Penetration Testing Report

Full Name : Rachael Ayomide Fanifosi
Program : HCPT
Date : 18/02/2025

Introduction

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

I. 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.

II. Scope

The scope of the penetration testing project for HTML injection and Cross Site Scripting includes identifying vulnerabilities in the web application's frontend. HTML injection and Cross Site Scripting points will be assessed to identify potential avenues for malicious code insertion. Results will be provided with recommendations for mitigation to enhance the application's security posture.

Application Name	**{Lab 1 -HTML injection}** **{Lab 2 – Cross Site Scripting}**

III. Summary

Outlined is an (Injection) Security assessment for the **Week {1} Labs**.

Total number of Sub-labs: 17

High	Medium	Low
5	4	8

High - **Number of Sub-lab with high difficulty level**

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

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

1. HTML Injection

1.1. HTML's are easy!

Reference	Risk Rating
Sub-lab-1: HTML's are easy!	Low
Tools Used	
Browser(Google Chrome browser), manual testing	
Vulnerability Description	
HTML injection is a type of web security vulnerability that allows an attacker to inject malicious HTML code into a webpage. The website is vulnerable to HTML injection attacks. This means that an attacker can inject malicious HTML code into the website. The vulnerability can be exploited to steal user data, take control of the website, or spread malware.	
How It Was Discovered	
It was discovered through Manual testing of the website's search function. During testing, it was found that the website allowed user-inputted HTML code to be executed. This was done by inputting the HTML code Hello into the search function, which was then executed by the website.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_1/html_injection_1.php	
Consequences of not Fixing the Issue	
If the vulnerability is not patched, an attacker can be able to write his own code to get the cookie information of the victim user, this HTML Injection attack leads to XSS attack. It can lead to data theft, website takeover, malware spread, reputation damage.	
Suggested Countermeasures	
1. Implement strict input validation and sanitize user-supplied data. 2. Use contextual output encoding to prevent script execution. 3. Deploy Content Security Policy (CSP) to restrict script sources. 4. Educate developers on secure coding practices. 5. Regularly audit and test for vulnerabilities.	
References	

https://owasp.org/www-community/Injection_Information
https://portswigger.net/web-security/cross-site-scripting/html-injection
https://en.wikipedia.org/wiki/HTML_injection

Proof of Concept

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

Payload: <h1>Hello</h1>

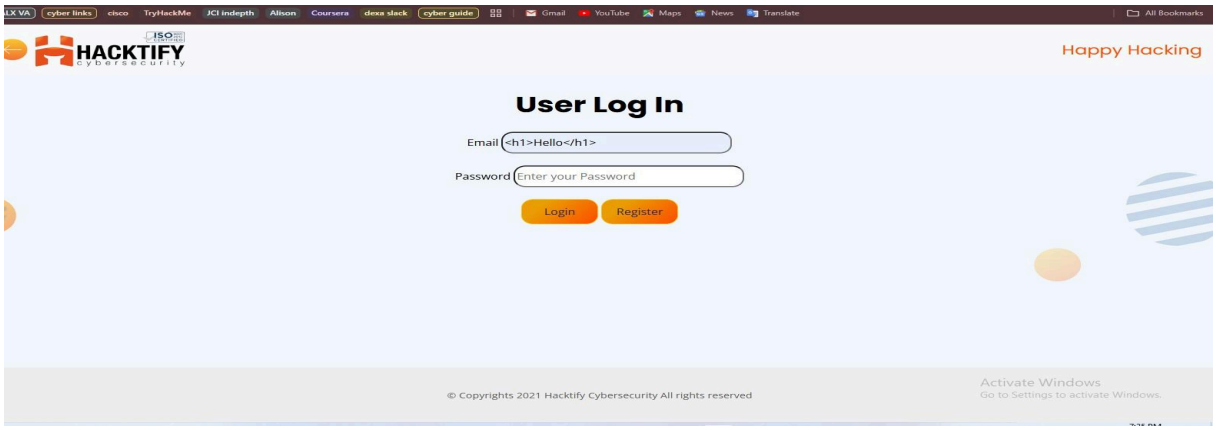
1.2. Let Me Store Them

Reference	Risk Rating
Sub-lab-2:	Low
Tools Used	
Browser(Google Chrome browser), manual testing	
Vulnerability Description	
HTML injection is a type of web security vulnerability that allows an attacker to inject malicious HTML code into a webpage. The website is vulnerable to HTML injection attacks. This means that an attacker can inject malicious HTML code into the website. The vulnerability can be exploited to steal user data, take control of the website, or spread malware.	
How It Was Discovered	
It was discovered through Manual testing of the website's search function. During testing, it was found that the website did not allow user-inputted HTML code to be executed. This was done by inputting the HTML code Hello into the search function, which was then not executed by the website.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_2/profile.php	

Consequences of Not Fixing the Issue
If the vulnerability isn't patched, an attacker can be able to write his own code to get the cookie information of the victim user, this HTML Injection attack leads to an XSS attack. It can lead to data theft, website takeover, malware spread, and reputation damage.
Suggested Countermeasures
1. Implement strict input validation and sanitize user-supplied data. 2. Use contextual output encoding to prevent script execution. 3. Deploy a Content Security Policy (CSP) to restrict script sources. 4. Educate developers on secure coding practices. 5. Regularly audit and test for vulnerabilities.
References
https://portswigger.net/web-security/cross-site-scripting/html-injection https://en.wikipedia.org/wiki/HTML_injection

Proof of Concept

This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab.
Payload: <h1>Hello</h1>



1.3. File names are also vulnerable!

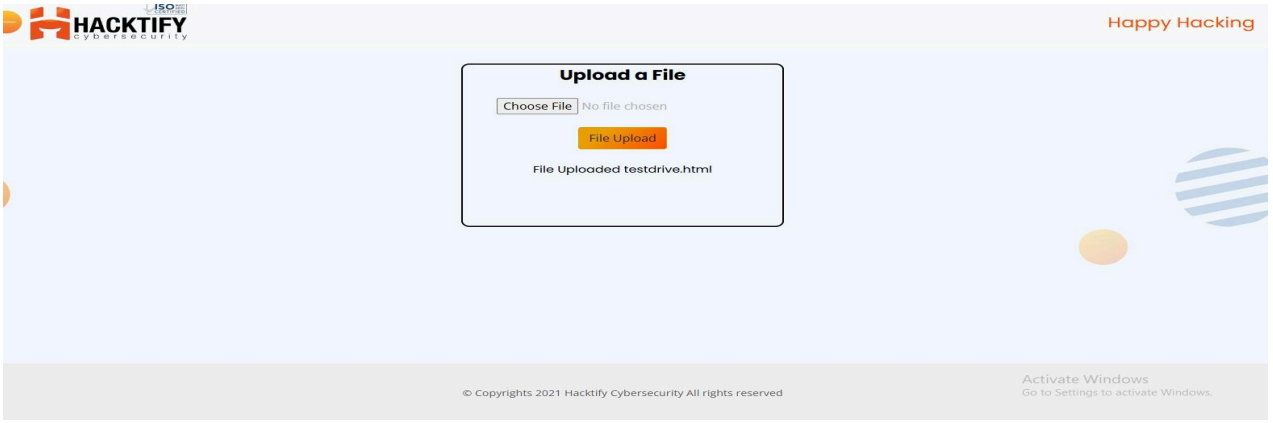
Reference	Risk Rating
Sub-lab-3: File names are also vulnerable!	**Low**
Tools Used	
Browser(Google Chrome browser), manual testing	

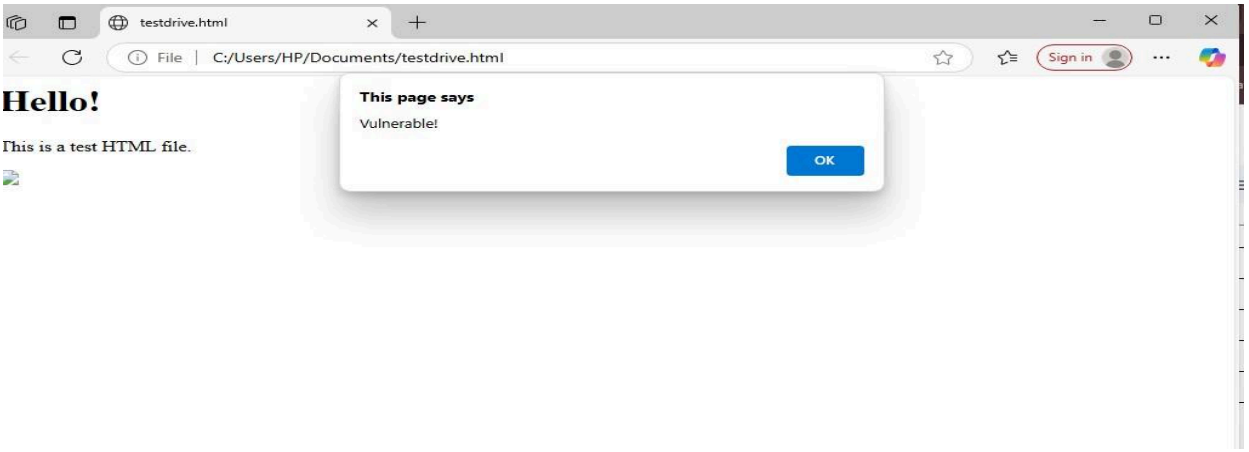
Vulnerability Description
HTML injection is a type of web security vulnerability that allows an attacker to inject malicious HTML code into a webpage. This can occur when user input is not properly validated or sanitized before being included in the webpage's output. Attackers can exploit HTML injection vulnerabilities to manipulate the appearance or functionality of the webpage, steal sensitive information such as login credentials or session cookies, or redirect users to malicious websites. This type of vulnerability can have serious consequences and requires proper input validation and output encoding to mitigate the risk.
How It Was Discovered
Browser View Page Sources (Ctrl + U)
Vulnerable URLs
https://labs.hacktify.in/HTML/html_lab/lab_3/html_injection_3.php
Consequences of Not Fixing the Issue
If the vulnerability isn't patched, an attacker can be able to write his own code to get the cookie information of the victim user, this HTML Injection attack leads to an XSS attack. It can lead to data theft, website takeover, malware spread, and reputation damage.
Suggested Countermeasures
1. Implement strict input validation and sanitize user-supplied data. 2. Use contextual output encoding to prevent script execution. 3. Deploy a Content Security Policy (CSP) to restrict script sources. 4. Educate developers on secure coding practices. 5. Regularly audit and test for vulnerabilities.
References
https://portswigger.net/web-security/cross-site-scripting/html-injection https://en.wikipedia.org/wiki/HTML_injection

Proof of Concept

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

Payload: <h1>Hello!</h1>
This is a test HTML file.
<img src="x" onerror="alert('Vulnerable!');">





1.4. File content and HTML Injection a perfect pair!

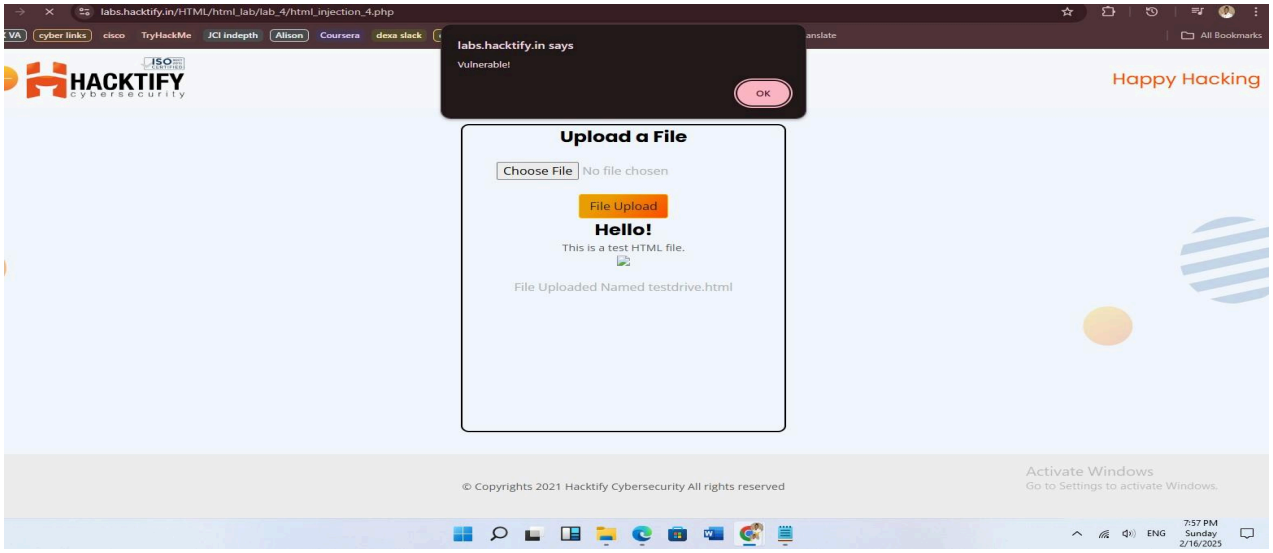
Reference	Risk Rating
Sub-lab-4: File content and HTML Injection are a perfect pair	**Medium**
Tools Used	
Browser(Google Chrome browser), manual testing	
Vulnerability Description	
HTML injection is a type of web security vulnerability that allows an attacker to inject malicious HTML code into a webpage. This can occur when user input is not properly validated or sanitized before being included in the webpage's output. Attackers can exploit HTML injection vulnerabilities to manipulate the appearance or functionality of the webpage, steal sensitive information such as login credentials or session cookies, or redirect users to malicious websites. This type of vulnerability can have serious consequences and requires proper input validation and output encoding to mitigate the risk.	
How It Was Discovered	

Browser View Page Sources (Ctrl + U)
Vulnerable URLs
https://labs.hacktify.in/HTML/html_lab/lab_4/html_injection_4.php
Consequences of Not Fixing the Issue
If the vulnerability isn't patched, an attacker can be able to write his own code to get the cookie information of the victim user, this HTML Injection attack leads to an XSS attack. It can lead to data theft, website takeover, malware spread, and reputation damage.
Suggested Countermeasures
1. Implement strict input validation and sanitize user-supplied data. 2. Use contextual output encoding to prevent script execution. 3. Deploy a Content Security Policy (CSP) to restrict script sources. 4. Educate developers on secure coding practices. 5. Regularly audit and test for vulnerabilities.
References
https://portswigger.net/web-security/cross-site-scripting/html-injection https://en.wikipedia.org/wiki/HTML_injection

Proof of Concept

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

Payload: <h1>Hello!</h1>
This is a test HTML file.
<img src="x" onerror="alert('Vulnerable!');">



1.5. Injecting HTML using URL!

Reference	Risk Rating
Sub-lab-5: Injecting HTML using URL!	**Medium**

Tools Used
Browser(Google Chrome browser), manual testing
Vulnerability Description
HTML injection is a type of web security vulnerability that allows an attacker to inject malicious HTML code into a webpage. This can occur when user input is not properly validated or sanitized before being included in the webpage's output. Attackers can exploit HTML injection vulnerabilities to manipulate the appearance or functionality of the webpage, steal sensitive information such as login credentials or session cookies, or redirect users to malicious websites. This type of vulnerability can have serious consequences and requires proper input validation and output encoding to mitigate the risk.
How It Was Discovered
Browser View Page Sources (Ctrl + U)
Vulnerable URLs
https://labs.hacktify.in/HTML/html_lab/lab_4/html_injection_4.php
Consequences of Not Fixing the Issue
If the vulnerability isn't patched, an attacker can be able to write his own code to get the cookie information of the victim user, this HTML Injection attack leads to an XSS attack.

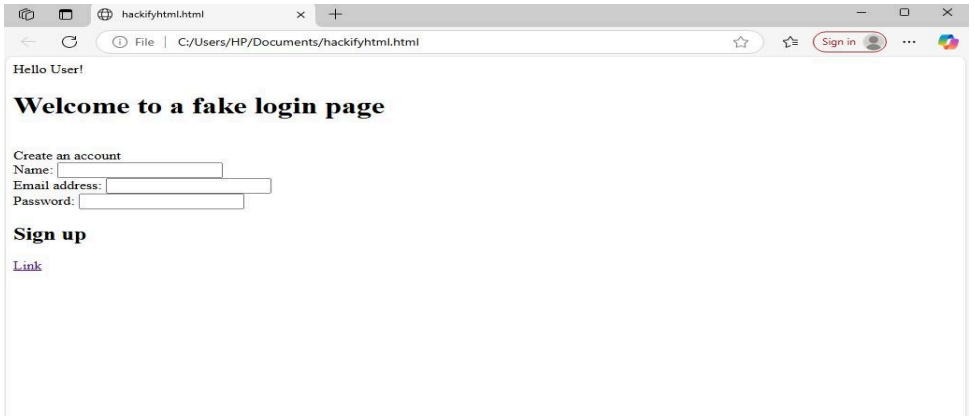
It can lead to data theft, website takeover, malware spread, and reputation damage.
Suggested Countermeasures
1. Implement strict input validation and sanitize user-supplied data. 2. Use contextual output encoding to prevent script execution. 3. Deploy a Content Security Policy (CSP) to restrict script sources. 4. Educate developers on secure coding practices. 5. Regularly audit and test for vulnerabilities.
References
https://portswigger.net/web-security/cross-site-scripting/html-injection https://en.wikipedia.org/wiki/HTML_injection

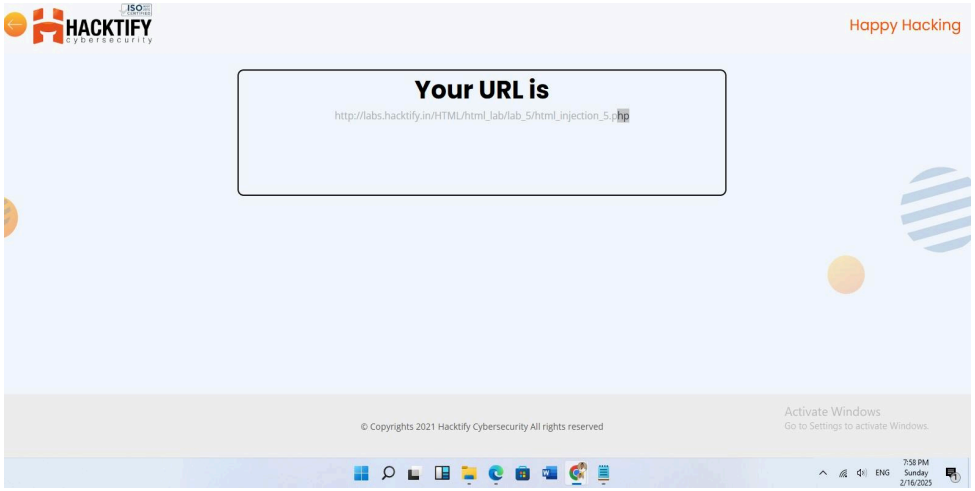
Proof of Concept

This section contains proof of the above vulnerabilities, such as a screenshot of the lab vulnerability.

Payload:

```
<html>
<head>
</head>
<body>
Hello User!
<br>
<h1> Welcome to a fake login page</h1>
<br>
Create an account
<br>
Name: <input type="name">
<br>
Email address: <input type="email">
<br>
Password: <input type="password">
<br>
<h2>Sign up</h2> <a
href="http://labs.hacktify.in/HTML/html_lab/lab_5/html_injection_5.php">Link</a>
</body>
</html>
```





1.6. Encode IT!

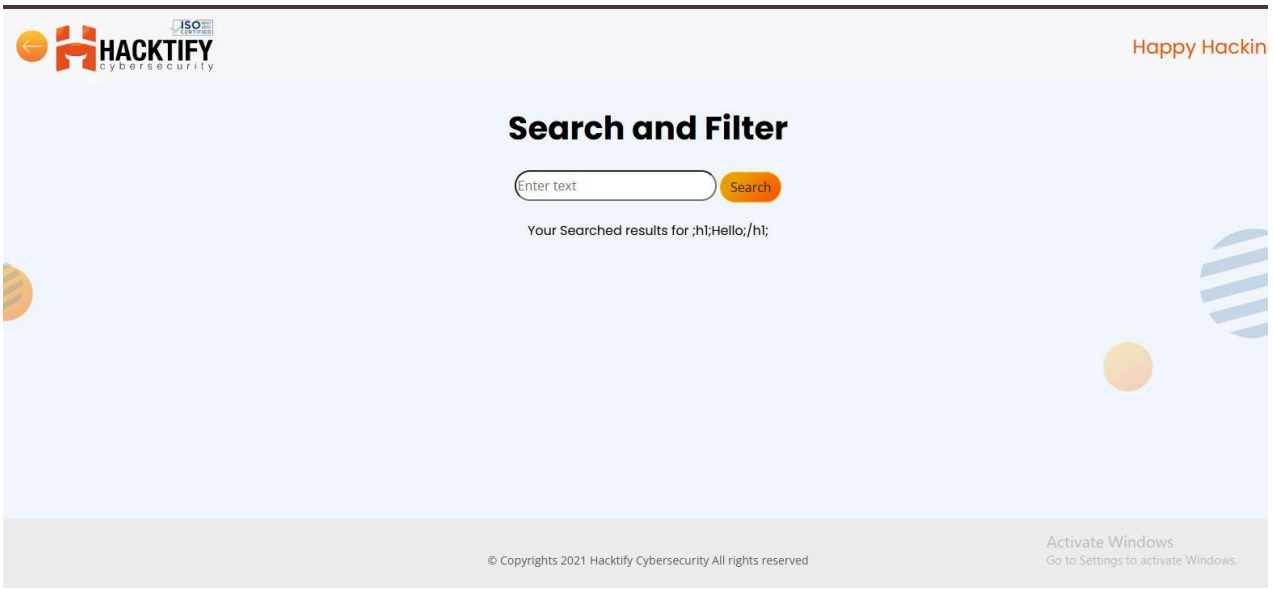
Reference	Risk Rating
Sub-lab-6: Encode IT!	High
Tools Used	
Browser(Google Chrome browser), manual testing	
Vulnerability Description	
HTML injection is a type of web security vulnerability that allows an attacker to inject malicious HTML code into a webpage. This can occur when user input is not properly validated or sanitized before being included in the webpage's output. Attackers can exploit HTML injection vulnerabilities to manipulate the appearance or functionality of the webpage, steal sensitive information such as login credentials or session cookies, or redirect users to malicious websites. This type of vulnerability can have serious consequences and requires proper input validation and output encoding to mitigate the risk.	

How It Was Discovered
Browser View Page Sources (Ctrl + U)
Vulnerable URLs
https://labs.hacktify.in/HTML/html_lab/lab_6/html_injection_6.php
Consequences of Not Fixing the Issue
If the vulnerability isn't patched, an attacker can be able to write his own code to get the cookie information of the victim user, this HTML Injection attack leads to an XSS attack. It can lead to data theft, website takeover, malware spread, and reputation damage.
Suggested Countermeasures
1. Implement strict input validation and sanitize user-supplied data. 2. Use contextual output encoding to prevent script execution. 3. Deploy a Content Security Policy (CSP) to restrict script sources. 4. Educate developers on secure coding practices. 5. Regularly audit and test for vulnerabilities.
References
https://portswigger.net/web-security/cross-site-scripting/html-injection https://en.wikipedia.org/wiki/HTML_injection

Proof of Concept

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

Payload: <h1>Hello</h1>



2. Cross Site Scripting

2.1. Let's Do IT!

Reference	Risk Rating
Sub-lab-1: **Let's Do IT!**	Low

Tools Used
Google Chrome Browser

Vulnerability Description
Cross-Site Scripting (XSS) is a security flaw in web applications that lets attackers sneak malicious scripts into web pages seen by other users. This happens when a website displays user-provided information without properly checking it first. In this lab, a "reflected" XSS vulnerability was found. Reflected XSS means the malicious script isn't saved on the website itself; it's only active when a user clicks a specially crafted link or submits a form. Attackers often use this type of XSS for phishing attacks, trying to trick users into giving up their login details or other sensitive information.

How It Was Discovered
Manual testing

Vulnerable URLs
https://labs.hacktify.in/HTML/xss_lab/lab_1/lab_1.php https://labs.hacktify.in/HTML/xss_lab/lab_1/lab_1.php?email=%3Cscript%3Ealert%28%27Hack.me%27%29%3C%2Fscript%3E

Consequences of not Fixing the Issue
Consequences of not Fixing the Issue If this vulnerability is not patched. Malicious scripts could be executed as soon as the user interacts with the

Suggested Countermeasures
Filter input on arrival: At the point where user input is received, filter as strictly as possible based on what is expected or valid input. Encode data on output: At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding. Use appropriate response headers: To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type- Options headers to ensure that browsers interpret the responses in the way you intend. Content Security Policy: As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur

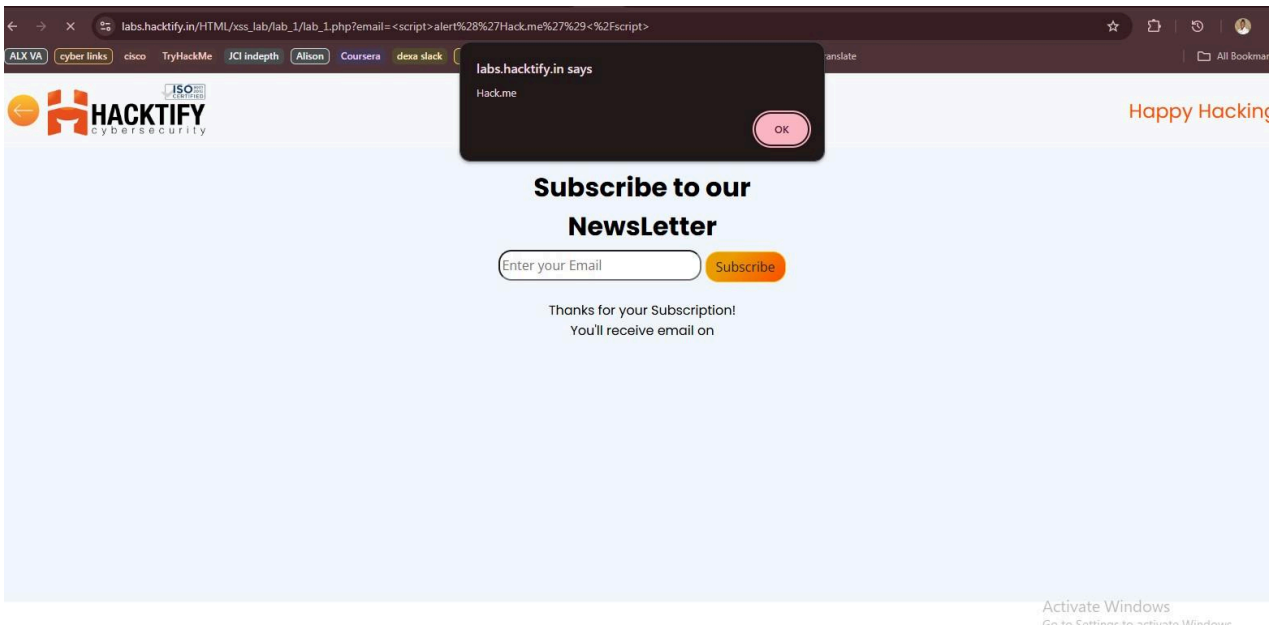
References

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

Payload: <script>alert(Hack.me)</script/>



2.2. Balancing is important in Life!

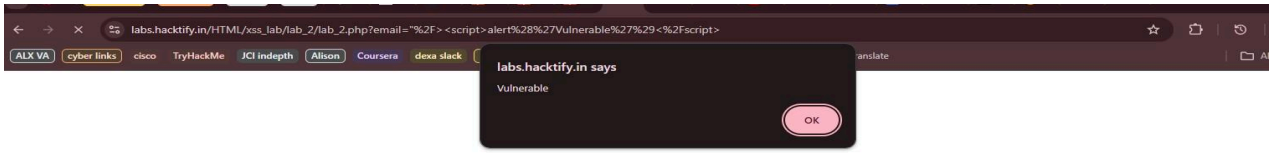
Reference	Risk Rating
Sub-lab-2: Balancing is important in Life!	Low
Tools Used	
Google Chrome Browser	
Vulnerability Description	
Cross-Site Scripting (XSS) is a security flaw in web applications that lets attackers sneak malicious scripts into web pages seen by other users. This happens when a website displays user-provided information without properly checking it first. In this lab, a "reflected" XSS vulnerability was found. Reflected XSS means the malicious script isn't saved on the website itself; it's only active when a user clicks a specially crafted link or submits a form. Attackers often use this type of XSS for phishing attacks, trying to trick users into giving up their login details or other sensitive information.	
How It Was Discovered	

Manual testing
Vulnerable URLs
https://labs.hacktify.in/HTML/xss_lab/lab_2/lab_2.php https://labs.hacktify.in/HTML/xss_lab/lab_2/lab_2.php?email=%22%2F%3E%3Cscript%3Ealert%28%27Vulnerable%27%29%3C%2Fscript%3E
Consequences of not Fixing the Issue
Consequences of not Fixing the Issue If this vulnerability is not patched. Malicious scripts could be executed as soon as the user interacts with the
Suggested Countermeasures
Filter input on arrival: At the point where user input is received, filter as strictly as possible based on what is expected or valid input. Encode data on output: At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding. Use appropriate response headers: To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type- Options headers to ensure that browsers interpret the responses in the way you intend. Content Security Policy: As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur
References
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.

Payload: “/><script>alert(‘Vulnerable’)</script>



2.3. XSS Is Everywhere!

Reference	Risk Rating
Sub-lab-3: XSS Is Everywhere!	Low

Tools Used

Google Chrome Browser

Vulnerability Description

Cross-Site Scripting (XSS) is a security flaw in web applications that lets attackers sneak malicious scripts into web pages seen by other users. This happens when a website displays user-provided information without properly checking it first. In this lab, a "reflected" XSS vulnerability was found. Reflected XSS means the malicious script isn't saved on the website itself; it's only active when a user clicks a specially crafted link or submits a form. Attackers often use this type of XSS for phishing attacks, trying to trick users into giving up their login details or other sensitive information.

How It Was Discovered

Manual testing

Vulnerable URLs

https://labs.hacktify.in/HTML/xss_lab/lab_3/lab_3.php
https://labs.hacktify.in/HTML/xss_lab/lab_3/lab_3.php?email=%22%2F%3E%3Cscript%3Ealert%28%27Vulnerable%27%29%3C%2Fscript%3E

Consequences of not Fixing the Issue

Consequences of not Fixing the Issue If this vulnerability is not patched. Malicious scripts could be executed as soon as the user interacts with the

Suggested Countermeasures

Filter input on arrival: At the point where user input is received, filter as strictly as possible based on what is expected or valid input.
Encode data on output: At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding.
Use appropriate response headers: To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type- Options headers to ensure that browsers interpret the responses in the way you intend.
Content Security Policy: As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur

References

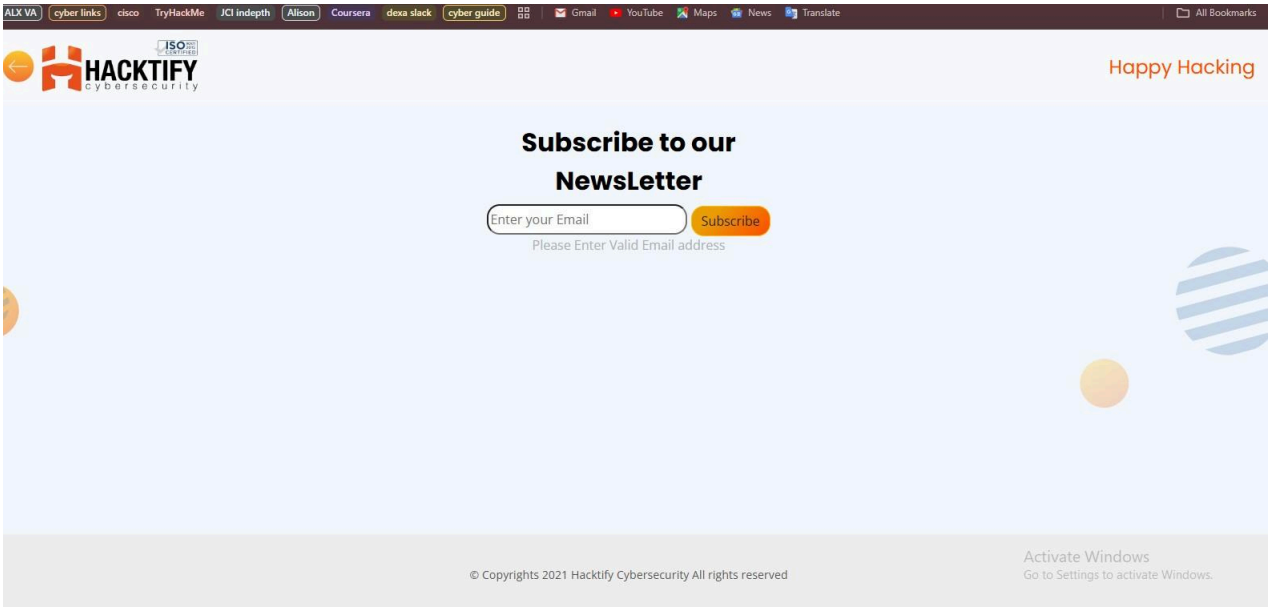
https://portswigger.net/web-security/cross-site-scripting

Proof of Concept

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

Payload: “/><script>alert(‘Vulnerable’)</script>

This shows that an attempt was made to inject malicious code into the email address field using the payload. However, the website responded with an error message saying “Please enter a valid email address” indicating that the input was not accepted. This suggests that the website has some level of protection against Reflected XSS attacks.



2.4. Alternatives Are A Must!

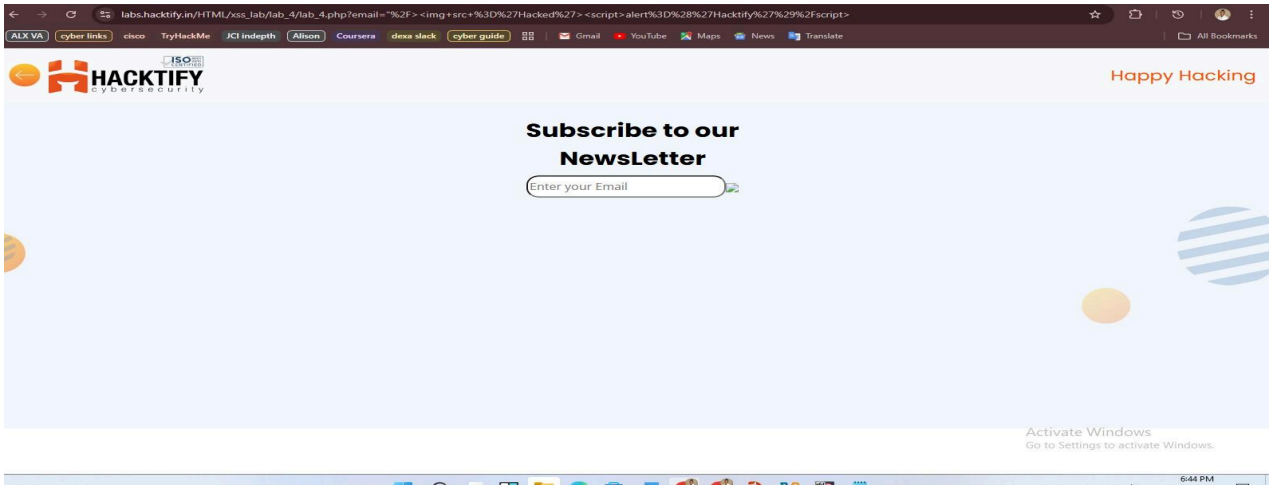
Reference	Risk Rating
Sub-lab-4: Alternatives Are A Must!	Medium
Tools Used	
Google Chrome Browser	
Vulnerability Description	
Cross-Site Scripting (XSS) is a security flaw in web applications that lets attackers sneak malicious scripts into web pages seen by other users. This happens when a website displays user-provided information without properly checking it first. In this lab, a "reflected" XSS vulnerability was found. Reflected XSS means the malicious script isn't saved on the website itself; it's only active when a user clicks a specially crafted link or submits a form. Attackers often use this type of XSS for phishing attacks, trying to trick users into giving up their login details or other sensitive information.	

How It Was Discovered
Manual testing
Vulnerable URLs
https://labs.hacktify.in/HTML/xss_lab/lab_4/lab_4.php https://labs.hacktify.in/HTML/xss_lab/lab_4/lab_4.php?email=%22%2F%3E%3Cimg+src+%3D%27Hacked%27%3E%3Cscript%3Ealert%3D%28%27Hacktify%27%29%2Fscript%3E
Consequences of not Fixing the Issue
Consequences of not Fixing the Issue If this vulnerability is not patched. Malicious scripts could be executed as soon as the user interacts with the
Suggested Countermeasures
Filter input on arrival: At the point where user input is received, filter as strictly as possible based on what is expected or valid input. Encode data on output: At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding. Use appropriate response headers: To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type- Options headers to ensure that browsers interpret the responses in the way you intend. Content Security Policy: As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur
References
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.

Payload: "/><img src ='Hacked'><script>alert=('Hacktify')/script>



2.5. Developer Hates Scripts!

Reference	Risk Rating
Sub-lab-5: Developer Hates Scripts!	**Low**

Tools Used

Google Chrome Browser

Vulnerability Description

Cross-Site Scripting (XSS) is a security flaw in web applications that lets attackers sneak malicious scripts into web pages seen by other users. This happens when a website displays user-provided information without properly checking it first. In this lab, a "reflected" XSS vulnerability was found. Reflected XSS means the malicious script isn't saved on the website itself; it's only active when a user clicks a specially crafted link or submits a form. Attackers often use this type of XSS for phishing attacks, trying to trick users into giving up their login details or other sensitive information.

How It Was Discovered

Manual testing

Vulnerable URLs

https://labs.hacktify.in/HTML/xss_lab/lab_5/lab_5.php
https://labs.hacktify.in/HTML/xss_lab/lab_5/lab_5.php?email=%22%2F%3E%3Cscript%3Ealert%28%27Vulnerable%27%29%3C%2Fscript%3E

Consequences of not Fixing the Issue

Consequences of not Fixing the Issue If this vulnerability is not patched. Malicious scripts could be executed as soon as the user interacts with the

Suggested Countermeasures

Filter input on arrival: At the point where user input is received, filter as strictly as possible based on what is expected or valid input.
Encode data on output: At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding.
Use appropriate response headers: To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type- Options headers to ensure that browsers interpret the responses in the way you intend.
Content Security Policy: As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur

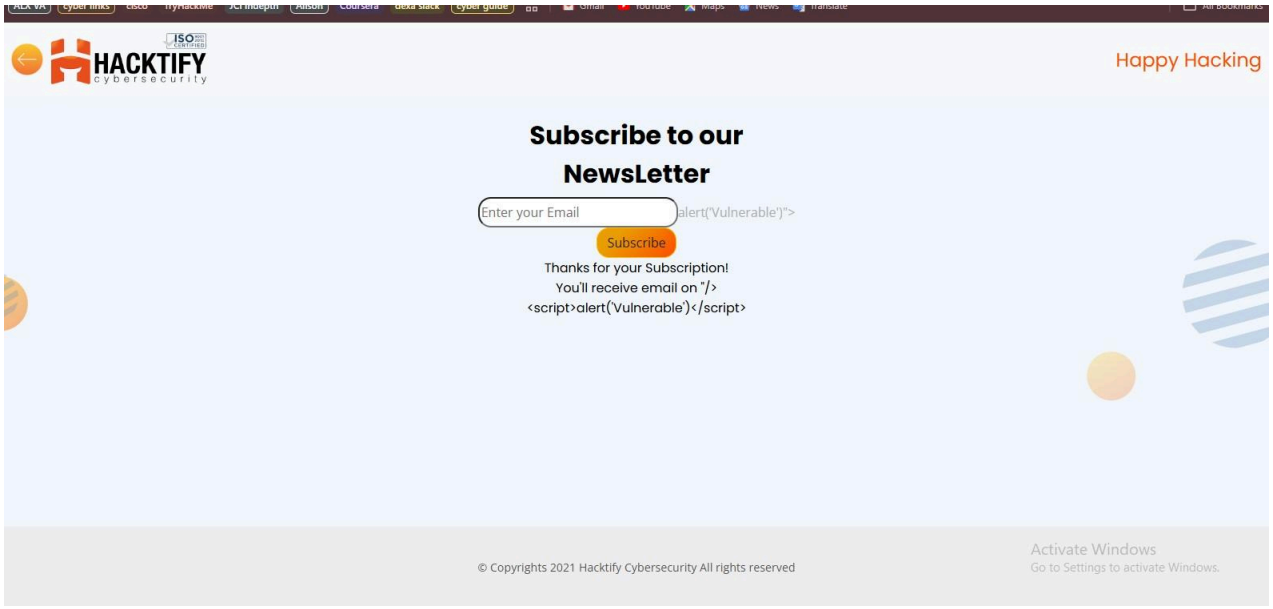
References

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.

payload:"/><script>alert('Vulnerable')</script>



2.6. Change The Variation

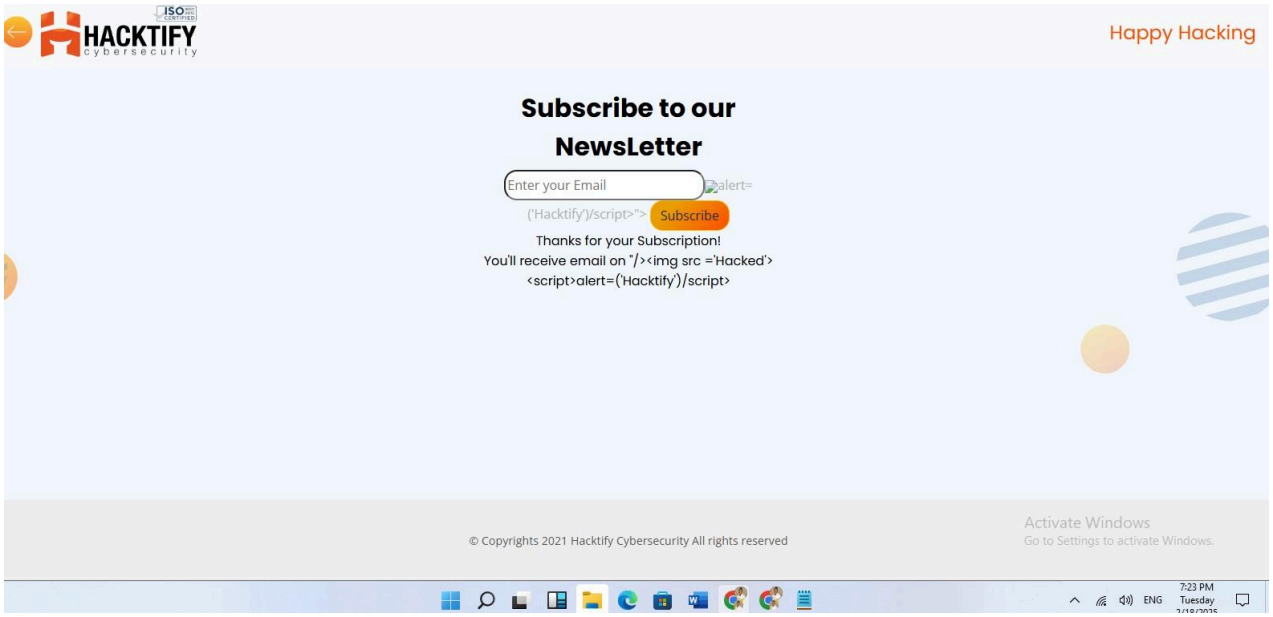
Reference	Risk Rating
Sub-lab-6: Change The Variation	Low
Tools Used	
Google Chrome Browser	
Vulnerability Description	
Cross-Site Scripting (XSS) is a security flaw in web applications that lets attackers sneak malicious scripts into web pages seen by other users. This happens when a website displays user-provided information without properly checking it first. In this lab, a "reflected" XSS vulnerability was found. Reflected XSS means the malicious script isn't saved on the website itself; it's only active when a user clicks a specially crafted link or submits a form. Attackers often use this type of XSS for phishing attacks, trying to trick users into giving up their login details or other sensitive information.	
How It Was Discovered	
Manual testing	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_6/lab_6.php https://labs.hacktify.in/HTML/xss_lab/lab_6/lab_6.php?email=%22%2F%3E%3Cimg+src+%3D%27Hacked%27%3E%3Cscript%3Ealert%3D%28%27Hacktify%27%29%2Fscript%3E	

Consequences of not Fixing the Issue
Consequences of not Fixing the Issue If this vulnerability is not patched. Malicious scripts could be executed as soon as the user interacts with the
Suggested Countermeasures
Filter input on arrival: At the point where user input is received, filter as strictly as possible based on what is expected or valid input. Encode data on output: At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding. Use appropriate response headers: To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type- Options headers to ensure that browsers interpret the responses in the way you intend. Content Security Policy: As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur
References
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.

Payload: **"/><img src ='Hacked'><script>alert=('Hacktify')/script>**



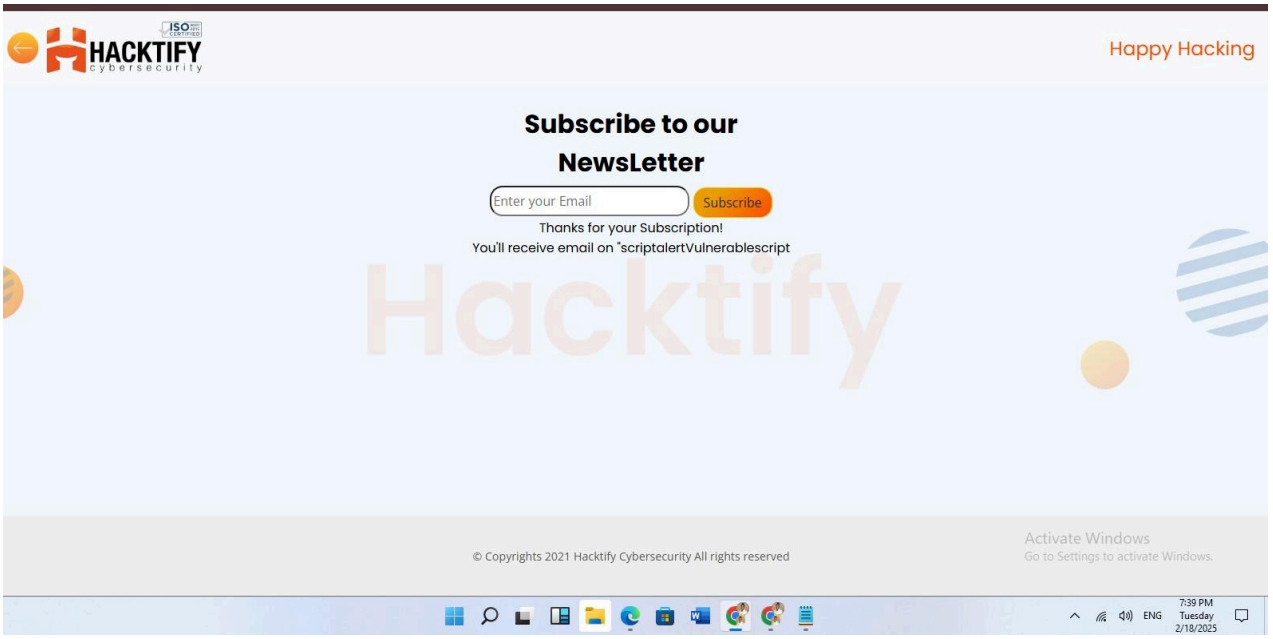
2.7. Encoding is Key?

Reference	Risk Rating
Sub-lab-7: Encoding is key?	Medium
Tools Used	
Google Chrome Browser	
Vulnerability Description	
Cross-Site Scripting (XSS) is a security flaw in web applications that lets attackers sneak malicious scripts into web pages seen by other users. This happens when a website displays user-provided information without properly checking it first. In this lab, a "reflected" XSS vulnerability was found. Reflected XSS means the malicious script isn't saved on the website itself; it's only active when a user clicks a specially crafted link or submits a form. Attackers often use this type of XSS for phishing attacks, trying to trick users into giving up their login details or other sensitive information.	
How It Was Discovered	
Manual testing	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_7/lab_7.php https://labs.hacktify.in/HTML/xss_lab/lab_7/lab_7.php?email=%22%2F%3E%3Cscript%3Ealert%28%27Vulnerable%27%29%3C%2Fscript%3E	
Consequences of not Fixing the Issue	
Consequences of not Fixing the Issue If this vulnerability is not patched. Malicious scripts could be executed as soon as the user interacts with the	
Suggested Countermeasures	
Filter input on arrival: At the point where user input is received, filter as strictly as possible based on what is expected or valid input. Encode data on output: At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding. Use appropriate response headers: To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type- Options headers to ensure that browsers interpret the responses in the way you intend. Content Security Policy: As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur	
References	

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.

Payload: "/><script>alert('Vulnerable')</script>



1.8. XSS with file upload (file name)!

Reference	Risk Rating
Sub-lab-8: XSS with file upload (file name)!	Low
Tools Used	
Google Chrome Browser	
Vulnerability Description	

Cross-Site Scripting (XSS) is a security flaw in web applications that lets attackers sneak malicious scripts into web pages seen by other users. This happens when a website displays user-provided information without properly checking it first. In this lab, a "reflected" XSS vulnerability was found. Reflected XSS means the malicious script isn't saved on the website itself; it's only active when a user clicks a specially crafted link or submits a form. Attackers often use this type of XSS for phishing attacks, trying to trick users into giving up their login details or other sensitive information.
How It Was Discovered
Manual testing
Vulnerable URLs
https://labs.hacktify.in/HTML/xss_lab/lab_8/lab_8.php
Consequences of not Fixing the Issue
Consequences of not Fixing the Issue If this vulnerability is not patched. Malicious scripts could be executed as soon as the user interacts with the
Suggested Countermeasures
Filter input on arrival: At the point where user input is received, filter as strictly as possible based on what is expected or valid input. Encode data on output: At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding. Use appropriate response headers: To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type- Options headers to ensure that browsers interpret the responses in the way you intend. Content Security Policy: As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur
References
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.

Payload: <html>

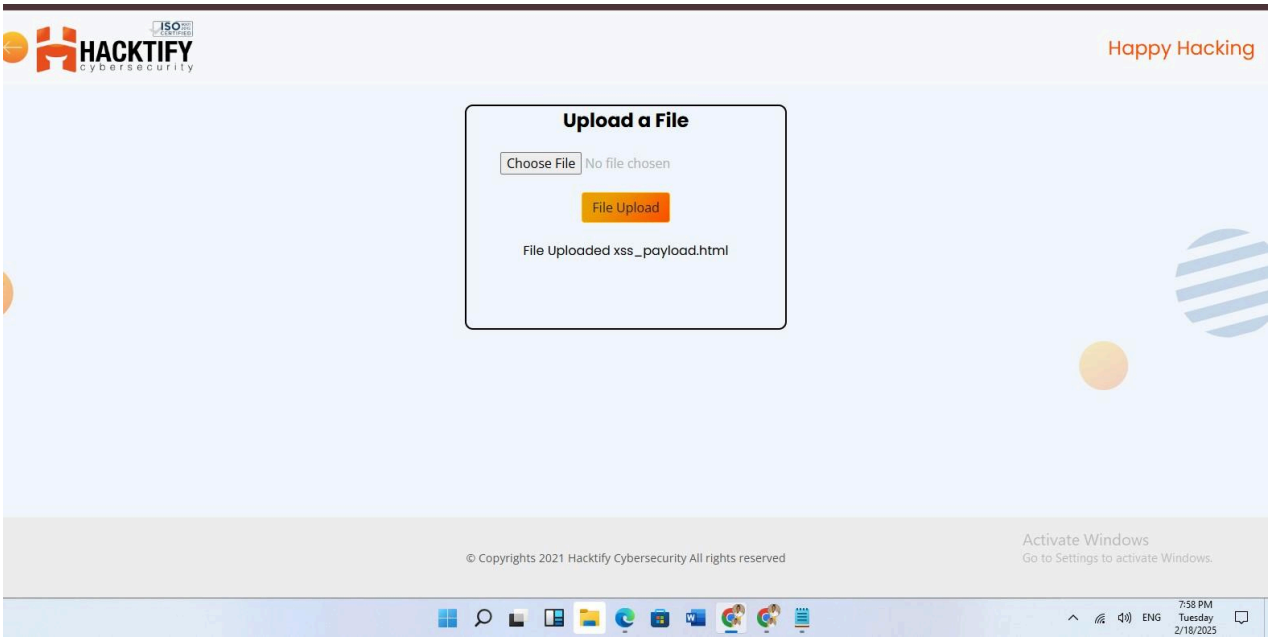
<body>

<script>alert('XSS');

alert('Vulnerable');</script>

</body>

</html>



2.9. XSS with File Upload (file content)!

Reference	Risk Rating
Sub-lab-9: XSS with File Upload (file content)!	High
Tools Used	
Google Chrome Browser	
Vulnerability Description	
Cross-Site Scripting (XSS) is a security flaw in web applications that lets attackers sneak malicious scripts into web pages seen by other users. This happens when a website displays user-provided information without properly checking it first. In this lab, a "reflected" XSS vulnerability was found. Reflected XSS means the malicious script isn't saved on the website itself; it's only active when a user clicks a specially crafted link or submits a form. Attackers often use this type of XSS for phishing attacks, trying to trick users into giving up their login details or other sensitive information.	
How It Was Discovered	
Manual testing	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_9/lab_9.php	

Consequences of not Fixing the Issue
Consequences of not Fixing the Issue If this vulnerability is not patched. Malicious scripts could be executed as soon as the user interacts with the
Suggested Countermeasures
Filter input on arrival: At the point where user input is received, filter as strictly as possible based on what is expected or valid input. Encode data on output: At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding. Use appropriate response headers: To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type- Options headers to ensure that browsers interpret the responses in the way you intend. Content Security Policy: As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur
References
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.

Payload: <html>

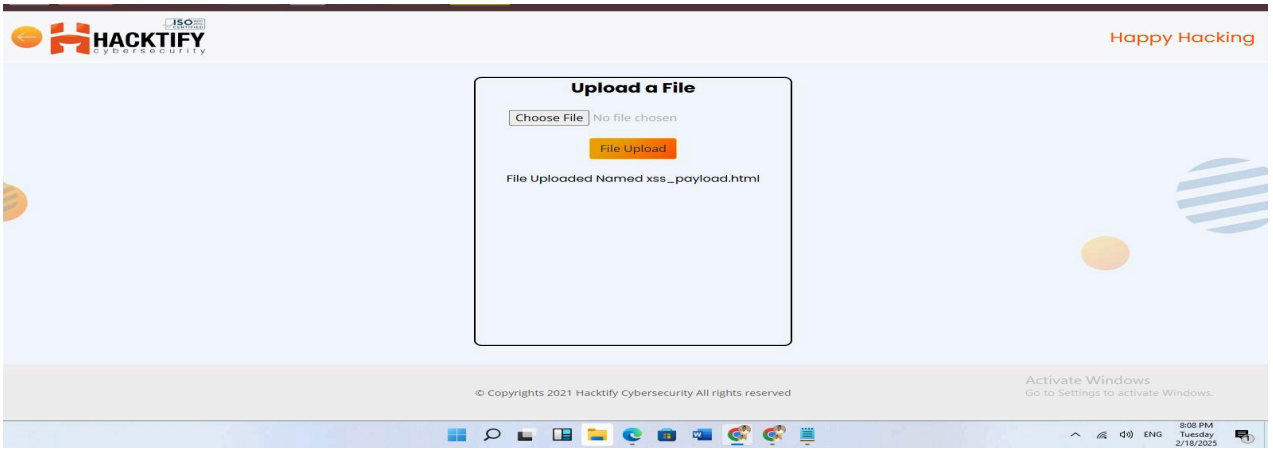
<body>

<script>alert('XSS');

alert('Vulnerable');</script>

</body>

</html>



2.10 Stored Everywhere!

Reference	Risk Rating
Sub-lab-10: Stored Everywhere!	Low

Tools Used
Google Chrome Browser

Vulnerability Description
Cross-Site Scripting (XSS) is a security flaw in web applications that lets attackers sneak malicious scripts into web pages seen by other users. This happens when a website displays user-provided information without properly checking it first. In this lab, a "reflected" XSS vulnerability was found. Reflected XSS means the malicious script isn't saved on the website itself; it's only active when a user clicks a specially crafted link or submits a form. Attackers often use this type of XSS for phishing attacks, trying to trick users into giving up their login details or other sensitive information.

How It Was Discovered
Manual testing

Vulnerable URLs
https://labs.hacktify.in/HTML/xss_lab/lab_10/profile.php

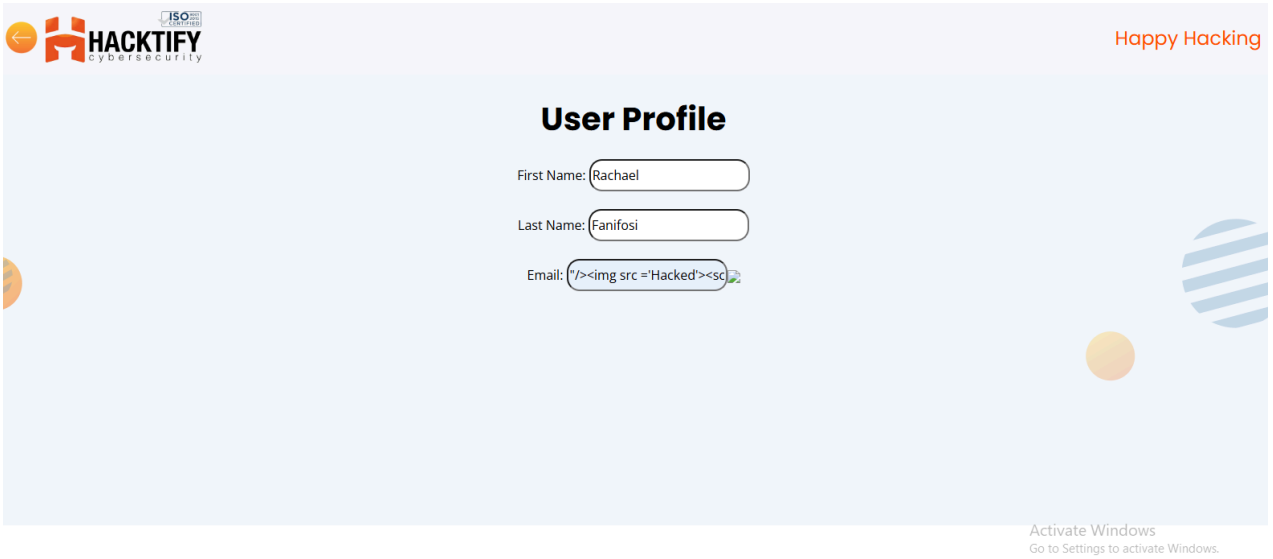
Consequences of not Fixing the Issue
Consequences of not Fixing the Issue If this vulnerability is not patched. Malicious scripts could be executed as soon as the user interacts with the

Suggested Countermeasures
Filter input on arrival: At the point where user input is received, filter as strictly as possible based on what is expected or valid input. Encode data on output: At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding. Use appropriate response headers: To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type- Options headers to ensure that browsers interpret the responses in the way you intend. Content Security Policy: As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur

References
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.

Payload: "/><img src ='Hacked'><script>alert=('Hacktify')/script>



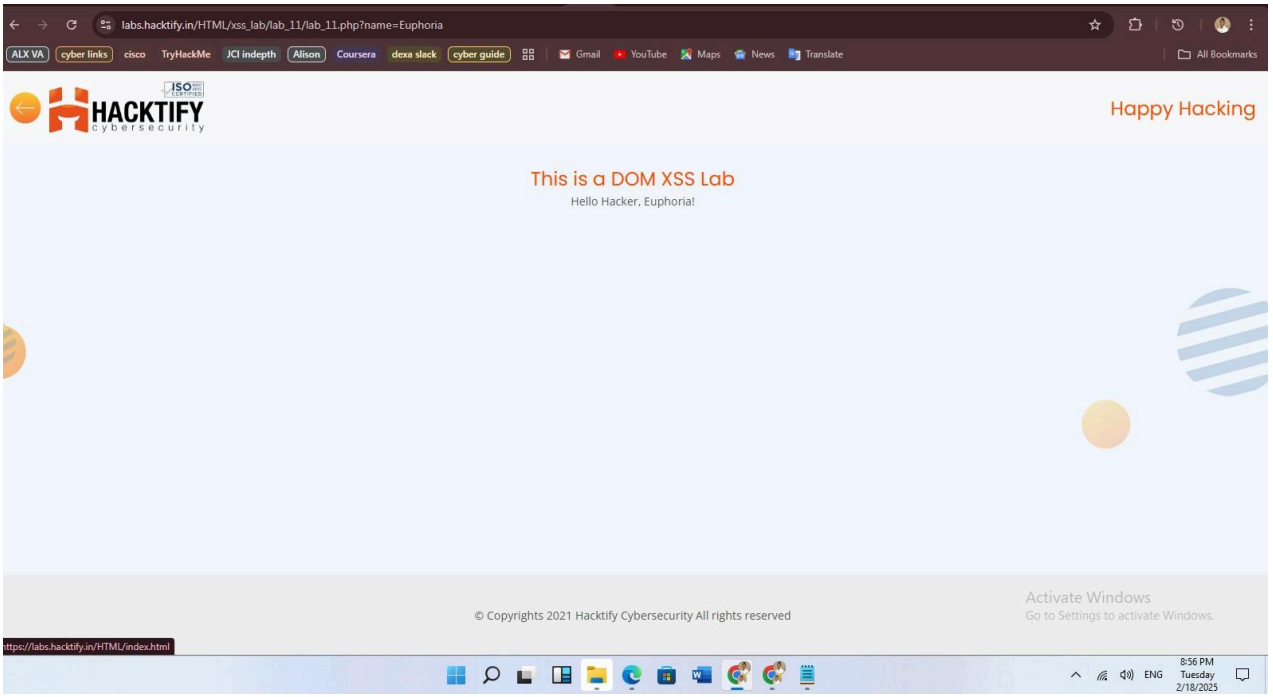
2.11. DOM's are love!

Reference	Risk Rating
Sub-lab-11: DOM's are love!	Hard
Tools Used	
Google Chrome Browser	
Vulnerability Description	
Cross-Site Scripting (XSS) is a security flaw in web applications that lets attackers sneak malicious scripts into web pages seen by other users. This happens when a website displays user-provided information without properly checking it first. In this lab, a "reflected" XSS vulnerability was found. Reflected XSS means the malicious script isn't saved on the website itself; it's only active when a user clicks a specially crafted link or submits a form. Attackers often use this type of XSS for phishing attacks, trying to trick users into giving up their login details or other sensitive information.	
How It Was Discovered	
Manual testing	
Vulnerable URLs	

https://labs.hacktify.in/HTML/xss_lab/lab_11/lab_11.php https://labs.hacktify.in/HTML/xss_lab/lab_11/lab_11.php?name=Euphoria
Consequences of not Fixing the Issue
Consequences of not Fixing the Issue If this vulnerability is not patched. Malicious scripts could be executed as soon as the user interacts with the
Suggested Countermeasures
Filter input on arrival: At the point where user input is received, filter as strictly as possible based on what is expected or valid input. Encode data on output: At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding. Use appropriate response headers: To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type- Options headers to ensure that browsers interpret the responses in the way you intend. Content Security Policy: As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur
References
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.

Payload: https://labs.hacktify.in/HTML/xss_lab/lab_11/lab_11.php?name=Euphoria


labs.hacktify.in/HTML/xss_lab/lab_11/lab_11.php?name=Euphoria
ALX VA
cyber links
cisco
TryHackMe
JCI indepth
Alison
Coursera
dexa slack
cyber guide
Gmail
YouTube
Maps
News
Translate
All Bookmarks
HACKTIFY
cybersecurity
Happy Hacking
This is a DOM XSS Lab
Hello Hacker, Euphoria!
© Copyrights 2021 Hacktify Cybersecurity All rights reserved
Activate Windows
Go to Settings to activate Windows.
https://labs.hacktify.in/HTML/index.html
ENG
8:56 PM
Tuesday
2/18/2025