

# Penetration Testing Report

**Full Name** : Pramod Kumar P

**Program** : HCPT

**Date** : 18-02-2025

## Introduction

This document provides a detailed Proof of Concept (PoC) for HTML Injection and Cross-Site Scripting (XSS) vulnerabilities encountered in the Hacktify.in Labs training platform. The document includes observed vulnerabilities, exploitation techniques, payloads, and mitigation strategies.

### 1.Objective

The objective of this assessment is to identify and exploit HTML Injection and Cross-Site Scripting (XSS) vulnerabilities in the target application. These vulnerabilities can allow attackers to manipulate the web page's content or execute malicious scripts in a user's browser, leading to data theft, phishing attacks, and other security risks.

### 2.Scope

- Modules Tested: HTML input fields, query parameters, and reflected content
- Testing Type: Black-box and Gray-box testing
- Vulnerability Types:
  - HTML Injection
  - DOM-based XSS
  - Reflected XSS

Hacktify.in	HTML Injection, XSS Injection
-------------	-------------------------------

### 3. Summary

During the security assessment, multiple injection points were identified that allowed malicious input to be reflected in the application's DOM. The vulnerabilities could be exploited to execute JavaScript, steal session cookies, deface the webpage, or redirect users to phishing sites.

**Total number of Sub-labs: 17 Sub-lab**

High	Medium	Low
04	05	08

**High** - Four Sub-labs with hard difficulty level.

**Medium** - Five of Sub-labs with Medium difficulty level.

**Low** - Nine 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	
Burp Suite	
Vulnerability Description	
HTML Injection allows attackers to manipulate web page content.	
How It Was Discovered	
Manual testing identified an input field rendering HTML tags.	
Vulnerable URLs	
<a href="https://hacktify.in/HTML/html_lab/lab_1/html_injection_1.php">https://hacktify.in/HTML/html_lab/lab_1/html_injection_1.php</a>	

### Consequences of not Fixing the Issue

Attackers can deface the webpage, perform phishing, or inject malicious links

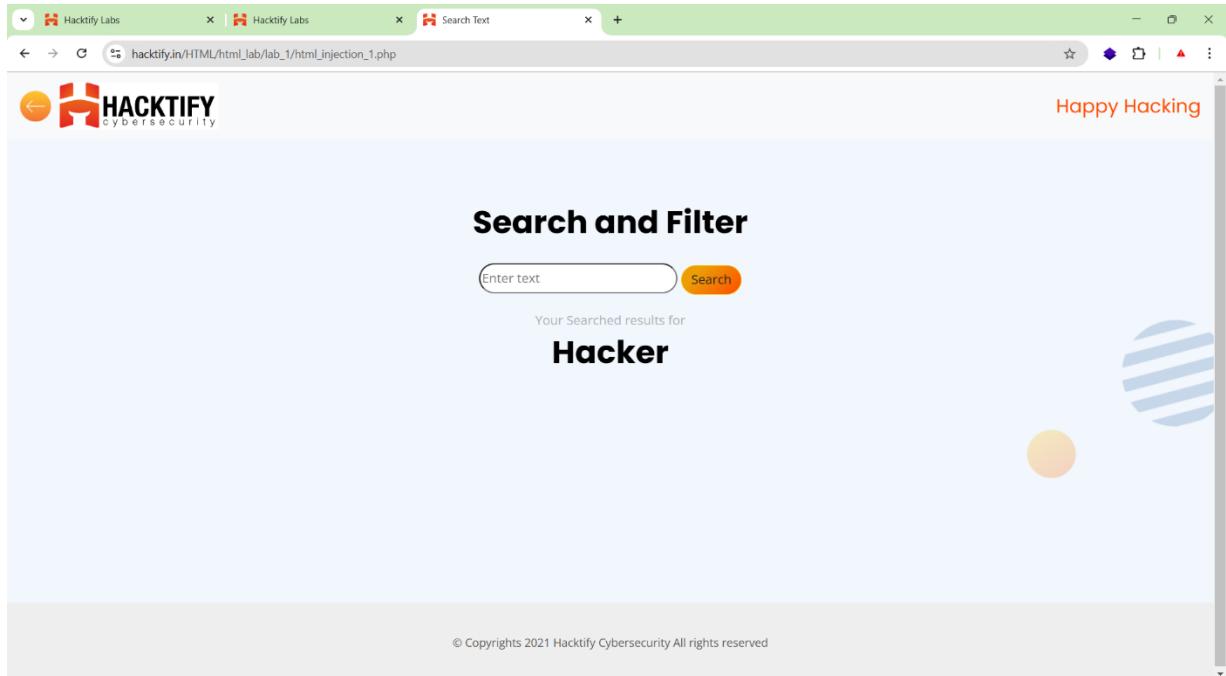
### Suggested Countermeasures

Sanitize user input by encoding HTML entities.

### References

OWASP HTML Injection Guide

## Proof of Concept



## 1.2. Let Me Store Them!

Reference	Risk Rating
Let Me Store Them!	Low
Tools Used	
Burp Suite	

## Vulnerability Description

HTML Injection enables attackers to modify displayed content on a webpage

## How It Was Discovered

Identified through manual testing by injecting HTML tags into input fields.

## Vulnerable URLs

[https://hacktify.in/HTML/html\\_lab/lab\\_2/html\\_injection\\_2.php](https://hacktify.in/HTML/html_lab/lab_2/html_injection_2.php)

## Consequences of not Fixing the Issue

Can lead to UI manipulation, phishing attempts, or misleading information display.

## Suggested Countermeasures

Implement input validation and encode user-supplied data.

## References

OWASP HTML Injection Guide

## Proof of Concept

The screenshot shows a user profile form on the Hacktify platform. The form includes fields for First Name, Last Name, Email, and Password. The 'First Name' field contains the value "Hacked". The 'Last Name' field contains the value "Successfully". The 'Email' field contains the value "sureshbabub.time@gmail.com". The 'Password' and 'Confirm Password' fields both contain the value "...". Below the form are two buttons: "Update" and "Log out". The page header features the Hacktify logo and the text "Happy Hacking". The footer contains the copyright notice "© Copyrights 2021 Hacktify Cybersecurity All rights reserved".

### 1.3. File Names Are Also Vulnerable!

Reference	Risk Rating
File Names Are Also Vulnerable!	Low
<b>Tools Used</b>	
Burp Suite, Web Browser	
<b>Vulnerability Description</b>	
The application allows users to upload HTML files without proper validation, leading to stored HTML injection.	
<b>How It Was Discovered</b>	
By manually uploading a crafted .html file containing injected HTML tags and observing the rendered output.	
<b>Vulnerable URLs</b>	
<a href="https://hacktify.in/HTML/html_lab/lab_3/html_injection_3.php">https://hacktify.in/HTML/html_lab/lab_3/html_injection_3.php</a>	
<b>Consequences of not Fixing the Issue</b>	
Attackers can execute <b>stored HTML injection</b> , misleading users or tricking them into phishing attacks.	
<b>Suggested Countermeasures</b>	
<ul style="list-style-type: none"><li>-Restrict file types to non-executable formats like .png, .jpg, or .pdf.</li><li>-Implement <b>server-side validation</b> to reject .html, .js, or other potentially harmful files.</li><li>-Enforce <b>Content Security Policy (CSP)</b> to mitigate content injection risks.</li></ul>	
<b>References</b>	
<a href="#">OWASP Unrestricted File Upload Guide</a>	

## Proof of Concept



### 1.4. File Content And HTML Injection A Perfect Pair!

Reference	Risk Rating
File Content And HTML Injection A Perfect Pair!	<b>Medium</b>
Tools Used	
Burp Suite, Web Browser	
Vulnerability Description	
The application allows uploading an HTML file without restrictions, leading to potential stored HTML injection.	
How It Was Discovered	
A manually crafted test.html file containing injected HTML was uploaded and successfully executed when accessed	

## Vulnerable URLs

[https://hacktify.in/HTML/html\\_lab/lab\\_4/html\\_injection\\_4.php](https://hacktify.in/HTML/html_lab/lab_4/html_injection_4.php)

## Consequences of not Fixing the Issue

- Stored HTML injection may mislead users with fake content.
- Attackers could exploit this for phishing or social engineering attacks.

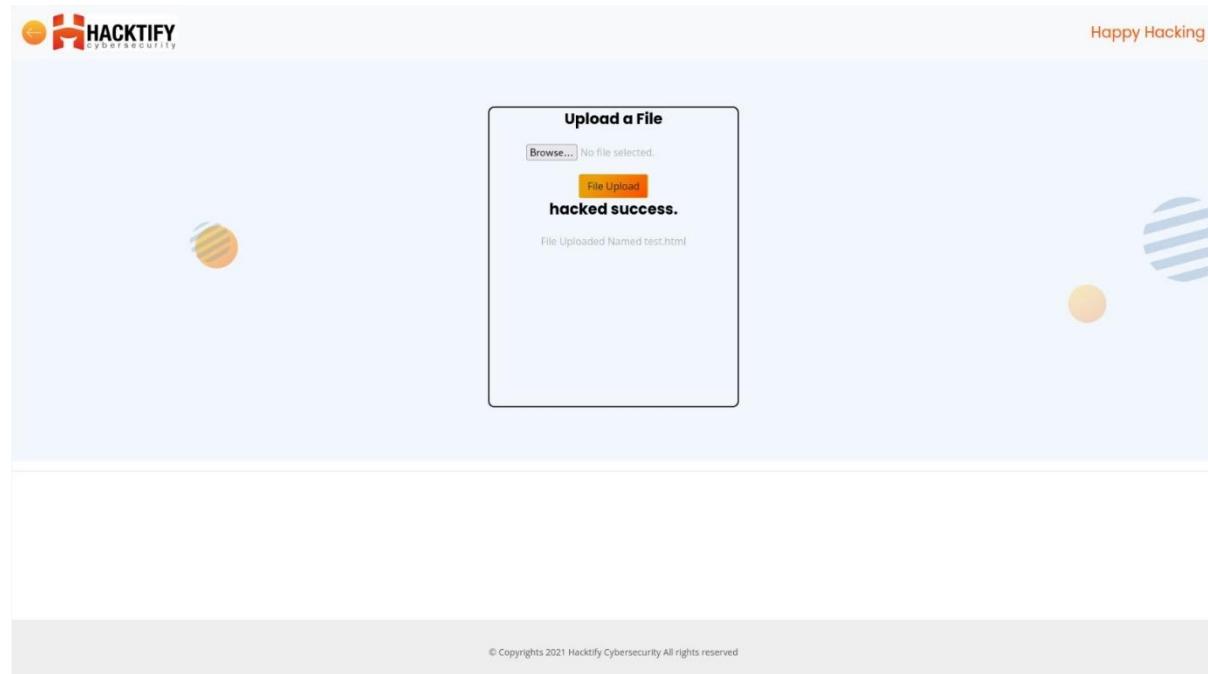
## Suggested Countermeasures

- Restrict file types to only necessary formats (e.g., .png, .jpg, .pdf).
- Validate and sanitize uploaded files on the server-side.
- Enforce **Content-Disposition: attachment** to prevent execution.

## References

[OWASP Unrestricted File Upload Guide](#)

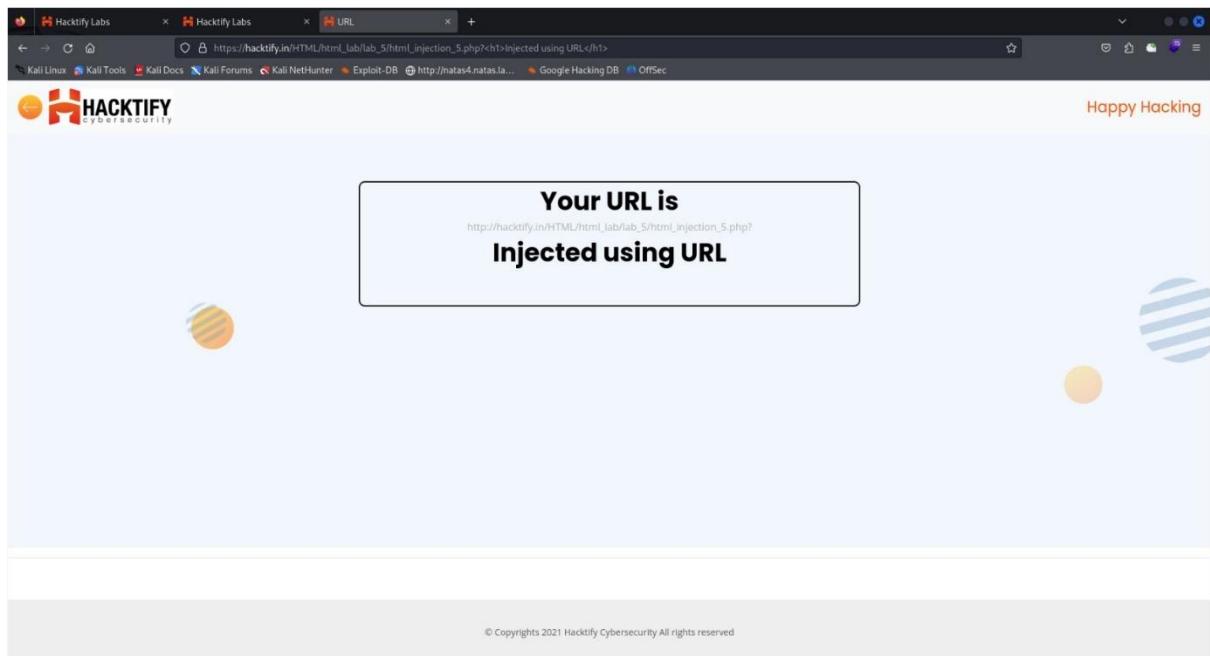
## Proof of Concept



## 1.5. Injecting HTML Using URL

Reference	Risk Rating
Injecting HTML Using URL	Medium
Tools Used	
Burp Suite	
Vulnerability Description	
The application is vulnerable to <b>HTML injection</b> , allowing attackers to manipulate page content by injecting HTML through URL parameters.	
How It Was Discovered	
Manually tested by injecting HTML elements into a parameterized URL, which was reflected on the webpage.	
Vulnerable URLs	
<a href="https://hacktify.in/HTML/html_lab/lab_5/html_injection_5.php?&lt;h1&gt;name&lt;/h1&gt;">https://hacktify.in/HTML/html_lab/lab_5/html_injection_5.php?&lt;h1&gt;name&lt;/h1&gt;</a>	
Consequences of not Fixing the Issue	
<ul style="list-style-type: none"><li>-Attackers can alter page content, misleading users.</li><li>-Could be used for phishing or social engineering attacks.</li></ul>	
Suggested Countermeasures	
<ul style="list-style-type: none"><li>-Sanitize user input before rendering.</li><li>-Use proper encoding to prevent direct HTML execution.</li><li>-Implement Content Security Policy (CSP) to mitigate injected content risks.</li></ul>	
References	
<a href="#">OWASP HTML Injection Guide</a>	

## Proof of Concept



### 1.6. Encoding It!

Reference	Risk Rating
Encoding it!	High
Tools Used	
Burp Suite, Web Browser	
Vulnerability Description	
The search functionality is vulnerable to <b>HTML injection</b> , allowing attackers to insert HTML tags, which are rendered on the webpage instead of being treated as plain text.	
How It Was Discovered	
Tested by entering encoded HTML tags (%3Ch3%3Ehacked%3C/h3%3E) into the search bar and observing their execution on the page.	
Vulnerable URLs	
<a href="https://hacktify.in/HTML/html_lab/lab_6/html_injection_6.php">https://hacktify.in/HTML/html_lab/lab_6/html_injection_6.php</a>	

### Consequences of not Fixing the Issue

- Attackers can manipulate displayed content.
- Could lead to phishing attacks, misleading information, or defacement.

### Suggested Countermeasures

- Implement input validation to allow only expected characters.
- Encode output before displaying user-generated input.
- Use security headers like **Content Security Policy (CSP)**.

### References

OWASP HTML Injection Guide

### Proof of Concept

The screenshot shows a search interface for 'hacked'. The search bar contains '%3Ch%33hacked%3C/h3%3E'. The search button is labeled 'Search'. Below the search bar, it says 'Your Searched results for hacked'. There are three decorative circular icons on the page: one orange icon on the left, one yellow icon on the right, and one blue-grey striped icon at the top right. At the bottom, there is a copyright notice: '© Copyrights 2021 Hacktify Cybersecurity All rights reserved'.

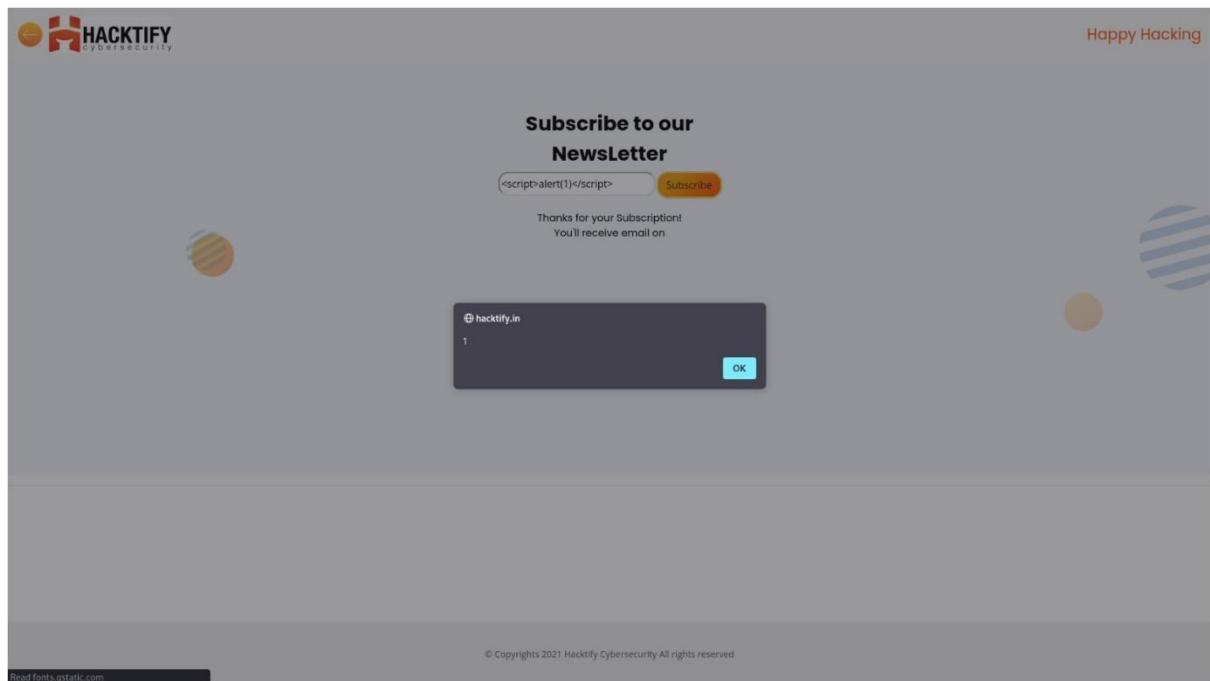
## 2. Cross Site Scripting

### 2.1. Let's Do It!

Reference	Risk Rating
Let's Do It!	Low

Tools Used
Burp Suite
Vulnerability Description
The search functionality does not properly sanitize user input, allowing <b>Cross-Site Scripting (XSS)</b> attacks. Injected scripts execute in the victim's browser, potentially leading to session hijacking or phishing.
How It Was Discovered
Tested by entering a simple XSS payload ( <code>&lt;script&gt;alert(1)&lt;/script&gt;</code> ) in the search bar and observing its execution in the browser.
Vulnerable URLs
<a href="https://hacktify.in/HTML/xss_lab/lab_1/lab_1.php">https://hacktify.in/HTML/xss_lab/lab_1/lab_1.php</a>
Consequences of not Fixing the Issue
-Malicious scripts can execute in users' browsers. -Attackers can steal session cookies, impersonate users, or redirect them to phishing sites.
Suggested Countermeasures
-Implement <b>input validation</b> to restrict special characters. -Encode user-generated content before displaying it. -Use <b>Content Security Policy (CSP)</b> to block inline scripts.
References
OWASP XSS Prevention Cheat Sheet

## Proof of Concept



## 2.2. Balancing Is Important In Life!

Reference	Risk Rating
Balancing Is Important In Life!	Low
Tools Used	
Burp Suite	
Vulnerability Description	
The search bar fails to properly sanitize input when passing the <b>email parameter</b> , allowing an attacker to inject JavaScript. The injected script executes in the victim's browser, leading to <b>Reflected XSS</b> .	
How It Was Discovered	
Manually tested by injecting "><script>alert(1)</script>" into the <b>email</b> parameter in the search bar and observing JavaScript execution.	
Vulnerable URLs	

[https://hacktify.in/HTML/xss\\_lab/lab\\_2/lab\\_2.php?email=%22%3E%3Cscript%3Ealert%281%29%3C%2Fscript%3E](https://hacktify.in/HTML/xss_lab/lab_2/lab_2.php?email=%22%3E%3Cscript%3Ealert%281%29%3C%2Fscript%3E)

### Consequences of not Fixing the Issue

- Attackers can execute malicious JavaScript in users' browsers.
- Can lead to **session hijacking, cookie theft, or phishing attacks**.
- Attackers may redirect users to malicious sites or keylog their inputs.

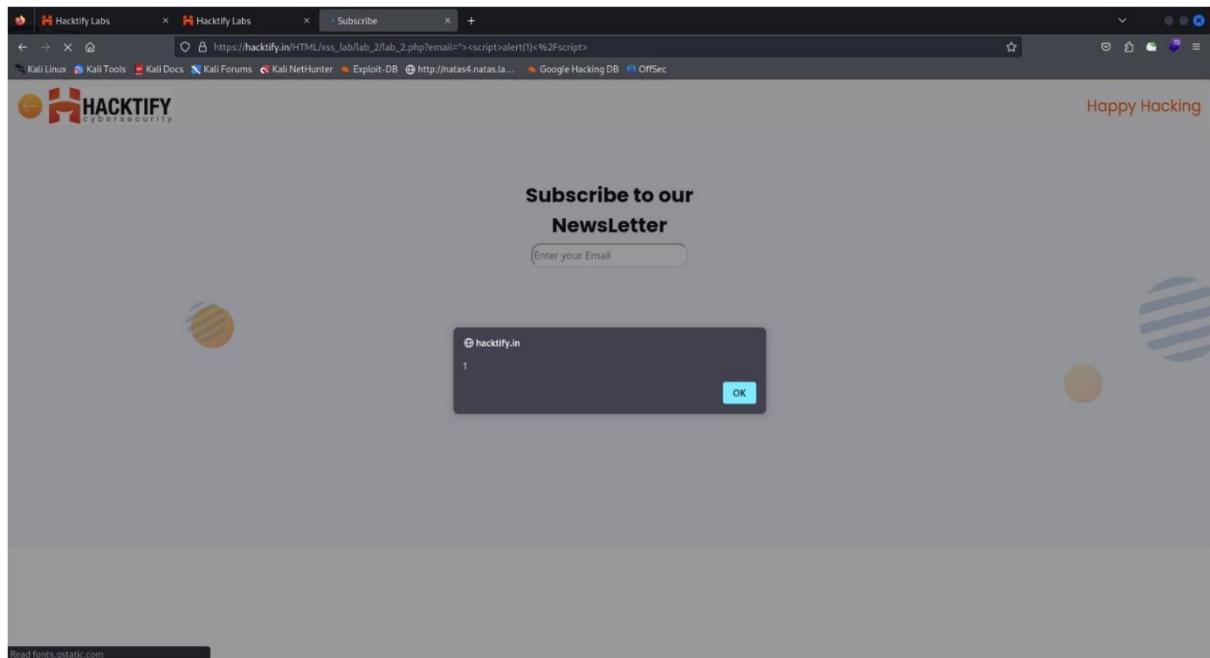
### Suggested Countermeasures

- Sanitize user inputs and encode special characters.
- Implement **output encoding** (e.g., `htmlspecialchars()` in PHP).
- Use **Content Security Policy (CSP)** to block inline JavaScript execution.

### References

OWASP XSS Prevention Cheat Sheet

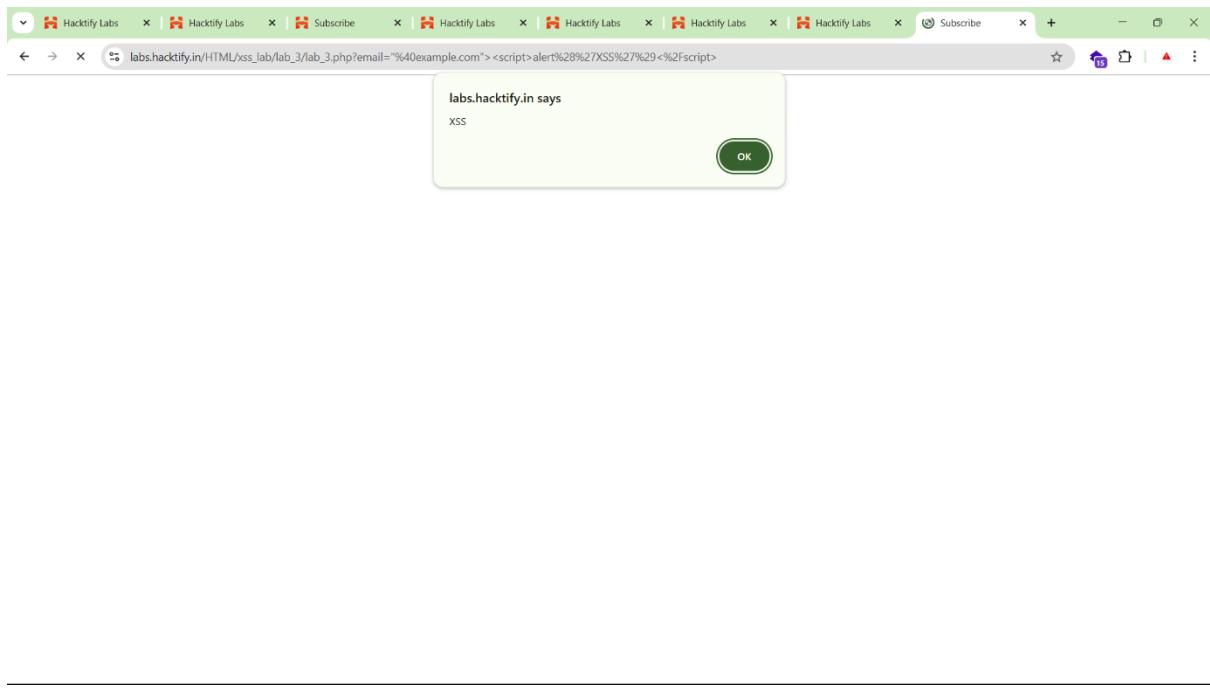
### Proof of Concept



## 2.3. XSS Is Everywhere!

Reference	Risk Rating
XSS Is Everywhere!	Low
Tools Used	
Burp Suite, Web Browser	
Vulnerability Description	
The application does not properly validate and sanitize user input in the <b>email</b> parameter of the search bar. By injecting JavaScript after breaking the expected email format, an attacker can execute <b>Reflected XSS</b> in the victim's browser.	
How It Was Discovered	
Manually tested by injecting the payload "@example.com"><script>alert('XSS')</script> into the <b>email</b> parameter and observing script execution.	
Vulnerable URLs	
<a href="https://hacktify.in/HTML/xss_lab/lab_3/lab_3.php?email=%22%40example.com%22%3E%3Cscript%3Ealert%28%27XSS%27%29%3C%2Fscript%3E">https://hacktify.in/HTML/xss_lab/lab_3/lab_3.php?email=%22%40example.com%22%3E%3Cscript%3Ealert%28%27XSS%27%29%3C%2Fscript%3E</a>	
Consequences of not Fixing the Issue	
<ul style="list-style-type: none"><li>-Attackers can execute JavaScript in the victim's browser.</li><li>-Users may be tricked into revealing sensitive data through phishing attacks.</li><li>-Attackers can steal session cookies and impersonate users.</li></ul>	
Suggested Countermeasures	
<b>Validate input:</b> Ensure only valid email formats are accepted.	
<b>Encode output:</b> Use <code>htmlspecialchars()</code> or similar encoding methods.	
<b>Implement CSP (Content Security Policy):</b> Block inline JavaScript execution.	
References	
OWASP XSS Prevention Cheat Sheet	

## Proof of Concept



### 2.4. Alternatives Are Must!

Reference	Risk Rating
Alternates Are Must!	Medium
Tools Used	
Burp Suite, Web Browser	
Vulnerability Description	
The application allows <b>unsanitized user input</b> in an input field, making it vulnerable to <b>Reflected XSS</b> . By injecting a JavaScript payload, an attacker can force victims to be redirected to a malicious website.	
How It Was Discovered	
Manually tested by injecting the payload "><script>window.location='http://example.com/'</script>" into an input field. Upon submission, the browser redirected to the specified URL.	
Vulnerable URLs	
<a href="https://hacktify.in/HTML/xss_lab/lab_4/lab_4.php">https://hacktify.in/HTML/xss_lab/lab_4/lab_4.php</a>	

## Consequences of not Fixing the Issue

**Phishing attacks:** Users can be redirected to fake login pages.

**Data theft:** Sensitive information can be harvested through social engineering.

**Session hijacking:** Attackers may steal cookies to impersonate users.

## Suggested Countermeasures

**Input validation:** Reject special characters like < > " '.

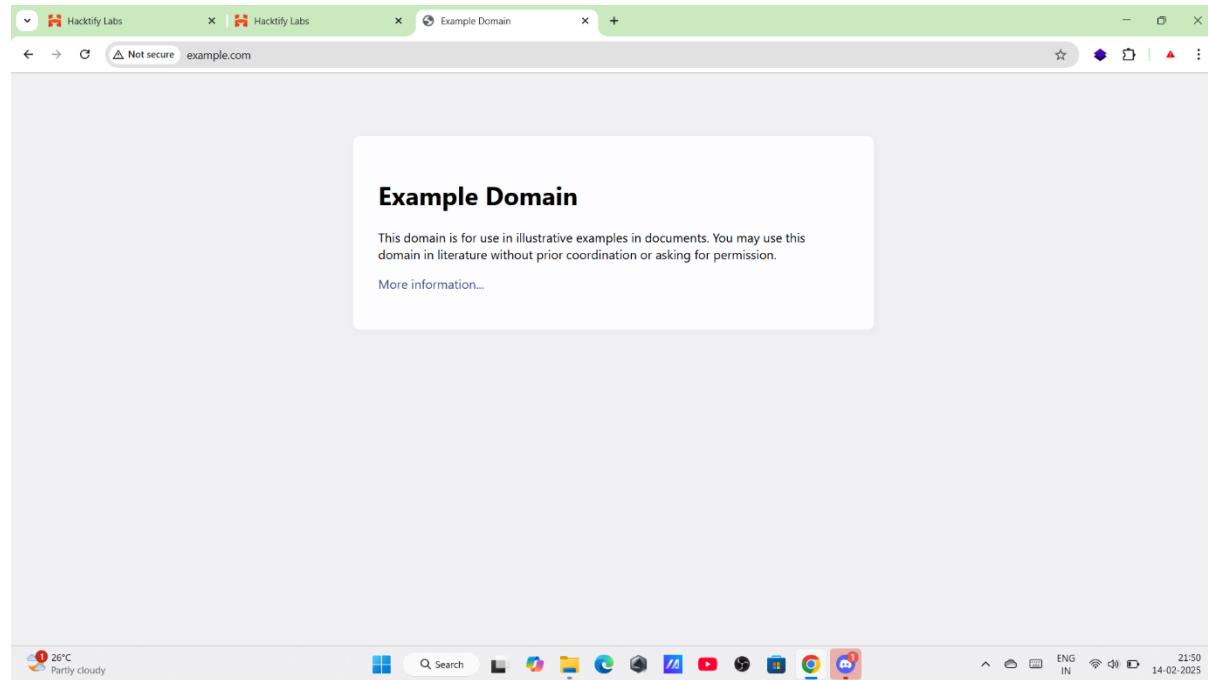
**Output encoding:** Encode user input before rendering in the DOM.

**CSP implementation:** Restrict inline JavaScript execution.

## References

OWASP XSS Prevention Cheat Sheet

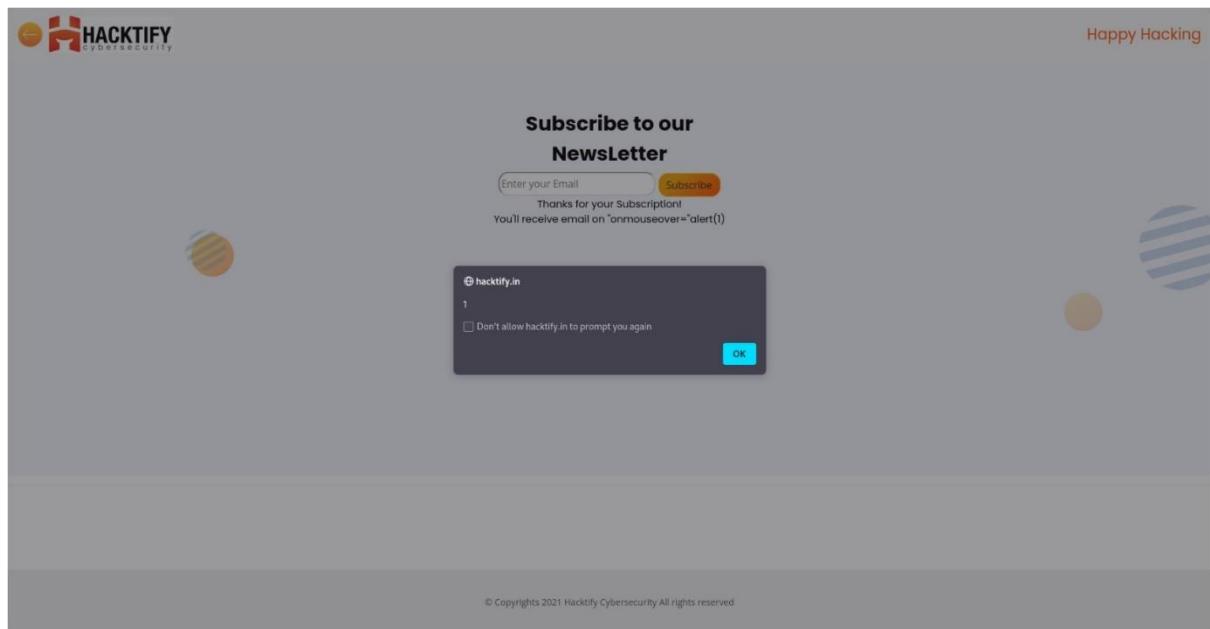
## Proof of Concept



## 2.5. Developer Hates Scripts!

Reference	Risk Rating
Developer Hates Scripts!	Hard
Tools Used	
Burp Suite	
Vulnerability Description	
The application does not properly sanitize input in the <b>search bar</b> , allowing attackers to inject <b>event-based XSS payloads</b> . This exploit triggers JavaScript execution when a user hovers over the injected element.	
How It Was Discovered	
Tested by injecting the <b>onmouseover</b> event into the search bar. When the result is displayed, hovering over it executed JavaScript.	
Vulnerable URLs	
<a href="https://hacktify.in/HTML/xss_lab/lab_5/lab_5.php">https://hacktify.in/HTML/xss_lab/lab_5/lab_5.php</a>	
Consequences of not Fixing the Issue	
<ul style="list-style-type: none"><li>-Attackers can execute JavaScript in users' browsers.</li><li>-Can be used for stealing cookies or session hijacking.</li><li>-Users may be tricked into performing unintended actions.</li></ul>	
Suggested Countermeasures	
<b>Input validation:</b> Restrict special characters and JavaScript event handlers.	
<b>Output encoding:</b> Ensure user input is rendered as text, not executable code.	
<b>CSP headers:</b> Block inline script execution with Content-Security-Policy.	
References	
<a href="#">OWASP XSS Prevention Cheat Sheet</a>	

## Proof of Concept



### 2.6. Change The Variation!

Reference	Risk Rating
Change The Variation!	<b>Hard</b>
Tools Used	
Burp Suite	
Vulnerability Description	
The application does not properly sanitize user input in the <b>input field</b> , allowing <b>image-based XSS injection</b> . The injected payload triggers JavaScript execution via the <b>onerror</b> event when the browser fails to load the image.	
How It Was Discovered	
Tested by injecting an image tag with an <b>onerror</b> event handler into the input field. Upon submission, the JavaScript executed as expected.	
Vulnerable URLs	
<a href="https://hacktify.in/HTML/xss_lab/lab_6/lab_6.php">https://hacktify.in/HTML/xss_lab/lab_6/lab_6.php</a>	

## Consequences of not Fixing the Issue

- Attackers can execute malicious JavaScript in the victim's browser.
- May lead to **session hijacking, data theft, or phishing attacks**.
- Can be used to inject fake UI elements for social engineering.

## Suggested Countermeasures

**Input validation:** Reject <script>, <img>, and other risky tags.

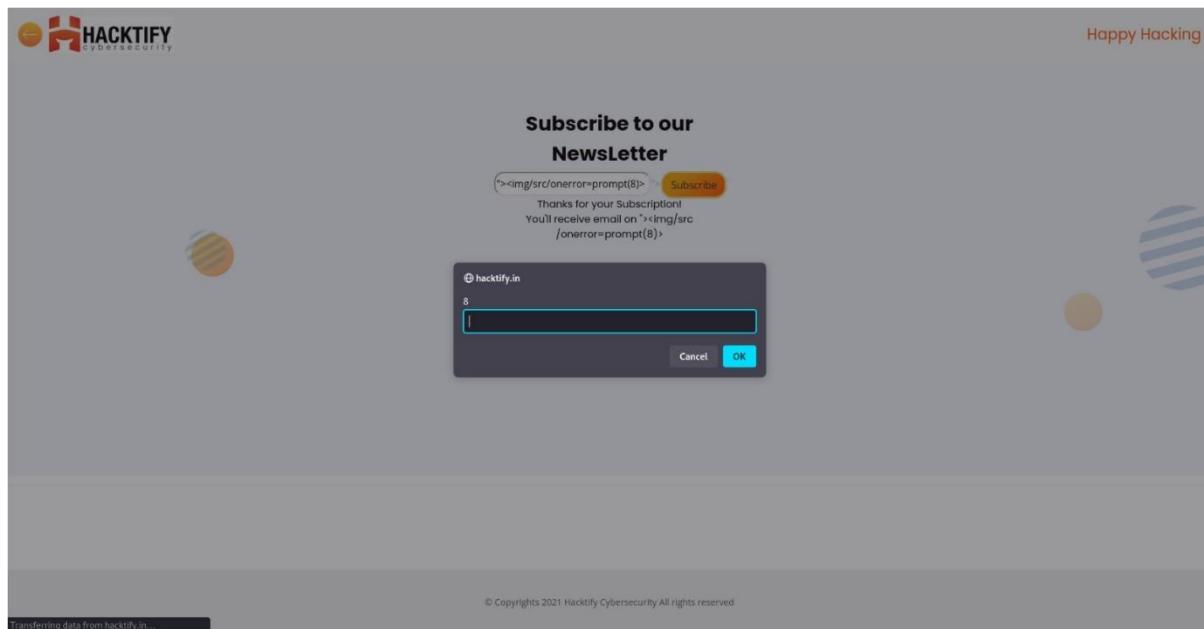
**Output encoding:** Ensure user input is properly escaped when displayed.

**Content Security Policy (CSP):** Restrict inline JavaScript execution..

## References

OWASP XSS Prevention Cheat Sheet

## Proof of Concept



## 2.7. Encoding Is The Key!

Reference	Risk Rating
Encoding Is The Key!	Medium

## Tools Used

Burp Suite, Web Browser

## Vulnerability Description

The application does not properly sanitize **URL-encoded input**, allowing an attacker to inject a **percent-encoded XSS payload**. When processed, the application decodes the input and executes the injected JavaScript.

## How It Was Discovered

Tested by injecting a **URL-encoded XSS payload**

(%3Cscript%3Ealert%280%29%3C%2Fscript%3E) into the input field. Upon submission, the browser executed the alert function.

## Vulnerable URLs

[https://hacktify.in/HTML/xss\\_lab/lab\\_7/lab\\_7.php](https://hacktify.in/HTML/xss_lab/lab_7/lab_7.php)

## Consequences of not Fixing the Issue

-Attackers can **bypass standard filtering** by using encoded payloads.

-**Stored or reflected XSS** may execute on the victim's browser.

-Can be used for **cookie theft, session hijacking, or phishing attacks**.

## Suggested Countermeasures

**Input validation:** Restrict and sanitize user input before processing.

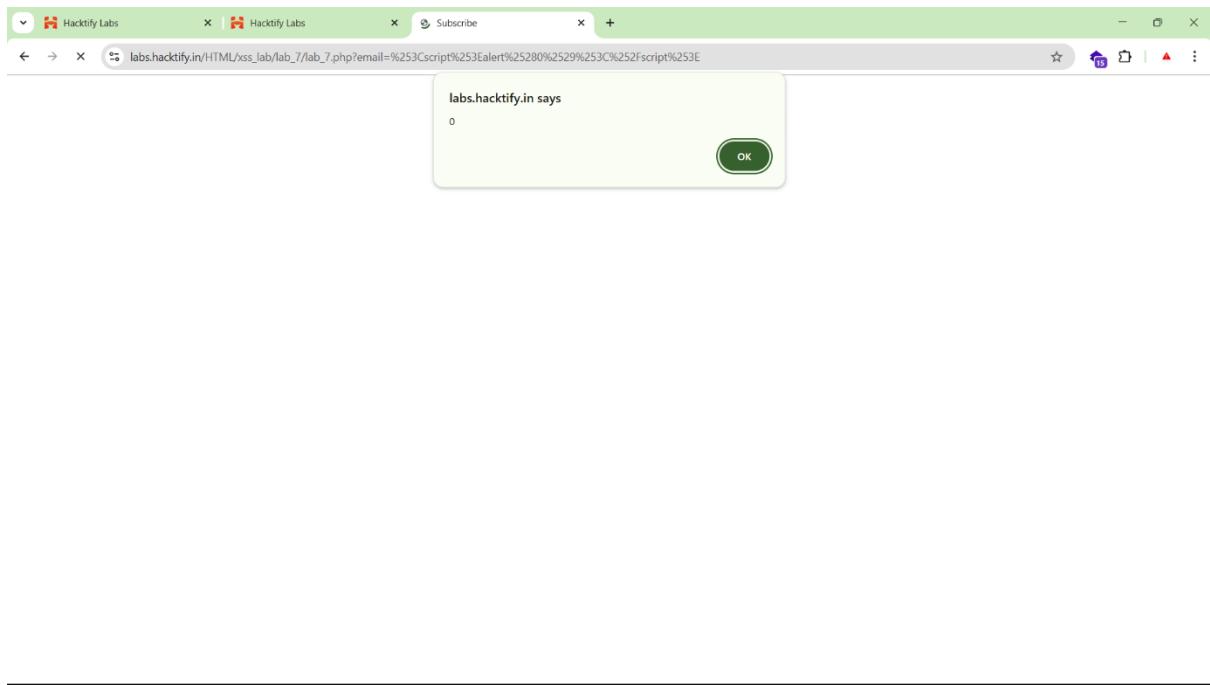
**Output encoding:** Use functions like htmlspecialchars() to prevent script execution.

**Implement CSP (Content Security Policy):** Block execution of inline scripts.

## References

OWASP XSS Prevention Cheat Sheet

## Proof of Concept



### 2.8. XSS With File Upload(File Name)

Reference	Risk Rating
XSS With File Upload (File Name)	Low
Tools Used	
Burp Suite	
Vulnerability Description	
The application allows unrestricted file uploads, permitting attackers to upload an <b>HTML file</b> containing malicious JavaScript. When a victim accesses the uploaded file, the JavaScript executes in their browser, leading to <b>Stored XSS</b> .	
How It Was Discovered	
By uploading an HTML file containing an <b>&lt;img&gt; tag with an onerror event</b> , which executes JavaScript when the browser fails to load the image.	
Vulnerable URLs	

[https://hacktify.in/HTML/xss\\_lab/lab\\_8/lab\\_8.php](https://hacktify.in/HTML/xss_lab/lab_8/lab_8.php)

### Consequences of not Fixing the Issue

**Stored XSS:** Every user accessing the uploaded file will execute the malicious script.

**Session hijacking:** Attackers can steal cookies and impersonate users.

**Phishing:** Users can be redirected to malicious sites.

### Suggested Countermeasures

**Restrict file types:** Only allow safe extensions like .png, .jpg, and .pdf.

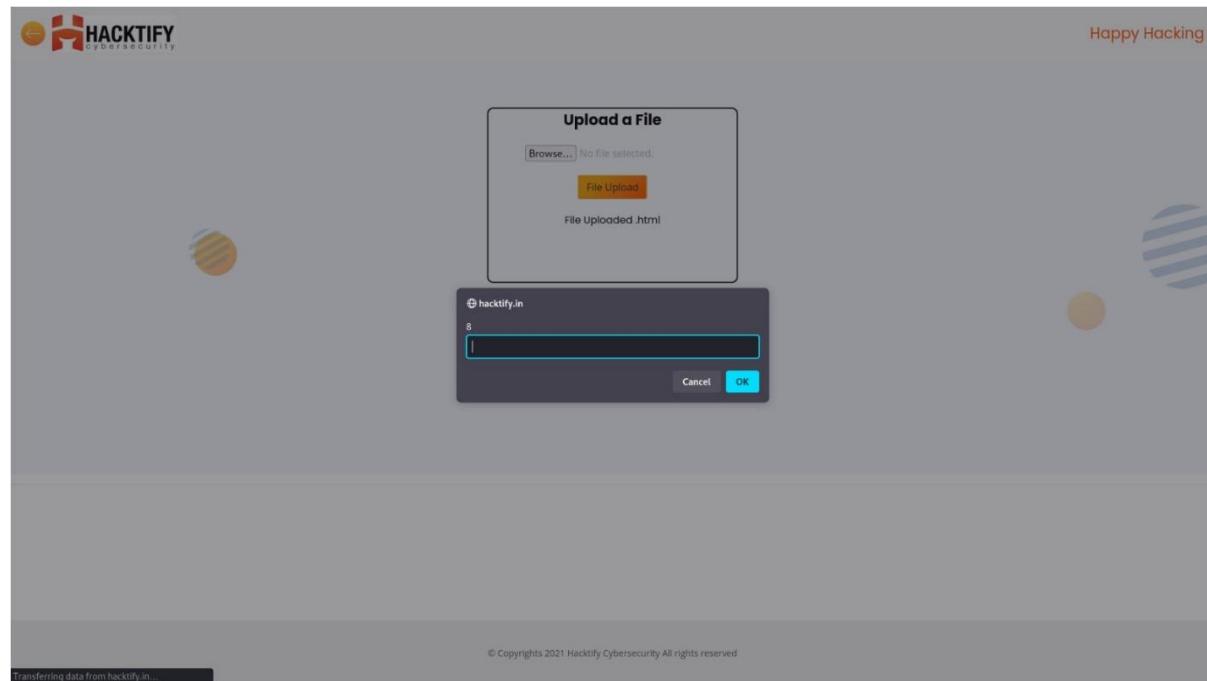
**Validate file contents:** Use **MIME type validation** to ensure the uploaded file matches its expected type.

**Deny execution:** Serve uploaded files with Content-Disposition: attachment headers to prevent execution.

### References

OWASP Unrestricted File Upload Guide

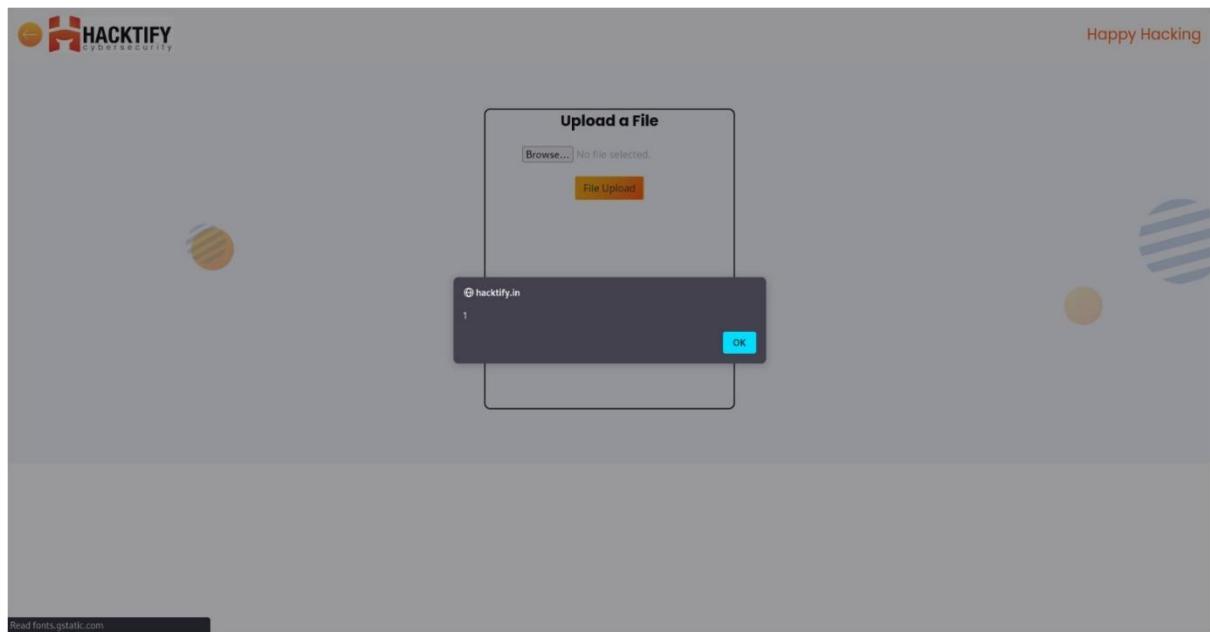
## Proof of Concept



## 2.9. XSS With File Upload (File Content)

Reference	Risk Rating
XSS With File Upload (File Content)	Medium
Tools Used	
Burp Suite	
Vulnerability Description	
<p>The application allows unrestricted .txt file uploads but fails to sanitize file content. If the server renders text files as <b>HTML instead of plain text</b>, an attacker can inject JavaScript into a .txt file, leading to <b>Stored XSS</b> when a victim views the file in a browser.</p>	
How It Was Discovered	
<p>By uploading a .txt file containing an XSS payload. When accessed in a browser, the JavaScript executed instead of being displayed as plain text.</p>	
Vulnerable URLs	
<a href="https://hacktify.in/HTML/xss_lab/lab_9/lab_9.php">https://hacktify.in/HTML/xss_lab/lab_9/lab_9.php</a>	
Consequences of not Fixing the Issue	
<p><b>Stored XSS:</b> The script executes every time a user opens the file.</p>	
<p><b>Session hijacking:</b> Attackers can steal cookies and impersonate users.</p>	
<p><b>Phishing:</b> Users may be tricked into executing malicious scripts.</p>	
Suggested Countermeasures	
<p><b>Restrict file types:</b> Prevent .txt files from being served as HTML.</p>	
<p><b>Set correct MIME types:</b> Ensure text files are served with Content-Type: text/plain instead of text/html.</p>	
<p><b>Sanitize user uploads:</b> Escape &lt;script&gt; and other dangerous HTML tags in uploaded files.</p>	
References	
<a href="#">OWASP Unrestricted File Upload Guide</a>	

## Proof of Concept



### 2.10. Stored Everywhere!

Reference	Risk Rating
Stored Everywhere!	Low
Tools Used	
Burp Suite	
Vulnerability Description	
The application <b>fails to sanitize user input</b> during registration, allowing <b>Stored XSS</b> . When a user registers with a malicious payload, the script gets stored in the database and executes when viewing the profile page.	
How It Was Discovered	
Tested by injecting a <b>JavaScript payload</b> in the registration form. Upon accessing the profile page, the script executed, confirming <b>Stored XSS</b> .	
Vulnerable URLs	
<a href="https://hacktify.in/HTML/xss_lab/lab_10/lab_10.php">https://hacktify.in/HTML/xss_lab/lab_10/lab_10.php</a>	

## Consequences of not Fixing the Issue

**Persistent attack vector:** The malicious script remains stored and executes every time the profile page is viewed.

**Session hijacking:** Attackers can steal user cookies and impersonate victims.

**Phishing & social engineering:** Users can be tricked into revealing sensitive information.

## Suggested Countermeasures

**Input validation:** Reject < > " ' & characters in user input.

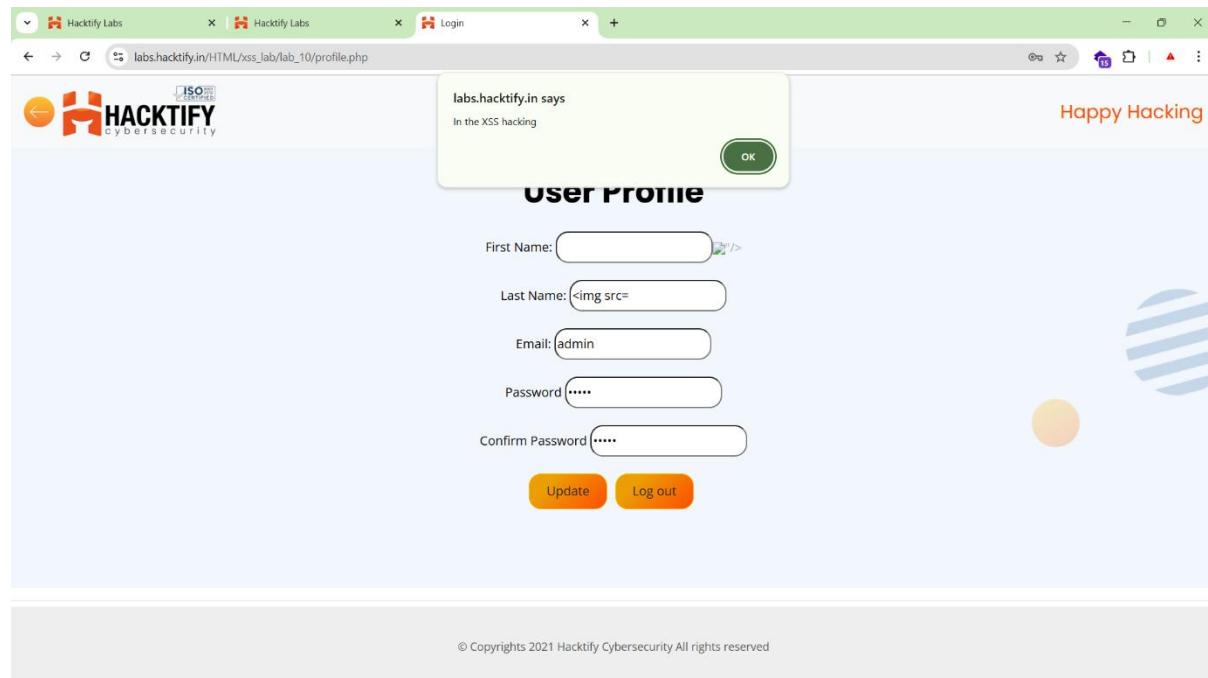
**Output encoding:** Use htmlspecialchars() to encode stored data before rendering.

**Content Security Policy (CSP):** Restrict inline JavaScript execution.

## References

OWASP XSS Prevention Cheat Sheet

## Proof of Concept



## 2.11. DOM's Are Love!

Reference	Risk Rating
DOM's Are Love!	Hard
Tools Used	
Burp Suite	
Vulnerability Description	
<p>The <b>name</b> parameter in dom.js is <b>directly used in the DOM</b> without proper sanitization, making the application vulnerable to <b>DOM-Based XSS</b>. This allows attackers to inject and execute JavaScript within the victim's browser when they visit a crafted URL.</p>	
How It Was Discovered	
<p>By viewing the <b>page source</b>, it was observed that the name parameter is dynamically handled in dom.js. Injecting a <b>URL-encoded script payload</b> resulted in execution within the DOM.</p>	
Vulnerable URLs	
<p><a href="https://hacktify.in/HTML/xss_lab/lab_11/lab_11.php?name=%3Cscript%3Ealert(1)%3C/script%3E">https://hacktify.in/HTML/xss_lab/lab_11/lab_11.php?name=%3Cscript%3Ealert(1)%3C/script%3E</a></p>	
Consequences of not Fixing the Issue	
<p><b>Client-side script execution:</b> Allows attackers to execute arbitrary JavaScript.</p>	
<p><b>Session hijacking:</b> Attackers can steal cookies using document.cookie.</p>	
<p><b>Phishing attacks:</b> Users can be redirected to malicious sites.</p>	
Suggested Countermeasures	
<p><b>Sanitize input:</b> Use JavaScript sanitization libraries like DOMPurify.</p>	
<p><b>Avoid innerHTML:</b> Use textContent instead to prevent script execution.</p>	
<p><b>Implement CSP (Content Security Policy):</b> Restrict inline JavaScript execution.</p>	
References	
<p>OWASP DOM-Based XSS Prevention Guide</p>	

## Proof of Concept

