Penetration Testing Report

Full Name: Mantone Malikhetla

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}	{2}	{5}

High	4	-	**Number of Sub-labs with hard difficulty level**
Medium	2	-	**Number of Sub-labs with Medium difficulty level**
Low	5	-	**Number of Sub-labs with Easy difficulty level**

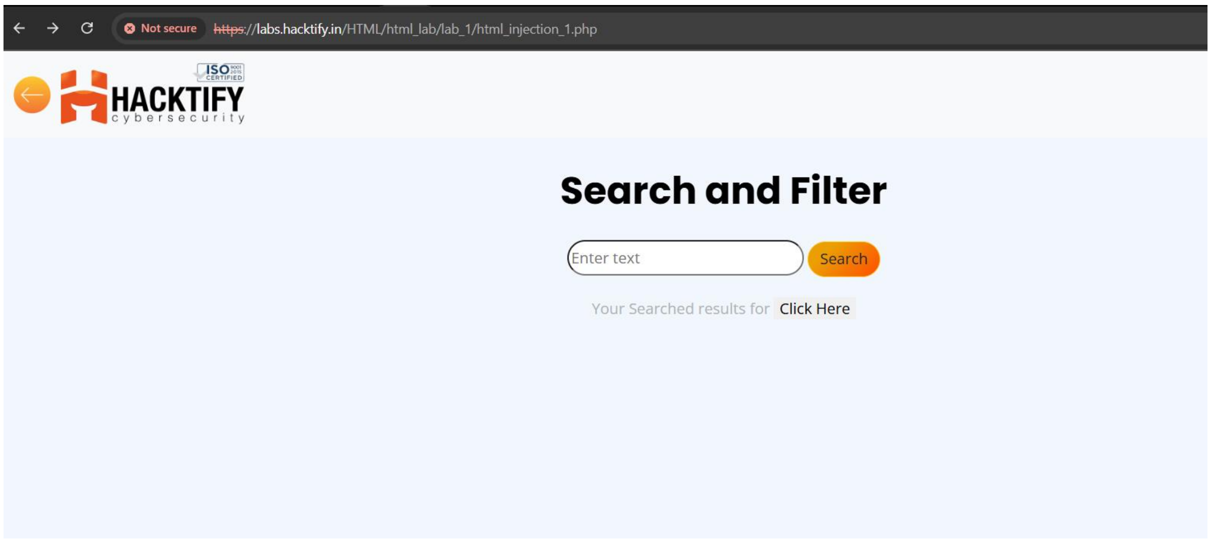
1. {Lab 1 HTML Injection}

1.1. {Sub-lab-1 HTML’s are easy}

Reference	Risk Rating
{Sub-lab-1 HTM’s are easy}	**Low**
Tools Used	
Browser	
Vulnerability Description	
Stored or Reflected HTML Injection vulnerability, where an attacker can inject raw HTML elements, such as buttons or other interactive elements, into the webpage. This injected button executes JavaScript, redirecting users to an arbitrary website when clicked. If an application fails to properly sanitize user input, attackers can manipulate the page’s HTML structure and execute unintended behaviors	
How It Was Discovered	
Automated Tools / Manual Analysis <button onclick="window.location.href='https://www.google.com'">Click Here</button>	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_1/html_injection_1.php	
Consequences of not Fixing the Issue	
Phishing Attacks Forced Redirects to Malicious Websites Defacement of Web Pages Session Hijacking or Data Theft	
Suggested Countermeasures	
Input Sanitization HTML Encoding Before Rendering Use Content Security Policy Disable Dangerous Attributes in Inputs eg onclick	
References	
https://www.imperva.com/learn/application-security/html-injection/ https://www.invicti.com/learn/html-injection/	

Proof of Concept

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


Not secure
https://labs.hacktify.in/HTML/html_lab/lab_1/html_injection_1.php
HACKTIFY cybersecurity
Search and Filter
Enter text
Search
Your Searched results for Click Here

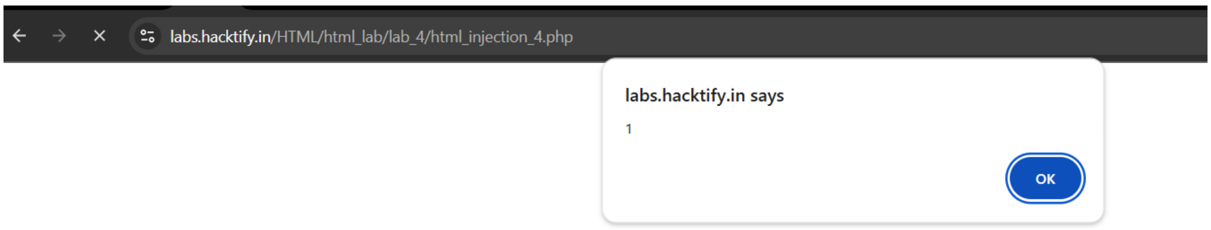
1.4. {Sub-lab-4 File Content and HTML Injection a perfect pair!}

Reference	Risk Rating
{Sub-lab-4 File Content and HTML Injection a perfect pair!}	**Low**
Tools Used	
Browser	
Vulnerability Description	
A HTML injection via file uploads that are later interpreted by the browser. User input is directly written to a file, and the file is later served with a text/html MIME type, leading to execution of injected content. If an application allows users to upload HTML or text files, and later renders them as HTML pages, this allows HTML injection. If input from a form is stored in a .html, .txt, or .php file, it can be served as a webpage. Improper Content-Type Handling If the uploaded file is stored as .txt but later served as text/html, HTML will be interpreted and executed by the browser. If the application allows arbitrary file extensions like .html, .shtml, or .php, attackers can upload script-containing files that get executed.	
How It Was Discovered	
Manual Analysis - <script>alert(1)</script>.jpg	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_4/html_injection_4.php	
Consequences of not Fixing the Issue	
Fake Login Forms for Phishing Automatic JavaScript Execution (Stored XSS) Redirecting Users to Malicious Sites	
Suggested Countermeasures	
Serve Uploaded Files with Safe MIME Types Sanitize User Input	
References	

https://www.imperva.com/learn/application-security/html-injection/ https://www.invicti.com/learn/html-injection/

Proof of Concept

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


labs.hacktify.in/HTML/html_lab/lab_4/html_injection_4.php

labs.hacktify.in says

1

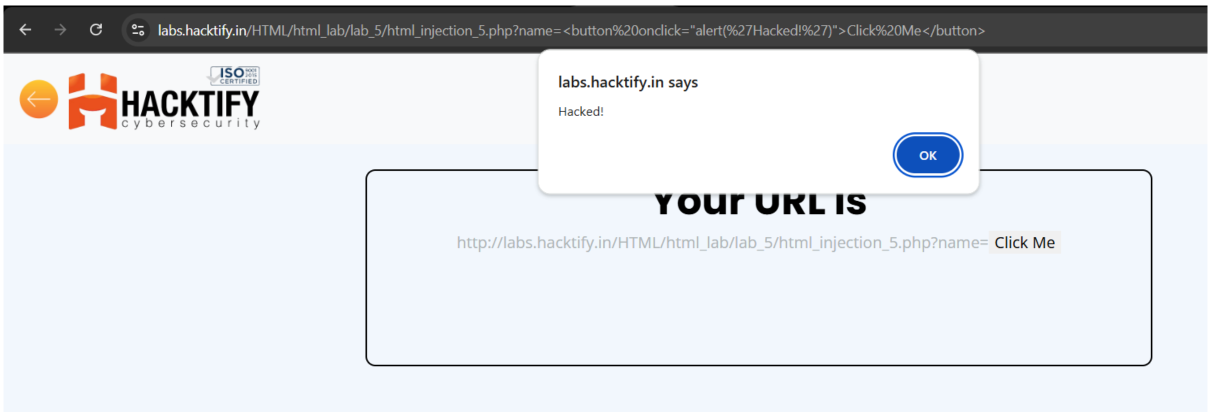
1.5. {Sub-lab-5 Injecting HTML using URL }

Reference	Risk Rating
{Sub-lab-5 Injecting HTML Using URL}	**Low**
Tools Used	
Browser	
Vulnerability Description	
HTML Injection via URL parameter vulnerability, allowing attackers to inject HTML elements dynamically by manipulating the name parameter in the URL. This works because the application directly renders user-supplied input without sanitization, allowing HTML elements (like <button>) to be interpreted by the browser.	
How It Was Discovered	
Automated Tools / Manual Analysis - https://labs.hacktify.in/HTML/html_lab/lab_5/html_injection_5.php?name=<button onclick="alert('Hacked!')">Click Me</button>	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_5/html_injection_5.php	
Consequences of not Fixing the Issue	
Phishing Attacks Forced Redirects to Malicious Websites Defacement of Web Pages JavaScript-Based Attacks	
Suggested Countermeasures	
Input Sanitization HTML Encoding Use libraries like DOMPurify	

Server-Side Filtering
References
https://www.imperva.com/learn/application-security/html-injection/ https://www.invicti.com/learn/html-injection/

Proof of Concept

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


labs.hacktify.in/HTML/html_lab/lab_5/html_injection_5.php?name=<button%20onclick="alert(%27Hacked!%27)">Click%20Me</button>

labs.hacktify.in says

Hacked!

Your URL is

http://labs.hacktify.in/HTML/html_lab/lab_5/html_injection_5.php?name= Click Me

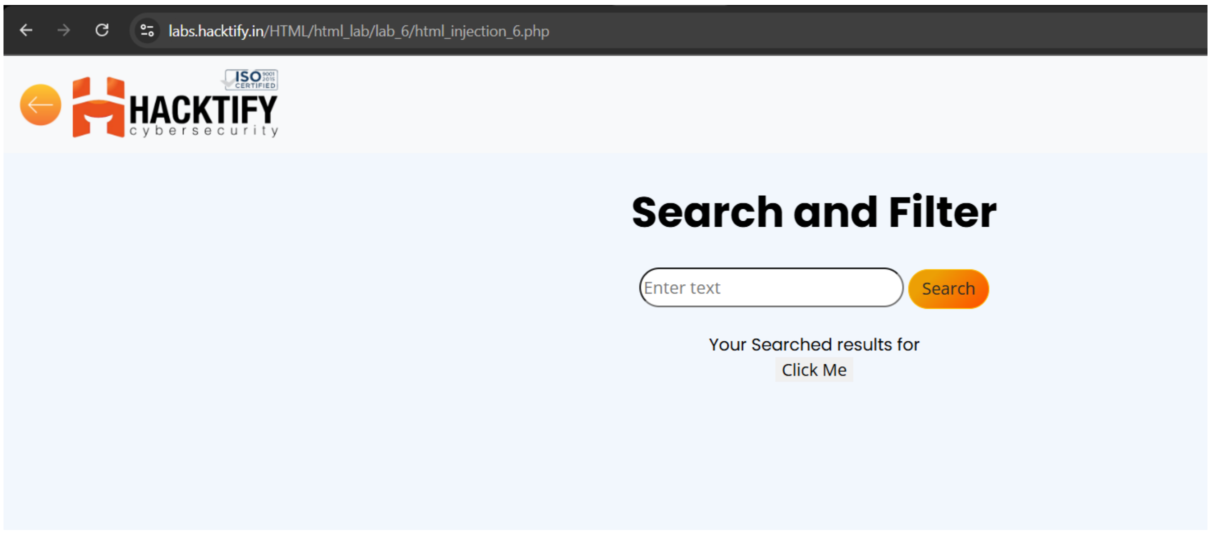
1.6. {Sub-lab-6 Encode IT!}

Reference	**Risk Rating**
{Sub-lab-6 Encode IT!}	**Low**
Tools Used	
Browser	
Vulnerability Description	
HTML Injection vulnerability with URL encoding, where an attacker can inject encoded HTML/JavaScript payloads into the application. This is because the application fails to decode and sanitize user input properly, allowing HTML to be interpreted by the browser after decoding.	
How It Was Discovered	
Manual Analysis - %3C%2Fh2%3E%3Cbutton%20onclick%3D%22alert%28%27Hacked%21%27%29%22%3EClick%20Me%3C%2Fbutton%3E	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_6/index.php	
Consequences of not Fixing the Issue	
Phishing Attacks Forced Redirects to Malicious Websites Defacement of Web Pages	
Suggested Countermeasures	
Input Sanitization	

HTML Encoding Before Rendering Use Content Security Policy
References
https://www.imperva.com/learn/application-security/html-injection/ https://www.invicti.com/learn/html-injection/

Proof of Concept

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


labs.hacktify.in/HTML/html_lab/lab_6/html_injection_6.php
HACKTIFY cybersecurity
Search and Filter
Enter text
Search
Your Searched results for
Click Me

2. {Lab 2 Cross Site Scripting}

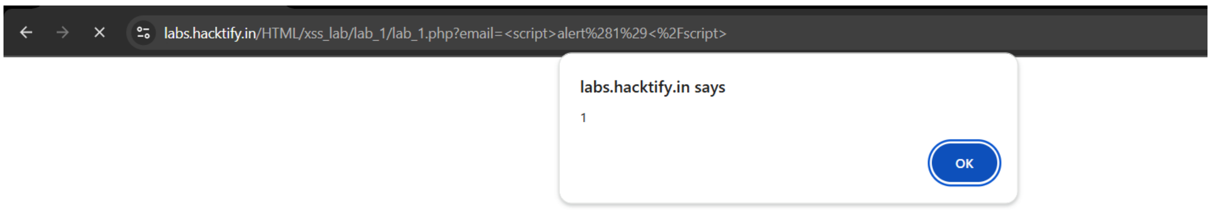
2.1. {Sub-lab-1 Let's Do IT!}

Reference	Risk Rating
{ Sub-lab-1 Let's Do IT!}	**Medium**
Tools Used	
Browser	
Vulnerability Description	
Reflected Cross-Site Scripting (XSS) is a web security vulnerability where an attacker injects malicious JavaScript into a website, which is then immediately reflected back to the victim without being stored on the server. The attack typically occurs when user input is included in the webpage without proper validation or sanitization. The application directly reflects user input into the page's response without sanitization.	

Attackers craft a malicious URL containing an XSS payload, when a victim clicks the link, the injected <script> executes in their browser.
How It Was Discovered
Manual Analysis – using <script>alert(1)</script> in the URL
Vulnerable URLs
https://labs.hacktify.in/HTML/xss_lab/lab_1/lab_1.php
Consequences of not Fixing the Issue
Session Hijacking - Attackers can steal session cookies using document.cookie, allowing them to impersonate users, including admins. Phishing Attacks - Users can be tricked into entering credentials on a fake login form injected into the site.
Suggested Countermeasures
Input Validation & Output Encoding Use Content Security Policy
References
https://www.invicti.com/learn/cross-site-scripting-xss/ https://owasp.org/www-community/attacks/xss/ 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



labs.hacktify.in/HTML/xss_lab/lab_1/lab_1.php?email=<script>alert%281%29<%2Fscript>

labs.hacktify.in says

1

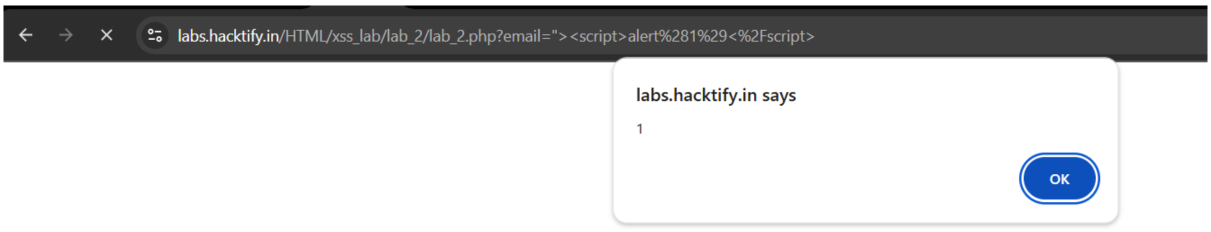
2.2. { Sub-lab-2 Balancing is Important in Life}

Reference	**Risk Rating**
{ Sub-lab-2 Balancing is Important in Life}	**Medium**
Tools Used	
Browser	
Vulnerability Description	
Reflected Cross-Site Scripting (XSS) due to improper handling of user input within an HTML attribute or tag. The key reason a payload like "><script>alert(1)</script> works is because the application fails to properly sanitize or encode user input before inserting it into the HTML response. The application injects user input directly into an HTML attribute or element without escaping special characters.	

">: Closes an existing attribute or tag. <script>alert(1)</script>: Injects a new <script> tag that executes alert(1). When the page loads, the browser interprets the script as part of the legitimate HTML structure and executes it
How It Was Discovered
Manual Analysis - "><script>alert(1)</script>
Vulnerable URLs
https://labs.hacktify.in/HTML/xss_lab/lab_2/lab_2.php
Consequences of not Fixing the Issue
Session Hijacking - Attackers can steal session cookies using document.cookie, allowing them to impersonate users, including admins. Phishing Attacks - Users can be tricked into entering credentials on a fake login form injected into the site.
Suggested Countermeasures
Input Validation & Output Encoding Use Content Security Policy (CSP) Sanitize User Input Before Processing
References
https://www.invicti.com/learn/cross-site-scripting-xss/ https://owasp.org/www-community/attacks/xss/ 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


labs.hacktify.in/HTML/xss_lab/lab_2/lab_2.php?email="><script>alert%281%29<%2Fscript>

labs.hacktify.in says

1

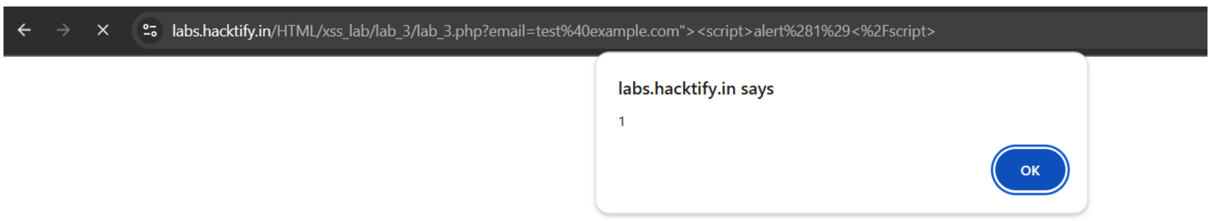
2.3. { Sub-lab-3 XSS is everywhere}

Reference	**Risk Rating**
{Sub-lab-3 XSS is everywhere}	**Medium**

Tools Used
Browser
Vulnerability Description
Reflected Cross-Site Scripting (XSS) vulnerability, where user-supplied input is reflected in the webpage without proper sanitization or encoding. This allows an attacker to inject arbitrary JavaScript code into the page, which executes when the victim loads the manipulated URL. The page loads user input from the URL and directly places it in the response. Injecting a script after a valid email since site expects an email format. This allows client-side validation bypass which tricks the validation while still injecting JavaScript.
How It Was Discovered
Manual Analysis - test@example.com"><script>alert(1)</script>
Vulnerable URLs
https://labs.hacktify.in/HTML/xss_lab/lab_3/lab_3.php
Consequences of not Fixing the Issue
Session Hijacking - Attackers can steal session cookies using document.cookie, allowing them to impersonate users, including admins. Phishing Attacks - Users can be tricked into entering credentials on a fake login form injected into the site.
Suggested Countermeasures
Input Validation & Output Encoding Use Content Security Policy (CSP) Sanitize User Input Before Processing
References
https://www.invicti.com/learn/cross-site-scripting-xss/ https://owasp.org/www-community/attacks/xss/ 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



labs.hacktify.in/HTML/xss_lab/lab_3/lab_3.php?email=test%40example.com"><script>alert%281%29<%2Fscript>

labs.hacktify.in says

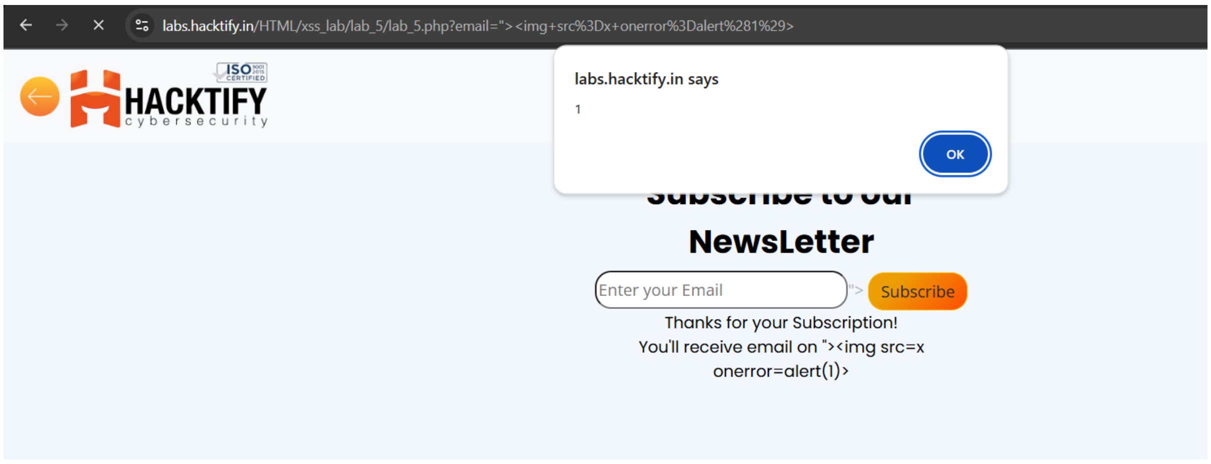
1

2.5. { Sub-lab-5 Developer hates scripts!}

Reference	Risk Rating
{Sub-lab-5 Developer hates scripts!}	**Medium**
Tools Used	
Browser	
Vulnerability Description	
Reflected Cross-Site Scripting (XSS) vulnerability, where user-supplied input is reflected in the webpage without proper sanitization or encoding. This allows an attacker to inject arbitrary HTML and JavaScript code, which executes when the victim loads the manipulated URL. The application dynamically inserts user input into the HTML structure without escaping special characters, if the input is not sanitized, an attacker can insert HTML tags like <img>, which will be directly interpreted by the browser. Since <script> has been blacklisted other tags that have not been blocked like <img> can be used. <img> tags allow the onerror attribute, which executes JavaScript when the image fails to load	
How It Was Discovered	
Manual Analysis - "><img src=x onerror=alert(1)>	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_5/lab_5.php	
Consequences of not Fixing the Issue	
Session Hijacking - Attackers can steal session cookies using document.cookie, allowing them to impersonate users, including admins. Phishing Attacks - Users can be tricked into entering credentials on a fake login form injected into the site.	
Suggested Countermeasures	
Escape User Input Properly Use Secure JavaScript DOM Manipulation Implement Content Security Policy	
References	
https://www.invicti.com/learn/cross-site-scripting-xss/ https://owasp.org/www-community/attacks/xss/ 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


```
labs.hacktify.in/HTML/xss_lab/lab_5/lab_5.php?email="><img+src%3Dx+onerror%3Dalert%281%29>
```

labs.hacktify.in says

1

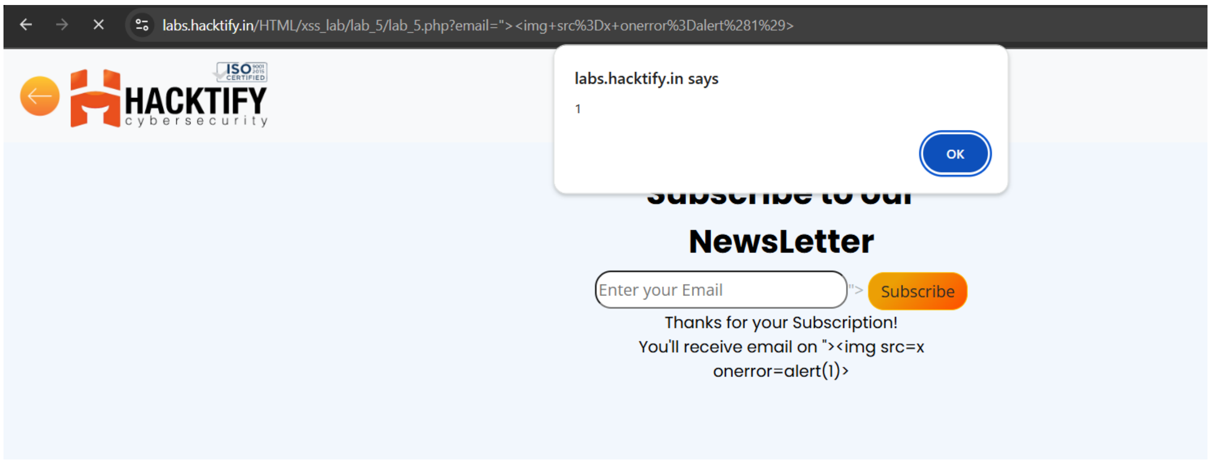
Thanks for your Subscription!
You'll receive email on "><img src=x onerror=alert(1)>

2.6. { Sub-lab-6 Change the variation}

Reference	**Risk Rating**
{Sub-lab-6 Change the variation}	**Medium**
Tools Used	
Browser	
Vulnerability Description	
Reflected Cross-Site Scripting (XSS) vulnerability, where user-supplied input is reflected in the webpage without proper sanitization or encoding. This allows an attacker to inject arbitrary HTML and JavaScript code, which executes when the victim loads the manipulated URL. The application dynamically inserts user input into the HTML structure without escaping special characters, if the input is not sanitized, an attacker can insert HTML tags like <img>, which will be directly interpreted by the browser. Since <script> has been blacklisted other tags that have not been blocked like <img> can be used. <img> tags allow the onerror attribute, which executes JavaScript when the image fails to load	
How It Was Discovered	
Manual Analysis - "><img src=x onerror=alert(1)>	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_6/lab_6.php	
Consequences of not Fixing the Issue	
Session Hijacking - Attackers can steal session cookies using document.cookie, allowing them to impersonate users, including admins. Phishing Attacks - Users can be tricked into entering credentials on a fake login form injected into the site.	
Suggested Countermeasures	
Escape User Input Properly Use Secure JavaScript DOM Manipulation Implement Content Security Policy	
References	
https://www.invicti.com/learn/cross-site-scripting-xss/ https://owasp.org/www-community/attacks/xss/ 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


labs.hacktify.in/HTML/xss_lab/lab_5/lab_5.php?email="><img+src%3Dx+onerror%3Dalert%281%29>

labs.hacktify.in says

1

Thanks for your Subscription!
You'll receive email on "><img src=x onerror=alert(1)>

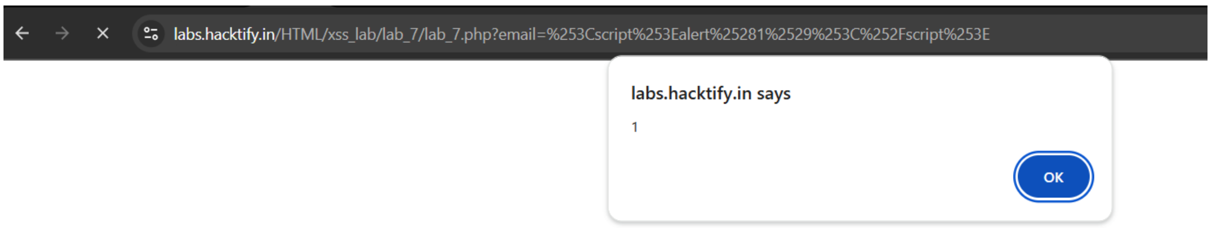
2.7. {Sub-lab-7 Encoding is key?}

Reference	Risk Rating
{Sub-lab-7 Encoding is key?}	**Medium**
Tools Used	
Browser	
Vulnerability Description	
Reflected Cross-Site Scripting (XSS) vulnerability, where user input is improperly processed and later reflected in the response without proper encoding or sanitization. The application prevents <script> injection but fails to account for encoded payloads, allowing attackers to bypass filtering mechanisms by using URL encoding. The application fails to properly encode special characters like <, >, and " before rendering them into the page. By URL encoding the payload, can trick the application into accepting and executing the script.	
How It Was Discovered	
Manual Analysis - %3Cscript%3Ealert%281%29%3C%2Fscript%3E	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_7/lab_7.php	
Consequences of not Fixing the Issue	
Session Hijacking - Attackers can steal session cookies using document.cookie, allowing them to impersonate users, including admins. Phishing Attacks - Users can be tricked into entering credentials on a fake login form injected into the site.	
Suggested Countermeasures	
Escape User Input Properly Use Secure DOM Manipulation	

Implement Content Security Policy Validate and Sanitize input
References
https://www.invicti.com/learn/cross-site-scripting-xss/ https://owasp.org/www-community/attacks/xss/ 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



labs.hacktify.in/HTML/xss_lab/lab_7/lab_7.php?email=%253Cscript%253Ealert%25281%2529%253C%252Fscript%253E

labs.hacktify.in says

1

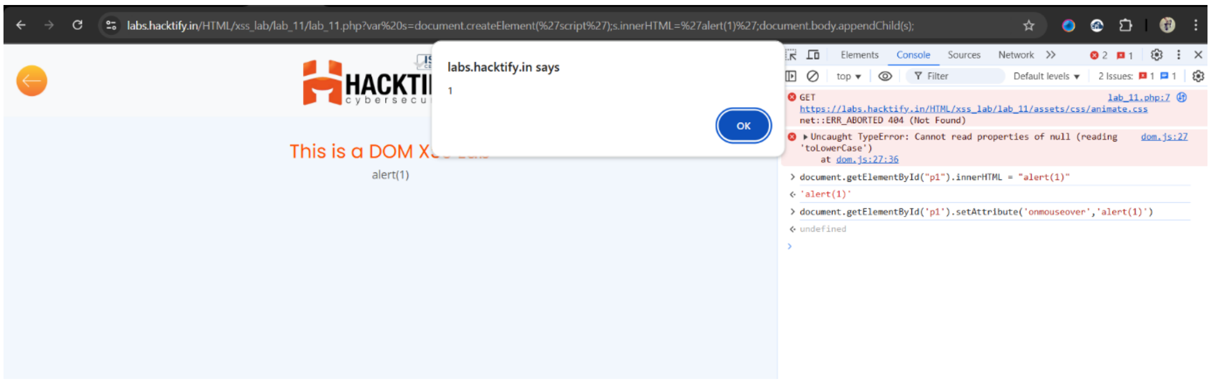
2.11. { Sub-lab-11 DOMs are Love!}

Reference	**Risk Rating**
{Sub-lab-11 DOMs are love!}	**Medium**
Tools Used	
Browser – Developer Tools - Console	
Vulnerability Description	
The vulnerability is a Reflected Cross-Site Scripting (XSS). This occurs when JavaScript processes user input from an untrusted source (like a URL parameter) and directly modifies the DOM without proper sanitization. The application retrieves user-controlled input from the UR then improperly handles it in the JavaScript context without sanitization. This enables JavaScript injection via document.createElement('script'), which the browser executes as an inline script.	
How It Was Discovered	
var%20s=document.createElement('script');s.innerHTML='alert(1)';document.body.appendChild(s);	
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 session cookies using document.cookie, allowing them to impersonate users, including admins.	

Phishing Attacks - Users can be tricked into entering credentials on a fake login form injected into the site.
Suggested Countermeasures
Avoid Insecure JavaScript Functions Validate and Filter User Input Implement Content Security Policy
References
https://www.invicti.com/learn/cross-site-scripting-xss/ https://owasp.org/www-community/attacks/xss/ 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



```
labs.hacktify.in/HTML/xss_lab/lab_11/lab_11.php?var%20s=document.createElement(%27script%27);s.innerHTML=%27alert(1)%27;document.body.appendChild(s);
```

labs.hacktify.in says

1

This is a DOM X

alert(1)

```
GET https://labs.hacktify.in/HTML/xss_lab/lab_11/assets/css/animate.css      lab_11.php:7
net::ERR_ABORTED 404 (Not Found)
▸Uncaught TypeError: Cannot read properties of null (reading                 dom.js:27
'toLowerCase')
    at dom.js:27:36
> document.getElementById("p1").innerHTML = "alert(1)"
< 'alert(1)'
> document.getElementById('p1').setAttribute('onmouseover','alert(1)')
< undefined
>
```