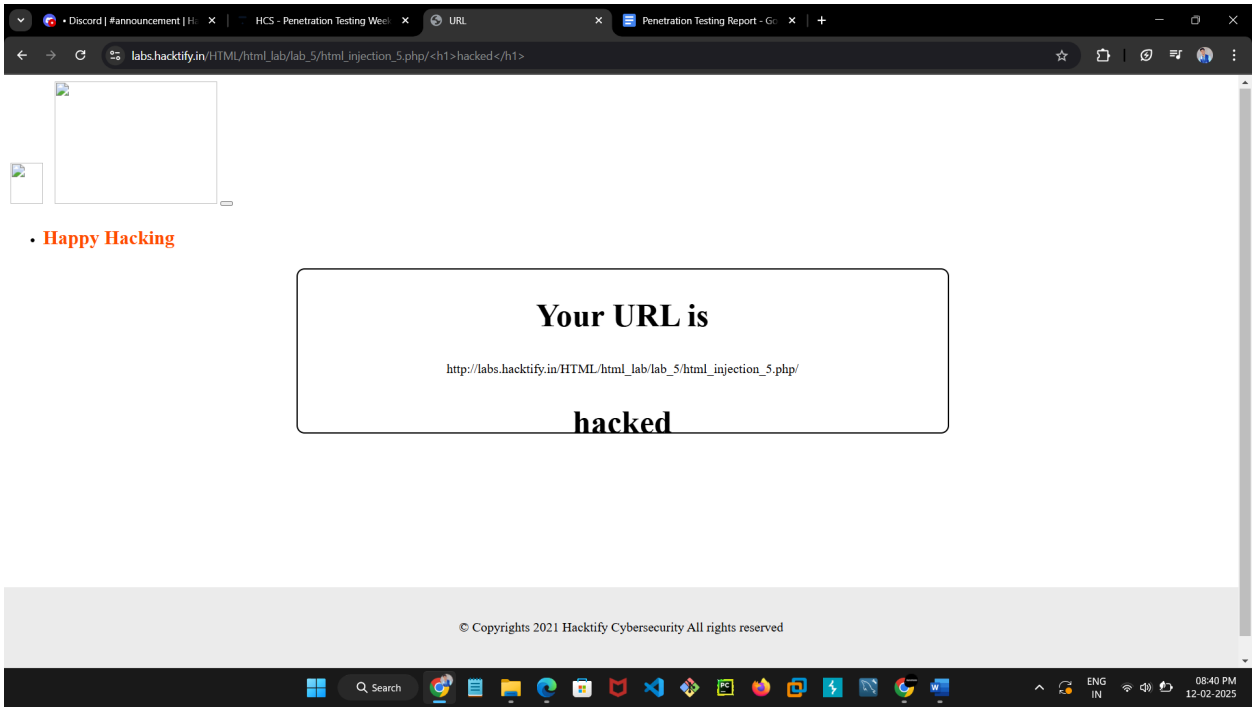


1.5. Injecting HTML using URL

Reference	**Risk Rating**
Injecting HTML using URL	**Medium**
Tools Used	
Manual analysis using html payloads.	
Vulnerability Description	
A vulnerability where unsanitized user input in the URL is directly rendered in the webpage.	
How It Was Discovered	
I tested the URL by appending HTML tags and observed that the page rendered the input as formatted HTML instead of plain text.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_5/html_injection_5.php	
Consequences of not Fixing the Issue	
1. Data Misinterpretation : Users may see misleading or fake content injected by attackers, causing confusion or reputational damage. 2. User Redirection : Attackers can manipulate the page to insert malicious links, tricking users into visiting harmful sites..	

Suggested Countermeasures
1. Properly escape user input by encoding HTML characters (`< > " &`) before rendering them in the UI. 2. Validate and sanitize URL parameters to allow only expected input formats. 3. Implement a Content Security Policy (CSP) to prevent the execution of injected content.
References
https://www.imperva.com/learn/application-security/html-injection/

Proof of Concept



Payload : /<h1>hacked</h1> appended in the url

1.6. Encode IT!

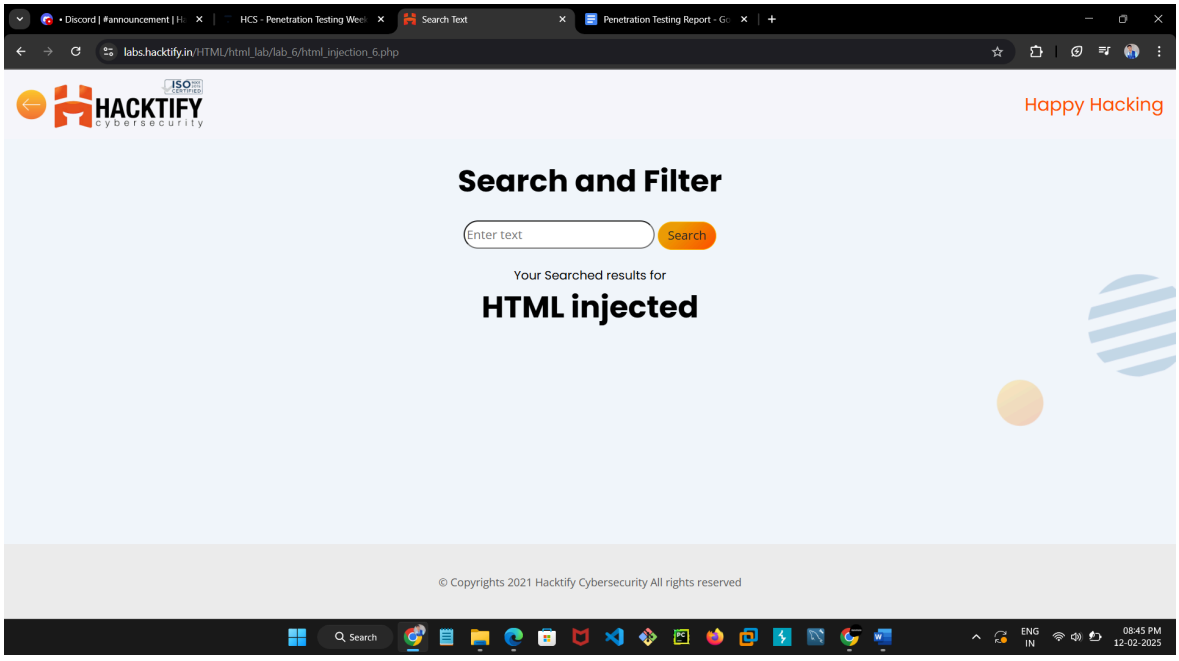
Reference	Risk Rating
Encode IT!	**Hard**
Tools Used	
CyberChef and html payloads.	
Vulnerability Description	
HTML Injection via Double Encoding vulnerability, where the application improperly decodes and renders encoded input. Direct input was treated as text, but after URL encoding, it was decoded and executed as HTML.	
How It Was Discovered	
I first entered hacked in the search field, but it was displayed as plain text. Then, I tried the URL-encoded version (%3Ch1%3Ehacked%3C%2Fh1%3E), and it got decoded and executed.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_6/html_injection_6.php	
Consequences of not Fixing the Issue	
1. Bypassing Input Filters :Attackers can evade basic protections and inject malicious HTML that executes in the UI.	

2. Stored or Reflected HTML Injection : Malicious content can persist in search results or be reflected back, misleading users or enabling phishing attacks.
Suggested Countermeasures
1. Decode and sanitize input properly before rendering to prevent double encoding bypass. 2. Enforce strict input validation to reject or escape HTML tags, even when encoded. 3. Use output encoding (e.g., HTML entity encoding) to ensure user input is displayed as text, not executed as HTML.
References
https://www.imperva.com/learn/application-security/html-injection/

Proof of Concept

Payload : %3Ch1%3EHTML%20injected%3C/h1%3E (url encoded)

(actual : <h1>HTML injected</h1>)



2. Cross Site Scripting

2.1. Let's Do IT!

Reference	**Risk Rating**
Let's Do IT!	**Low**
Tools Used	
Manual Analysis, using xss payloads	
Vulnerability Description	
Reflected XSS, when user input, such as from a search field, is immediately returned by the application without proper validation. This allows attackers to inject and execute malicious scripts in the victim's browser.	
How It Was Discovered	
Manual Analysis using xss payloads	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_1/lab_1.php	
Consequences of not Fixing the Issue	
1. An attacker can steal session cookies and impersonate the victim, gaining unauthorized access to their account.	

2. XSS can be used to craft convincing phishing pages that steal user credentials or other sensitive information.
3. Attackers can inject malicious scripts that download and execute malware on the victim's device.

Suggested Countermeasures

1. Input Validation and Sanitization – Validate and sanitize all user input to remove potentially malicious scripts before processing or displaying it.
2. Context-Aware Output Encoding – Encode output based on the specific context (HTML, JavaScript, URL, etc.) to prevent script execution.
3. Content Security Policy (CSP) – Implement a CSP to restrict the execution of inline scripts and limit the sources from which scripts can be loaded.

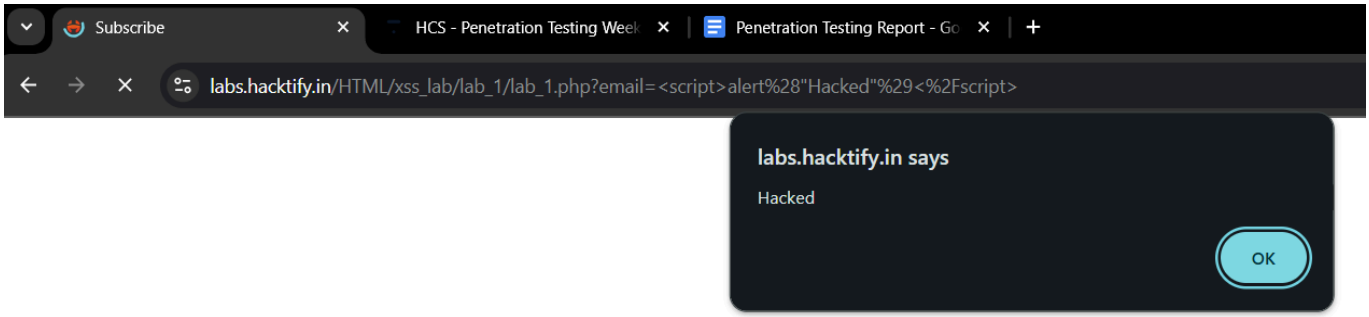
References

https://owasp.org/www-community/attacks/xss/

https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html

Proof of Concept

Payload used : <script>alert("Hacked")</script>



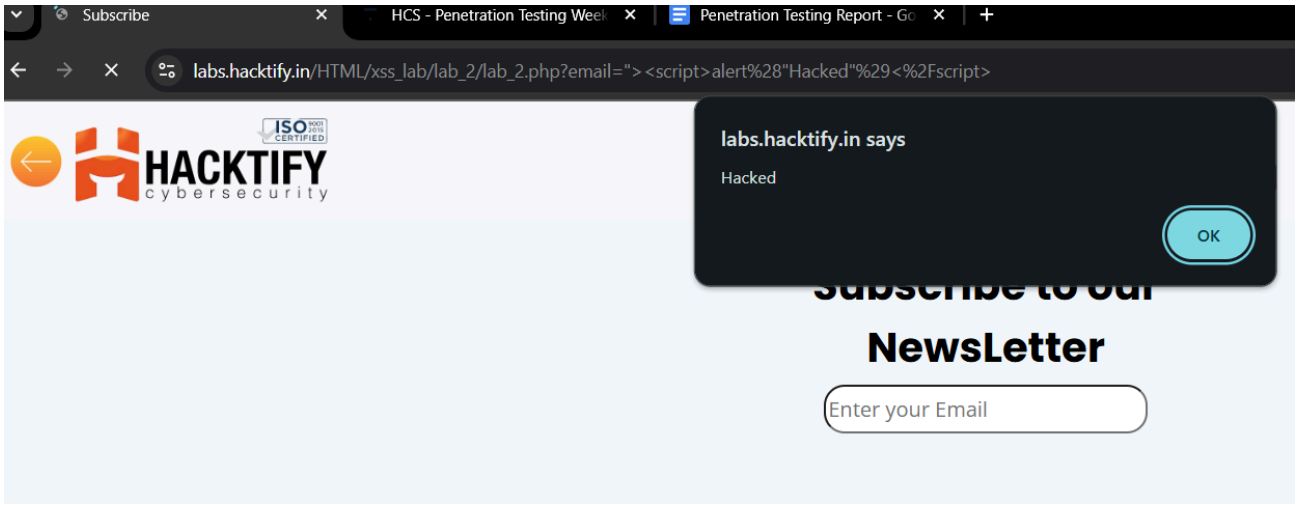
2.2. Balancing is Important in Life!

Reference	**Risk Rating**
Balancing is Important in Life!	**Low**
Tools Used	
Manual analysis using xss payloads	
Vulnerability Description	
RXSS triggered by balancing the payload involves carefully crafting an input that manipulates the application's parser to break out of existing contexts and execute malicious scripts	
How It Was Discovered	
Manual Analysis, balancing the xss payload.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_2/lab_2.php	
Consequences of not Fixing the Issue	
1. Data theft by injecting scripts that exfiltrate sensitive information. 2. Defacement of web pages by modifying displayed content. 3. Execution of unauthorized actions on behalf of the victim.	
Suggested Countermeasures	

1. Implement strict input validation to reject untrusted characters and patterns. 2. Use proper context-aware output encoding to prevent breaking out of intended contexts. 3. Apply a robust Content Security Policy (CSP) to limit script execution sources.
References
https://owasp.org/www-community/attacks/xss/ https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html

Proof of Concept

Payload : "><script>alert("Hacked")</script>



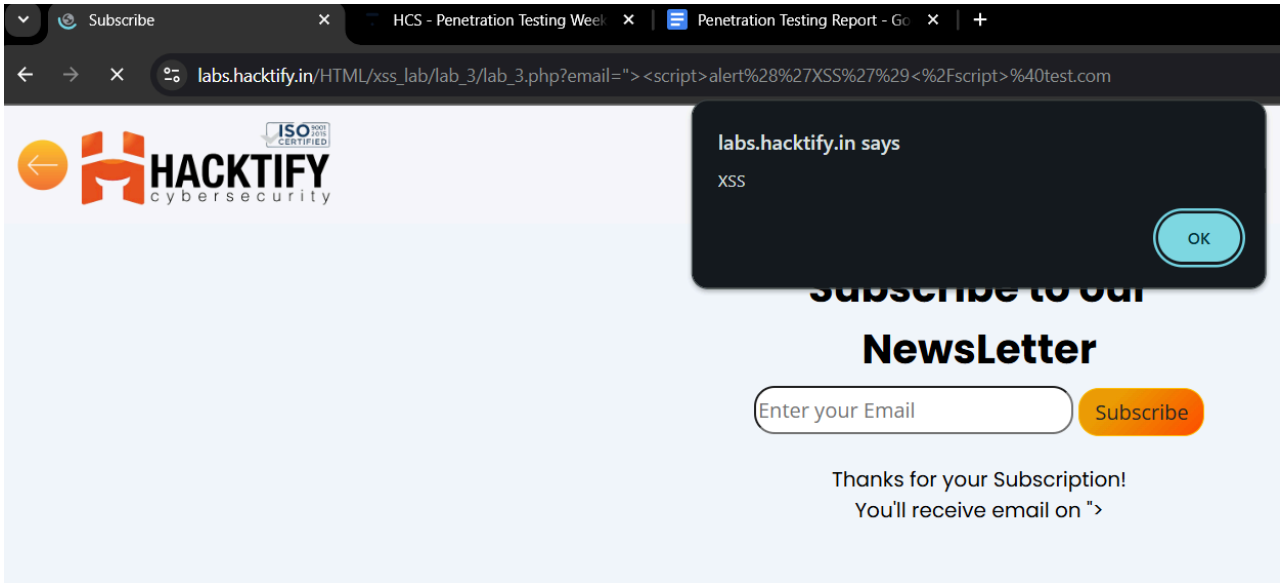
2.3. XSS is everywhere!

Reference	**Risk Rating**
XSS is everywhere!	**Low**
Tools Used	
Manual analysis using xss payloads	
Vulnerability Description	
RXSS when an application improperly validates email input, allowing an attacker to inject malicious scripts. Instead of enforcing strict email format rules, the application accepts inputs that include characters like "> or <script>, enabling the injection of executable JavaScript.	
How It Was Discovered	
Manual Analysis, using the xss payload.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_3/lab_3.php	
Consequences of not Fixing the Issue	
1. Session Hijacking: An attacker can steal session cookies and impersonate the victim, gaining unauthorized access to their account. 2. Phishing Attacks: XSS can be used to craft convincing phishing pages that steal user credentials or other sensitive information.	

3. Malware Distribution: Attackers can inject malicious scripts that download and execute malware on the victim's device.
Suggested Countermeasures
1. Input Validation and Sanitization: Validate and sanitize all user input to remove potentially malicious scripts before processing or displaying it. 2. Context-Aware Output Encoding: Encode output based on the specific context (HTML, JavaScript, URL, etc.) to prevent script execution. 3. Content Security Policy (CSP): Implement a CSP to restrict the execution of inline scripts and limit the sources from which scripts can be loaded.
References
https://owasp.org/www-community/attacks/xss/ https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html

Proof of Concept

Payload
"><script>alert('XSS')</script>@test.com (Only @ and . is validated)



2.4. Alternatives are must!

Reference	**Risk Rating**
Alternatives are must!	**Medium**
Tools Used	
Manual analysis using xss payloads	
Vulnerability Description	
Bypass of weak XSS sanitization, where an application attempts to mitigate XSS by filtering specific JavaScript functions (e.g., alert()) but fails to comprehensively prevent script Execution. (Reflected XSS)	
How It Was Discovered	
Manual Analysis, using the xss payload.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_4/lab_4.php	
Consequences of not Fixing the Issue	
1. Bypass of Security Controls – Attackers evade weak filtering, making the app falsely appear secure. 2. Expanded Attack Surface – Alternative scripts enable data theft, DOM manipulation, and credential stealing.	

3. Gateway to Advanced Exploits – Can lead to keylogging, session hijacking, and deeper attack

Suggested Countermeasures

1. Proper Output Encoding – Use context-aware encoding (e.g., HTML, JavaScript) to prevent script execution.
2. Whitelist-Based Input Validation – Allow only expected input formats instead of blocking specific keywords.
3. Strong Content Security Policy (CSP) – Restrict inline scripts and enforce secure script sources.

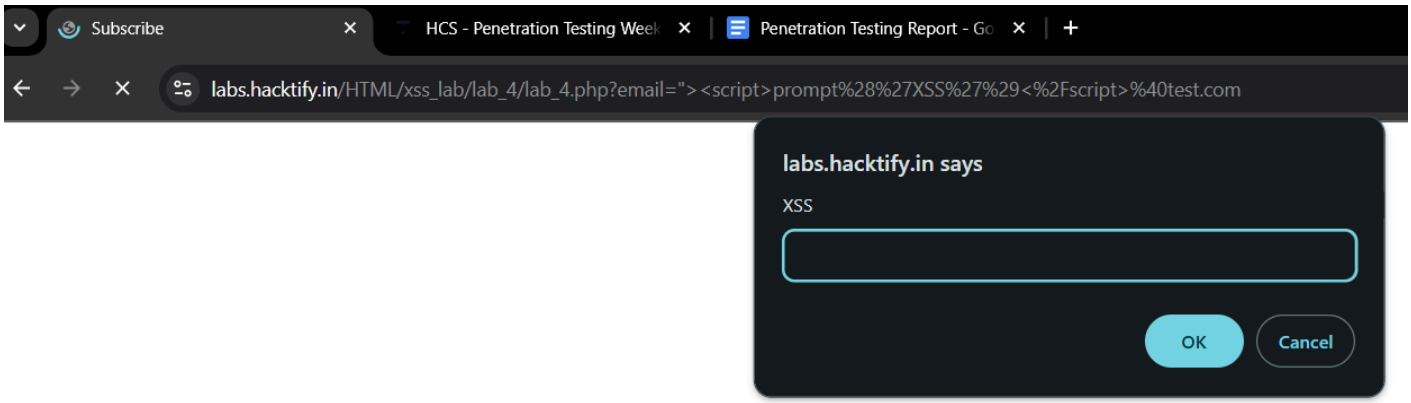
References

https://owasp.org/www-community/attacks/xss/

https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html

Proof of Concept

Payload : "><script>prompt('XSS')</script>@test.com



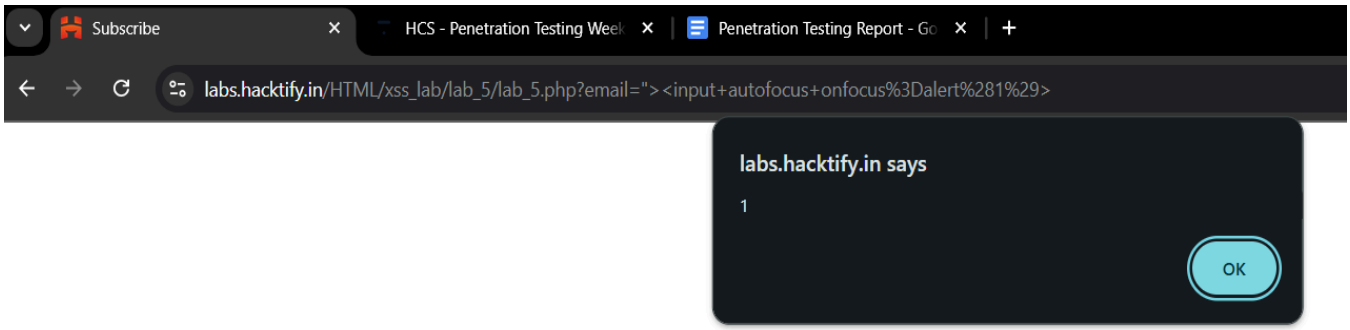
2.5. Developer hates scripts!

Reference	Risk Rating
Developer hates scripts!	**Hard**
Tools Used	
Manual analysis using xss payloads	
Vulnerability Description	
Reflected Scriptless XSS is a vulnerability where an attacker executes JavaScript without using <script> tags by exploiting improper input handling. It occurs due to insufficient validation and encoding.	
How It Was Discovered	
Manual Analysis, using the xss payload.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_5/lab_5.php	
Consequences of not Fixing the Issue	
1. Bypassing Traditional XSS Filters – Attackers evade basic script tag filters, making detection and mitigation harder.	

2. Persistent Exploitation in Modern Apps – Scriptless techniques work even in restricted environments, leading to long-term security risks. 3. Seamless User Manipulation – Attackers can silently steal data, hijack sessions, or perform unauthorized actions without obvious script execution.
Suggested Countermeasures
1. Proper Output Encoding – Encode user input to prevent unintended execution. 2. Strong CSP – Block inline scripts and unsafe JavaScript execution. 3. Strict Input Validation – Only allow safe and expected input formats.
References
https://owasp.org/www-community/attacks/xss/ https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html

Proof of Concept

Payload : "><input autofocus onfocus=alert(1)>



2.6. Change the Variation!

Reference	Risk Rating
Change the Variation!	**Hard**
Tools Used	
Manual analysis using xss payloads	
Vulnerability Description	
Stored XSS occurs when malicious input is permanently stored in the application and later executed in users' browsers. Even without <script> tags, attackers can trigger an alert using event handlers, javascript: URLs, or HTML attribute injections due to improper input validation and encoding.	
How It Was Discovered	
Manual Analysis, using the xss payload.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_6/lab_6.php	
Consequences of not Fixing the Issue	
1. Repeated Execution – The malicious code runs every time a user accesses the infected page.	

2. Hard to Detect & Remove – Since the payload is stored, it can go unnoticed for a long time.

Suggested Countermeasures

1. Proper Output Encoding – Prevents stored input from executing as code.
2. Strict Input Validation – Blocks harmful input before storing it.
3. Strong CSP – Restricts unauthorized script execution.

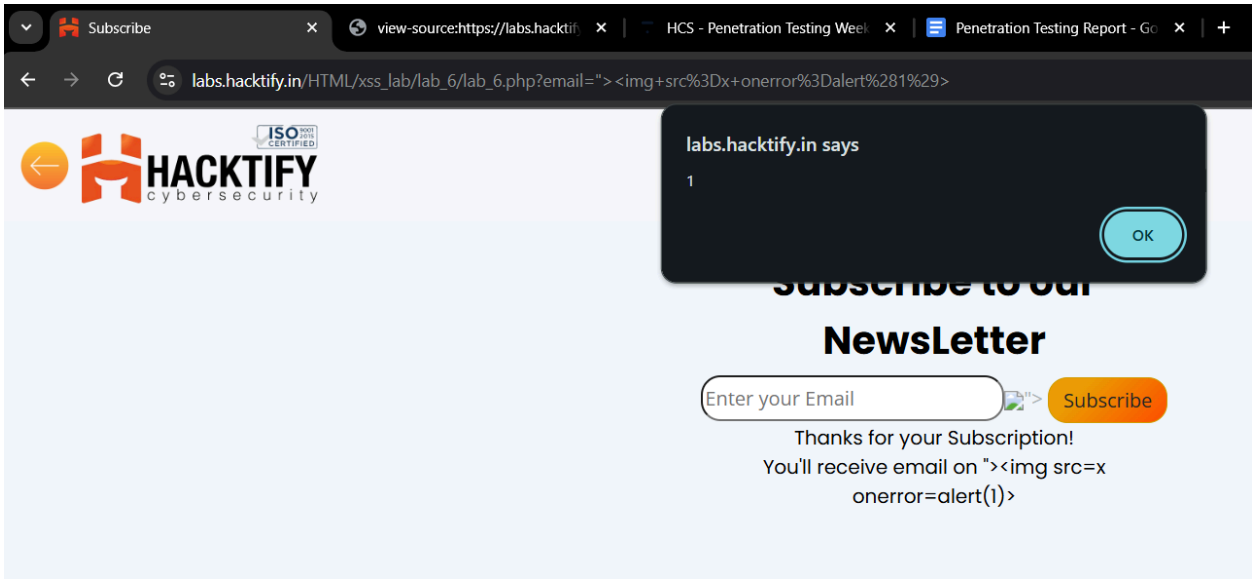
References

https://owasp.org/www-community/attacks/xss/

https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html

Proof of Concept

Payload
"><img src=x onerror=alert(1)>



2.7. Encoding is the key?

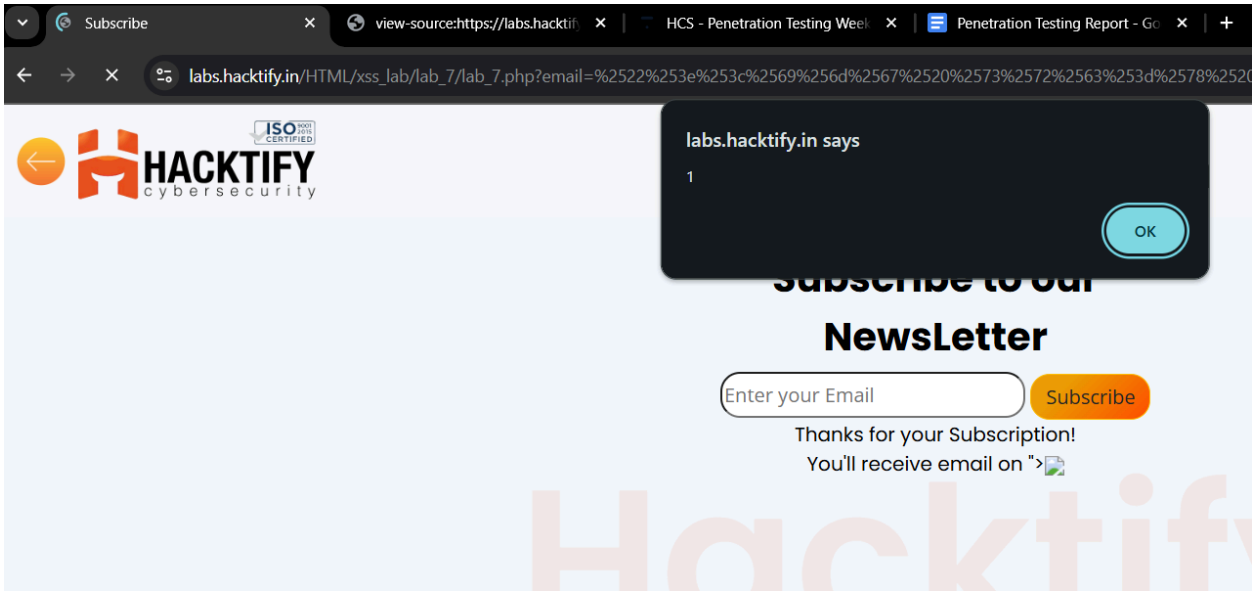
Reference	**Risk Rating**
Encoding is the key?	**Medium**
Tools Used	
Burpsuite (proxy and encoder)	
Vulnerability Description	
Stored XSS via an encoded payload in an email field occurs when malicious input is saved and later decoded improperly, allowing script execution. This leads to persistent attacks, affecting users by enabling session hijacking, data theft, or phishing.	
How It Was Discovered	
Manual Analysis, using encoded xss payload.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_7/lab_7.php	
Consequences of not Fixing the Issue	
1. Repeated Execution – The malicious code runs every time a user accesses the infected page. 2. Hard to Detect & Remove – Since the payload is stored, it can go unnoticed for a long time.	

3. Serious Data Theft – Attackers can steal user credentials, session cookies, or sensitive information.
Suggested Countermeasures
1. Proper Input Validation – Filter and sanitize all user inputs. 2. Secure Output Handling – Encode data before displaying it. 3. Strong CSP Implementation – Restrict unauthorized script execution.
References
https://owasp.org/www-community/attacks/xss/ https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html

Proof of Concept

Payload
%22%3e%3c%69%6d%67%20%73%72%63%3d%78%20%6f%6e%65%72%72%6f%72%3d%61%6c%65%72%74%28%31%29%3e

Actual : "><img src=x onerror=alert(1)>



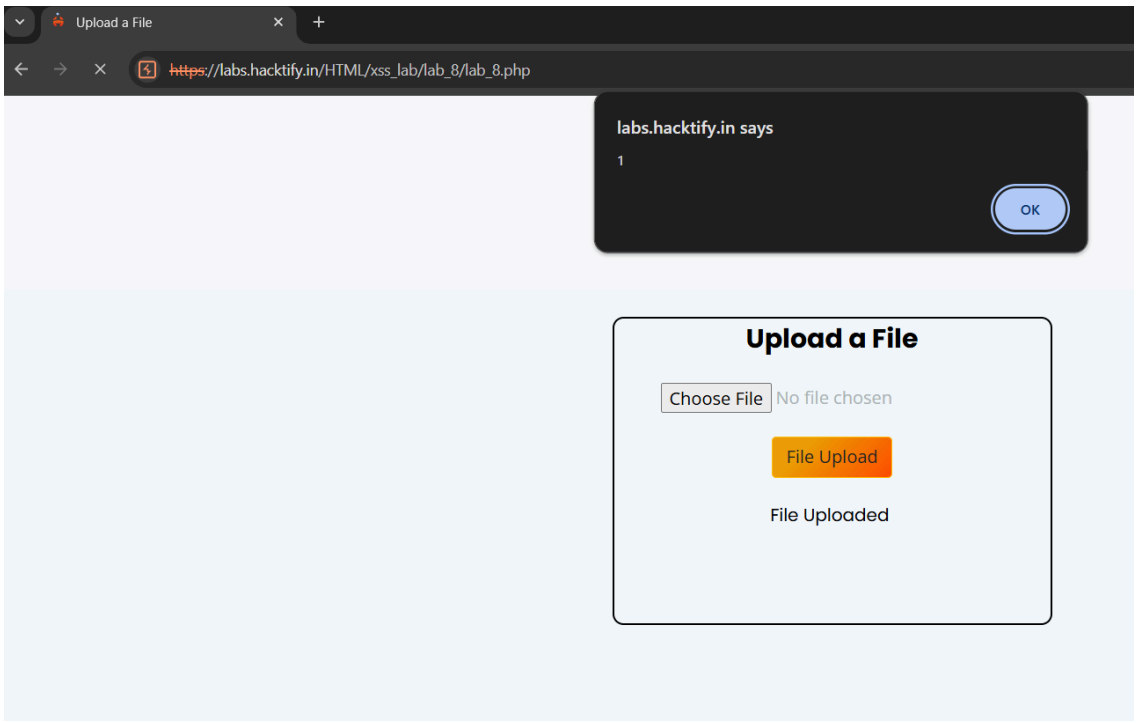
2.8. XSS with File Upload (file name)

Reference	**Risk Rating**
XSS with File Upload (file name)	**Easy**
Tools Used	
Burpsuite	
Vulnerability Description	
A file upload XSS vulnerability where an attacker uploads a file with a malicious filename containing an XSS payload. If the application displays the filename without proper encoding, the script executes in users' browsers.	
How It Was Discovered	
Manual Analysis, using the xss payload.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_8/lab_8.php	
Consequences of not Fixing the Issue	
1. Automatic Script Execution – Malicious code runs when the filename is displayed. 2. User Account Compromise – Attackers can steal session cookies and credentials. 3. System Exploitation – Can lead to further attacks, including malware injection.	

Suggested Countermeasures
1. Sanitize & Encode Filenames – Ensure filenames are properly escaped before display. 2. Restrict File Naming Rules – Allow only alphanumeric filenames and remove special characters. 3. Use Content Security Policy (CSP) – Prevent execution of injected scripts in case of exposure.
References
https://owasp.org/www-community/attacks/xss/ https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html

Proof of Concept

Payload
<img src=x onerror=alert(1)>

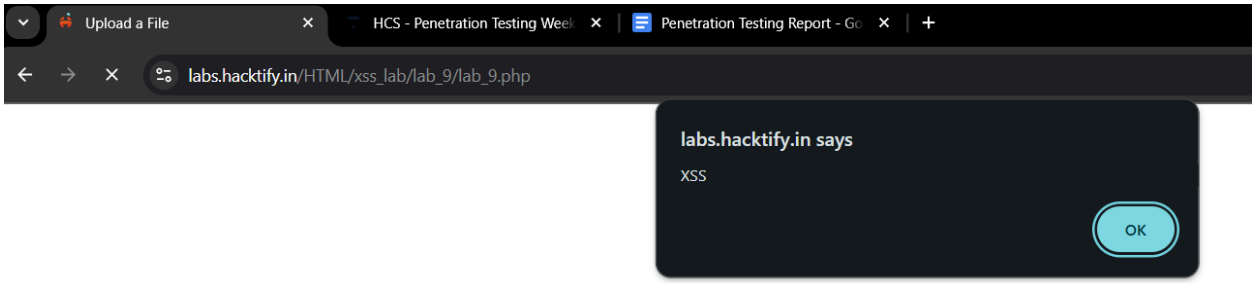


2.9. XSS with File Upload (File Content)

Reference	**Risk Rating**
XSS with File Upload (File Content)	**Medium**
Tools Used	
Manual analysis using xss payloads stored in a file	
Vulnerability Description	
Stored XSS vulnerability where an application displays file contents without proper sanitization, allowing embedded XSS payloads to execute.	
How It Was Discovered	
Manual Analysis, using the xss payload.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_9/lab_9.php	
Consequences of not Fixing the Issue	
1. Auto-Execution on File View – Any user viewing the file gets affected as the script runs automatically. 2. Admin Panel Exploitation – If an admin views the file, attackers can escalate privileges or gain control. 3. Malware Injection – The attacker can use the vulnerability to load additional malicious scripts.	

Suggested Countermeasures
1. Sanitize & Encode File Content – Prevent execution by properly handling displayed content. 2. Restrict Allowed File Types – Only permit safe formats that cannot contain executable scripts. 3. Implement Strong CSP – Block inline scripts and enforce secure content loading policies.
References
https://owasp.org/www-community/attacks/xss/ https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html

Proof of Concept



2.10. Stored Everywhere!

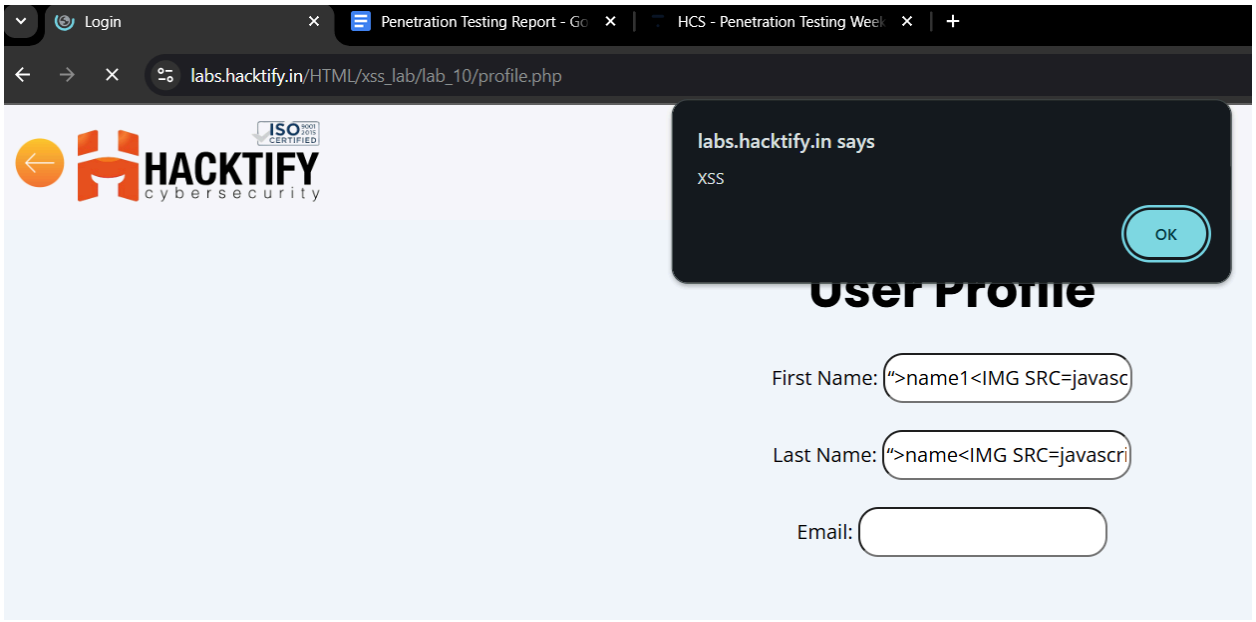
Reference	**Risk Rating**
Stored Everywhere!	**Easy**
Tools Used	
Manual analysis using xss payloads	
Vulnerability Description	
Stored XSS vulnerability where an attacker injects a malicious script into a user registration form. The application stores the payload in the database without proper sanitization. When the victim logs in and views their profile page, the stored payload executes	
How It Was Discovered	
Manual Analysis, using the xss payload.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_10/lab_10.php	
Consequences of not Fixing the Issue	
1. Automatic Execution on Login – The script runs as soon as the user views their profile. 2. Account Hijacking – Attackers can steal session cookies and take over accounts. 3. Admin Exploitation – If an admin views the infected profile, attackers can escalate privileges.	

Suggested Countermeasures
1. Sanitize & Encode User Input – Prevent stored data from executing as scripts. 2. Validate Input Strictly – Allow only safe characters in form fields. 3. Implement CSP – Block inline scripts and restrict execution to trusted sources.
References
https://owasp.org/www-community/attacks/xss/ https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html

Proof of Concept

Payload
“>name<IMG SRC=javascript:alert(“XSS”)>

Email "><script>alert('XSS')</script>@test.com



2.11. DOM's are love!

Reference	Risk Rating
DOM's are love!	**Hard**
Tools Used	
Manual analysis using xss payloads and chrome dev tools	
Vulnerability Description	
DOM XSS where user input is processed by JavaScript in the browser without proper sanitization, allowing attackers to modify the page and execute scripts.	
How It Was Discovered	
Manual Analysis, using the xss payload.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_11/lab_11.php	
Consequences of not Fixing the Issue	
1. Client-Side Exploitation – The attack runs directly in the victim's browser without server interaction. 2. Session Hijacking – Attackers can steal cookies and gain unauthorized access. 3. Phishing & Redirection – Users can be tricked into visiting malicious sites.	

Suggested Countermeasures
1. Use Safe JavaScript Methods – Avoid innerHTML, document.write(), and use textContent or createElement(). 2. Validate & Sanitize Input – Ensure user input is properly filtered before use. 3. Implement Content Security Policy (CSP) – Restrict inline scripts and untrusted sources.
References
https://owasp.org/www-community/attacks/xss/ https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html

Proof of Concept

Payload
In the url ?coin-btc<img src=x onerror=confirm("dom-xss")>