Week 1 Penetration Testing Report

Full Name: Atharva Jagdale
Program: HCPT
Date: 13-02-2025

Introduction

This report document hereby describes the proceedings and results of a Black Box security assessment conducted against Week **2 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 2 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	**{HTML Injection}, {Cross-Site Scripting}**

3. Summary

Outlined is a Black Box Application Security assessment for Week **2 Labs**.

Total number of Sub-labs: 8 Sub-labs

High	Medium	Low
1	3	4

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

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

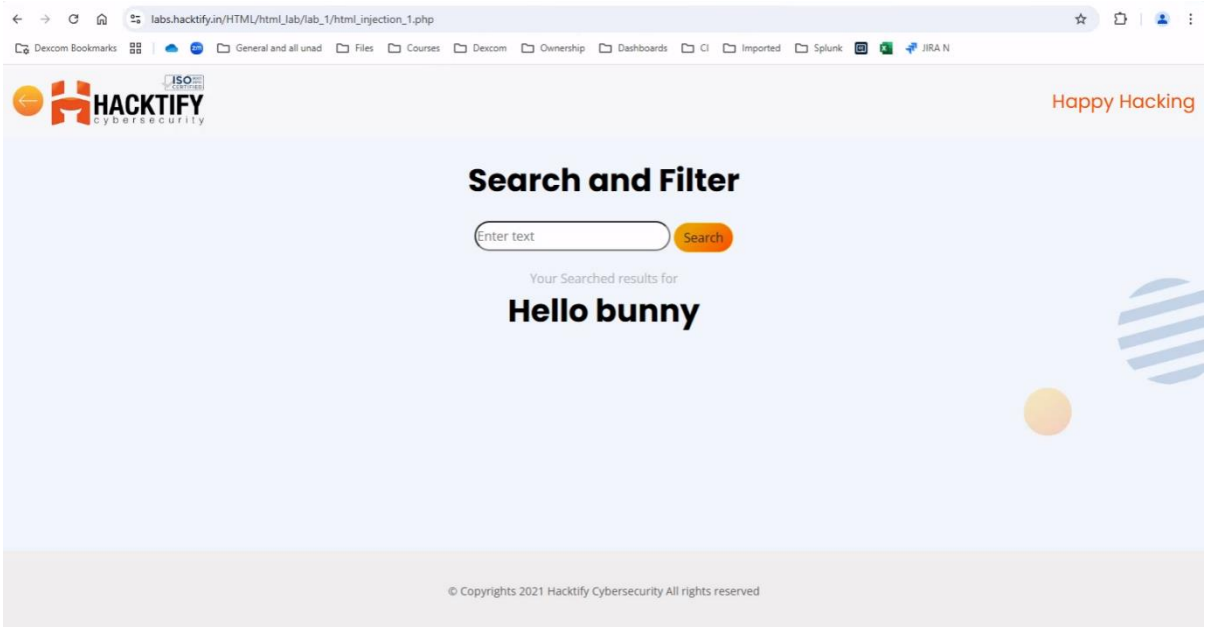
Low - **Number of Sub-labs with Easy difficulty level**

1. {HTML Injection}

1.1. {HTML's are easy!}

Reference	Risk Rating
HTML's are easy!	**Low**
Tools Used	
Google chrome	
Vulnerability Description	
I found this vulnerability by using basic HTML tags i.e., h1 tag.	
How It Was Discovered	
Manual Analysis.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_1/lab_1.php	
Consequences of not Fixing the Issue	
This can lead to managing and accessing other web pages and it can lead to phishing other users.	
Suggested Countermeasures	
Sanitize the input by validating canonical tags (<>) and slash (/). or < > ( ) ' " / \ * ; : = { }.	
References	
I used basic knowledge of HTML to discover this vulnerability.	
Payload Used	
Hello bunny	

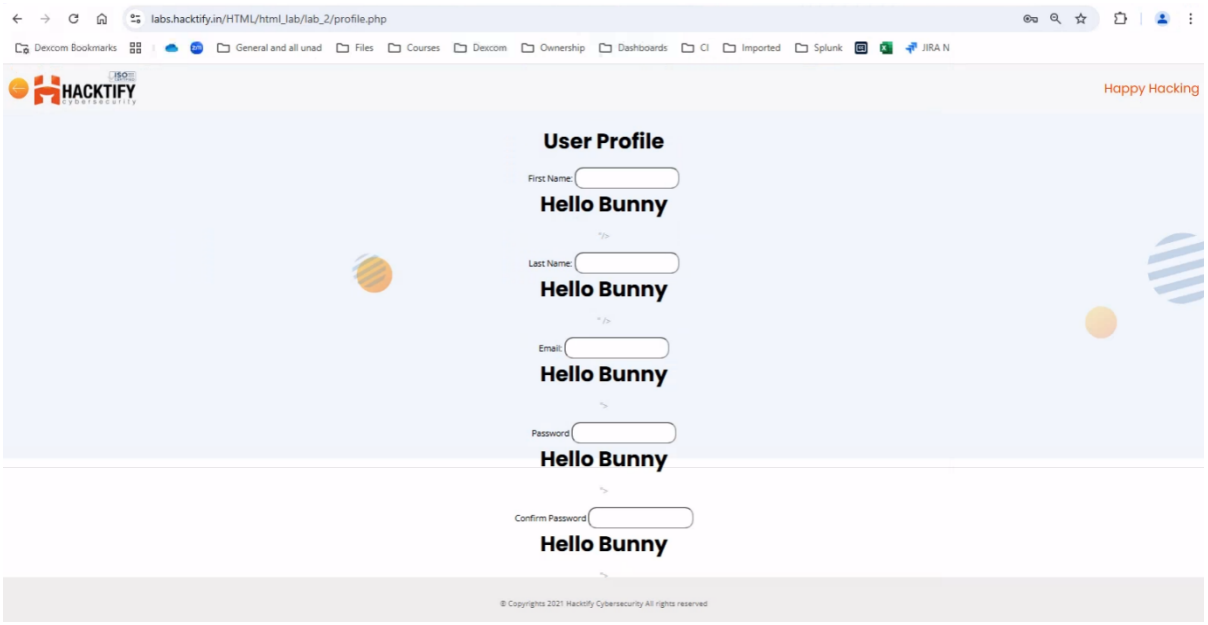
Proof of Concept



1.2. {Let me store them}

Reference	Risk Rating
Let me store them	**Low**
Tools Used	
Google chrome	
Vulnerability Description	
I found this vulnerability by trying multiple HTML tags on different input fields. Here I can run every HTML valid code to inject malicious code.	
How It Was Discovered	
Manual Analysis.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/ html_lab/lab_2/lab_2.php	
Consequences of not Fixing the Issue	
To steal another person's identity.	
Suggested Countermeasures	
Sanitize the input by validating canonical tags (<>) and slash (/). or < > ( ) ‘ “ / \ * ; : = { }.	
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	
Payload Used	
"> Hello Bunny	

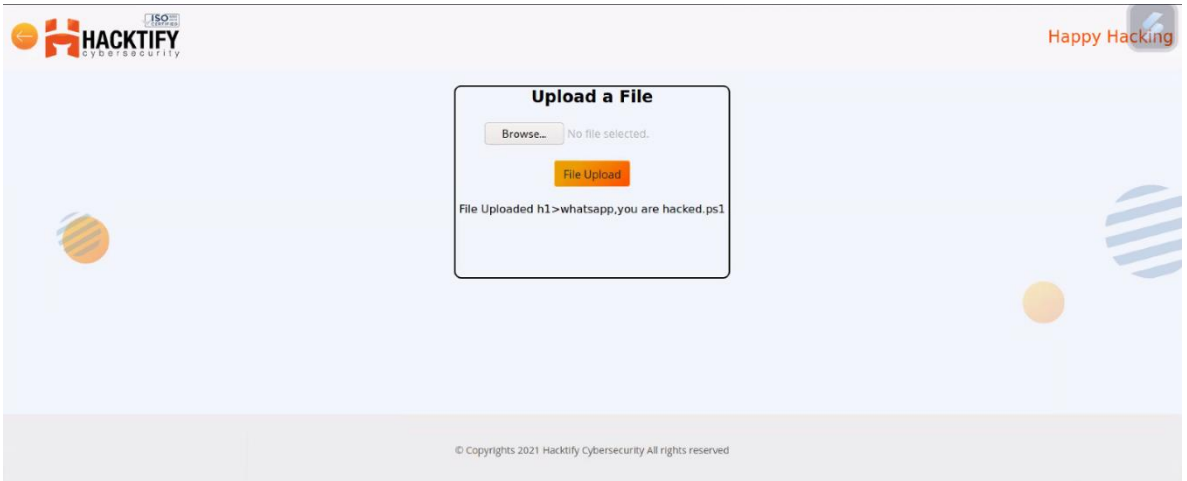
Proof of Concept



1.3. {File names are also vulnerable}

Reference	Risk Rating
File names are vulnerable	**Low**
Tools Used	
Google chrome and Burp Suite	
Vulnerability Description	
I found this vulnerability by intercepting a web app on burpsuite and changing the filename on it to back on the web to a successful run.	
How It Was Discovered	
Manual Analysis as well as automated analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/ html_lab/lab_3/lab_3.php	
Consequences of not Fixing the Issue	
Malicious file names may distract the user and it can cause the user to do something illegal or wrong on the web.	
Suggested Countermeasures	
Sanitize the input by validating canonical tags (<>) and slash (/). or < > ( ) ‘ “ / \ * ; : = { } and it should not accept any of these tags by external resources also.	
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	
Payload Used	
whatsapp,you are hacked.ps1	

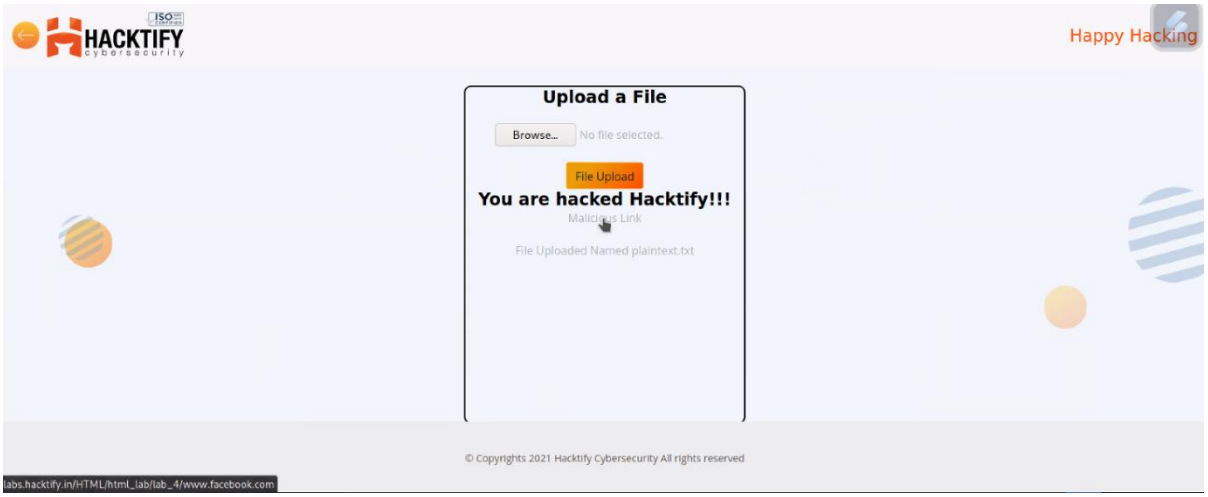
Proof of Concept



1.4. {File content and HTML Injection a perfect pair}

Reference	Risk Rating
File content and HTML Injection are a perfect pair	**Medium**
Tools Used	
Google chrome	
Vulnerability Description	
I found this vulnerability by just uploading malicious files in the file uploading section with different file extensions and containing malicious code in it.	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/ html_lab/lab_4/lab_4.php	
Consequences of not Fixing the Issue	
It can lead to taking down an entire website and the website will show wrong data content on the website.	
Suggested Countermeasures	
Sanitization of file extensions that it should accept only required files with valid and specified file extensions	
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	
Payload Used	
You are Hacked in a text file and then uploaded over there	

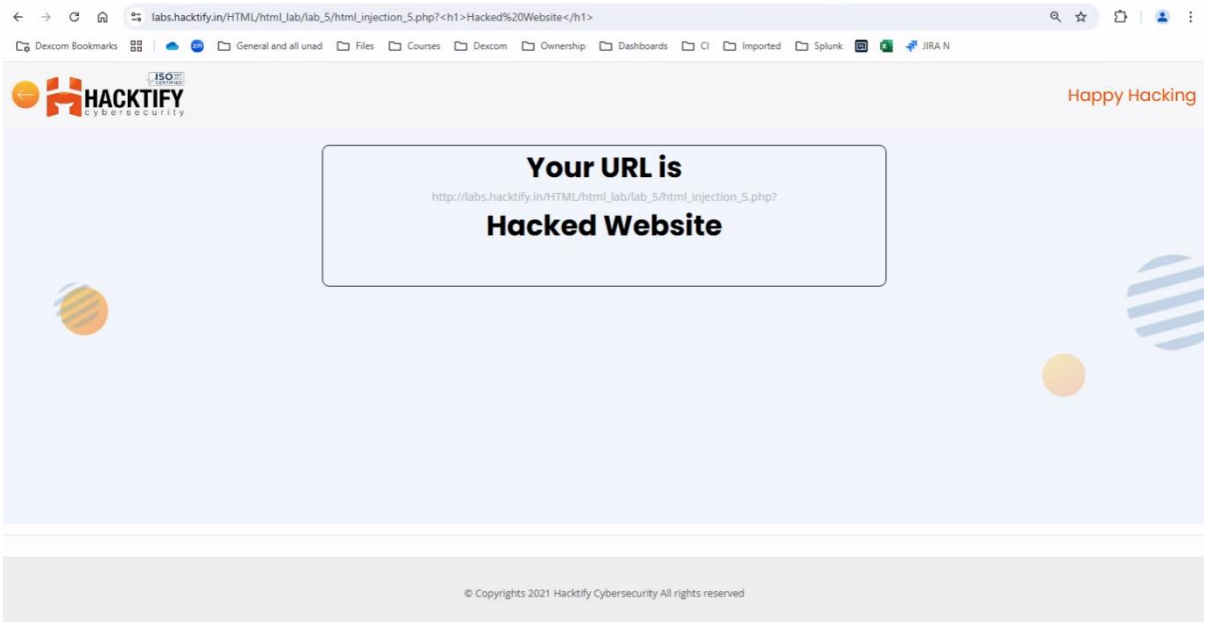
Proof of Concept



1.5. {Injecting HTML using URL}

Reference	Risk Rating
Injecting HTML using URL	**Medium**
Tools Used	
Google chrome	
Vulnerability Description	
I found this vulnerability by tampering the actual website of the code and injecting valid HTML code as malicious code or payload	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/ html_lab/lab_5/lab_5.php	
Consequences of not Fixing the Issue	
It can allow attackers to modify the URL and modify the web pages	
Suggested Countermeasures	
Invalid URL should not work or should display errors to the user	
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	
Payload Used	
? Hacked Website in the URL	

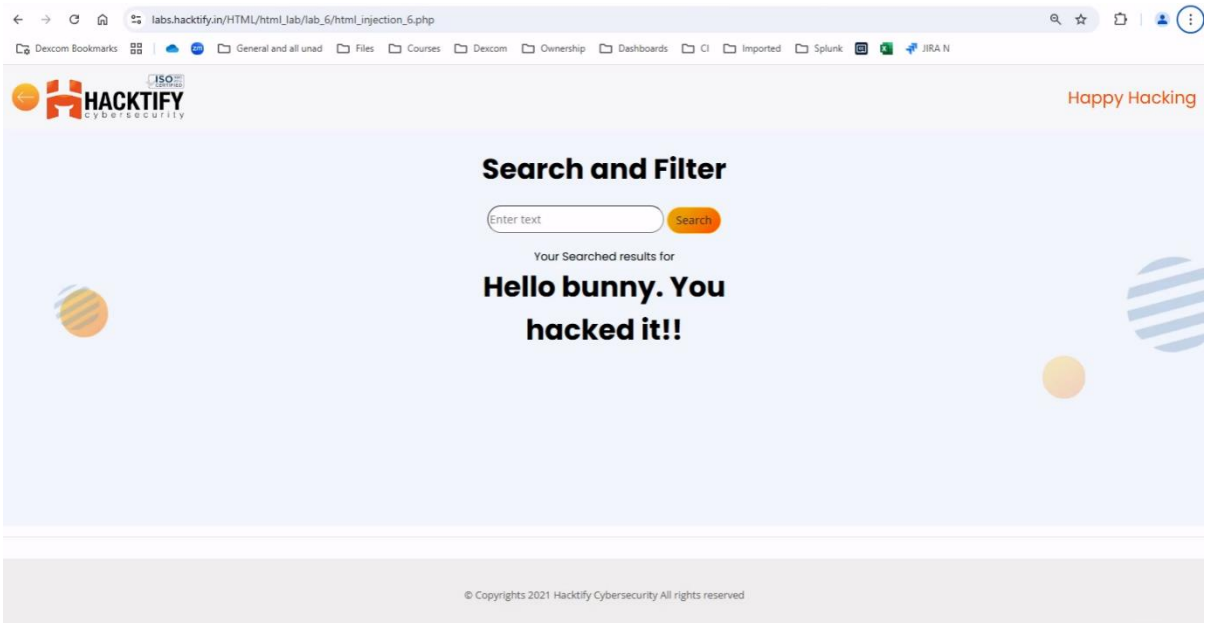
Proof of Concept



1.6. {Encode it}

Reference	Risk Rating
Encode it	**High**
Tools Used	
Google chrome	
Vulnerability Description	
I found this vulnerability by entering basic HTML valid code by encoding it into URL encoder	
How It Was Discovered	
Manual Analysis and automated analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/ html_lab/lab_6/lab_6.php	
Consequences of not Fixing the Issue	
This can lead to managing and accessing other web pages and it can lead to phishing other users.	
Suggested Countermeasures	
Sanitize the input by validating canonical tags (<>) and slash (/). or < > ( ) ‘ “ / \ * ; : = { } also encoding with the help of other external tools	
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	
Payload Used	
<h1?Hello bunny. You hacked it!!</h1> in URL Encoder	

Proof of Concept

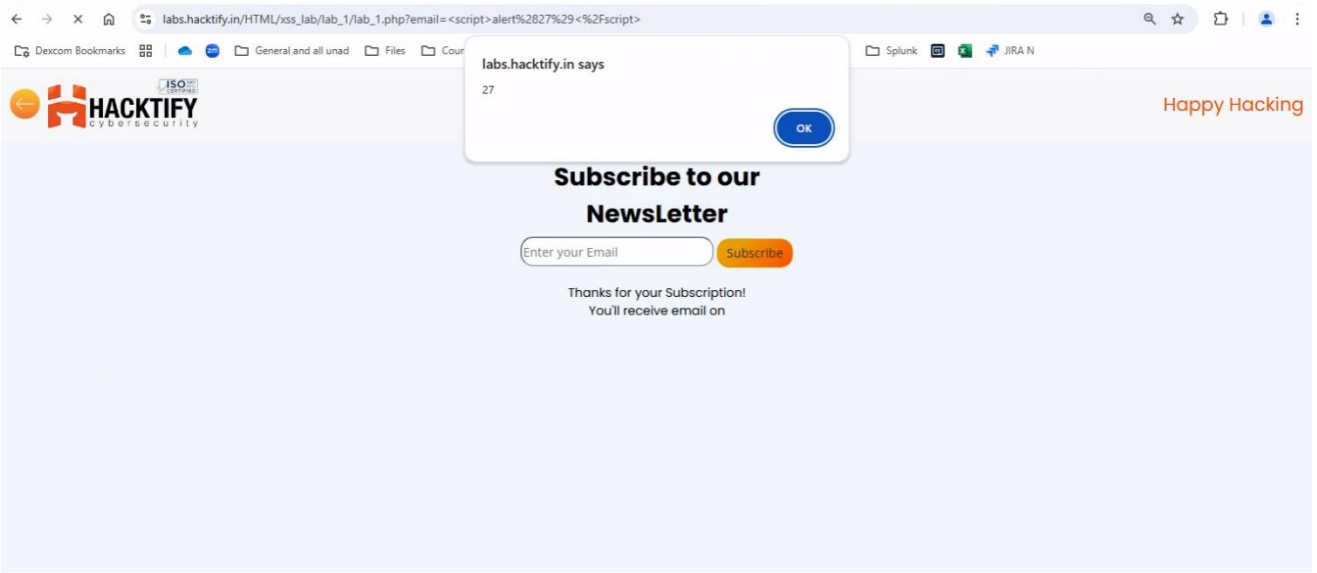


2. {Cross-Site Scripting}

2.1. {Let's do it!}

Reference	Risk Rating
Let's Do IT!	**Low**
Tools Used	
Google Chrome	
Vulnerability Description	
I found this vulnerability by entering simple javascript code into the client side reflection field, i.e., i successfully performed reflected xss.	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
labs.hacktify.in/HTML/xss_lab/lab_1/lab_1.php	
Consequences of not Fixing the Issue	
XSS enables an attacker to inject client-side code, so with this, an attacker will be able to capture client-side cookies which can capture account credentials of any user.	
Suggested Countermeasures	
Developers must sanitize the input of any user by denying entry of any tag or any special character.	
References	
I found this basic vulnerability by entering a basic JavaScript tag. So, I did not get any help.	
Payload Used	
<script>alert(27)</script>	

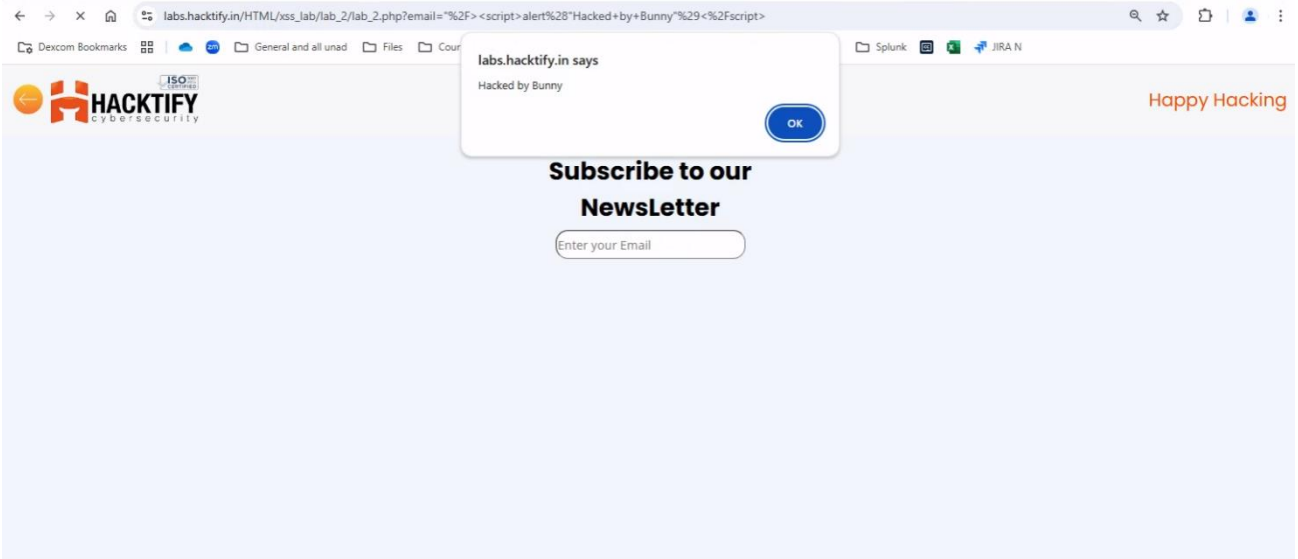
Proof of Concept



2.2. {Balancing is important in life}

Reference	Risk Rating
Balancing is important in life	**Low**
Tools Used	
Google chrome	
Vulnerability Description	
I found this vulnerability by modifying the URL after getting valid output as actually developed. Also, it will allow an attacker to carry out any action that the user is able to perform.	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
labs.hacktify.in/HTML/xss_lab/lab_2/lab_2.php	
Consequences of not Fixing the Issue	
Carry out any action that the user can perform.	
Suggested Countermeasures	
Developers must sanitize the input of any user by denying entry of any tag or any special character. And Also, illegal links of any web page should not display any valid output.	
References	
https://github.com/payloadbox/xss-payload-list	
Payload Used	
"/><script>alert("Hacked by Bunny")</script>	

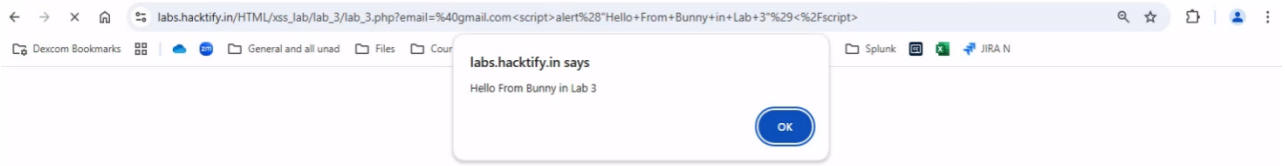
Proof of Concept



2.3. {XSS is everywhere!}

Reference	Risk Rating
XSS is everywhere!	**Low**
Tools Used	
Google chrome	
Vulnerability Description	
I found this vulnerability by modifying the URL after getting valid output as actually developed. Here, the developer has given validation while developing this web page as the text field must contain a valid email address which will end with “@gmail.com”. If any other input is entered, then an error message will display called “Enter a valid email address”. So, to bypass this I simple inserted my payload which will end with provided validation for input, i.e., “@gmail.com”	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
labs.hacktify.in/HTML/xss_lab/lab_3/lab_3.php	
Consequences of not Fixing the Issue	
Carry out any action that the user can perform.	
Suggested Countermeasures	
Developers must sanitize the input of any user by denying entry of any tag or any special character if it is not a valid entry for that text field.	
References	
https://github.com/payloadbox/xss-payload-list	
Payload Used	
@gmail.com<script>alert("Hello From Bunny in Lab 3")</script>	

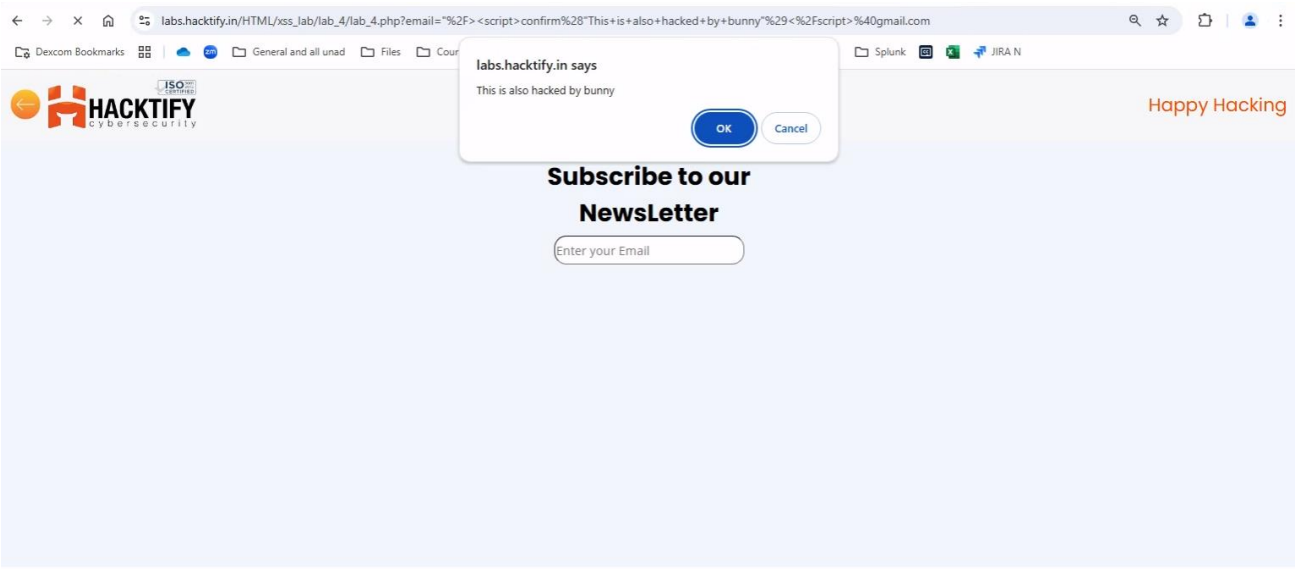
Proof of Concept



2.4. {Alternatives are a must!}

Reference	Risk Rating
Alternatives are a must	**Medium**
Tools Used	
Google chrome	
Vulnerability Description	
I found this vulnerability by modifying the URL after getting valid output as actually developed. But, here the developer has sanitized the tag which is commonly used in JavaScript, i.e., alert(). So, to bypass this situation, I used the confirm() tag and I successfully got what I expected. Also, I am able to successfully run prompt() tag instead of alert() and alternative of confirm()	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
labs.hacktify.in/HTML/xss_lab/lab_4/lab_4.php	
Consequences of not Fixing the Issue	
Carry out any action that the user can perform.	
Suggested Countermeasures	
Such web pages must be sanitized by denying the input of JavaScript tags which are not required in the text fields.	
References	
https://github.com/payloadbox/xss-payload-list	
Payload Used	
"/><script>confirm("This is also hacked by bunny")</script>@gmail.com	

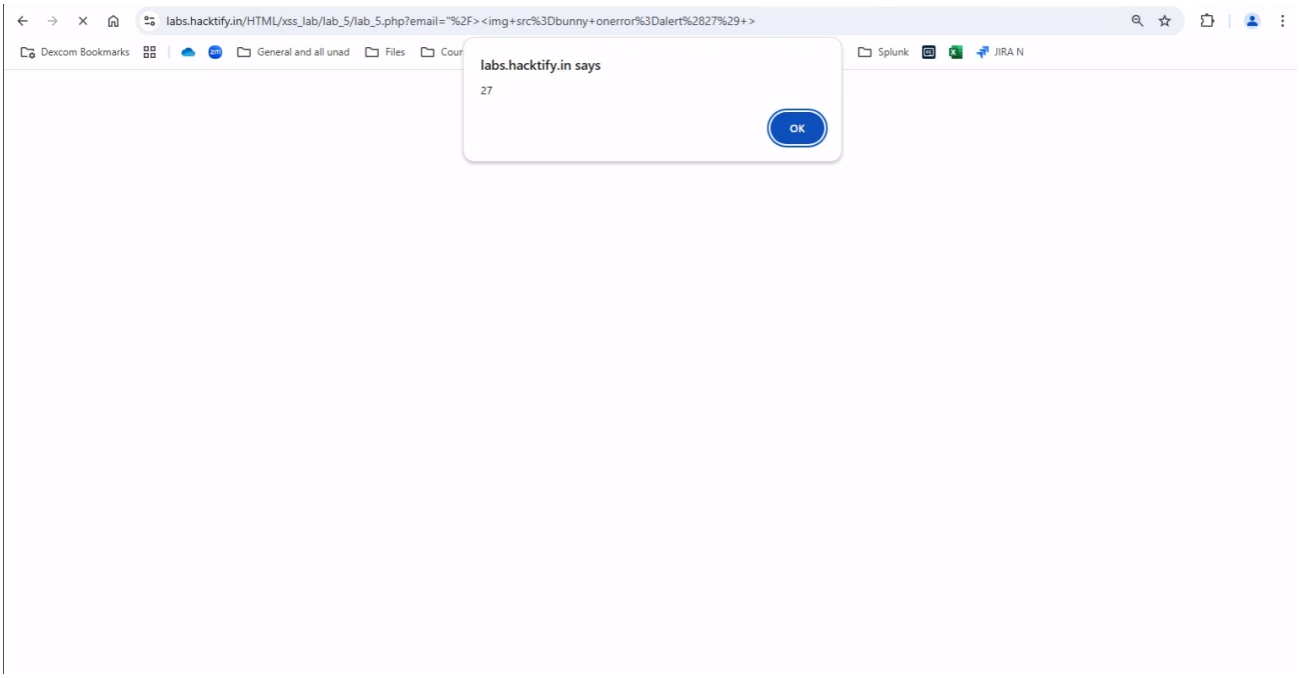
Proof of Concept



2.5. {Developer hates script!}

Reference	Risk Rating
Developers hate scripts!	**Hard**
Tools Used	
Google chrome	
Vulnerability Description	
I found this vulnerability by using different javascript tags like <img>, <svg>, etc.	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
labs.hacktify.in/HTML/xss_lab/lab_5/lab_5.php	
Consequences of not Fixing the Issue	
Carry out any action that the user can perform.	
Suggested Countermeasures	
Developers must sanitize the input of any user by denying entry of any tag or any special character if it is not a valid entry for that text field.	
References	
https://github.com/payloadbox/xss-payload-list	
Payload Used	
"/><img src=bunny onerror=alert(27) >	

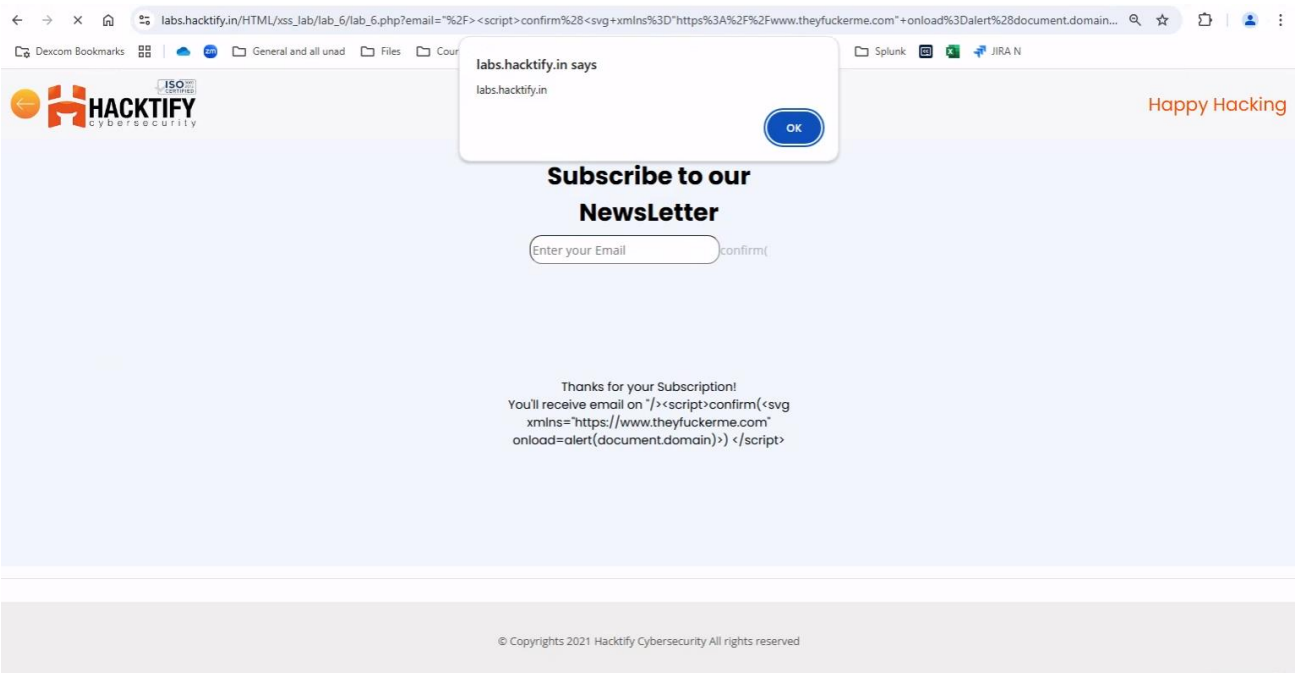
Proof of Concept



2.6. {Change the variation!}

Reference	Risk Rating
Change the variation!	**Hard**
Tools Used	
Google chrome	
Vulnerability Description	
I found this vulnerability by merging two different tags within a single command as payload to bypass the situation which the developer has created. I used, <script> tag in which I used <svg> tag to bypass it.	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
labs.hacktify.in/HTML/xss_lab/lab_6/lab_6.php	
Consequences of not Fixing the Issue	
Carry out any action that the user can perform.	
Suggested Countermeasures	
Developers must sanitize the input of any user by denying entry of any tag or any special character if it is not a valid entry for that text field.	
References	
https://github.com/payloadbox/xss-payload-list	
Payload Used	
"/><script>confirm(<svg xmlns="https://www.theyfuckerme.com" onload=alert(document.domain)>) </script>	

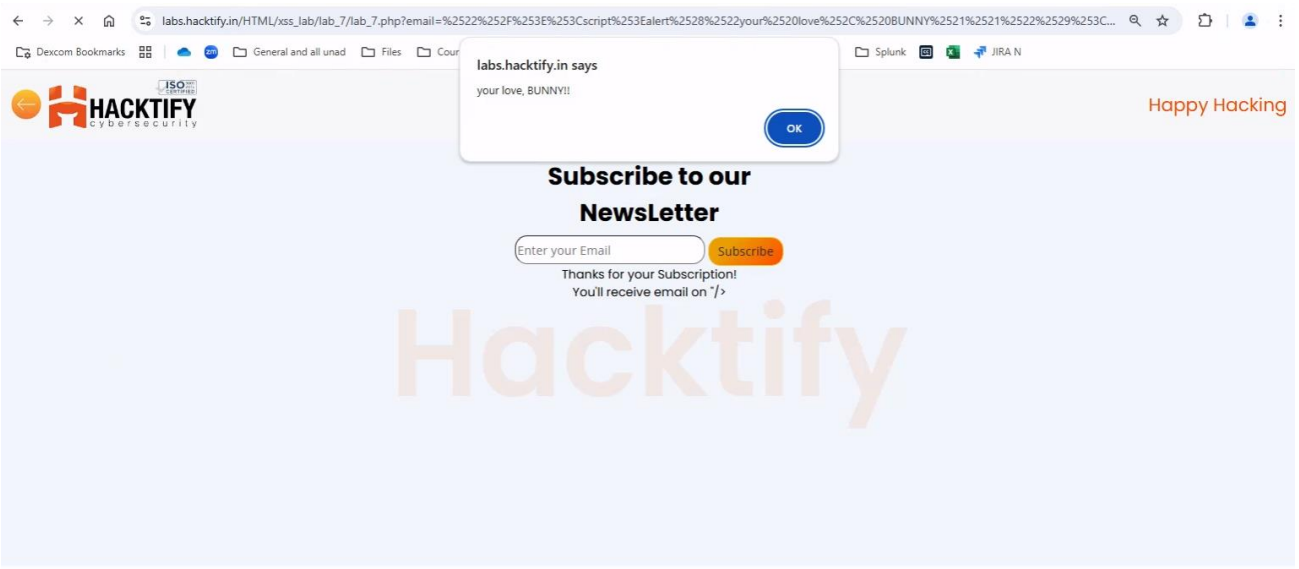
Proof of Concept



2.7. {Encoding is the key?}

Reference	Risk Rating
Encoding is the key?	**Medium**
Tools Used	
Google chrome	
Vulnerability Description	
This vulnerability was found by URL encoding a simple javascript command with the help of Burp Suite.	
How It Was Discovered	
Manual Analysis and Automated Analysis	
Vulnerable URLs	
labs.hacktify.in/HTML/xss_lab/lab_7/lab_7.php	
Consequences of not Fixing the Issue	
Carry out any action that the user can perform.	
Suggested Countermeasures	
Filtering inputs at the arrival of input data which will run through text fields.	
References	
https://github.com/payloadbox/xss-payload-list	
Payload Used	
Plain Text - "/><script>alert("your love, BUNNY!!")</script> **Encoded Payload** - %22%2F%3E%3Cscript%3Ealert%28%22your%20love%2C%20BUNNY%21%21%22%29%3C%2Fscript%3E	

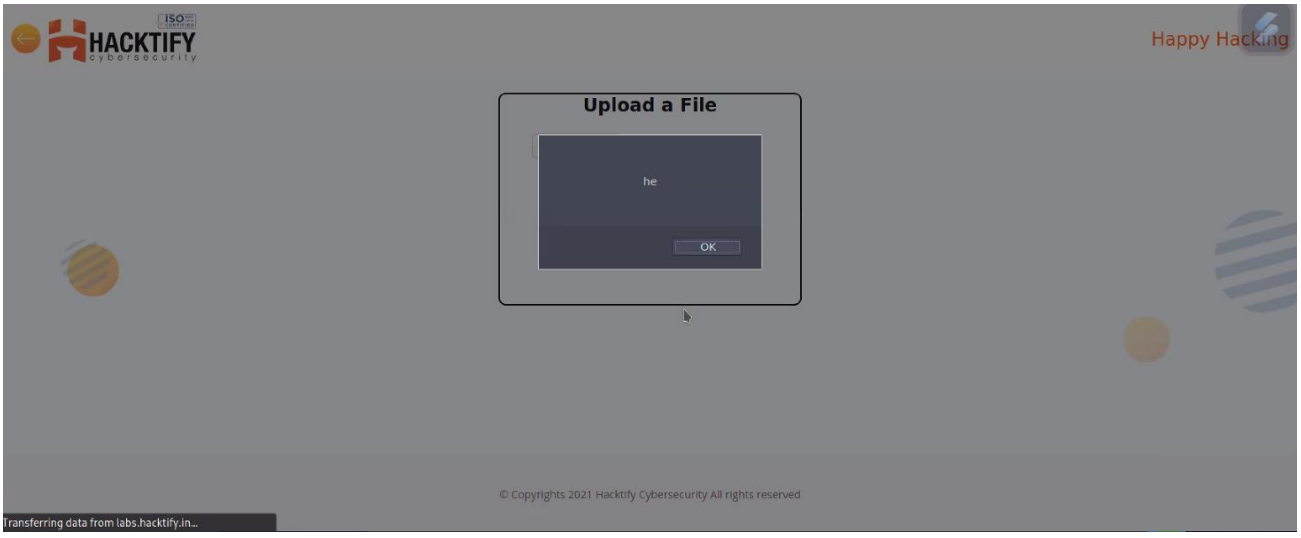
Proof of Concept



2.8. {XSS with file upload (file name)}

Reference	Risk Rating
XSS with file upload (file name)	**Low**
Tools Used	
Google chrome and Burp Suite	
Vulnerability Description	
I found this vulnerability by intercepting a file upload request onto Burp Suite tool and renaming the file name to insert payload into filename to bypass all the validation to load payload into the web pages and database to store payload into it.	
How It Was Discovered	
Manual Analysis and Automated Analysis	
Vulnerable URLs	
labs.hacktify.in/HTML/xss_lab/lab_8/lab_8.php	
Consequences of not Fixing the Issue	
Carry out any action that the user can perform.	
Suggested Countermeasures	
Using appropriate HTTP headers and using Content Security Policy	
References	
https://github.com/payloadbox/xss-payload-list	
Payload Used	
"/><img src=bunny onerror=alert(27) >	

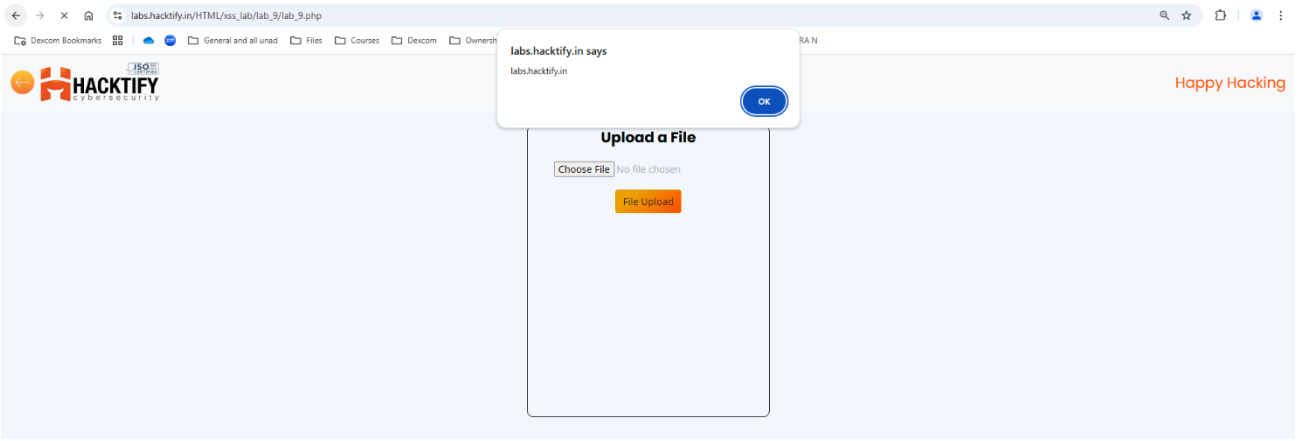
Proof of Concept



2.9. {XSS with file upload (file content)}

Reference	Risk Rating
XSS with file upload (file content)	**Medium**
Tools Used	
Google chrome	
Vulnerability Description	
I found this vulnerability by uploading a malicious code file into the file upload section. In malicious code, I used <script> tag to design a malicious code.	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
labs.hacktify.in/HTML/xss_lab/lab_9/lab_9.php	
Consequences of not Fixing the Issue	
Carry out any action that the user can perform.	
Suggested Countermeasures	
Using appropriate HTTP headers and using Content Security Policy	
References	
https://github.com/payloadbox/xss-payload-list	
Payload Used	
<script>alert(document.domain)</script>	

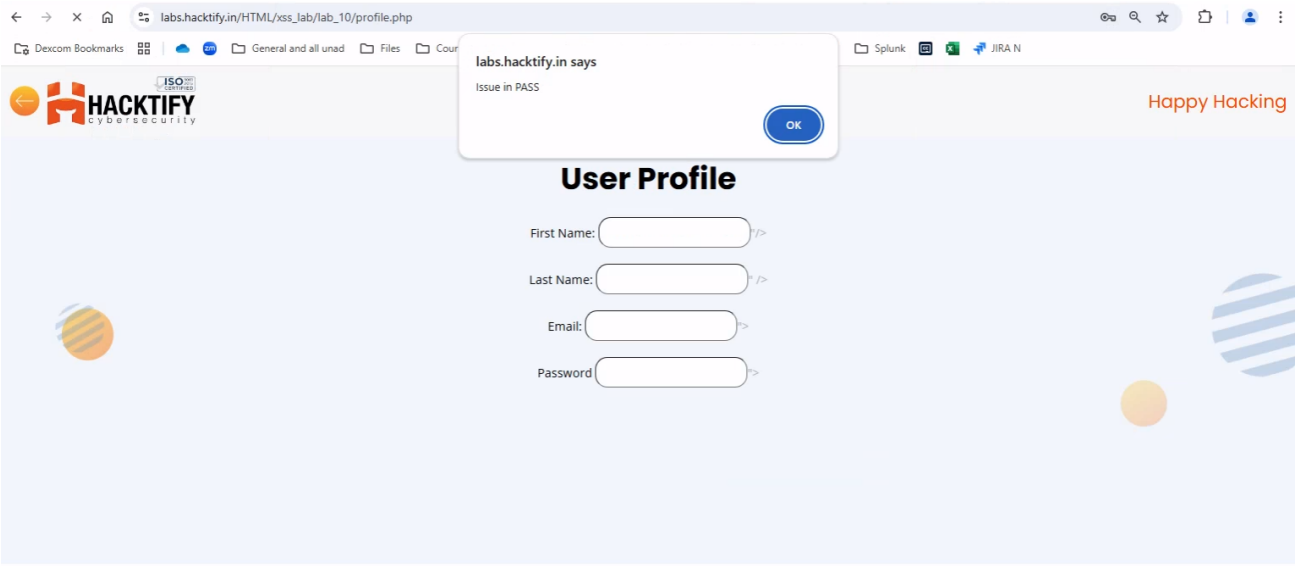
Proof of Concept



2.10. {Stored Everywhere}

Reference	Risk Rating
Stored Everywhere	**Low**
Tools Used	
Google chrome	
Vulnerability Description	
I found this vulnerability by registering my into register web page. I inserted a simple <script> tag in all fields which are required to fill for registration. After registration, I tried logging in into the webpage and after the login request, I was able to verify that I successfully implemented a stored XSS attack. Here, these credentials will be stored into the database which can affect other used credentials as well as other database related features.	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
labs.hacktify.in/HTML/xss_lab/lab_10/lab_10.php	
Consequences of not Fixing the Issue	
Carry out any action that the user can perform.	
Suggested Countermeasures	
Using appropriate HTTP headers and using Content Security Policy	
References	
https://github.com/payloadbox/xss-payload-list	
Payload Used	
1. "/><script>alert("Issue in FNAME")</script> 2. "/><script>alert("Issue in LNAME")</script> 3. "/><script>alert("Issue in EMAIL")</script> 4. "/><script>alert("Issue in PASS")</script>	

Proof of Concept



2.11. {DOM's are Love}

Reference	Risk Rating
DOM's are Love	**Hard**
Tools Used	
Google chrome	
Vulnerability Description	
I found this vulnerability by inspecting the source code of the web page and finding out what variables are used in this section. And later on tried to load payload using <img> tag.	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
labs.hacktify.in/HTML/xss_lab/lab_11/lab_11.php	
Consequences of not Fixing the Issue	
Carry out any action that the user can perform.	
Suggested Countermeasures	
Using appropriate HTTP headers and using Content Security Policy	
References	
https://github.com/payloadbox/xss-payload-list	
Payload Used	
Payload - name=bunny<img src=bugs onerror=alert("buggeebunny")> URL - https://labs.hacktify.in/HTML/xss_lab/lab_11/lab_11.php?name=bunny%3Cimg%20src=bugs%20onerror=alert(%22buggeebunny%22)%3E	

Proof of Concept