Penetration Testing Report

Full Name: Aashutosh Thakur
Program: HCPT
Date: 17 Feb 2025

Introduction

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

1. Objective

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

2. Scope

This section defines the scope and boundaries of the project.

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

3. Summary

Outlined is a Black Box Application Security assessment for the **Week {1} and Week {2} Labs**.

Total number of Sub-labs:

High	Medium	Low
{4}	{3}	{10}

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

Medium - **3 Sub-labs with Medium difficulty level**

Low - **10 Sub-labs with Easy difficulty level**

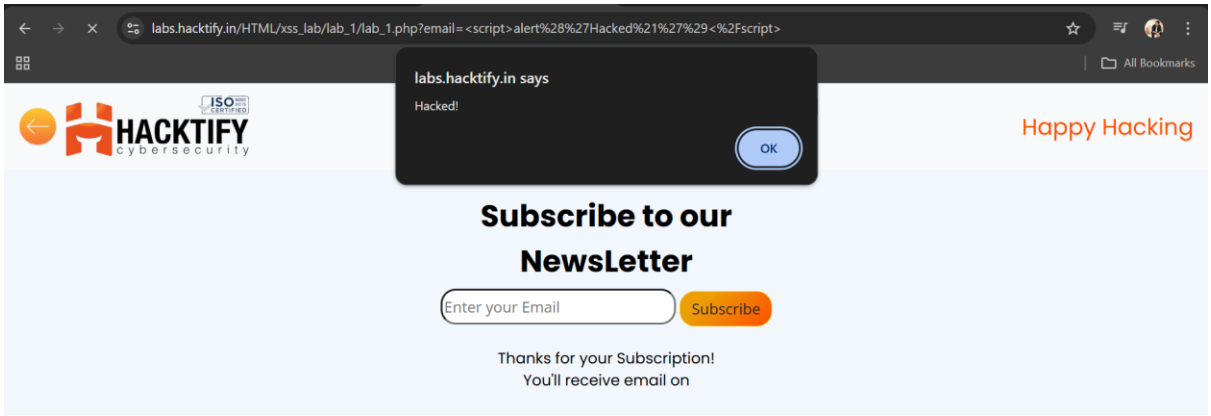
1. Cross Site Scripting

1.1. Let's Do It

Reference	Risk Rating
Sub-lab-1: Let's Do It	**Low**
Tools Used	
Browser Inspector (Automated tools)	
Vulnerability Description	
Cross-Site Scripting (XSS) is a vulnerability that allows an attacker to inject malicious scripts into web pages viewed by other users. This type of vulnerability occurs when a web application includes untrusted data without proper validation or escaping.	
How It Was Discovered	
The vulnerability was discovered by using the automated tool, Browser Inspector, which identified points where user input was not properly sanitized, allowing JavaScript injection.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_1/lab_1.php?email=%3Cscript%3Ealert%28%27Hacked%21%27%29%3C%2Fscript%3E	
Consequences of not Fixing the Issue	
If this vulnerability is not patched, an attacker could execute malicious scripts in a victim's browser. This could lead to the theft of sensitive data, session hijacking, defacement of the site, or the spread of malware.	
Suggested Countermeasures	
Input Validation: Ensure that all user inputs are validated, escaping any characters that could be interpreted as code. **Use of Security Headers:** Implement HTTP security headers such as Content Security Policy (CSP) to restrict the sources from which scripts can be loaded. **Output Encoding:** Use context-sensitive output encoding to prevent the execution of injected scripts. **Sanitize HTML Input:** Use libraries or frameworks that sanitize inputs and prevent harmful code execution. **Regular Security Audits:** Conduct regular security audits using automated tools and manual testing to identify and fix vulnerabilities.	
References	
• https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html	

Proof of Concept

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

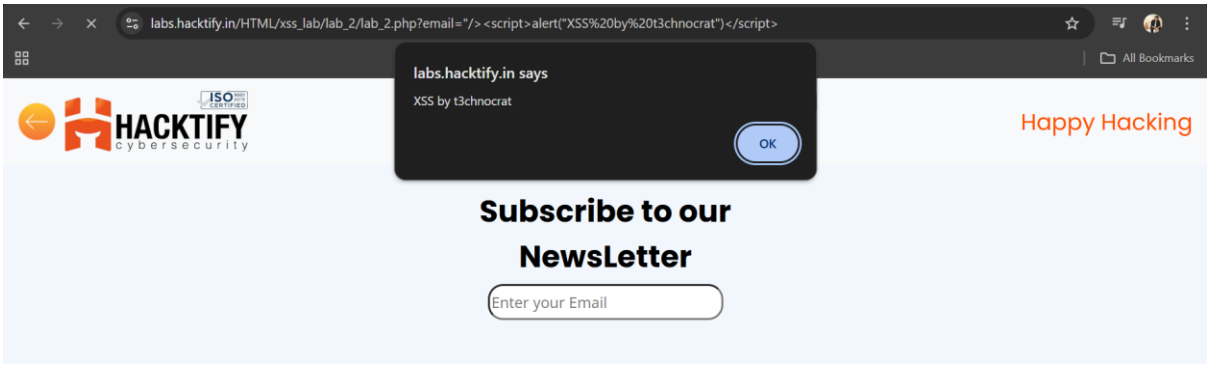


1.2. Balancing is important in Life!

Reference	Risk Rating
Sub-lab-2: Balancing is important in life!	**Low**
Tools Used	
Browser Inspector (Automated tools)	
Vulnerability Description	
Cross-Site Scripting (XSS) is a vulnerability that allows an attacker to inject malicious scripts into web pages viewed by other users. This type of vulnerability occurs when a web application includes untrusted data without proper validation or escaping.	
How It Was Discovered	
The vulnerability was discovered by using the automated tool, Browser Inspector, which identified points where user input was not properly sanitized, allowing JavaScript injection.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_2/lab_2.php?email=%22/%3E%3Cscript%3Ealert(%22XSS%20by%20t3chnocrat%22)%3C/script%3E	
Consequences of not Fixing the Issue	
If this vulnerability is not patched, an attacker could execute malicious scripts in a victim's browser. This could lead to the theft of sensitive data, session hijacking, defacement of the site, or the spread of malware.	
Suggested Countermeasures	
Input Validation: Ensure that all user inputs are validated, escaping any characters that could be interpreted as code. **Use of Security Headers:** Implement HTTP security headers such as Content Security Policy (CSP) to restrict the sources from which scripts can be loaded. **Output Encoding:** Use context-sensitive output encoding to prevent the execution of injected scripts. **Sanitize HTML Input:** Use libraries or frameworks that sanitize inputs and prevent harmful code execution. **Regular Security Audits:** Conduct regular security audits using automated tools and manual testing to identify and fix vulnerabilities.	
References	
https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/	

Proof of Concept

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

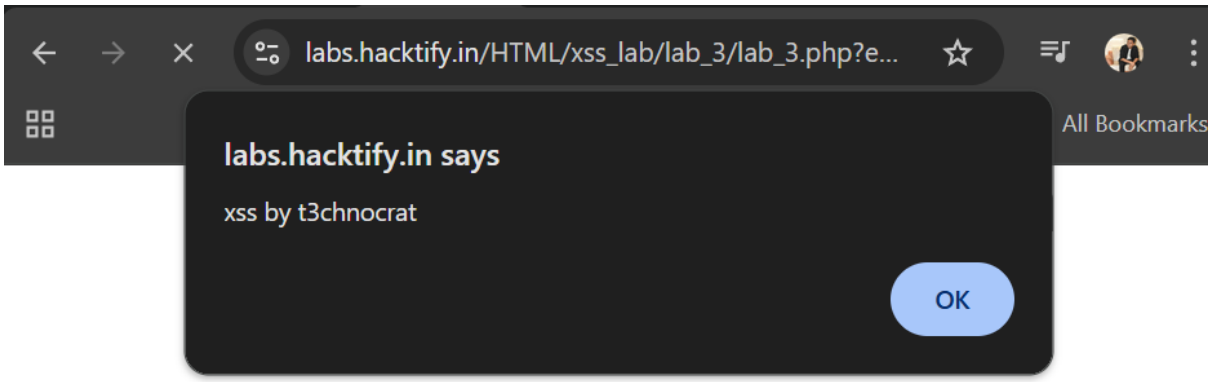


1.3. XSS is everywhere!

Reference	Risk Rating
Sub-lab-3: XSS is everywhwere!	**Low**
Tools Used	
Browser Inspector (Automated tools)	
Vulnerability Description	
Cross-Site Scripting (XSS) is a vulnerability that allows an attacker to inject malicious scripts into web pages viewed by other users. This type of vulnerability occurs when a web application includes untrusted data without proper validation or escaping.	
How It Was Discovered	
The vulnerability was discovered by using the automated tool, Browser Inspector, which identified points where user input was not properly sanitized, allowing JavaScript injection.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_3/lab_3.php?email=test@gmail.com%3Cscript%3Ealert(%22xss%20by%20t3chnocrat%22)%3C/script%3E	
Consequences of not Fixing the Issue	
If this vulnerability is not patched, an attacker could execute malicious scripts in a victim's browser. This could lead to the theft of sensitive data, session hijacking, defacement of the site, or the spread of malware.	
Suggested Countermeasures	
Input Validation: Ensure that all user inputs are validated, escaping any characters that could be interpreted as code. **Use of Security Headers:** Implement HTTP security headers such as Content Security Policy (CSP) to restrict the sources from which scripts can be loaded. **Output Encoding:** Use context-sensitive output encoding to prevent the execution of injected scripts. **Sanitize HTML Input:** Use libraries or frameworks that sanitize inputs and prevent harmful code execution. **Regular Security Audits:** Conduct regular security audits using automated tools and manual testing to identify and fix vulnerabilities.	
References	
https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/	

Proof of Concept

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



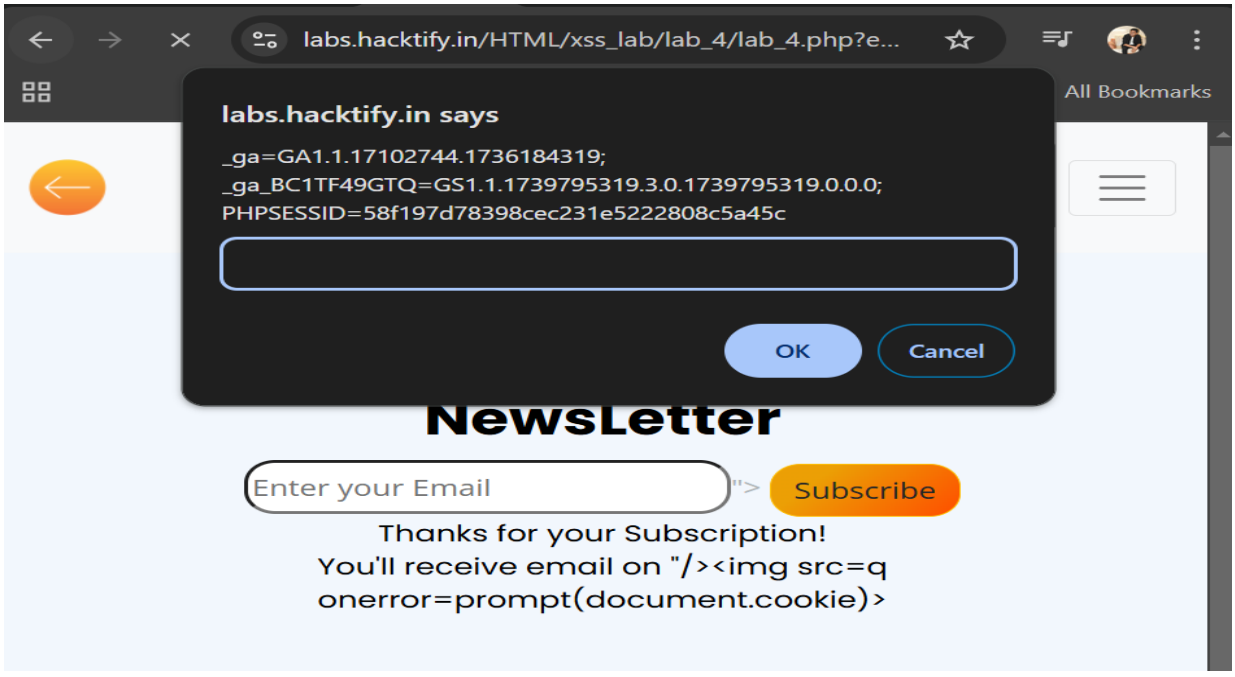
1.4. Alternatives are must!

Reference	Risk Rating
Sub-lab-4: Alternatives are must!	**Medium**
Tools Used	
Browser Inspector (Automated tools)	
Vulnerability Description	
Cross-Site Scripting (XSS) is a vulnerability that allows an attacker to inject malicious scripts into web pages viewed by other users. This type of vulnerability occurs when a web application includes untrusted data without proper validation or escaping.	
How It Was Discovered	
The vulnerability was discovered by using the automated tool, Browser Inspector, which identified points where user input was not properly sanitized, allowing JavaScript injection.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_4/lab_4.php?email=%22/%3E%3Cimg%20src=q%20onerror=prompt(document.cookie)%3E	
Consequences of not Fixing the Issue	

If this vulnerability is not patched, an attacker could execute malicious scripts in a victim's browser. This could lead to the theft of sensitive data, session hijacking, defacement of the site, or the spread of malware.
Suggested Countermeasures
Input Validation: Ensure that all user inputs are validated, escaping any characters that could be interpreted as code. **Use of Security Headers:** Implement HTTP security headers such as Content Security Policy (CSP) to restrict the sources from which scripts can be loaded. **Output Encoding:** Use context-sensitive output encoding to prevent the execution of injected scripts. **Sanitize HTML Input:** Use libraries or frameworks that sanitize inputs and prevent harmful code execution. **Regular Security Audits:** Conduct regular security audits using automated tools and manual testing to identify and fix vulnerabilities.
References
https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/

Proof of Concept

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



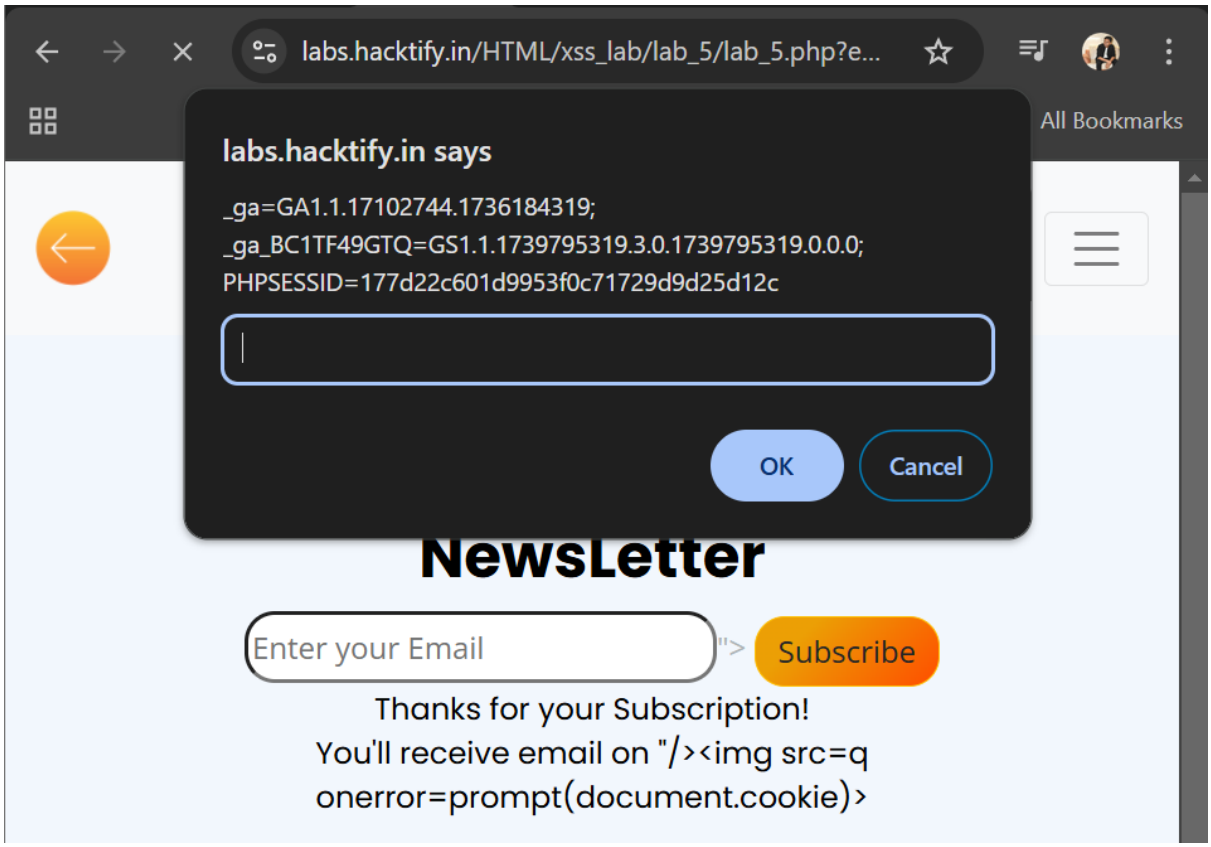
1.5. Developer hates scripts!

Reference	Risk Rating
Sub-lab-5: Developer hates scripts!	**Hard**
Tools Used	
Browser Inspector (Automated tools)	
Vulnerability Description	
Cross-Site Scripting (XSS) is a vulnerability that allows an attacker to inject malicious scripts into web pages viewed by other users. This type of vulnerability occurs when a web application includes untrusted data without proper validation or escaping.	
How It Was Discovered	
The vulnerability was discovered by using the automated tool, Browser Inspector, which identified points where user input was not properly sanitized, allowing JavaScript injection.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_5/lab_5.php?email=%22/%3E%3Cimg%20src=q%20onerror=prompt(document.cookie)%3E	
Consequences of not Fixing the Issue	
If this vulnerability is not patched, an attacker could execute malicious scripts in a victim's browser. This could lead to the theft of sensitive data, session hijacking, defacement of the site, or the spread of malware.	
Suggested Countermeasures	
Input Validation: Ensure that all user inputs are validated, escaping any characters that could be interpreted as code. **Use of Security Headers:** Implement HTTP security headers such as Content Security Policy (CSP) to restrict the sources from which scripts can be loaded. **Output Encoding:** Use context-sensitive output encoding to prevent the execution of injected scripts. **Sanitize HTML Input:** Use libraries or frameworks that sanitize inputs and prevent harmful code execution.	

Regular Security Audits: Conduct regular security audits using automated tools and manual testing to identify and fix vulnerabilities.
References
• https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

Proof of Concept

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



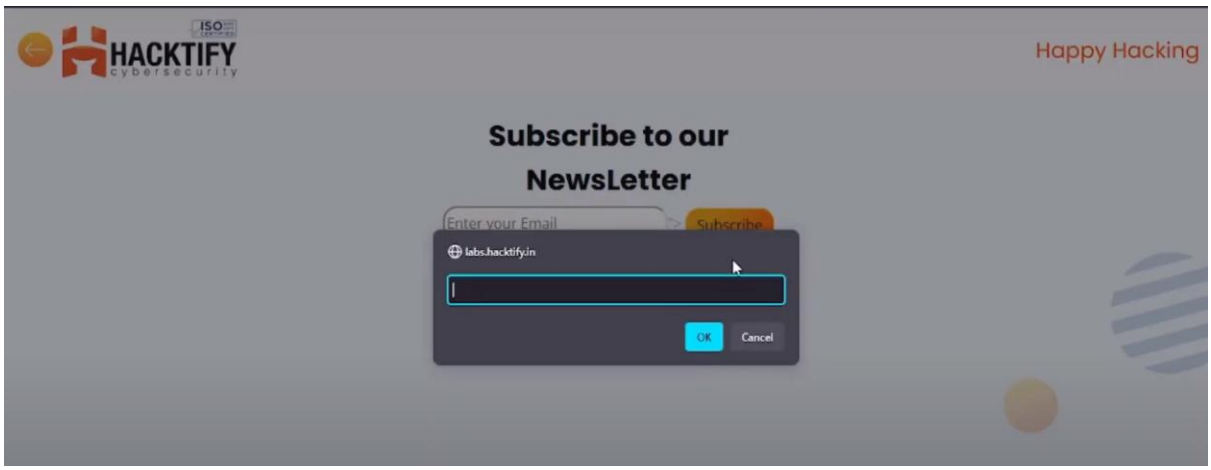
1.6. Change the variation!

Reference	Risk Rating
Sub-lab-6: Change the variation!	**Hard**
Tools Used	
Browser Inspector (Automated tools)	
Vulnerability Description	

Cross-Site Scripting (XSS) is a vulnerability that allows an attacker to inject malicious scripts into web pages viewed by other users. This type of vulnerability occurs when a web application includes untrusted data without proper validation or escaping.
How It Was Discovered
The vulnerability was discovered by using the automated tool, Browser Inspector, which identified points where user input was not properly sanitized, allowing JavaScript injection.
Vulnerable URLs
https://labs.hacktify.in/HTML/xss_lab/lab_6/lab_6.php?email=%22%2F%3E%3Cimage+src%3Dq+onerror-prompt%28document.cookies%29%3E
Consequences of not Fixing the Issue
If this vulnerability is not patched, an attacker could execute malicious scripts in a victim's browser. This could lead to the theft of sensitive data, session hijacking, defacement of the site, or the spread of malware.
Suggested Countermeasures
Input Validation: Ensure that all user inputs are validated, escaping any characters that could be interpreted as code. **Use of Security Headers:** Implement HTTP security headers such as Content Security Policy (CSP) to restrict the sources from which scripts can be loaded. **Output Encoding:** Use context-sensitive output encoding to prevent the execution of injected scripts. **Sanitize HTML Input:** Use libraries or frameworks that sanitize inputs and prevent harmful code execution. **Regular Security Audits:** Conduct regular security audits using automated tools and manual testing to identify and fix vulnerabilities.
References
• https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

Proof of Concept

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

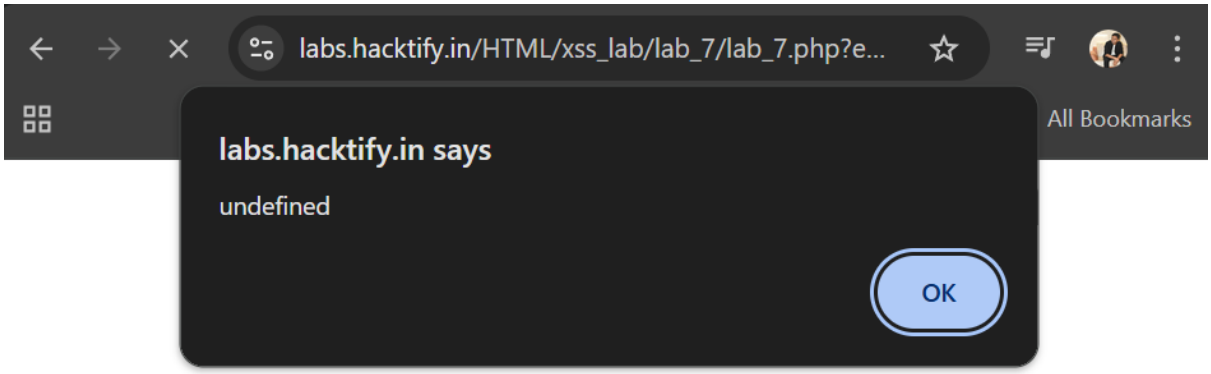


1.7. Encoding is the key?

Reference	Risk Rating
Sub-lab-7: Encoding is the key?	**Medium**
Tools Used	
Browser Inspector (Automated tools)	
Vulnerability Description	
Cross-Site Scripting (XSS) is a vulnerability that allows an attacker to inject malicious scripts into web pages viewed by other users. This type of vulnerability occurs when a web application includes untrusted data without proper validation or escaping.	
How It Was Discovered	
The vulnerability was discovered by using the automated tool, Browser Inspector, which identified points where user input was not properly sanitized, allowing JavaScript injection.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_7/lab_7.php?email=%253Cscript%253Ealert%2528document.cookies%2529%253C%252Fscript%253E	
Consequences of not Fixing the Issue	
If this vulnerability is not patched, an attacker could execute malicious scripts in a victim's browser. This could lead to the theft of sensitive data, session hijacking, defacement of the site, or the spread of malware.	
Suggested Countermeasures	
Input Validation: Ensure that all user inputs are validated, escaping any characters that could be interpreted as code. **Use of Security Headers:** Implement HTTP security headers such as Content Security Policy (CSP) to restrict the sources from which scripts can be loaded. **Output Encoding:** Use context-sensitive output encoding to prevent the execution of injected scripts. **Sanitize HTML Input:** Use libraries or frameworks that sanitize inputs and prevent harmful code execution. **Regular Security Audits:** Conduct regular security audits using automated tools and manual testing to identify and fix vulnerabilities.	
References	
• https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html	

Proof of Concept

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

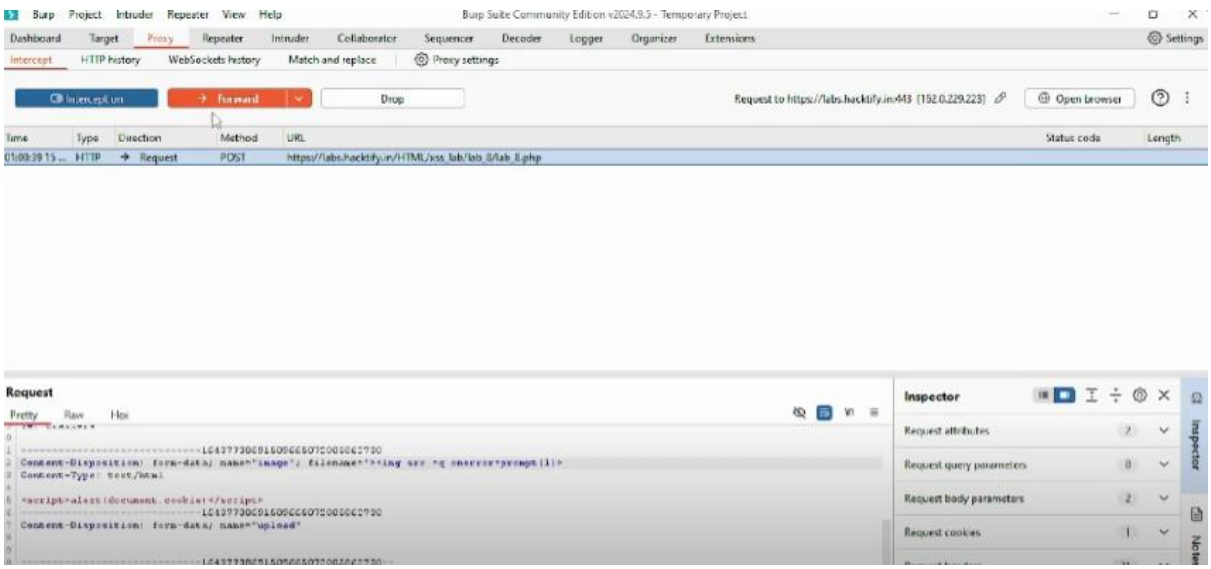


1.8. XSS with File Upload(file name)

Reference	Risk Rating
Sub-lab-8: XSS with File Upload(File Name)	**Low**
Tools Used	
Brup Suite Community Edition	
Vulnerability Description	
Cross-Site Scripting (XSS) is a vulnerability that allows an attacker to inject malicious scripts into web pages viewed by other users. This type of vulnerability occurs when a web application includes untrusted data without proper validation or escaping.	
How It Was Discovered	
The vulnerability was discovered by using the automated tool, Browser Inspector, which identified points where user input was not properly sanitized, allowing JavaScript injection.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_8/lab_8.php	
Consequences of not Fixing the Issue	
If this vulnerability is not patched, an attacker could execute malicious scripts in a victim's browser. This could lead to the theft of sensitive data, session hijacking, defacement of the site, or the spread of malware.	
Suggested Countermeasures	
Input Validation: Ensure that all user inputs are validated, escaping any characters that could be interpreted as code. **Use of Security Headers:** Implement HTTP security headers such as Content Security Policy (CSP) to restrict the sources from which scripts can be loaded. **Output Encoding:** Use context-sensitive output encoding to prevent the execution of injected scripts. **Sanitize HTML Input:** Use libraries or frameworks that sanitize inputs and prevent harmful code execution. **Regular Security Audits:** Conduct regular security audits using automated tools and manual testing to identify and fix vulnerabilities.	
References	
• https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html	

Proof of Concept

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



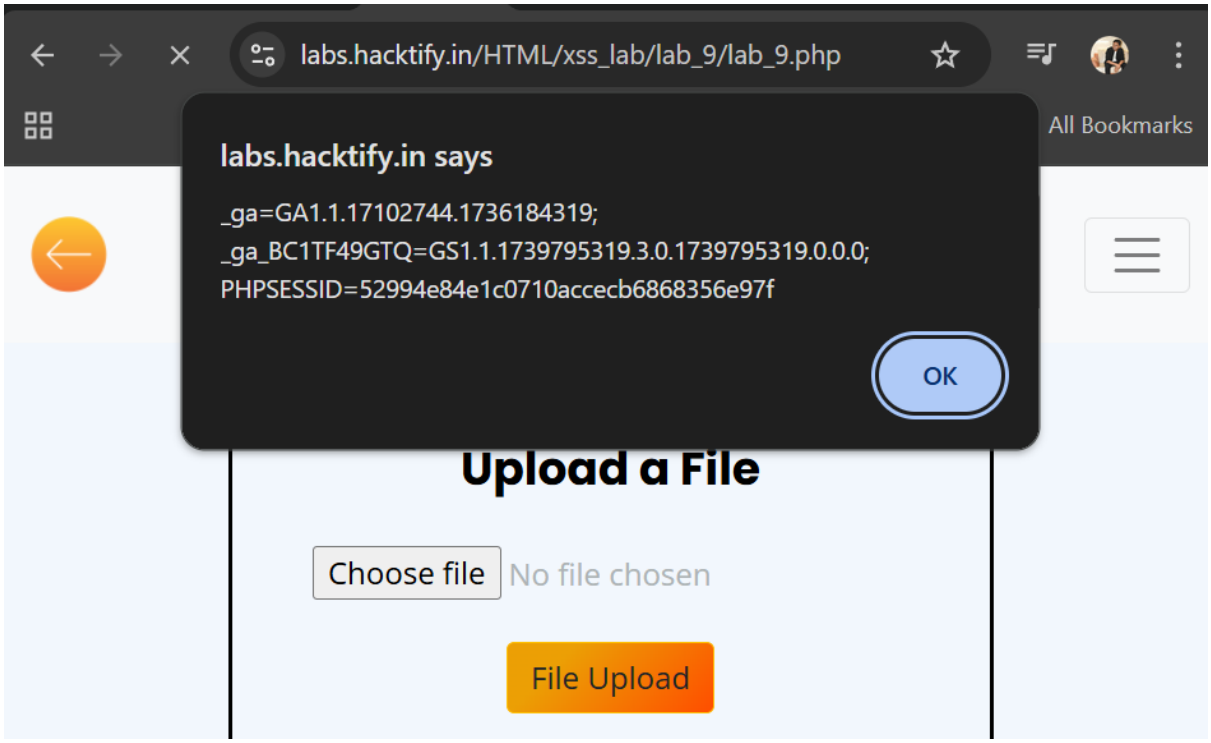
1.9. XSS with File Upload(File Content)

Reference	Risk Rating
Sub-lab-9: XSS with File Upload(File Content)	
Tools Used	
Brup Suite Community Edition	
Vulnerability Description	
Cross-Site Scripting (XSS) is a vulnerability that allows an attacker to inject malicious scripts into web pages viewed by other users. This type of vulnerability occurs when a web application includes untrusted data without proper validation or escaping.	
How It Was Discovered	
The vulnerability was discovered by using the automated tool, Browser Inspector, which identified points where user input was not properly sanitized, allowing JavaScript injection.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_9/lab_9.php	
Consequences of not Fixing the Issue	
If this vulnerability is not patched, an attacker could execute malicious scripts in a victim's browser. This could lead to the theft of sensitive data, session hijacking, defacement of the site, or the spread of malware.	
Suggested Countermeasures	
Input Validation: Ensure that all user inputs are validated, escaping any characters that could be interpreted as code. **Use of Security Headers:** Implement HTTP security headers such as Content Security Policy (CSP) to restrict the sources from which scripts can be loaded. **Output Encoding:** Use context-sensitive output encoding to prevent the execution of injected scripts.	

Sanitize HTML Input: Use libraries or frameworks that sanitize inputs and prevent harmful code execution. **Regular Security Audits:** Conduct regular security audits using automated tools and manual testing to identify and fix vulnerabilities.
References
• https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

Proof of Concept

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



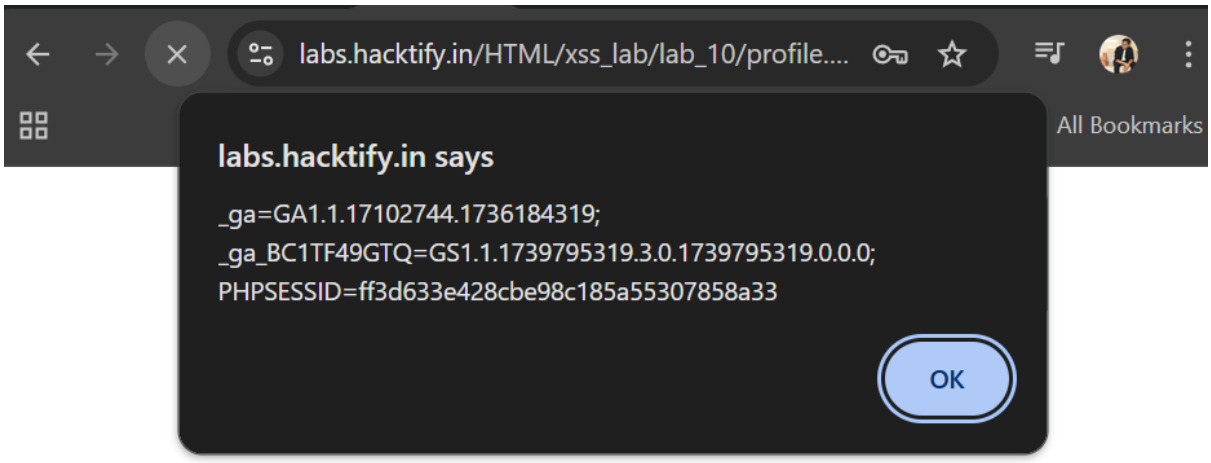
1.10. Stored Everywhere!

Reference	Risk Rating
Sub-lab-10: Stored Everywhere	Low
Tools Used	
Browser Inspector (Automated tools)	
Vulnerability Description	
Cross-Site Scripting (XSS) is a vulnerability that allows an attacker to inject malicious scripts into web pages viewed by other users. This type of vulnerability occurs when a web application includes untrusted data without proper validation or escaping.	

How It Was Discovered
The vulnerability was discovered by using the automated tool, Browser Inspector, which identified points where user input was not properly sanitized, allowing JavaScript injection.
Vulnerable URLs
https://labs.hacktify.in/HTML/xss_lab/lab_10/profile.php
Consequences of not Fixing the Issue
If this vulnerability is not patched, an attacker could execute malicious scripts in a victim's browser. This could lead to the theft of sensitive data, session hijacking, defacement of the site, or the spread of malware.
Suggested Countermeasures
Input Validation: Ensure that all user inputs are validated, escaping any characters that could be interpreted as code. **Use of Security Headers:** Implement HTTP security headers such as Content Security Policy (CSP) to restrict the sources from which scripts can be loaded. **Output Encoding:** Use context-sensitive output encoding to prevent the execution of injected scripts. **Sanitize HTML Input:** Use libraries or frameworks that sanitize inputs and prevent harmful code execution. **Regular Security Audits:** Conduct regular security audits using automated tools and manual testing to identify and fix vulnerabilities.
References
• https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

Proof of Concept

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

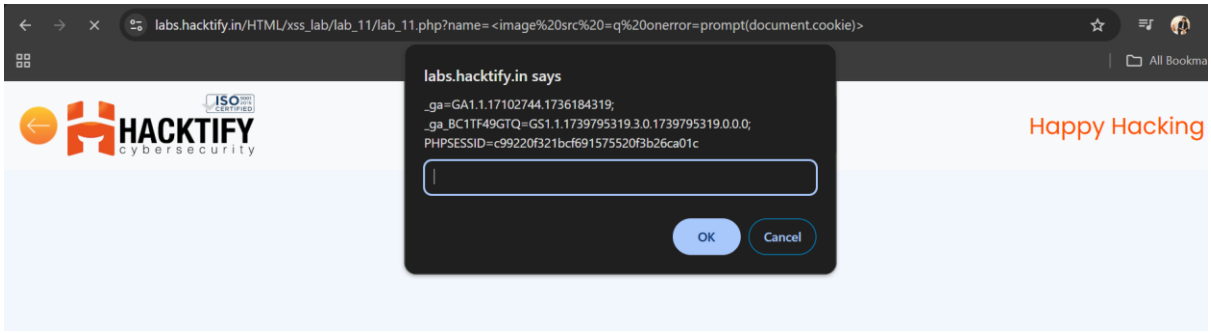


1.11. DOM'S are love!

Reference	Risk Rating
Sub-lab-11: DOM'S are love	Hard
Tools Used	
Browser Inspector (Automated tools)	
Vulnerability Description	
Cross-Site Scripting (XSS) is a vulnerability that allows an attacker to inject malicious scripts into web pages viewed by other users. This type of vulnerability occurs when a web application includes untrusted data without proper validation or escaping.	
How It Was Discovered	
The vulnerability was discovered by using the automated tool, Browser Inspector, which identified points where user input was not properly sanitized, allowing JavaScript injection.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/xss_lab/lab_11/lab_11.php?name=%3Cimage%20src%20=q%20onerror=prompt(document.cookie)%3E	
Consequences of not Fixing the Issue	
If this vulnerability is not patched, an attacker could execute malicious scripts in a victim's browser. This could lead to the theft of sensitive data, session hijacking, defacement of the site, or the spread of malware.	
Suggested Countermeasures	
Input Validation: Ensure that all user inputs are validated, escaping any characters that could be interpreted as code. **Use of Security Headers:** Implement HTTP security headers such as Content Security Policy (CSP) to restrict the sources from which scripts can be loaded. **Output Encoding:** Use context-sensitive output encoding to prevent the execution of injected scripts. **Sanitize HTML Input:** Use libraries or frameworks that sanitize inputs and prevent harmful code execution. **Regular Security Audits:** Conduct regular security audits using automated tools and manual testing to identify and fix vulnerabilities.	
References	
• https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html	

Proof of Concept

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



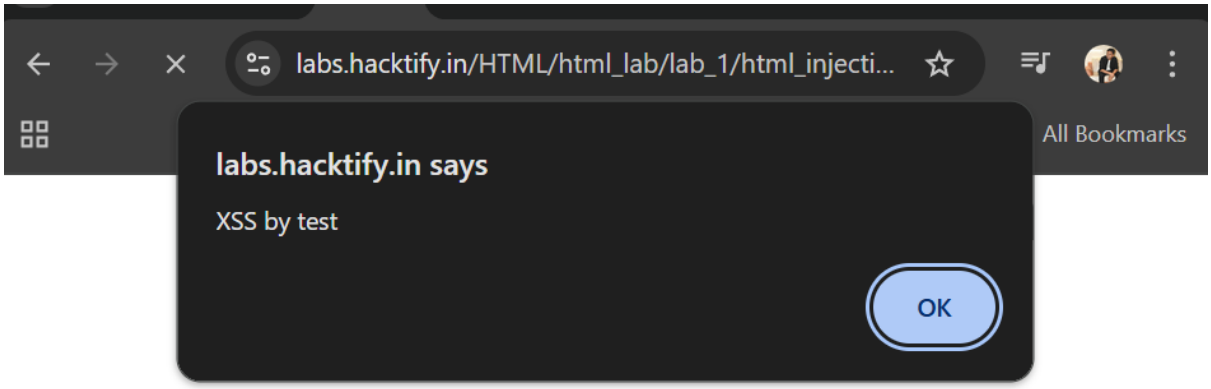
2. HTML Injection

2.1. HTML's are easy!

Reference	Risk Rating
Sub-lab-1: HTML's are easy!	**Low**
Tools Used	
Browser Developer Tools (Inspector) Burp Suite OWASP ZAP	
Vulnerability Description	
HTML Injection occurs when an application does not properly sanitize user inputs before rendering them in the browser. Attackers can inject malicious HTML content, which can modify the webpage's structure, display unauthorized content, or even execute malicious scripts in certain cases.	
How It Was Discovered	
The vulnerability was identified using: • **Automated Tools:** Burp Suite and OWASP ZAP were used to intercept and modify HTTP requests. • **Manual Analysis:** Browser Developer Tools (Inspector) helped locate input fields that reflected user input directly onto the webpage.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_1/html_injection_1.php	
Consequences of not Fixing the Issue	
If HTML Injection is not patched, it can lead to: • **Defacement of Web Pages:** Attackers can modify the website's appearance by injecting custom HTML. • **Phishing Attacks:** Malicious links or fake login forms can be inserted to steal user credentials. • **Disruption of Functionality:** Attackers can insert broken HTML or JavaScript, causing layout issues or crashes.	
Suggested Countermeasures	
Input Validation: Ensure all user inputs are validated against a whitelist of allowed characters. **Output Encoding:** Encode user-generated content to prevent it from being interpreted as HTML. **Content Security Policy (CSP):** Implement CSP headers to limit the execution of injected content. **Sanitization Libraries:** Use libraries such as OWASP Java HTML Sanitizer to remove unwanted tags. **Regular Security Audits:** Continuously test and monitor the application for injection vulnerabilities.	
References	
https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/	

Proof of Concept

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



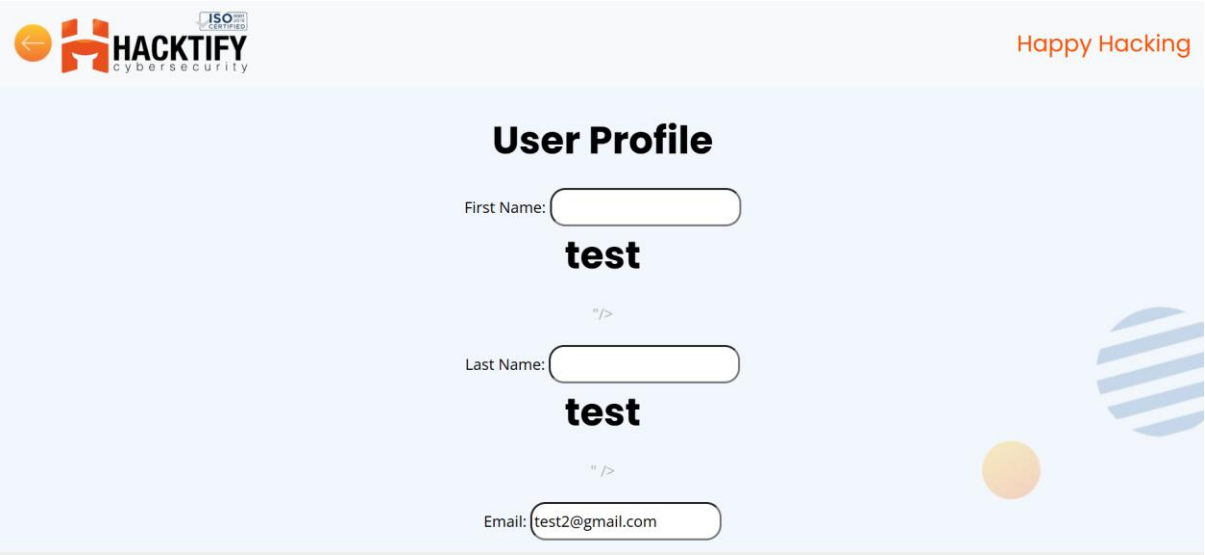
2.2. Let me Store them!

Reference	Risk Rating
Sub-lab-2: Let me Store them!	**Low**
Tools Used	
Browser Developer Tools (Inspector) Burp Suite OWASP ZAP	
Vulnerability Description	
HTML Injection occurs when an application does not properly sanitize user inputs before rendering them in the browser. Attackers can inject malicious HTML content, which can modify the webpage's structure, display unauthorized content, or even execute malicious scripts in certain cases.	
How It Was Discovered	
The vulnerability was identified using: • **Automated Tools:** Burp Suite and OWASP ZAP were used to intercept and modify HTTP requests. • **Manual Analysis:** Browser Developer Tools (Inspector) helped locate input fields that reflected user input directly onto the webpage.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_2/profile.php	
Consequences of not Fixing the Issue	
If HTML Injection is not patched, it can lead to: • **Defacement of Web Pages:** Attackers can modify the website's appearance by injecting custom HTML. • **Phishing Attacks:** Malicious links or fake login forms can be inserted to steal user credentials. • **Disruption of Functionality:** Attackers can insert broken HTML or JavaScript, causing layout issues or crashes.	

Suggested Countermeasures
Input Validation: Ensure all user inputs are validated against a whitelist of allowed characters. **Output Encoding:** Encode user-generated content to prevent it from being interpreted as HTML. **Content Security Policy (CSP):** Implement CSP headers to limit the execution of injected content. **Sanitization Libraries:** Use libraries such as OWASP Java HTML Sanitizer to remove unwanted tags. **Regular Security Audits:** Continuously test and monitor the application for injection vulnerabilities.
References
https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/

Proof of Concept

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



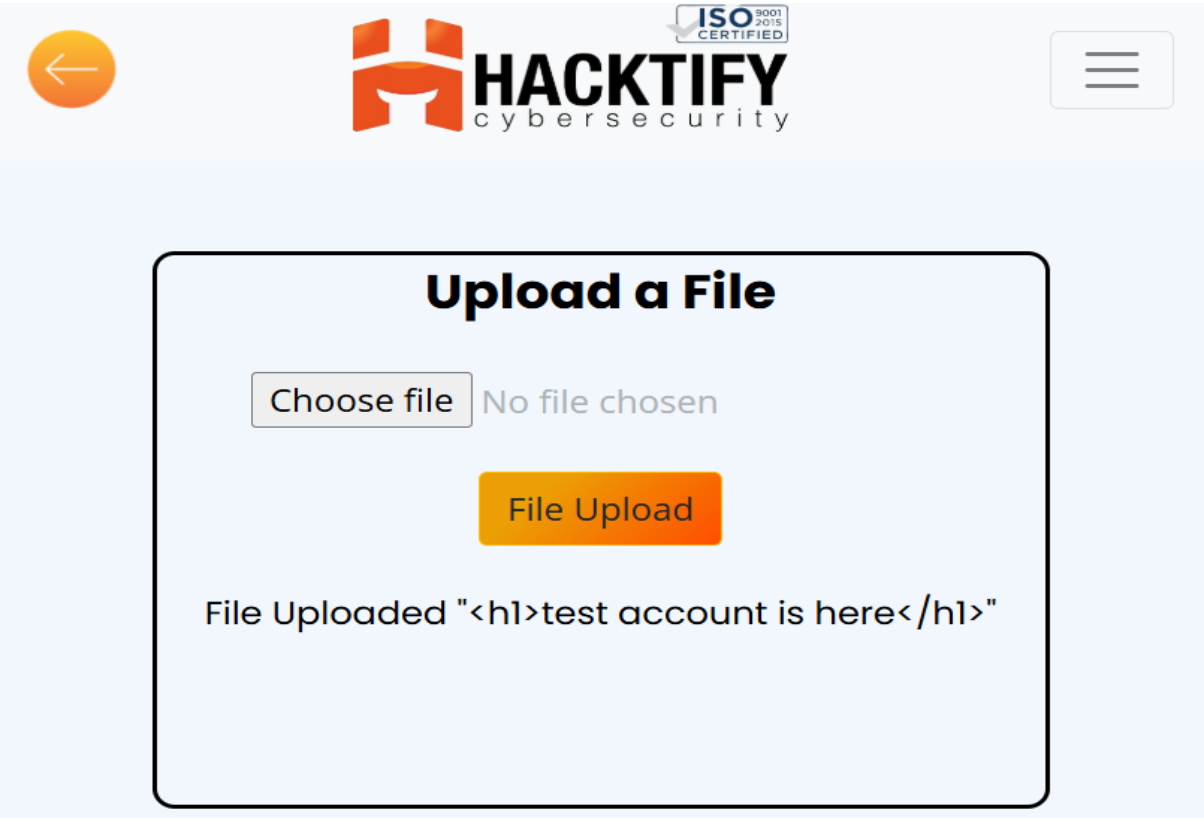
2.3. File Names are also vulnerables!

Reference	Risk Rating
Sub-lab-3: File Names are also vulnerables!	**Low**
Tools Used	
Browser Developer Tools (Inspector) Burp Suite OWASP ZAP	
Vulnerability Description	
HTML Injection occurs when an application does not properly sanitize user inputs before rendering them in the browser. Attackers can inject malicious HTML content, which can modify the webpage's structure, display unauthorized content, or even execute malicious scripts in certain cases.	
How It Was Discovered	
The vulnerability was identified using: • **Automated Tools:** Burp Suite and OWASP ZAP were used to intercept and modify HTTP requests.	

• **Manual Analysis:** Browser Developer Tools (Inspector) helped locate input fields that reflected user input directly onto the webpage.
Vulnerable URLs
https://labs.hacktify.in/HTML/html_lab/lab_3/html_injection_3.php
Consequences of not Fixing the Issue
If HTML Injection is not patched, it can lead to: • **Defacement of Web Pages:** Attackers can modify the website's appearance by injecting custom HTML. • **Phishing Attacks:** Malicious links or fake login forms can be inserted to steal user credentials. • **Disruption of Functionality:** Attackers can insert broken HTML or JavaScript, causing layout issues or crashes.
Suggested Countermeasures
Input Validation: Ensure all user inputs are validated against a whitelist of allowed characters. **Output Encoding:** Encode user-generated content to prevent it from being interpreted as HTML. **Content Security Policy (CSP):** Implement CSP headers to limit the execution of injected content. **Sanitization Libraries:** Use libraries such as OWASP Java HTML Sanitizer to remove unwanted tags. **Regular Security Audits:** Continuously test and monitor the application for injection vulnerabilities.
References
https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/

Proof of Concept

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

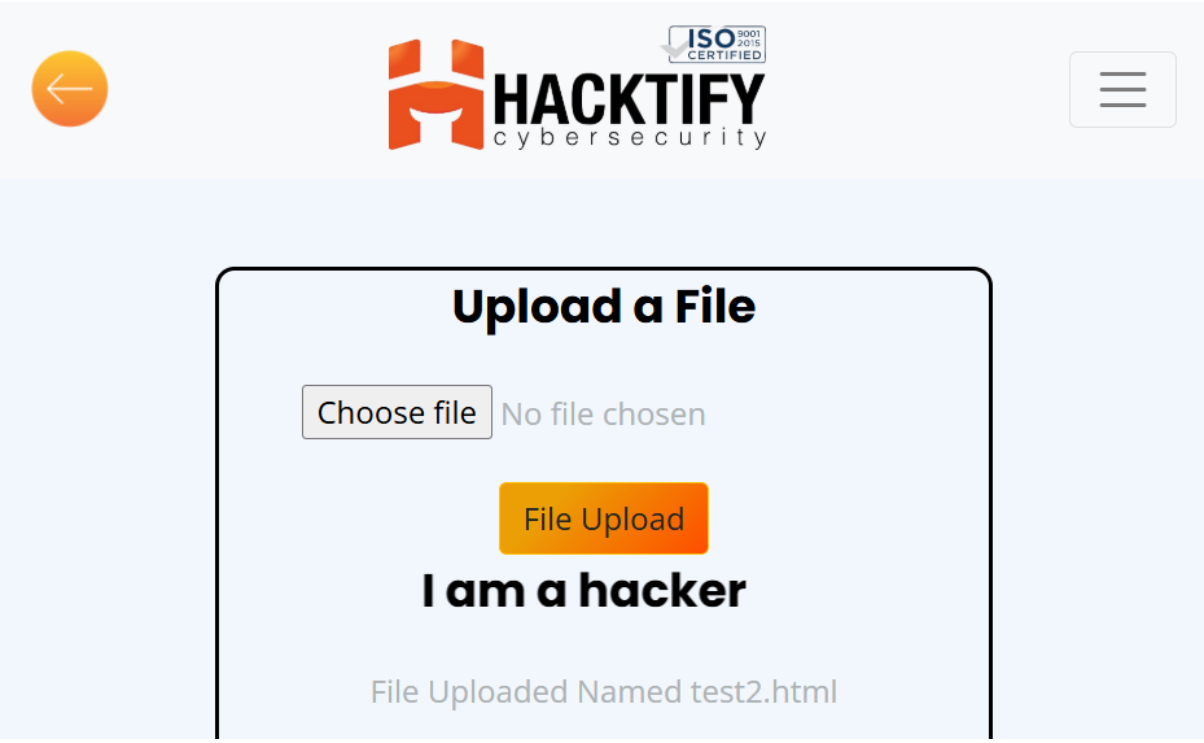


2.4. File Content and HTML Injection a perfect pair!

Reference	Risk Rating
Sub-lab-4: File Content and HTML Injection a perfect pair!	
Tools Used	
Browser Developer Tools (Inspector) Burp Suite OWASP ZAP	
Vulnerability Description	
HTML Injection occurs when an application does not properly sanitize user inputs before rendering them in the browser. Attackers can inject malicious HTML content, which can modify the webpage's structure, display unauthorized content, or even execute malicious scripts in certain cases.	
How It Was Discovered	
The vulnerability was identified using: • **Automated Tools:** Burp Suite and OWASP ZAP were used to intercept and modify HTTP requests. • **Manual Analysis:** Browser Developer Tools (Inspector) helped locate input fields that reflected user input directly onto the webpage.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_4/html_injection_4.php	
Consequences of not Fixing the Issue	
If HTML Injection is not patched, it can lead to: • **Defacement of Web Pages:** Attackers can modify the website's appearance by injecting custom HTML. • **Phishing Attacks:** Malicious links or fake login forms can be inserted to steal user credentials. • **Disruption of Functionality:** Attackers can insert broken HTML or JavaScript, causing layout issues or crashes.	
Suggested Countermeasures	
Input Validation: Ensure all user inputs are validated against a whitelist of allowed characters. **Output Encoding:** Encode user-generated content to prevent it from being interpreted as HTML. **Content Security Policy (CSP):** Implement CSP headers to limit the execution of injected content. **Sanitization Libraries:** Use libraries such as OWASP Java HTML Sanitizer to remove unwanted tags. **Regular Security Audits:** Continuously test and monitor the application for injection vulnerabilities.	
References	
https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/	

Proof of Concept

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



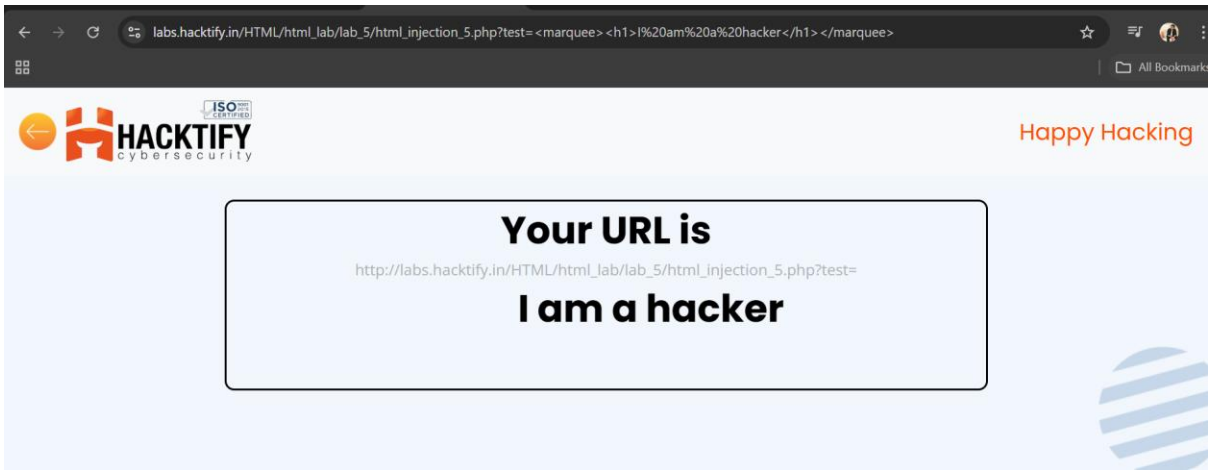
2.5. Injecting HTML using URL

Reference	Risk Rating
Sub-lab-5: Injecting HTML using URL	**Medium**
Tools Used	
Browser Developer Tools (Inspector) Burp Suite OWASP ZAP	
Vulnerability Description	
HTML Injection occurs when an application does not properly sanitize user inputs before rendering them in the browser. Attackers can inject malicious HTML content, which can modify the webpage's structure, display unauthorized content, or even execute malicious scripts in certain cases.	
How It Was Discovered	
The vulnerability was identified using: • **Automated Tools:** Burp Suite and OWASP ZAP were used to intercept and modify HTTP requests. • **Manual Analysis:** Browser Developer Tools (Inspector) helped locate input fields that reflected user input directly onto the webpage.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/html_lab/lab_5/html_injection_5.php?test=%3Cmarquee%3E%3Ch1%3EI%20am%20a%20hacker%3C/h1%3E%3C/marquee%3E	
Consequences of not Fixing the Issue	
If HTML Injection is not patched, it can lead to: • **Defacement of Web Pages:** Attackers can modify the website's appearance by injecting custom HTML. • **Phishing Attacks:** Malicious links or fake login forms can be inserted to steal user credentials. • **Disruption of Functionality:** Attackers can insert broken HTML or JavaScript, causing layout issues or crashes.	

Suggested Countermeasures
Input Validation: Ensure all user inputs are validated against a whitelist of allowed characters. **Output Encoding:** Encode user-generated content to prevent it from being interpreted as HTML. **Content Security Policy (CSP):** Implement CSP headers to limit the execution of injected content. **Sanitization Libraries:** Use libraries such as OWASP Java HTML Sanitizer to remove unwanted tags. **Regular Security Audits:** Continuously test and monitor the application for injection vulnerabilities.
References
https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/

Proof of Concept

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



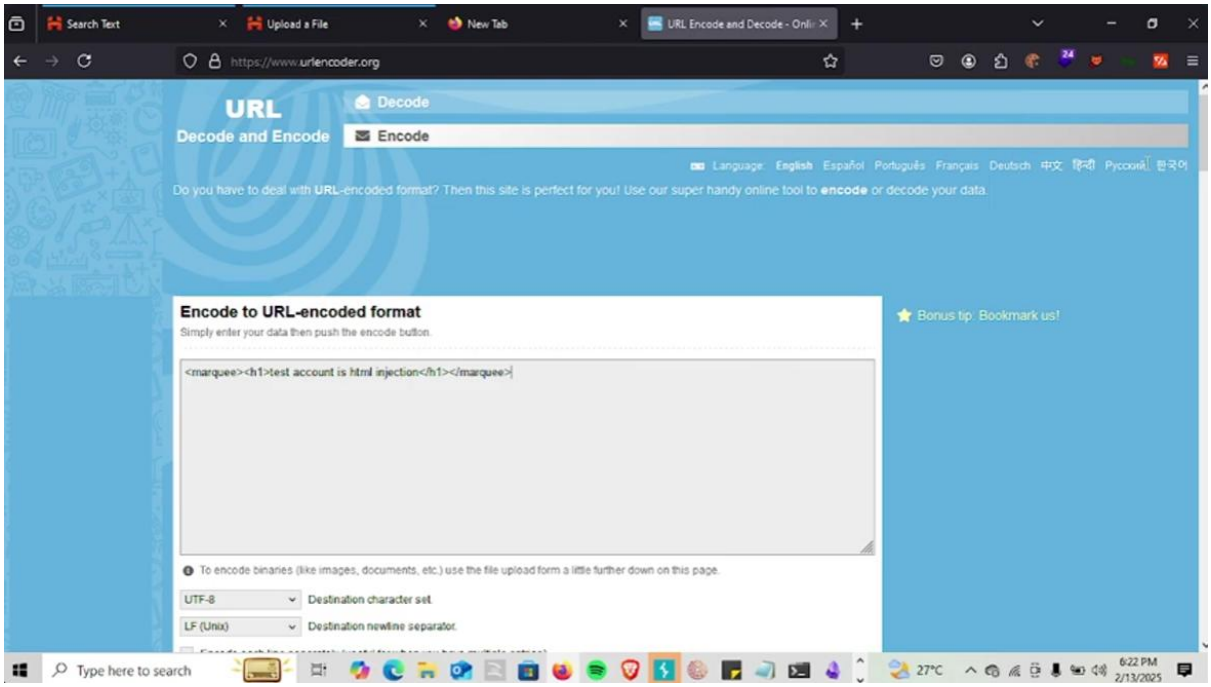
2.6. Injecting HTML using URL

Reference	Risk Rating
Sub-lab-5: Injecting HTML using URL	**Medium**
Tools Used	
Browser Developer Tools (Inspector) Burp Suite OWASP ZAP	
Vulnerability Description	
HTML Injection occurs when an application does not properly sanitize user inputs before rendering them in the browser. Attackers can inject malicious HTML content, which can modify the webpage's structure, display unauthorized content, or even execute malicious scripts in certain cases.	
How It Was Discovered	
The vulnerability was identified using: • **Automated Tools:** Burp Suite and OWASP ZAP were used to intercept and modify HTTP requests. • **Manual Analysis:** Browser Developer Tools (Inspector) helped locate input fields that reflected user input directly onto the webpage.	
Vulnerable URLs	

https://labs.hacktify.in/HTML/html_lab/lab_6/html_injection_6.php
Consequences of not Fixing the Issue
If HTML Injection is not patched, it can lead to: • **Defacement of Web Pages:** Attackers can modify the website's appearance by injecting custom HTML. • **Phishing Attacks:** Malicious links or fake login forms can be inserted to steal user credentials. • **Disruption of Functionality:** Attackers can insert broken HTML or JavaScript, causing layout issues or crashes.
Suggested Countermeasures
Input Validation: Ensure all user inputs are validated against a whitelist of allowed characters. **Output Encoding:** Encode user-generated content to prevent it from being interpreted as HTML. **Content Security Policy (CSP):** Implement CSP headers to limit the execution of injected content. **Sanitization Libraries:** Use libraries such as OWASP Java HTML Sanitizer to remove unwanted tags. **Regular Security Audits:** Continuously test and monitor the application for injection vulnerabilities.
References
https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/

Proof of Concept

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



NOTES:

- **Everything mentioned inside {} has to be changed based on your week, labs and sub-labs.**

- **If you have 2 labs in same week you need to mention that, if not ignore those mentions for lab 2.**
- **Here it is given with 2 Sub-labs vulnerability, you need to add all the sub-labs based on your labs.**
- **Don’t forget to add the screenshot of the vulnerability in the proof of concept.**
- **Add only 1 screenshot in the Proof of Concept section.**
- **This NOTE session is only for your reference, don’t forget to delete this in the report you submit.**