

# Penetration Testing Report

Full Name: G M Sohanur Rahman

Program: HCPT

Date: 22 Feb, 2024

## 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	{Clickjacking}, {HTML Injection}
------------------	----------------------------------

## 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
{1}	{3}	{4}

- High - Encode IT!
- Medium - File Content And HTML Injection A Perfect Pair!, Injecting HTML Using URL!, Re-Hijack!

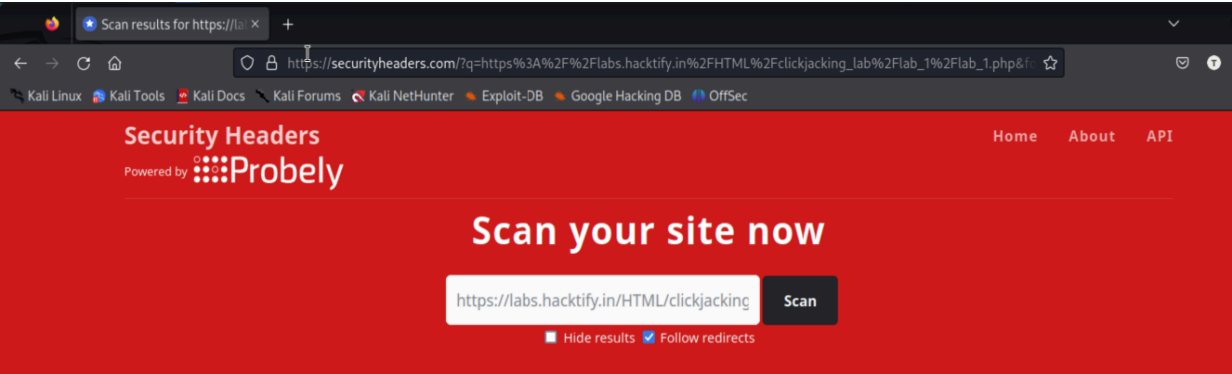
Low - HTML's Are Easy!, Let Me Store Them!, File Names Are Also Vulnerable!,  
,Let's Hijack!

## 1. {Clickjacking}

### 1.1. {Let's Hijack!}

Reference	Risk Rating
{Let's hijack}	Low
<b>Tools Used</b>	
Security headers	
<b>Vulnerability Description</b>	
Clickjacking, often called a "UI redress attack," is a type of attack where a user is deceived into interacting with a different website than intended. This is achieved by overlaying the genuine web page with transparent or opaque layers, making the user believe they are clicking on the visible page's elements, while in reality, they click on elements from a hidden page. Essentially, it's a technique where attackers manipulate website interfaces to trick users into performing unintended actions on another site.	
<b>How It Was Discovered</b>	
Automated Tools	
<b>Vulnerable URLs</b>	
<a href="https://labs.hacktify.in/HTML/clickjacking_lab/lab_1/lab_1.php">https://labs.hacktify.in/HTML/clickjacking_lab/lab_1/lab_1.php</a>	
<b>Consequences of not Fixing the Issue</b>	
Not fixing clickjacking vulnerabilities can lead to severe consequences, such as unauthorized actions by users, data theft, the spread of malware, financial fraud, and damage to an organization's reputation. These vulnerabilities can also result in legal and regulatory penalties for compromised data protection. Furthermore, leaving clickjacking issues unaddressed increases the attack surface, inviting more sophisticated attacks and undermining the trust and safety of web applications for users.	
<b>Suggested Countermeasures</b>	
Content Security Policy (CSP)	
<b>References</b>	
<a href="https://owasp.org/www-community/attacks/Clickjacking">https://owasp.org/www-community/attacks/Clickjacking</a>	

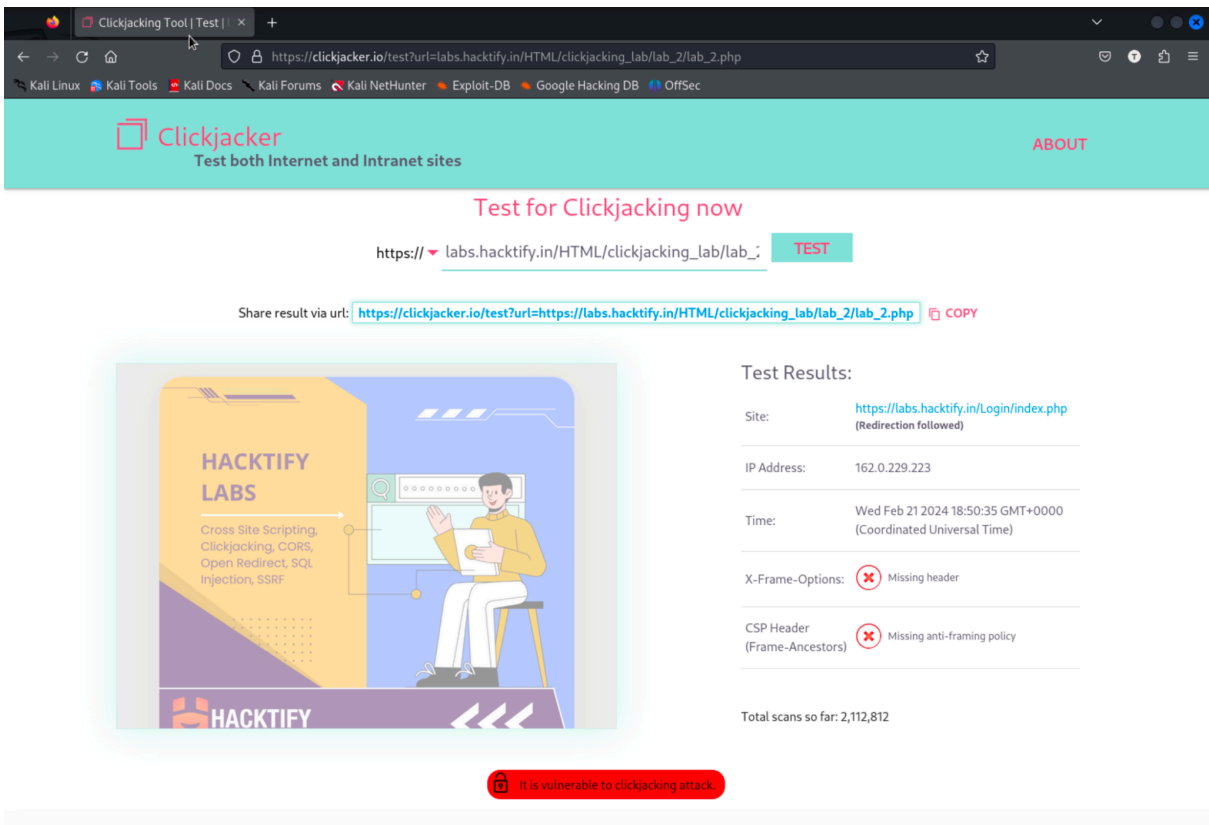
## Proof of Concept



## 1.2. {Re-Hijack!}

Reference	Risk Rating
{Re-Hijack}	Medium
<b>Tools Used</b>	
Automated tool	
<b>Vulnerability Description</b>	
Clickjacking, or a "UI redress attack," involves deceiving a user into interacting with hidden website elements by misleading them into clicking on seemingly harmless content on a decoy site. This interface-based attack exploits overlay techniques to manipulate user actions unknowingly.	
<b>How It Was Discovered</b>	
Automated Tools	
<b>Vulnerable URLs</b>	
https://labs.hacktify.in/HTML/clickjacking_lab/lab_2/lab_2.php	
<b>Consequences of not Fixing the Issue</b>	
Failing to address clickjacking vulnerabilities can result in users making unintended clicks that lead to sensitive information theft, malware installation, and unauthorized transactions. It can damage an organization's reputation, lead to legal consequences, and heighten the risk of more sophisticated cyber attacks, undermining the security and trustworthiness of web applications.	
<b>Suggested Countermeasures</b>	
To combat clickjacking, organizations can employ several strategies, including frame busting scripts, implementing Content Security Policy (CSP) headers to control who can frame their content, and using the X-Frame-Options header to restrict how content is embedded. Additionally, conducting regular security audits, educating users on safe browsing practices, and adopting modern web technologies with built-in protections are critical. These measures, when combined, offer a robust defense against clickjacking attacks, enhancing web application security and user trust.	
<b>References</b>	
https://owasp.org/www-community/attacks/Clickjacking	

# Proof of Concept



## 2. {HTML Injection}

### 2.1. {HTML's Are Easy!}

Reference	Risk Rating
{HTML's Are Easy!}	Low
Tools Used	
Manual	
Vulnerability Description	
HTML injection is a type of vulnerability that occurs when a website allows user input to be directly included on its pages without proper validation or encoding. This can allow attackers to inject arbitrary HTML code into the web page, which will be rendered and executed by the browser of anyone viewing the infected page. The consequences of such an attack can range from benign, such as modifying the appearance of the webpage, to malicious activities like stealing user session cookies, redirecting the user to phishing sites, or executing scripts in the context of the website (cross-site scripting, or XSS).	
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	

HTML injection can lead to web page manipulation, user data theft, session hijacking, and redirection to malicious sites. This compromises website integrity, user privacy, and can damage an organization's reputation, potentially leading to financial loss, legal repercussions, and a loss of trust from users and customers.

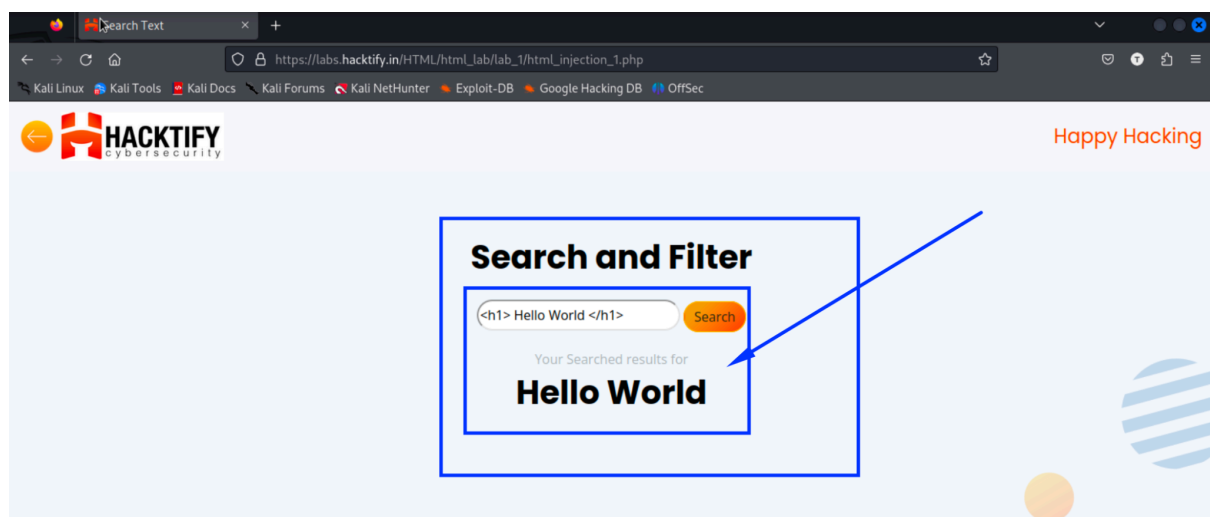
#### Suggested Countermeasures

To mitigate HTML injection risks, it's essential to enforce strict input validation, employ output encoding to neutralize malicious code, use secure APIs that automatically handle input safely, implement Content Security Policy (CSP) to restrict resource loading, and conduct regular security audits. Additionally, educating developers on secure coding practices is crucial. These measures collectively help safeguard web applications against HTML injection, protecting both the integrity of the websites and the privacy of their users.

#### References

[https://owasp.org/www-project-web-security-testing-guide/v41/4-Web\\_Application\\_Security\\_Testing/11-Client\\_Side\\_Testing/03-Testing\\_for\\_HTML\\_Injection](https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/11-Client_Side_Testing/03-Testing_for_HTML_Injection)

## Proof of Concept



## 2.2. {Let Me Store Them!}

Reference	Risk Rating
{Let Me Store Them!}	Low
Tools Used	
Manual	

<b>Vulnerability Description</b>
Stored HTML injection, a subtype of HTML injection, occurs when malicious HTML code is permanently stored on a target server, such as in a database, message forum, visitor log, or comment field. This code is then served to users whenever they access the affected page. The consequences of such an attack can be more severe than those of a typical HTML injection because the malicious code can affect multiple users over a longer period without the need for repeated injection attempts.
<b>How It Was Discovered</b>
Manual Analysis
<b>Vulnerable URLs</b>
<a href="https://labs.hacktify.in/HTML/html_lab/lab_2/register.php">https://labs.hacktify.in/HTML/html_lab/lab_2/register.php</a>
<b>Consequences of not Fixing the Issue</b>
Stored HTML injection can lead to persistent cross-site scripting (XSS) attacks, where attackers can steal cookies, session tokens, or other sensitive information from users. It can also allow attackers to deface websites, spread malware, perform phishing attacks, and potentially gain unauthorized access to user accounts or sensitive data stored on the server.
<b>Suggested Countermeasures</b>
To defend against stored HTML injection, it's crucial to rigorously validate and sanitize user inputs before storing them, encode outputs to neutralize malicious HTML code, implement Content Security Policy (CSP) for added security, and regularly perform security audits to identify vulnerabilities. Educating developers on secure coding practices is also essential to prevent such attacks. These measures collectively help safeguard web applications from the persistent threats posed by stored HTML injection, ensuring the protection of user data and maintaining the integrity of the web platform.
<b>References</b>
<a href="https://www.esecforte.com/responsible-vulnerability-disclosure-cve-2019-12863-stored-html-injection-vulnerability-in-solarwinds-orion-platform-2018-4-hf3-npm-12-4-netpath-1-1-4/">https://www.esecforte.com/responsible-vulnerability-disclosure-cve-2019-12863-stored-html-injection-vulnerability-in-solarwinds-orion-platform-2018-4-hf3-npm-12-4-netpath-1-1-4/</a>

## Proof of Concept

Browser tabs: Kali Linux, Kali Tools, Kali Docs, Kali Forums, Kali NetHunter, Exploit-DB, Google Hacking DB, OffSec

Address bar: [https://labs.hacktify.in/HTML/html\\_lab/lab\\_2/profile.php](https://labs.hacktify.in/HTML/html_lab/lab_2/profile.php)

Logo: HACKTIFY cybersecurity

Text: Happy Hacking

### User Profile

First Name:

Last Name:

Email:

Password:

Confirm Password:

## 2.3. {File Names are also vulnerable!}

Reference	Risk Rating
{File Names are also vulnerable!}	Low
<b>Tools Used</b>	
Manual	
<b>Vulnerability Description</b>	
File name vulnerability in HTML injection refers to a specific scenario where an application allows users to upload files with names that include HTML code or scripts. When these file names are displayed on a webpage without proper sanitization or encoding, the embedded HTML or script can be executed in the context of the user's browser. This form of vulnerability is particularly concerning because it combines the risks associated with file uploads and HTML injection, potentially leading to cross-site scripting (XSS) attacks or other malicious outcomes.	
<b>How It Was Discovered</b>	
Manual Analysis	
<b>Vulnerable URLs</b>	
<a href="https://labs.hacktify.in/HTML/html_lab/lab_3/html_injection_3.php">https://labs.hacktify.in/HTML/html_lab/lab_3/html_injection_3.php</a>	
<b>Consequences of not Fixing the Issue</b>	
Execute scripts in the context of other users' sessions, leading to session hijacking, data theft, or spreading malware. Phish for sensitive information by injecting forms or redirecting users to malicious sites. Deface the website by injecting arbitrary HTML content.	
<b>Suggested Countermeasures</b>	
To counter file name vulnerabilities in HTML injection, enforce strict input validation to filter or sanitize file names during uploads, use output encoding to neutralize executable code when displaying file	


names on web pages, and implement Content Security Policy (CSP) to restrict script execution. These strategies collectively strengthen web application defenses, preventing attackers from exploiting these vulnerabilities to conduct malicious activities.

#### References

[https://owasp.org/www-project-web-security-testing-guide/latest/4-Web\\_Application\\_Security\\_Testing/11-Client-side\\_Testing/03-Testing\\_for\\_HTML\\_Injection](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/03-Testing_for_HTML_Injection)

## Proof of Concept

```
95     <div class="container">
96         <center>
97             <div class="containers">
98                 <div class="contain">
99                     <h1>Upload a File</h1><br />
100                     <div class="welcome-image">
101                         <form action="html_injection_3.php" method="POST" enctype="multipart/form-data">
102                             <input type="file" name="image"><br><br>
103                             <button type="submit" name="upload" class="btn btn-warning">File Upload</button>
104                         </form>
105                     </div>
106                     <center><br><h2>File Uploaded <b>pole.txt</b></h2></center>
107                 </div>
108             </center>
109         </div>
110     </div>
111 </div>
112 </section>
113 <hr />
114 <div class="footer">
115     <p>© Copyrights 2021 Hacktify Cybersecurity All rights reserved</p>
116 </div>
117 </div>
118 </body>
119 </html>
120
121
```



## 2.4. {File Content and HTML Injection a perfect pair!}

Reference	Risk Rating
{File Content and HTML Injection a perfect pair!}	Medium
Tools Used	
Manual	
Vulnerability Description	
File Content and HTML Injection vulnerabilities occur when an application allows user-supplied content, which can include malicious HTML or script code, to be uploaded and then served or executed without proper sanitization. This can happen in various contexts, such as uploading documents, images with metadata, or any file type where the content or metadata is not securely validated or encoded before being displayed to users or executed by the web application.	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
<a href="https://labs.hacktify.in/HTML/html_lab/lab_4/html_injection_4.php">https://labs.hacktify.in/HTML/html_lab/lab_4/html_injection_4.php</a>	
Consequences of not Fixing the Issue	
The consequences of these vulnerabilities are similar to those of traditional HTML injection, but with potentially broader impact due to the nature of file content being served or executed. Cross-site	



Scripting (XSS) attacks, where attackers execute scripts in the context of other users' sessions. Phishing attacks by injecting malicious content that appears legitimate. Session hijacking and redirection to malicious sites. Stealing or manipulating sensitive information. Defacing websites or displaying unauthorized content.

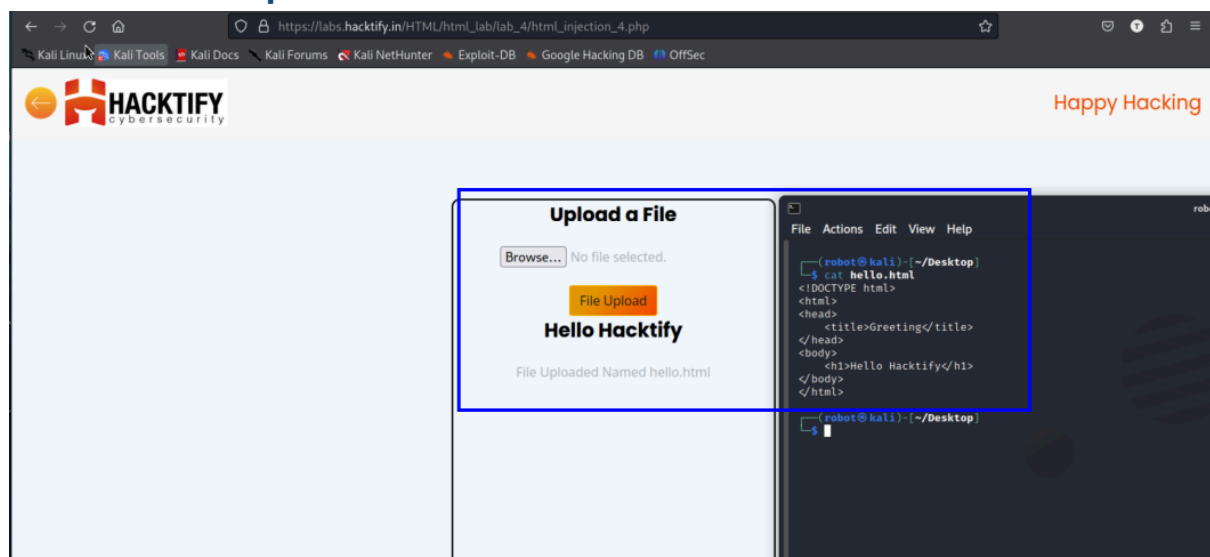
#### Suggested Countermeasures

To mitigate File Content and HTML Injection vulnerabilities, ensure comprehensive validation and sanitization of all user-supplied content, including file uploads and metadata, to prevent malicious code execution. Implementing Content Security Policy (CSP) is crucial for limiting the impact of any injection, by restricting resource loading and script execution from unauthorized sources. Secure file handling practices, such as isolating user-uploaded content on a separate domain and encoding all output displayed to users, are essential. Regular security assessments of file upload and handling processes further help in identifying and addressing vulnerabilities, solidifying the security of web applications against these types of attacks.

#### References

<https://www.acunetix.com/vulnerabilities/web/html-injection/>

## Proof of Concept



## 2.5. {Injecting HTML using URL!}

Reference	Risk Rating
{Injecting HTML using URL!}	Medium
Tools Used	
Manual	
Vulnerability Description	
HTML injection through URL parameters is a web security vulnerability that occurs when a web application accepts unvalidated or unencoded user input from the URL query string and dynamically includes it in the webpage content. Attackers can exploit this by crafting a malicious URL with HTML or JavaScript code as part of the query parameters. When a user visits this URL, the code is executed	

within the context of the webpage, leading to potential security breaches such as Cross-Site Scripting (XSS).

#### How It Was Discovered

Manual Analysis

#### Vulnerable URLs

[https://labs.hacktify.in/HTML/html\\_lab/lab\\_5/html\\_injection\\_5.php](https://labs.hacktify.in/HTML/html_lab/lab_5/html_injection_5.php)

#### Consequences of not Fixing the Issue

HTML injection through URL parameters can have severe consequences, including Cross-Site Scripting (XSS) attacks that allow attackers to steal sensitive information, hijack user sessions, and spread malware. This vulnerability undermines the security and integrity of web applications, eroding user trust and potentially exposing organizations to legal and financial repercussions. It emphasizes the critical need for robust input validation and output encoding practices to protect against such security threats.

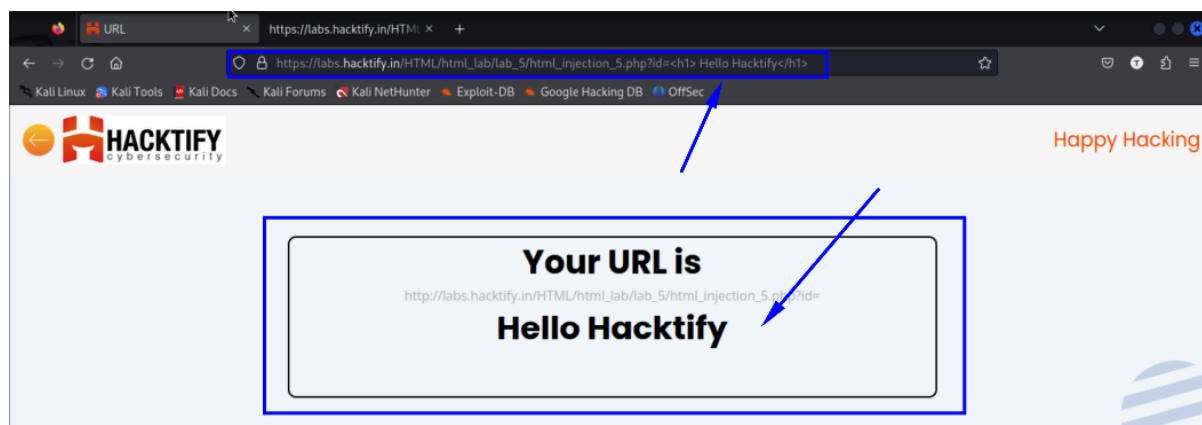
#### Suggested Countermeasures

To guard against HTML injection via URL parameters, enforce strict input validation to ensure only expected data is processed. Utilize output encoding to escape HTML entities, preventing execution of malicious code. Implementing a Content Security Policy (CSP) can also help mitigate potential attacks by restricting the sources from which scripts can be executed. These measures, combined with regular security audits and adopting secure coding practices, can significantly reduce the risk of HTML injection, safeguarding both the application and its users.

#### References

<https://www.cgisecurity.com/articles/xss-faq.shtml>

## Proof of Concept



## 2.6. {Encode IT!}

Reference	Risk Rating
{Encode IT!}	High
Tools Used	
Manual	
Vulnerability Description	

Blind SSRF attacks are typically carried out by sending a crafted request to a vulnerable web application. The request is designed to cause the web application to make a request to an external server. The attacker can then use the response from the external server to infer information about the web application's internal network or to carry out further attacks.

#### How It Was Discovered

Manual Analysis

#### Vulnerable URLs

[https://labs.hacktify.in/HTML/html\\_lab/lab\\_6/html\\_injection\\_6.php](https://labs.hacktify.in/HTML/html_lab/lab_6/html_injection_6.php)

#### Consequences of not Fixing the Issue

Ignoring blind Server-Side Request Forgery (SSRF) vulnerabilities can have dire consequences, including unauthorized access to internal networks, bypassing of security mechanisms, data breaches, service disruptions, and potential server compromise. Such vulnerabilities expose sensitive systems to attackers, leading to regulatory and compliance issues, financial penalties, and reputational damage.

#### Suggested Countermeasures

To mitigate blind SSRF vulnerabilities, adopt strict input validation and sanitization measures, and enforce allowlists for outbound requests to limit interactions to trusted destinations. Employ network segmentation and the principle of least privilege to restrict server access within the network. Additionally, implement robust monitoring and logging of outbound requests to quickly identify and respond.

#### References

<https://portswigger.net/web-security/ssrf/blind>

## Proof of Concept

