Web Penetration Testing Report

Full Name: Geshom Lukoshi
Program: HCPT
Date: 19th February, 2025

Introduction

This report document hereby describes the proceedings and results of the web site vulnerabilities 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

The scope of the penetration testing labs by Hactify Cyber Security for HTML injection and XSS – cross site scripting includes identifying vulnerabilities in web site. The testing will focus on detecting XSS vulnerabilities that could lead to unauthorized actions by users. HTML injection point entries will be assessed to identify potential avenues for malicious code insertion. The boundaries of the labs exclude testing of the backend systems and network infrastructure. The results will be provided with recommendations for mitigation to enhance the web site security posture.

Application Name	**Lab 1 – XSS (Cross Site Scripting)** **Lab 2 – HTML Injection**

3. Summary

Outlined is the Hackitify web site Security assessment for the **Week 1 Labs**.

Total number of Sub-labs: (17) Sub-labs

High	Medium	Low
4	4	9

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. Lab 1 HTML Injection

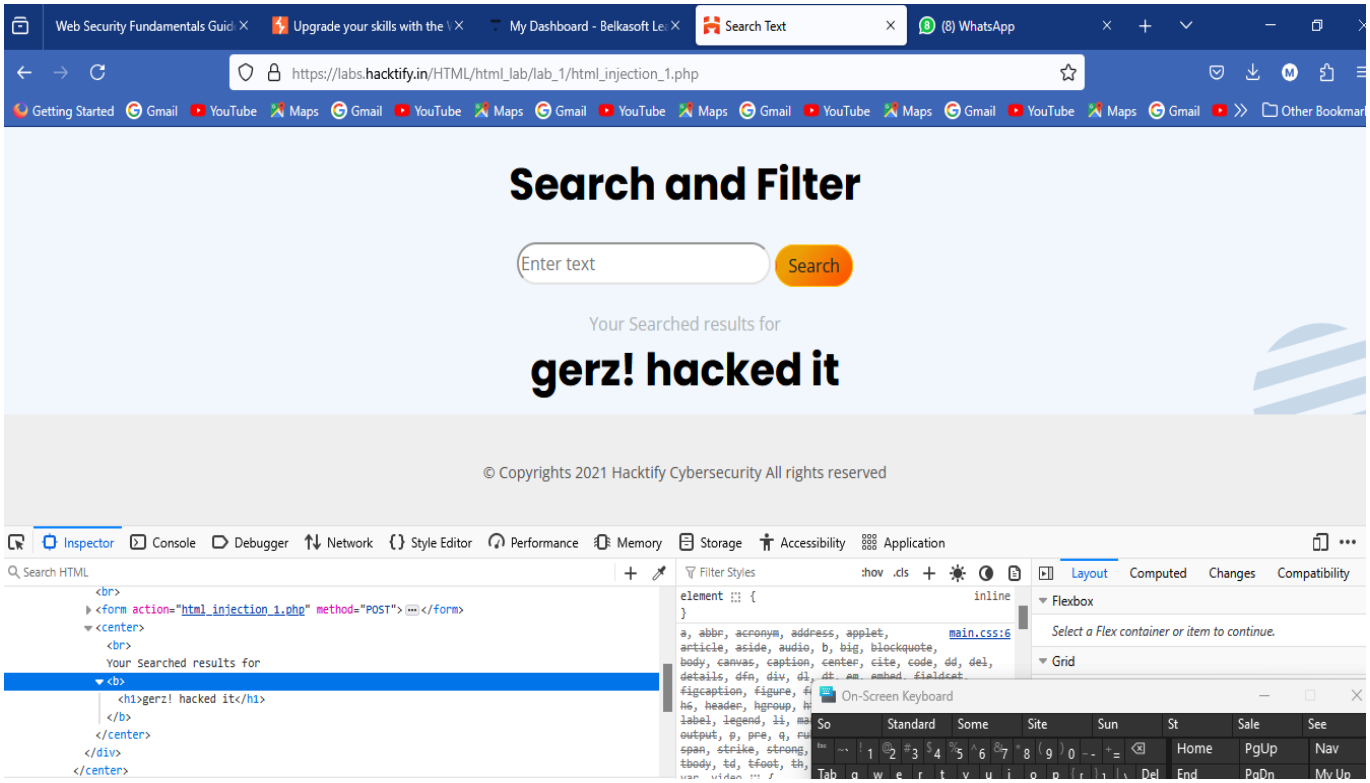
1.1. Sub-lab-1 HTML are easy

Reference	**Risk Rating**
Sub-lab-1 HTML are easy	**Low**
Tools Used	
Browser "View Page Source" is used to find the vulnerability.	
Vulnerability Description	
HTML injection is a type of web security Vulnerability that let attackers 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	
Automated Tools – Browser View Source Page	
Vulnerable URLs	
https://labs.hackify.in/HTML/html_lab/lab_1/html_injection_1.php	
Consequences of not Fixing the Issue	
1. It can Lead to an XSS attack 2. Can Lead to Cross-Site Request Forgery 3. Compliance Issues 4. Data Breaches (data theft) 5. Increased Risk of Future Attacks	
Suggested Countermeasures	
1. Implement a strict input validation and sanitize user-supplied data. 2. Deploy Content Security Policy (CSP) to restrict input sources. 3. Encoding (HTML encoding, URL encoding etc.) 4. Educate developers on secure coding practices 5. Regularly audit and test for vulnerabilities is a must	
References	

https://owasp.org
https://www.acunetix.com
https://github.com
https://www.imperva.com

Proof of Concept

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



1.2. Sub-lab-2 Let me store them!

Reference	Risk Rating
Sub-lab-2: Let me store them	**Low**
Tools Used	
Browser tools “By using the inspector tool, I inspected the source code of the page, and by balancing the script the payload got executed”	
Vulnerability Description	

HTML injection is a type of web security Vulnerability that let attackers inject malicious HTML code into a webpage. This can occur when a user input is not properly validated or sanitized before being included in the webpage's output. When this happens 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. In this regard the vulnerability was discovered after trying different entry points, then the source code of the page was inspected using a browser tool, the value placeholder in the source code was observed to be equal to the input I supplied in the field. By balancing the script with the ("/>) followed by the scripts, the input got stored on the client-side showing the script worked. This type of vulnerability can have serious consequences and requires proper input validation and output encoding to mitigate the risk.

How It Was Discovered

Automated Tools "By inspecting the source code of the page using the inspector tool of the browser after trying with the entry points of injecting".

Vulnerable URLs

https://labs.hacktify.in/HTML/html_lab/lab_2/profile.php

Consequences of not Fixing the Issue

1. Business Disruption
2. Increased Risk of Future Attacks
3. Malware Distribution
4. Data Theft (breaches)
5. Intellectual Property theft
6. Website Defacement

Suggested Countermeasures

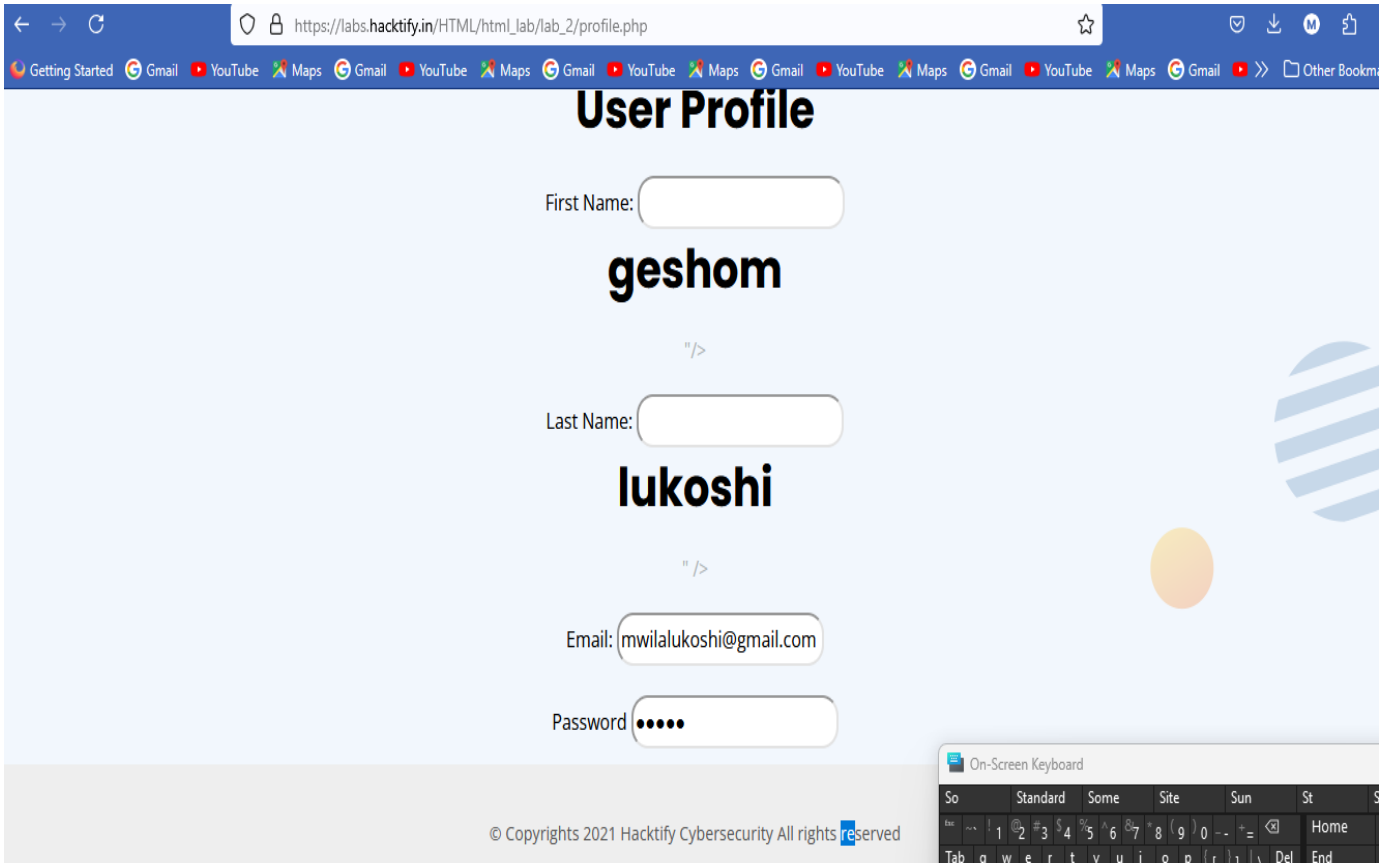
1. Input Validation and Sanitization
2. Implement a Content Security Policy (CSP)
3. Regular Security Audits
4. By deploying a Web Application Firewall

References

https://owasp.org
https://www.acunetix.com
https://github.com
https://www.imperva.com
https://www.wallarm.com

Proof of Concept

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



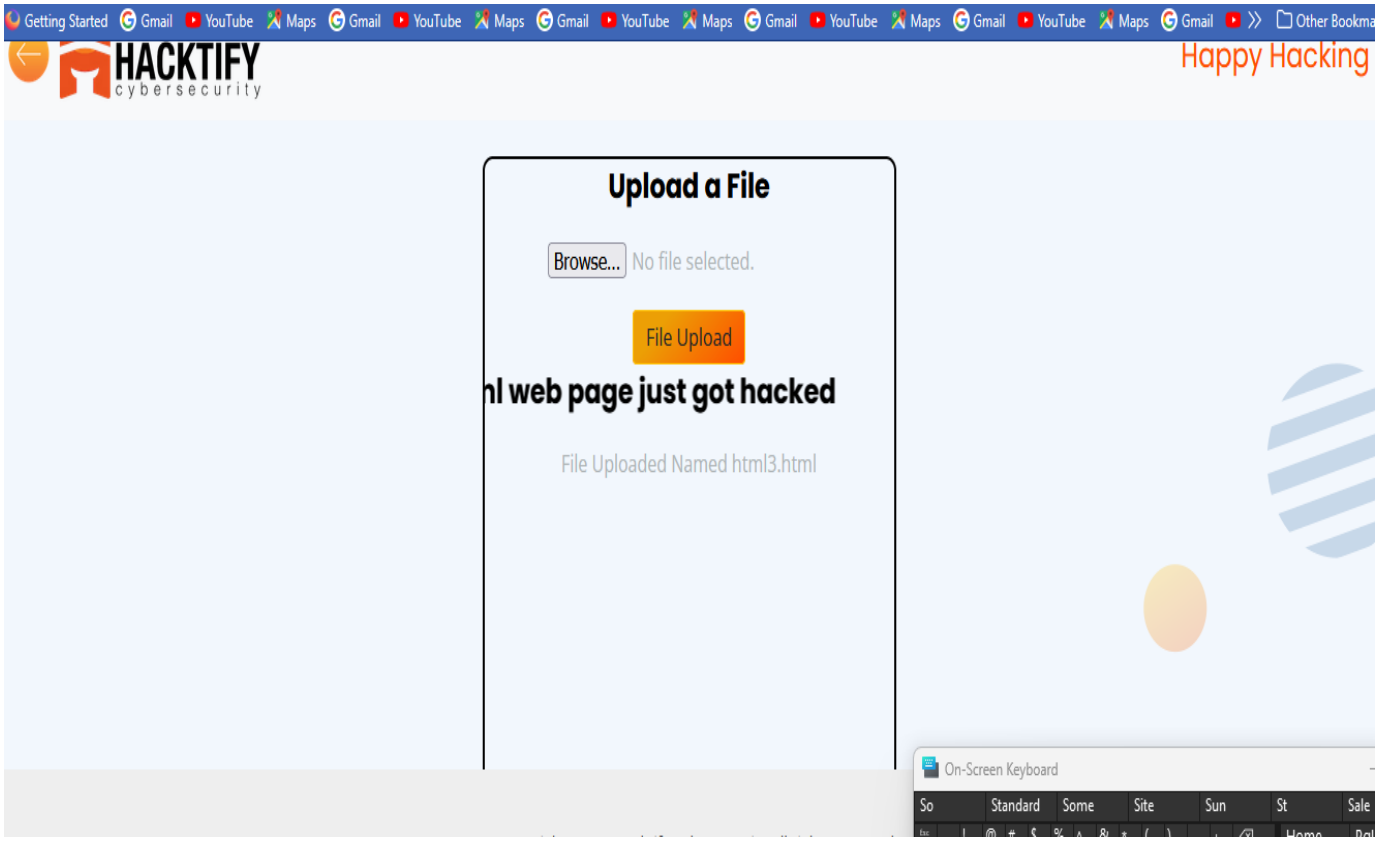
1.3. Sub-lab-3 File Names are also vulnerable!

Reference	Risk Rating
Sub-lab-2: File names are also vulnerable	**Low**
Tools Used	
Tools that you have used to find the vulnerability.	
Vulnerability Description	
HTML injection is a type of web security Vulnerability that let attackers inject malicious HTML code into a webpage. This can occur when a user input is not properly validated or sanitized before being included in the webpage's output. When this happens 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. In this regard the vulnerability was discovered after trying different entry points, then the source code of the page was inspected using a browser tool, the value placeholder in the source code was observed to be equal to the input I supplied in the field. By balancing the script with the ("/>) followed by the scripts, the input got stored on the client-side showing the script worked.	
How It Was Discovered	
Automated Tools / Manual Analysis	
Vulnerable URLs	

https://labs.hacktify.in/HTML/html_lab/lab_3/html_injection_3.php
Consequences of not Fixing the Issue
1. Business Disruption 2. Increased Risk of Future Attacks 3. Malware Distribution 4. Data Theft (breaches) 5. Website Defacement
Suggested Countermeasures
1. Input Validation and Sanitization 2. Implement a Content Security Policy (CSP) 3. Regular Security Audits 4. By deploying a Web Application Firewall
References
https://owasp.org https://portswigger.net https://www.acunetix.com

Proof of Concept

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



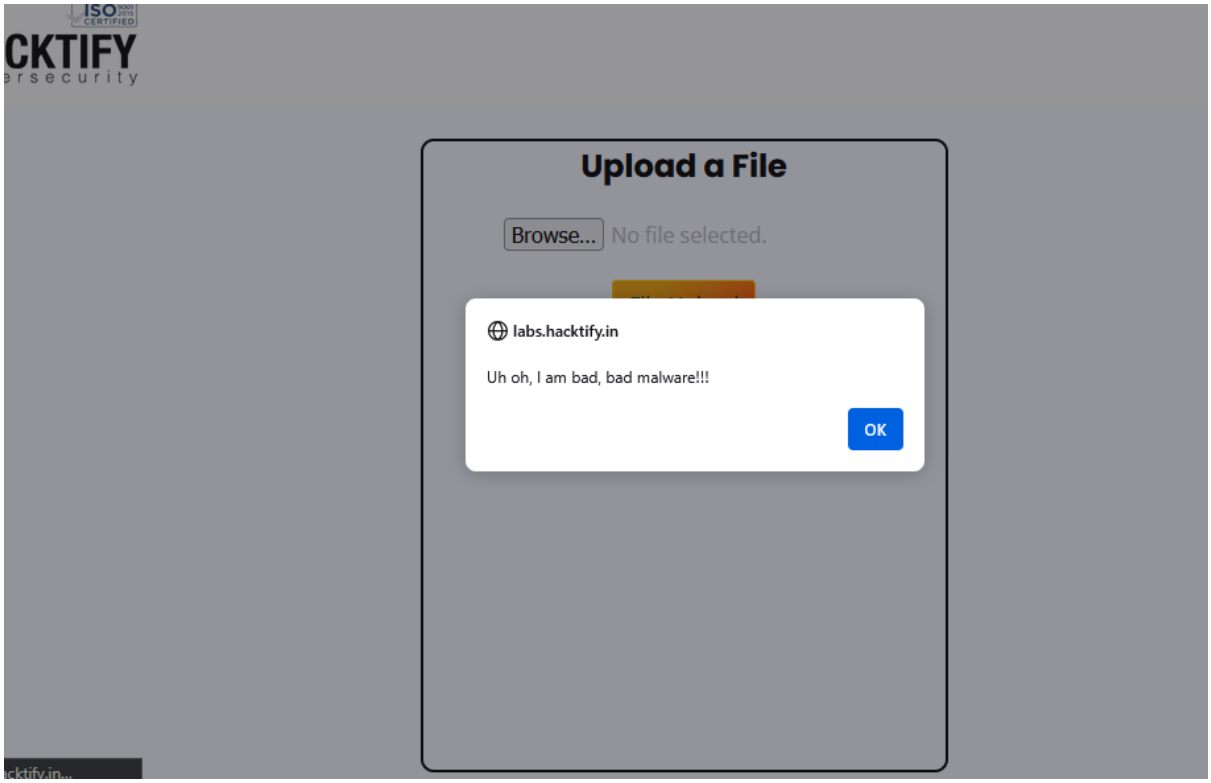
1.4. Sub-lab-4 File content and HTML injection a perfect pair!

Reference	Risk Rating
Sub-lab-4: File content and Html injection a perfect pair!	**Low**
Tools Used	
Sublime text editor “used to craft a payload”	
Vulnerability Description	
File upload vulnerabilities, are a type of HTML injection that happens when a web server allows users to upload files to its filesystem without sufficiently validating things like their name, type, contents, or size. Failing to properly enforce restrictions on these could mean that even a basic image upload function can be used to upload arbitrary and potentially dangerous files instead. This could even include server-side script files that enable remote code execution. Now since the site could not sanitize the input given to it, it allowed uploading a file with the malicious content. After uploading the file, the content inside got executed giving the threat actor vulnerability to exploit. This type of vulnerability can have serious impact and requires proper validation of the file type being uploaded for legitimate concerns and also Validation of the file content before uploading can at least help countering the vulnerabilities.	
How It Was Discovered	
Manual Analysis “By uploading the file with malicious content”	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_4/html_injection_4.php	

Consequences of not Fixing the Issue
1. Defacement of the website 2. Takeover of user session (Hijacking) 3. Malware distribution 4. Stealing of sensitive data (data theft)
Suggested Countermeasures
1. Validate file type – validating the type of file being uploaded for legitimate concerns 2. Validate file content – Validate the content of the uploaded file 3. Use secure file upload handling practices, such as storing uploaded files in a secure directory 4. Use Content Security Policy (CSP)
References
https://www.utep.edu/information-resources/iso/security-awareness/technical-security-resources/what-is-html-injection.html https://portswigger.net/web-security/file-upload https://cwe.mitre.org/data/definitions/434.html https://cognisys.co.uk/blog/file-upload-vulnerabilities/

Proof of Concept

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



1.5. Sub-lab-5 Injecting using URL!

Reference	Risk Rating
Sub-lab-4: Injecting using URL	**Low**
Tools Used	
Manual analysis “Browser”	
Vulnerability Description	

URL HTML injection, also known as "reflected XSS via URL," occurs when an attacker injects malicious HTML code into a website's URL. Here's a breakdown of the process:

Step 1: Identify Vulnerable Parameters
An attacker identifies URL parameters that are vulnerable to injection. These parameters are often used to customize content or provide user input.

Step 2: Craft Malicious URL
The attacker crafts a malicious URL by adding HTML code to the vulnerable parameter. The goal is to inject malicious code that will be executed by the user's browser.

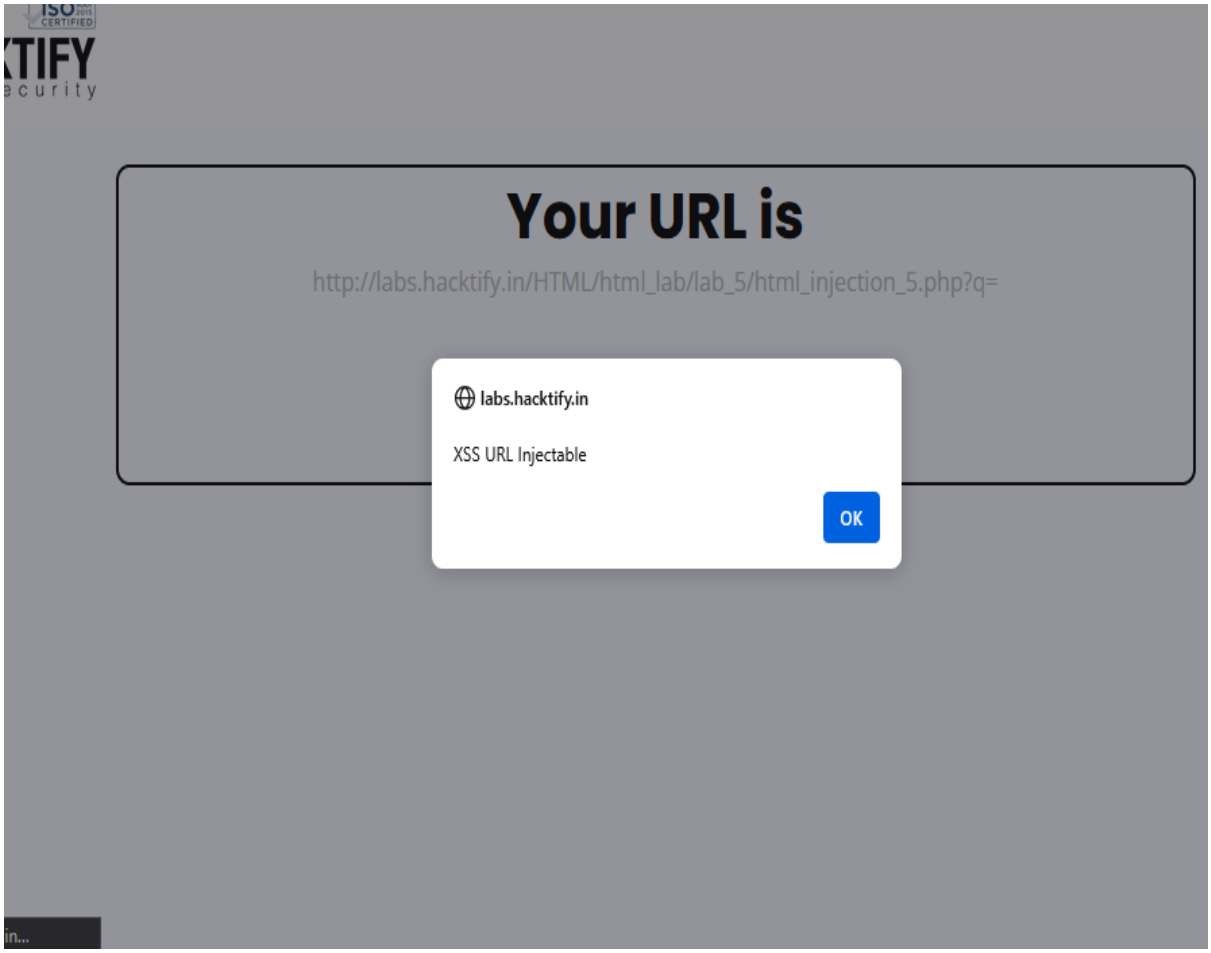
Step 3: Trick User into Clicking
The attacker tricks a user into clicking the malicious URL. This can be done through phishing emails, social engineering, or other tactics.

Step 4: Malicious Code Execution
When the user clicks the malicious URL, the injected HTML code is executed by their browser.

This type of injection can have a serious impact and requires Use URL encoding, validating and sanitizing user input and the use of prepared statements.
How It Was Discovered
Automated Tools / Manual Analysis
Vulnerable URLs
https://labs.hacktify.in/HTML/html_lab/lab_5/html_injection_5.php?q=%3Cscript%3Ealert(%20%27XSS%20URL%20Injectable%27%20)%3C/script%3E
Consequences of not Fixing the Issue
1. Can lead to Cross-Site Scripting 2. Session Hijacking 3. Malware Distribution 4. System Compromise 5. Reputation Damage
Suggested Countermeasures
1. Use URL encoding 2. Validate and Sanitize user input 3. Use prepared statements 4. Implement Content Security Policy (CSP)
References
https://www.invicti.com/learn/html-injection/ https://0x00eh.medium.com/html-injection-reflected-url-bwapp-923953a5a501 https://www.vaadata.com/blog/xss-cross-site-scripting-vulnerabilities-principles-types-of-attacks-exploitations-and-security-best-practices/ https://www.utep.edu/information-resources/iso/security-awareness/technical-security-resources/what-is-html-injection.html https://example.com/search?q=

Proof of Concept

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



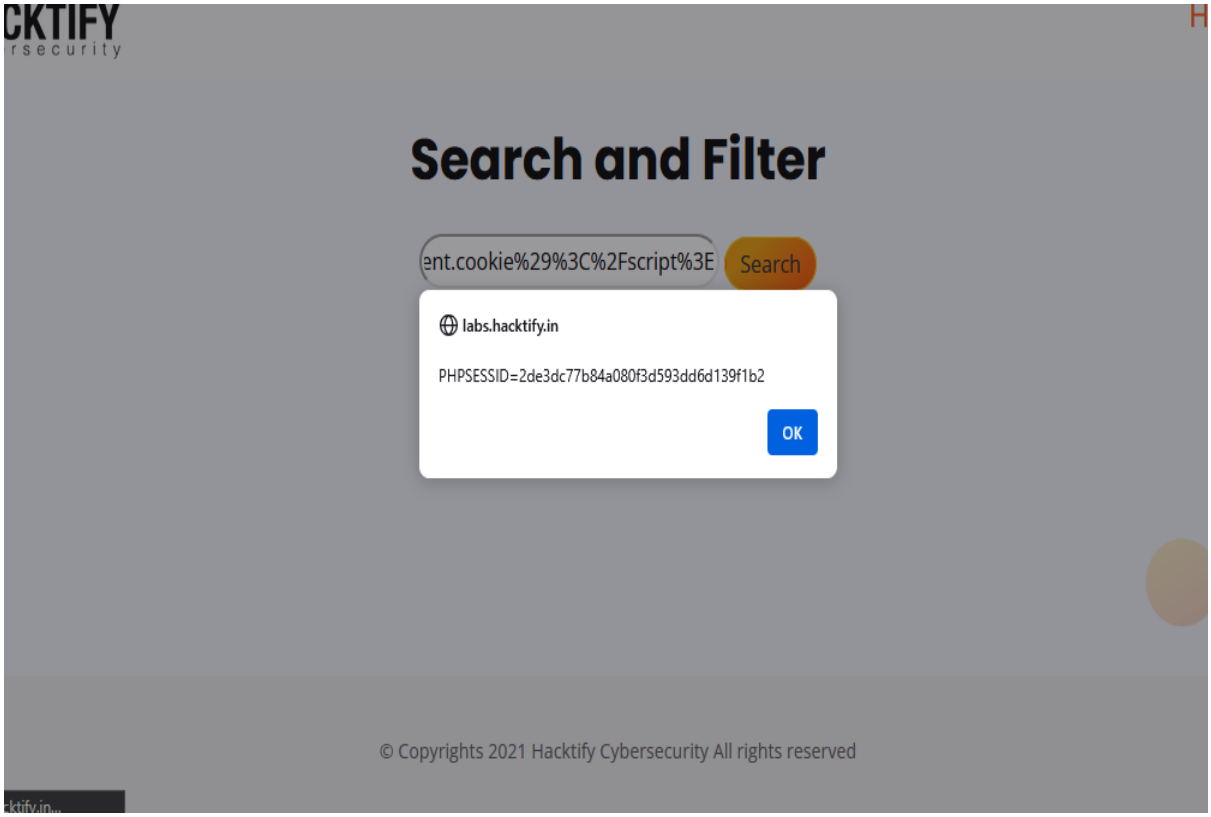
1.6. Sub-lab-6 Encode it!

Reference	Risk Rating
Sub-lab-6: Encode it!	**High**
Tools Used	
Tools that you have used to find the vulnerability.	
Vulnerability Description	
XSS (Cross-Site Scripting) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. Cross-Site Scripting vulnerabilities normally allow an attacker to masquerade as a victim user, or to carry out any actions that the user is able to perform, and to access any of the user's data. I tried all the entry points and discovered the login form was prone to malicious scripts. But it required the script to be encoded to an URL format, I then supplied it into the email login form. The payload got executed and the pop up was shown with the session ID.	
How It Was Discovered	

Manual Analysis “The vulnerability was discovered after trying the entry points with scripts and payloads, then the script was encoded into the URL format and input it into the form, and it got executed.”
Vulnerable URLs
https://labs.hacktify.in/HTML/html_lab/lab_6/html_injection_6.php
Consequences of not Fixing the Issue
1. Can lead to Cross-Site Scripting 2. Malware Distribution 3. Can lead to Cross-Site Request Forgery (CSRF) 4. Session Hijacking
Suggested Countermeasures
1. Filter the data received on the client side 2. Validate user inputs 3. Use the CSP (Content Security Policy) 4. Encode (escape) input and out-put data
References
https://www.vaadata.com/blog/xss-cross-site-scripting-vulnerabilities-principles-types-of-attacks-exploitations-and-security-best-practices/ https://portswigger.net/web-security/cross-site-scripting/dom-based#:~:text=This%20enables%20attackers%20to%20execute,causes%20execution%20of%20arbitrary%20JavaScript.

Proof of Concept

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



2. Lab 2 Sub-lab-2 XSS – Cross Site Scripting

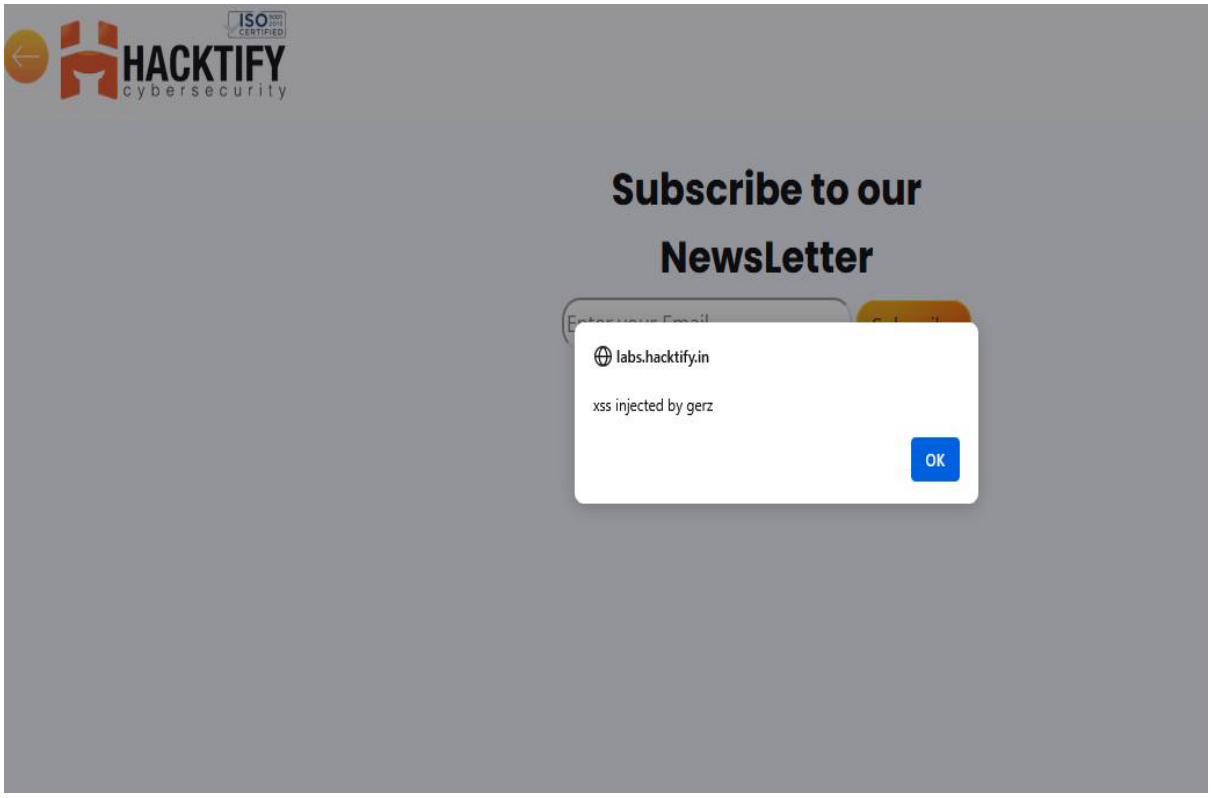
2.1. Sub-lab-1 Let's Do It!

Reference	**Risk Rating**
Sub-lab-1 Let's Do It!	**Low**
Tools Used	
Manual analysis "Browser tools"	
Vulnerability Description	
About the vulnerability and its working	
How It Was Discovered	
Manual Analysis "By trying all the entry points with scripts"	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_1/lab_1.php?email=%3Cscript%3Ealert%28%22xss+injected+by+gerz%22%29%3C%2Fscript%3E	
Consequences of not Fixing the Issue	
1. Can lead to Cross-Site Scripting 2. Session Hijacking 3. Malware Distribution	

4. System Compromise
Suggested Countermeasures
1. Output Encoding 2. Input Validation and Sanitization of Libraries 3. Secure Coding Practices 4. Content Security Policy (CSP)
References
https://www.invicti.com/learn/html-injection/ https://0x00eh.medium.com/html-injection-reflected-url-bwapp-923953a5a501 https://www.vaadata.com/blog/xss-cross-site-scripting-vulnerabilities-principles-types-of-attacks-exploitations-and-security-best-practices/ https://www.utep.edu/information-resources/iso/security-awareness/technical-security-resources/what-is-html-injection.html

Proof of Concept

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



2.2. Sub-lab-2: Balancing is Important

Reference	Risk Rating
Sub-lab-2: Balancing is Important in Life!	**Low**
Tools Used	
Vulnerability Description	
Reflected Cross-Site Scripting this is the type of XSS allows payloads to exploit vulnerabilities in web applications that allow untrusted data (such as user input) to be executed as code in the client's browser. When a web application does not properly validate or sanitize user input, an attacker can craft a payload that includes malicious code and inject it into a web page through an XSS vulnerability.	
How It Was Discovered	
Automated Tools "Browser tool, I inspected the source code and balanced that script"	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_2/lab_2.php?email=%22%2F%3E%3Cscript%3Ealert%28%22xss+injected+by+me+%22%29%3C%2Fscript%3E	
Consequences of not Fixing the Issue	
1. Denial of Service (DoS) 2. Data Breaches 3. Competitive Disadvantage 4. Session Hijacking	
Suggested Countermeasures	
1. Patch Management 2. Secure Coding Practices 3. Code Refactoring 4. Security Information and Event Management (SIEM) 5. Penetration Testing	
References	
https://www.hackerone.com https://blog.vidocsecurity.com/blog/why-you-never-get-reflected-xss-to-execute-balancing-payloads/ https://owasp.org https://portswigger.net	

Proof of Concept

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

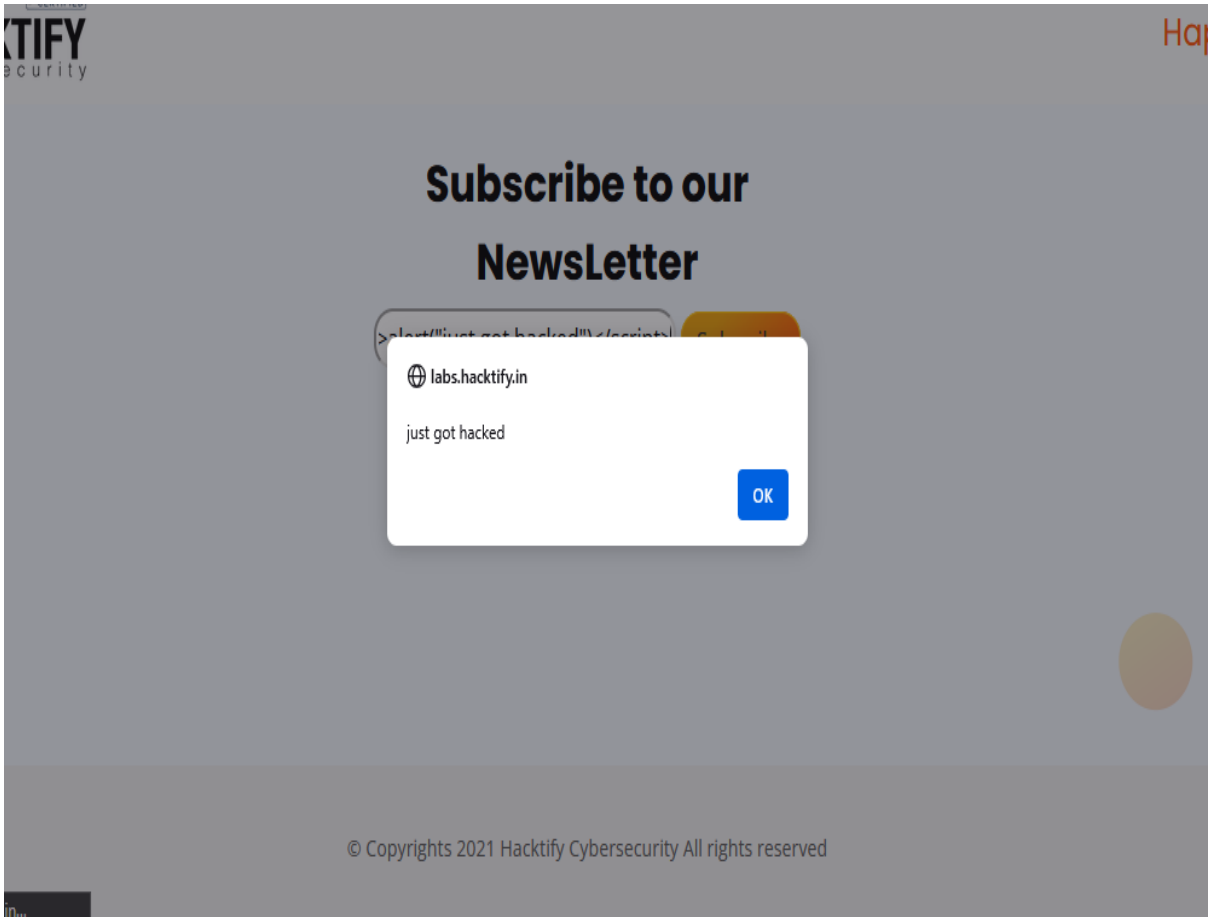
2.3. Sub-lab-3: XSS is everywhere

Reference	Risk Rating
Sub-lab-3: XSS is everywhere!	**Low**
Tools Used	
Vulnerability Description	
Cross-site scripting (XSS) is a type of computer security vulnerability typically found in web applications which enable malicious attackers to inject client-side script into web pages viewed by other users. An exploited cross-site scripting vulnerability can be used by attackers to bypass access controls such as the same origin policy.	
How It Was Discovered	
Automated Tools “I tried different inputs (scripts), the code passed unvalidated input from the request back to the client	

Vulnerable URLs
https://labs.hacktify.in/HTML/xss_lab/lab_3/lab_3.php?email=geshom%40gmail.com%3Cscript%3Ealert%28%22just+hacked+this%22%29%3C%2Fscript%3E
Consequences of not Fixing the Issue
1. Stealing of sensitive data 2. Takeover of user sessions 3. Reputation Damage 4. Business Disruption
Suggested Countermeasures
5. Output Encoding 6. Input Validation and Sanitization of Libraries 7. Secure Coding Practices 8. Content Security Policy (CSP)
References
https://stackoverflow.com/questions/2233015/what-is-the-general-concept-behind-xss https://www.techtarget.com/searchsecurity/definition/cross-site-scripting https://owasp.org https://portswigger.net

Proof of Concept

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



2.4. Sub-lab-4: Alternatives are must!

Reference	**Risk Rating**
Sub-lab-4: Alternatives are Must!	**Medium**
Tools Used	
Browser	
Vulnerability Description	
XSS vulnerability, this is the type web application security flaw that allows attackers to inject malicious scripts (typically JavaScript) into a website, which is then executed by the user's browser, potentially enabling them to steal sensitive data like cookies, redirect users to malicious sites, or manipulate the webpage content, all while appearing to come from a trusted source; essentially giving attackers control over a victim's browser session by exploiting a website's failure to properly sanitize user input.	
How It Was Discovered	
Automated Tools “By trying different entry points and different scripts payload”	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_4/lab_4.php?email=%22%2F%3E%3Cimg+src%3Dq+onerror%3Dprompt%28document.cookie%29%3E	
Consequences of not Fixing the Issue	

1. Website Defacement
2. Phishing Attacks
3. Data Theft
4. Session Hijacking

Suggested Countermeasures

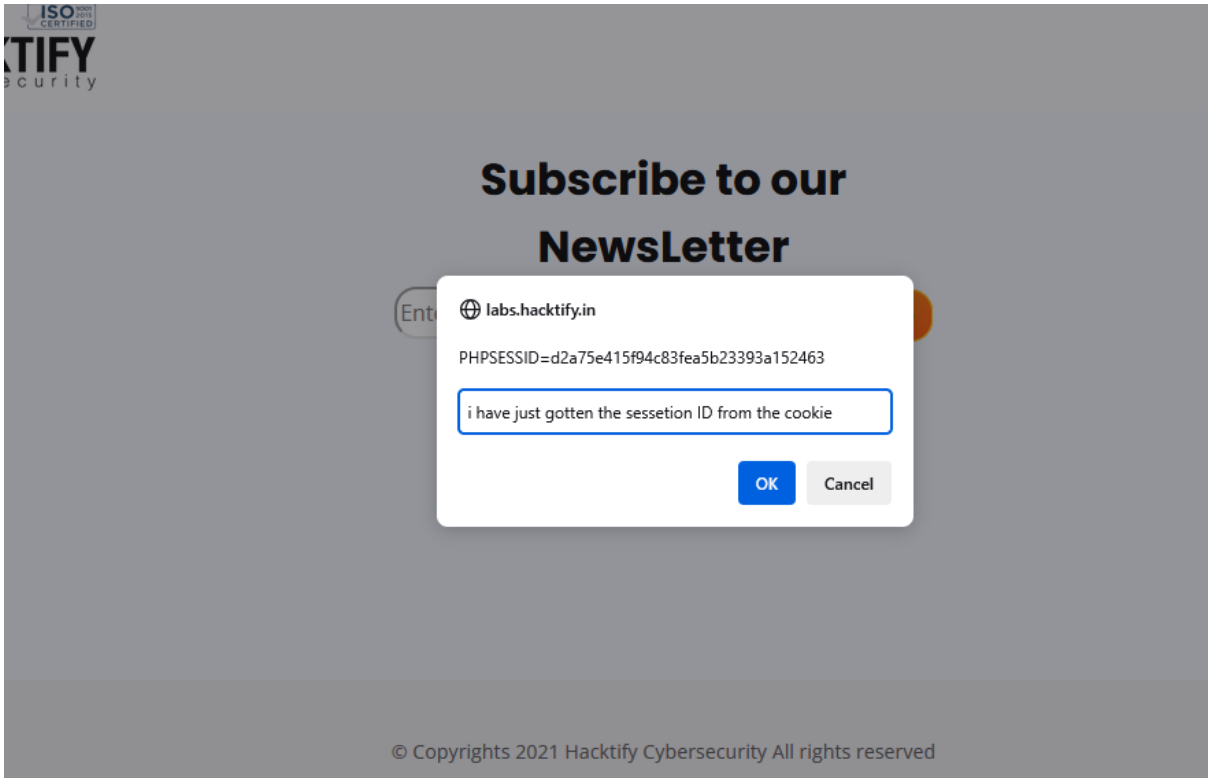
1. Output Encoding
2. Input Validation and Sanitization of Libraries
3. Secure Coding Practices
4. Regular Security Audits and Testing

References

https://cheatsheetseries.owasp.org/cheatsheets/XSS_filter_Evasion_cheat_html
https://owasp.org
https://portswigger.net
https://www.immuniweb.com
https://www.wallarm.com

Proof of Concept

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

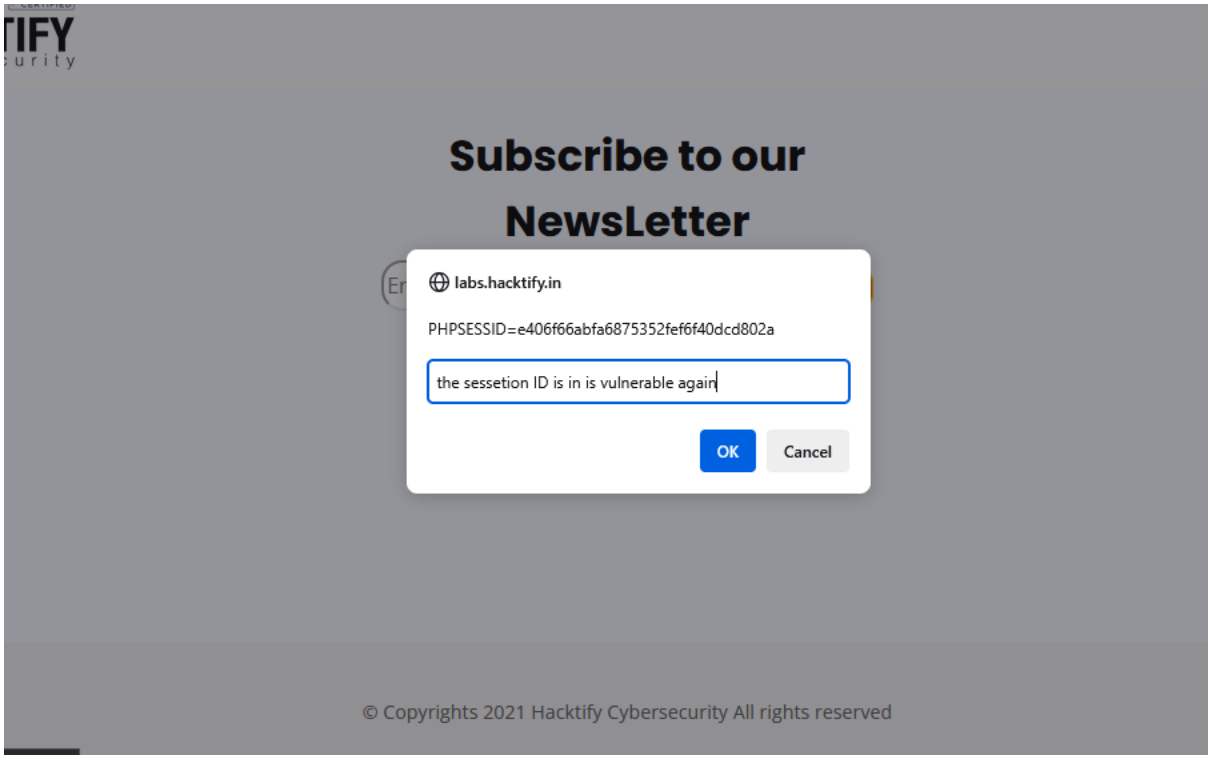


2.5. Sub-lab-5: Developers hate Scripts!

Reference	Risk Rating
Sub-lab-5: Developers hate Script!	**High**
Tools Used	
Tools that you have used to find the vulnerability.	
Vulnerability Description	
Cross-Site Scripting (XSS) vulnerability, these are a web application security flaws that allows attackers to inject malicious code (typically JavaScript) into a website, which is then executed by the user's browser, potentially enabling them to steal sensitive data like cookies, redirect users to malicious sites, or manipulate the webpage content, all while appearing to come from a trusted source; essentially giving attackers control over a victim's browser session by exploiting a website's failure to properly sanitize user input.	
How It Was Discovered	
Automated Tools	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_5/lab_5.php?email=%22%2F%3E%3Cimg+src%3Dq+onerror%3Dprompt%28document.cookie%29%3E	
Consequences of not Fixing the Issue	
1. Website Defacement 2. Phishing Attacks 3. Data Theft 4. Session Hijacking	
Suggested Countermeasures	
1. Content Security Policy 2. Output Encoding 3. Input Validation	
References	
https://owasp.org https://brightsec.com https://portswigger.net https://aptori.dev	

Proof of Concept

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



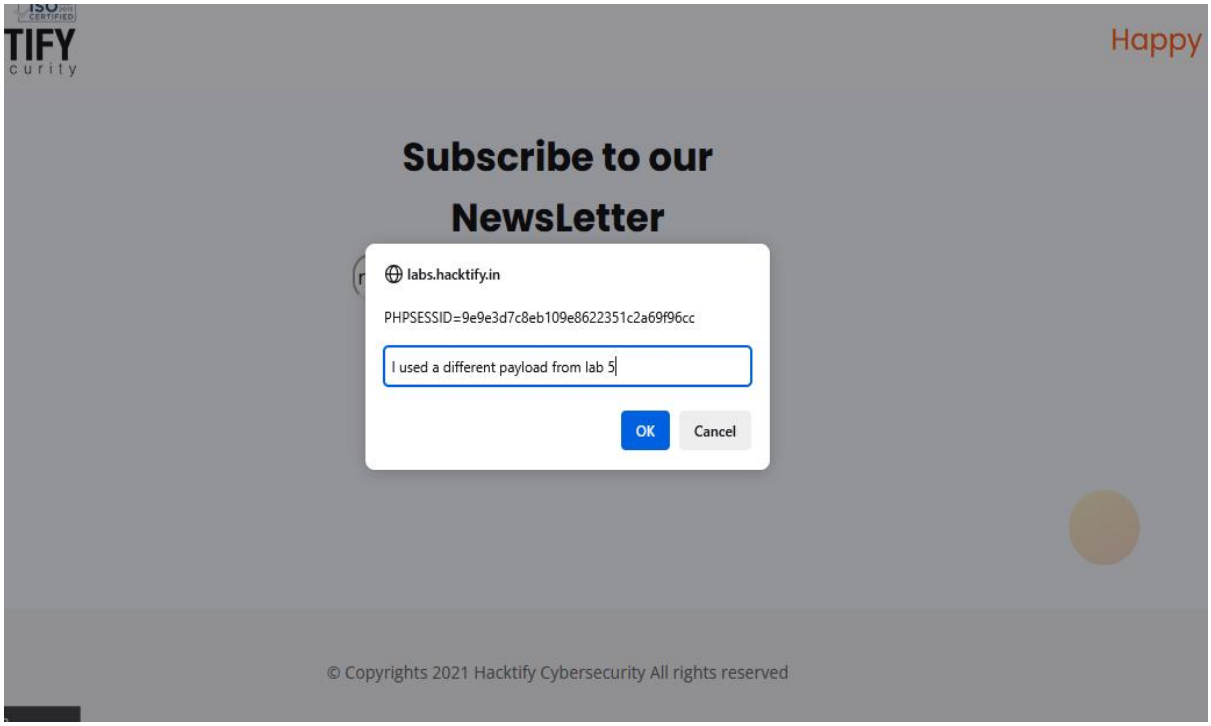
2.6. Sub-lab- 6: Change the Variation

Reference	Risk Rating
Sub-lab-6: Change the Variation	**High**
Tools Used	
Tools that you have used to find the vulnerability.	
Vulnerability Description	
This type of Vulnerability flaw that allows attackers to inject malicious code (typically JavaScript) into a website, which is then executed by the user's browser, potentially enabling them to steal sensitive data like cookies, redirect users to malicious sites, or manipulate the webpage content, all while appearing to come from a trusted source; essentially giving attackers control over a victim's browser session by exploiting a website's failure to properly sanitize user input.	
How It Was Discovered	
Automated Tools “Browser by trying different scripts”	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_1/lab_1.php?email=%22%2F%3E%3Cimg+src%3Dq+onerror%3Dprompt%28document.cookie%29%3E	
Consequences of not Fixing the Issue	
1. Phishing 2. Data Theft	

3. Malware Distribution 4. Regulatory Fines 5. Website Defacement
Suggested Countermeasures
1. Output Encoding 2. Input Validation and Sanitization of Libraries 3. Secure Coding Practices 4. Regular Security Audits and Testing
References
https://owasp.org https://portswigger.net https://www.acunetix.com https://www.blackduck.com

Proof of Concept

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

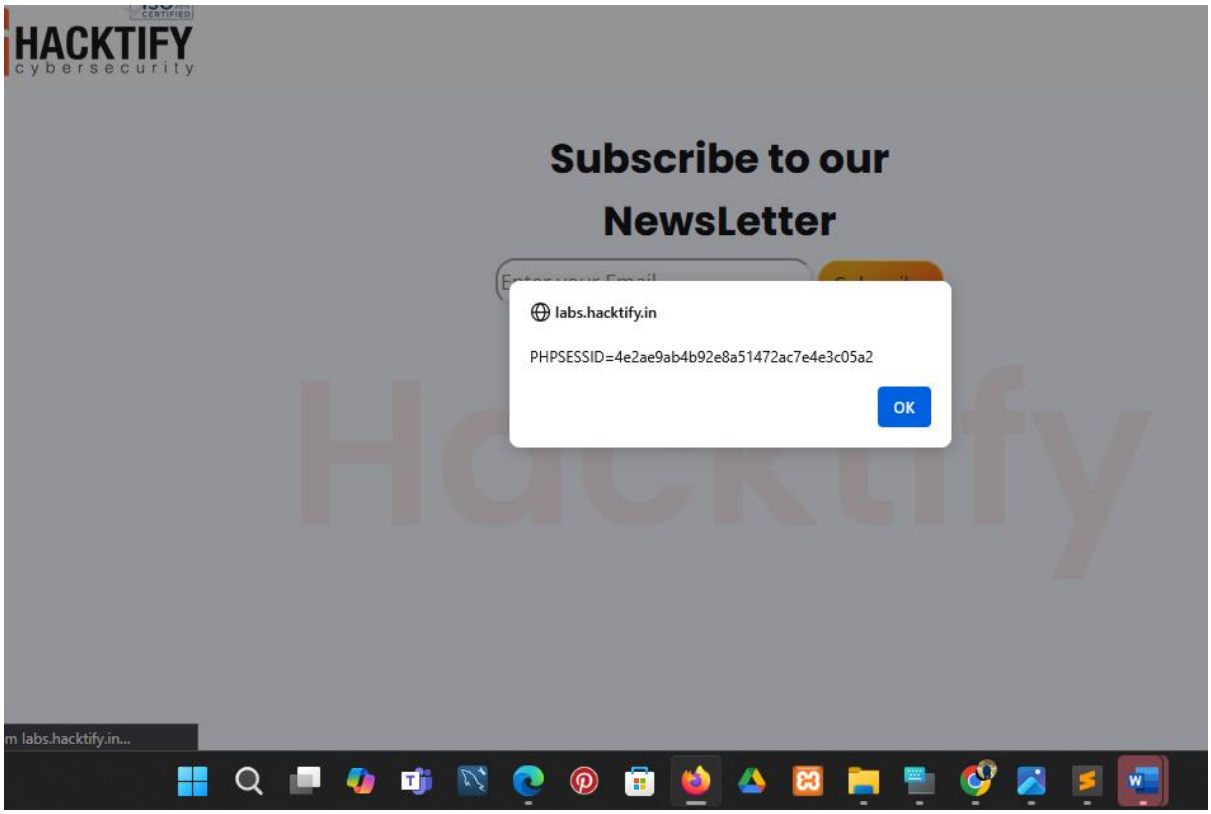


2.7. Sub-lab-7 Encoding is the Key

Reference	Risk Rating
Sub-lab-: Encoding is the Key!	**Medium**
Tools Used	
Web site https://www.urlencoder.org/ for encoding the script	
Vulnerability Description	
XSS (Cross-Site Scripting) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. Cross-Site Scripting vulnerabilities normally allow an attacker to masquerade as a victim user, or to carry out any actions that the user is able to perform, and to access any of the user's data. I tried all the entry points and discovered the login form was prone to malicious scripts. But it required the script to be encoded to an URL format, I then supplied it into the email login form. The payload got executed and the pop up was shown with the session ID.	
How It Was Discovered	
Manual Analysis "The vulnerability was discovered after trying the entry points with scripts and payloads, then the script was encoded into the URL format and input it into the form, and it got executed."	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_7/lab_7.php?email=%253Cscript%253Ealert%2528document.cookie%2529%253C%252Fscript%253E	
Consequences of not Fixing the Issue	
5. Can lead to Cross-Site Scripting 6. Malware Distribution 7. Can lead to Cross-Site Request Forgery (CSRF) 8. Session Hijacking	
Suggested Countermeasures	
1. Filter the data received on the client side 2. Validate user inputs 3. Use the CSP (Content Security Policy) 4. Encode (escape) input and out-put data	
References	
https://www.vaadata.com/blog/xss-cross-site-scripting-vulnerabilities-principles-types-of-attacks-exploitations-and-security-best-practices/ https://portswigger.net/web-security/cross-site-scripting/dom-based#:~:text=This%20enables%20attackers%20to%20execute,causes%20execution%20of%20arbitrary%20JavaScript.	

Proof of Concept

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



2.8. Sub-lab-8 XSS with file upload (file content)

Reference	**Risk Rating**
{Sub-lab-8: XSS with file upload (file content)	**Low**
Tools Used	
Vulnerability Description	
How It Was Discovered	
Automated Tools	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_8/lab_8.php	
Consequences of not Fixing the Issue	
1. Stealing of sensitive data 2. Takeover of user sessions	

3. Malware distribution 4. Defacement of the website
Suggested Countermeasures
1. Validate file types 2. Validate file content 3. Use secure file upload handling practices 4. Use Content Security Policy (CSP 5. Use a Web Application Firewall (WAF)
References
https://github.com/payloadbox/xss-payload-list https://testphp.vulnweb.com https://www.invicti.com/learn/cross-site-scripting-xss/

Proof of Concept

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

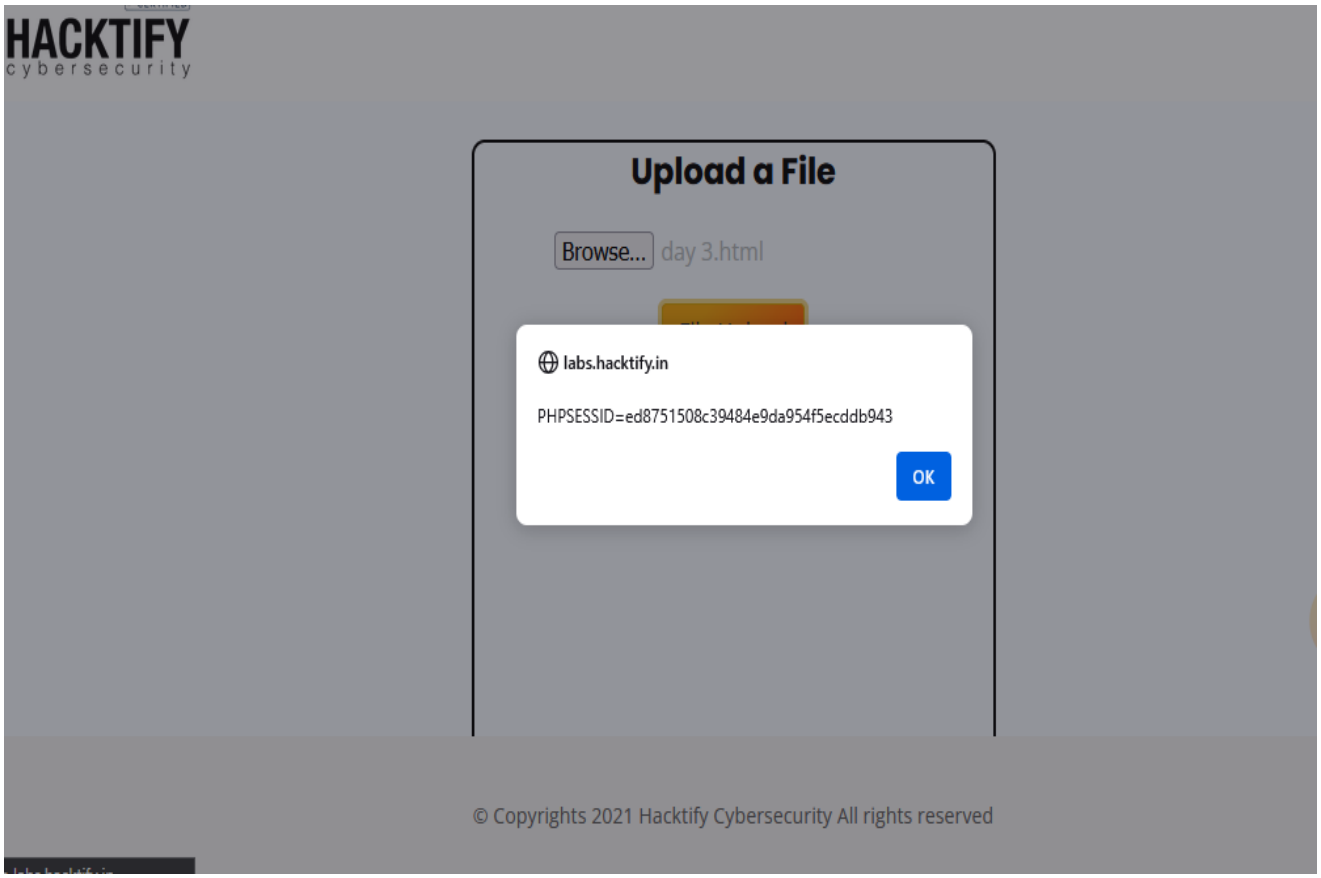
2.9. Sub-lab-9 XSS with file upload (file content)

Reference	Risk Rating
Sub-lab-9: XSS with file upload (file content)	**Medium**
Tools Used	
A crafted HTML payload which was uploaded to the vulnerable page	
Vulnerability Description	
XSS with file upload is the type of XSS that occurs when an attacker uploads a malicious file to a website, which is then executed by other users, allowing the attacker to steal sensitive data or take control of their sessions. There, are different type of XSS file upload vulnerabilities namely: 1. *Upload of HTML/JavaScript files*: An attacker uploads an HTML or JavaScript file that contains malicious code, which is then executed by other users. 2. *Upload of image files with embedded code*: An attacker uploads an image file that contains embedded malicious code, such as JavaScript or Flash code. 3. *Upload of files with malicious extensions*: An attacker uploads a file with a malicious extension, such as a PHP or ASP file, which is then executed by the server. If this, threats are not patched they would allow threat actors to steal data, session hijacking etc. The counter measures to put in order to mitigate this vulnerability would be, Validate the fie type, Validate the file content, using of CSP's.	
How It Was Discovered	
Manual Analysis "by uploading different types of files and file extensions	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_9/lab_9.php	
Consequences of not Fixing the Issue	
1. Stealing of sensitive data 2. Takeover of user sessions 3. Malware distribution 4. Defacement of the website	
Suggested Countermeasures	
1. Validate file type 2. Validate file content 3. Use secure file upload handling practices 4. Use Content Security Policy (CSP) 5. Use a Web Application Firewall (WAF)	
References	

0arbitrary%20JavaScript
https://github.com/payloadbox/xss-payload-list
https://testphp.vulnweb.com
https://www.invicti.com/learn/cross-site-scripting-xss/
https://portswigger.net/web-security/cross-site-scripting/dom-based#:~:text=This%20enables%20attackers%20to%20execute,causes%20execution%20of%2

Proof of Concept

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



2.10. Sub-lab-10 Stored Everywhere

Reference	Risk Rating
Sub-lab-2: Stored Everywhere!	**Low**
Tools Used	
Browser “Tried all the Entry points”	
Vulnerability Description	

Stored XSS, this is a type of XSS (Cross-Site Scripting) also known as persistent or second-order XSS. The stored XSS arises when an application receives data from an untrusted source and it includes that data within later HTTP responds in an unsafe way. This type of XSS happens when the server saves the supplied input (payload) somewhere into the server, (i.e.) Database, cache server. And, when this happens the payload gets stored into the database or cache server, every time the web page or application loads the payload executes.

How It Was Discovered

This was discovered by testing the fields which stores values in a server, in this case a malicious script was supplied in a f_name and l_name fields.

Vulnerable URLs

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

Consequences of not Fixing the Issue

1. Data Theft
2. Session Hijacking
3. Loss of Customer Trust
4. Phishing
5. Downtime
6. Reputation Damage

Suggested Countermeasures

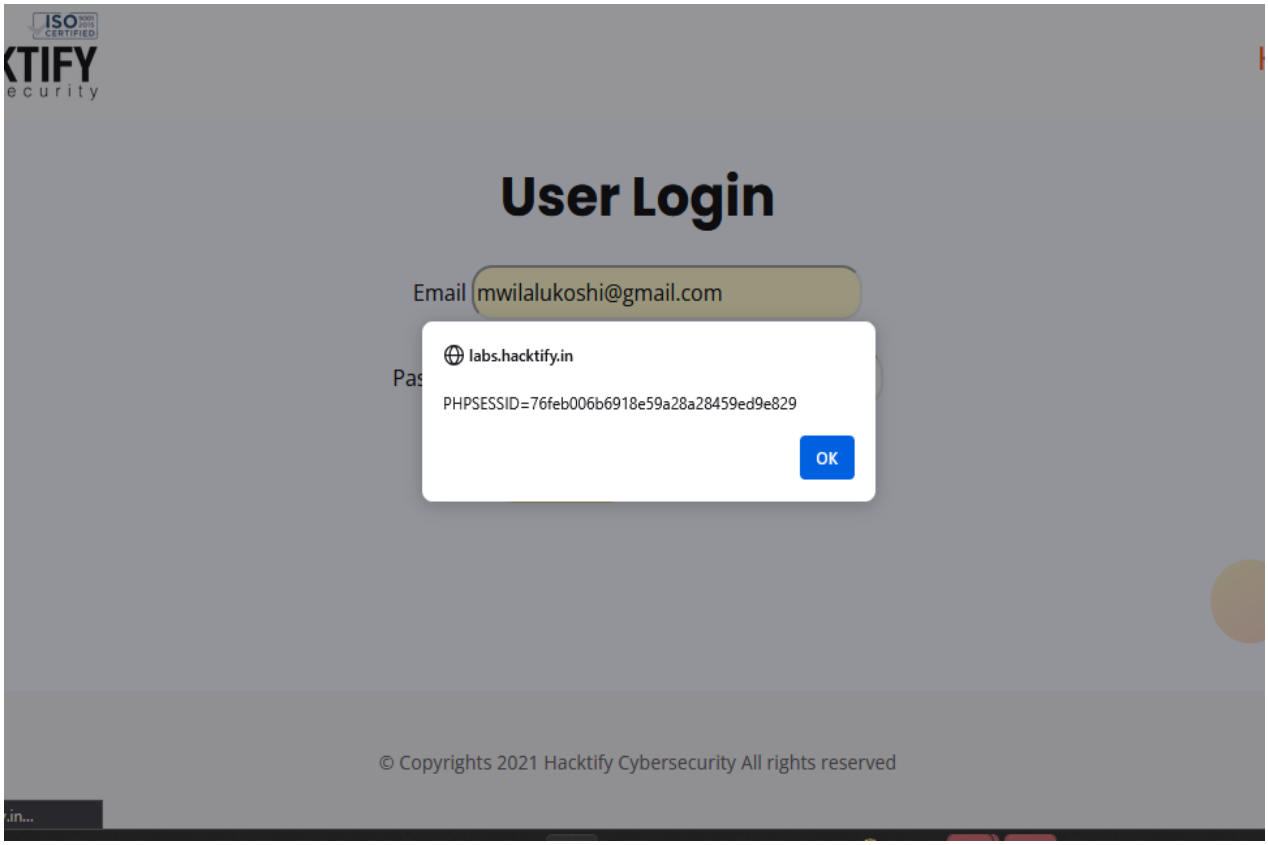
1. Output Encoding
2. Input Validation and Sanitization
3. Content Security Policy (COP)
4. Secure Coding Practices
5. User Education and Awareness

References

https://portswigger.net/web-security/cross-site-scripting/dom-based#:~:text=This%20enables%20attackers%20to%20execute,causes%20execution%20of%20arbitrary%20JavaScript
https://github.com/payloadbox/xss-payload-list
https://testphp.vulnweb.com
https://www.invicti.com/learn/cross-site-scripting-xss/

Proof of Concept

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



2.11. Sub-lab-11 DOM's are Love

Reference	Risk Rating
Sub-lab-11: DOM's are Love!	**High**
Tools Used	
Browser "View Page Source, DOM.JS" was used to find the vulnerability.	
Vulnerability Description	
DOM-based Cross-Site Scripting is the de-facto name for XSS bugs, which are the result of active browser-side content on a page, typically javascript, obtaining user input and then doing something unsafe with it, which leads to execution of the injected code. The DOM, or Document Object Model, is the structural format used to represent documents in a browser. The DOM enables dynamic scripts such as JavaScript to reference components of the document such as a form field or a session cookie. The DOM is also used by the browser for security - for example to limit scripts on different domains from obtaining session cookies for other domains. A DOM-based XSS vulnerability may occur when active content, such as a JavaScript function, is modified by a specially crafted request such that a DOM element that can be controlled by an attacker.	
How It Was Discovered	

By Manual Analysis " I identified the entry point in the (url) parameter, and it was injectable and after inspecting the DOM.js using the browser's developer tool to inspect the DOM and verified if the malicious code executed. It got executed.

Vulnerable URLs

https://labs.hacktify.in/HTML/xss_lab/lab_11/lab_11.php?name=%3Cimage%20src=x%20onerror=(document.cookie)%3E

Consequences of not Fixing the Issue

Security Risks

1. Malware Distribution – Attackers can inject malicious scripts to distribute malware.
2. Data theft – Malicious scripts can steal sensitive user data.
3. Session Hijacking – Attackers can hijack user sessions, allowing impersonation

Financial Consequences

1. Reputation Damage – A DOM-based XSS vulnerability can damage a company's reputation.
2. Litigation Costs – Companies may face lawsuits from affected users, resulting in litigation costs.
3. Regulatory Fines – Organizations that fail to protect user data may face regulatory fines And penalties.

Operational Consequences

1. System Compromise – Malicious scripts can compromise the underlaying system, leading to data corruption, system crashing, or other operational issues.
2. Downtime – Remediation efforts may require taking the website or application offline resulting in downtime and lost revenue.
3. Resource Intensive – Fixing DOM-based XSS vulnerability can be resource-intensive requiring time and effort from development and security teams.

Suggested Countermeasures

1. Input Validation
2. Output Encoding
3. And implementing of Content Security Policies
4. Use a Web Application Firewall (WFF) to detect and block malicious traffic.

References

https://portswigger.net/web-security/cross-site-scripting/dom-based#:~:text=This%20enables%20attackers%20to%20execute,causes%20execution%20of%20arbitrary%20JavaScript
https://github.com/payloadbox/xss-payload-list
https://wiki.owasp.org/index.php/Testing_for_DOM-based_Cross_site_scripting_(OTG-CLIENT-001)

Proof of Concept

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