**Penetration Testing Report**

**Full Name: Saurav Pundir  
Program: HCS**

**Date: 12-02-2025**

**Introduction**

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

**1. Objective**

The aim of this study is to investigate and analyze HTML injection and clickjacking vulnerabilities as demonstrated in **Week 1 Labs**. The objective of this assessment is to identify, document, and understand these vulnerabilities thoroughly, outlining remediation strategies and providing recommendations to mitigate the risks associated with HTML injection and clickjacking in web applications.

**2. Scope**

This section defines the scope and boundaries of the project.

|  |  |
| --- | --- |
| **Application Name** | **Cross Site Scripting, HTML Injection** |

**3. Summary**

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

**Total number of Sub-labs: 17 Sub-labs**

|  |  |  |
| --- | --- | --- |
| **High** | **Medium** | **Low** |
| **4** | **5** | **8** |

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

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

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

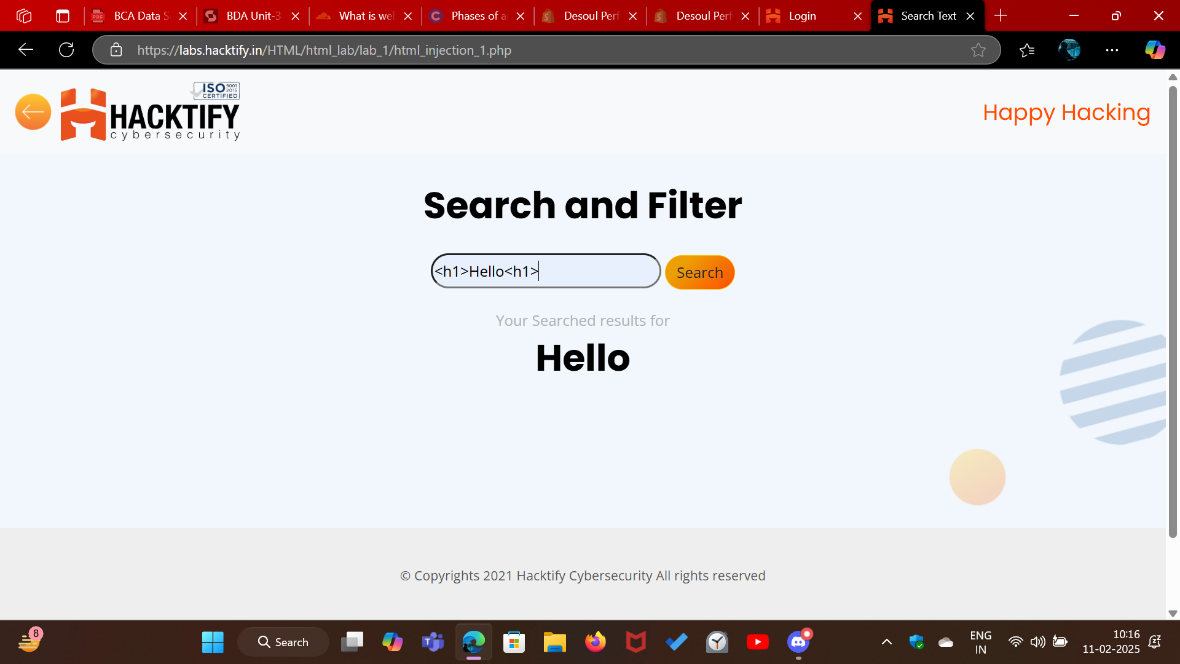
# 1. HTML Injection

# 1.1. HTML's Are Easy!

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| HTML's Are Easy! | **Low** |
| **Tools Used** | |
| Exploiting a straightforward HTML injection method via the search bar functionality (No tool used). | |
| **Vulnerability Description** | |
| The vulnerability allows an attacker to inject malicious HTML code into the search bar of the website. When the search query containing the injected code is submitted, the website's server processes it and responds with a page that includes the injected code. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/html\_lab/lab\_1/html\_injection\_1.php | |
| **Consequences of not Fixing the Issue** | |
| Failure to patch this vulnerability could lead to execute arbitrary code in the victim's browser. This could lead to theft of sensitive information, such as session cookies or user credentials, manipulation of page content, redirection to malicious websites, or other unauthorized actions. | |
| **Suggested Countermeasures** | |
| Implement robust input validation and sanitization procedures to mitigate the risk of HTML injection attacks. | |
| **References** | |
| <https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/03-Testing_for_HTML_Injection> | |

# 

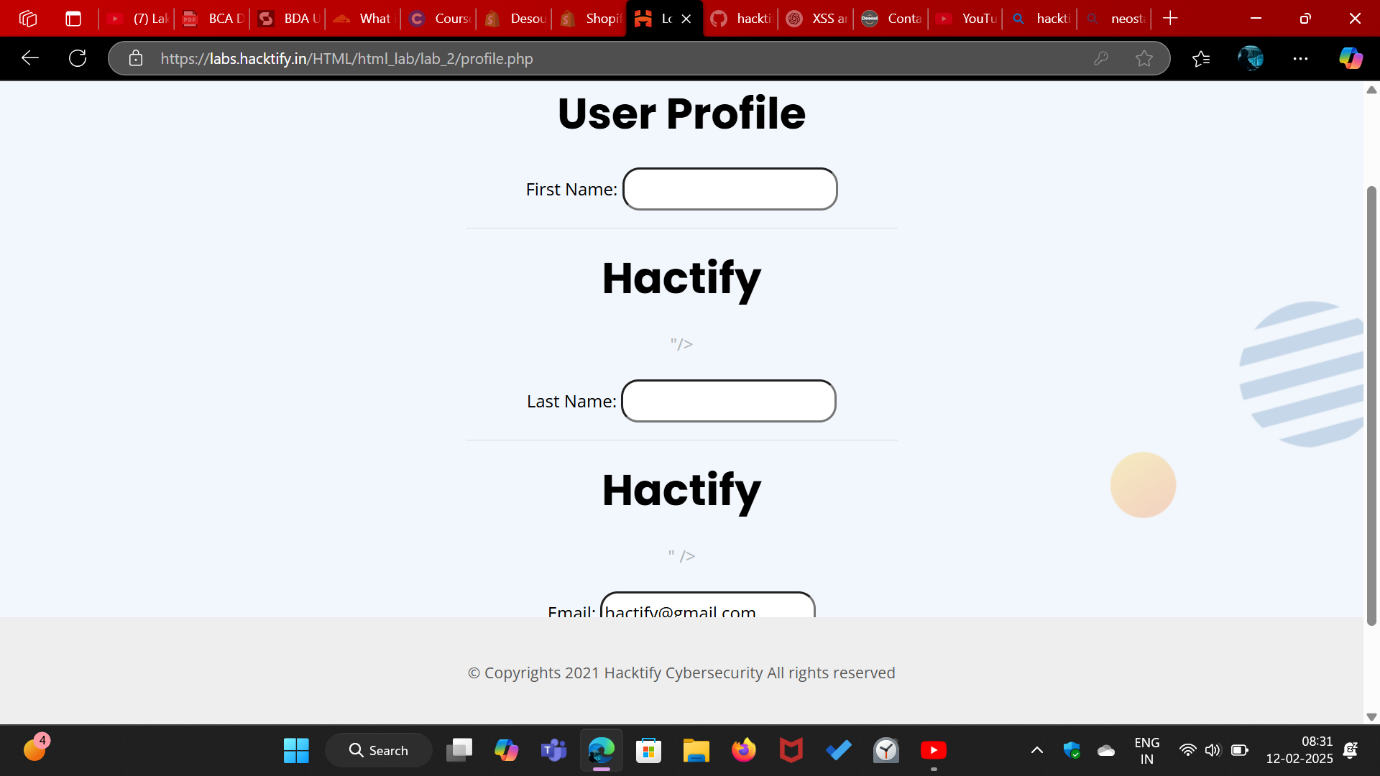
# Proof of Concept



# 1.2. Let Me Store Them!

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| Let Me Store Them! | **Low** |
| **Tools Used** | |
| Manual analysis | |
| **Vulnerability Description** | |
| Observed that input text field is executing injected HTML code. | |
| **How It Was Discovered** | |
| Through manual analysis of input fields and subsequent observation of injected HTML code. | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/html\_lab/lab\_2/html\_injection\_2.php | |
| **Consequences of not Fixing the Issue** | |
| Risk of data theft, unauthorized access, reputational damage, and legal liabilities. | |
| **Suggested Countermeasures** | |
| Implement strict input validation, output encoding, and regular security updates. | |
| **References** | |
| <https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/03-Testing_for_HTML_Injection> | |

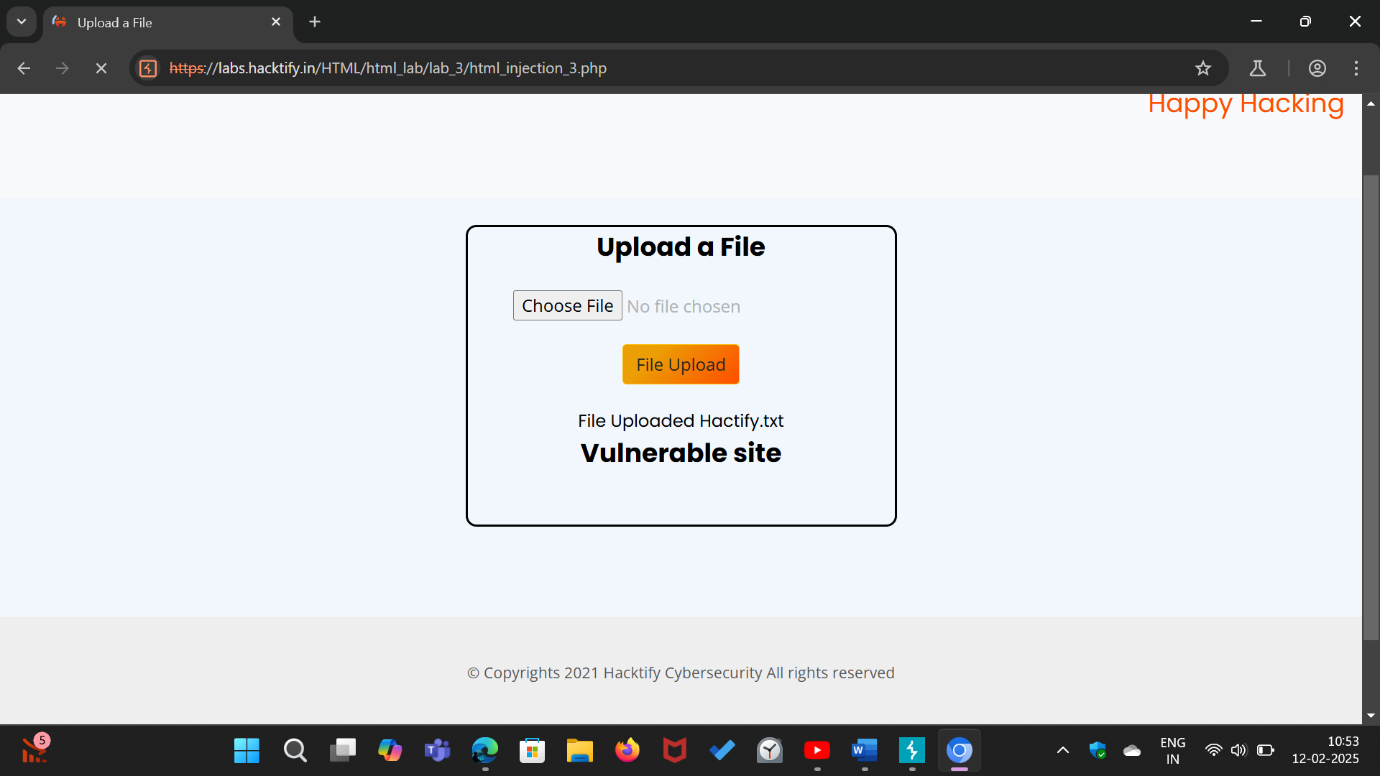
# Proof of Concept

****

# 1.3. File Names are also vulnerable!

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| File Names are also vulnerable! | **Low** |
| **Tools Used** | |
| Burp Suite for intercepting and modifying requests and Dev tool for analyzing the Code using inspect element. | |
| **Vulnerability Description** | |
| HTML injection vulnerability exploited by altering the file name field to inject HTML tags, which are reflected back and executed by the web application, potentially leading to XSS attacks. | |
| **How It Was Discovered** | |
| Manual analysis by intercepting and modifying requests using Burp Suite. | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/html\_lab/lab\_3/html\_injection\_3.php | |
| **Consequences of not Fixing the Issue** | |
| Risk of XSS attacks, data manipulation, and unauthorized access to sensitive information, leading to compromised security, reputation damage, and legal liabilities. | |
| **Suggested Countermeasures** | |
| Implement strict input validation, sanitize user-supplied data, and employ content security policies to mitigate HTML injection vulnerabilities. | |
| **References** | |
| <https://portswigger.net/burp/documentation/desktop/external-browser-config>  <https://portswigger.net/burp/documentation/desktop/getting-started/modifying-http-requests> | |

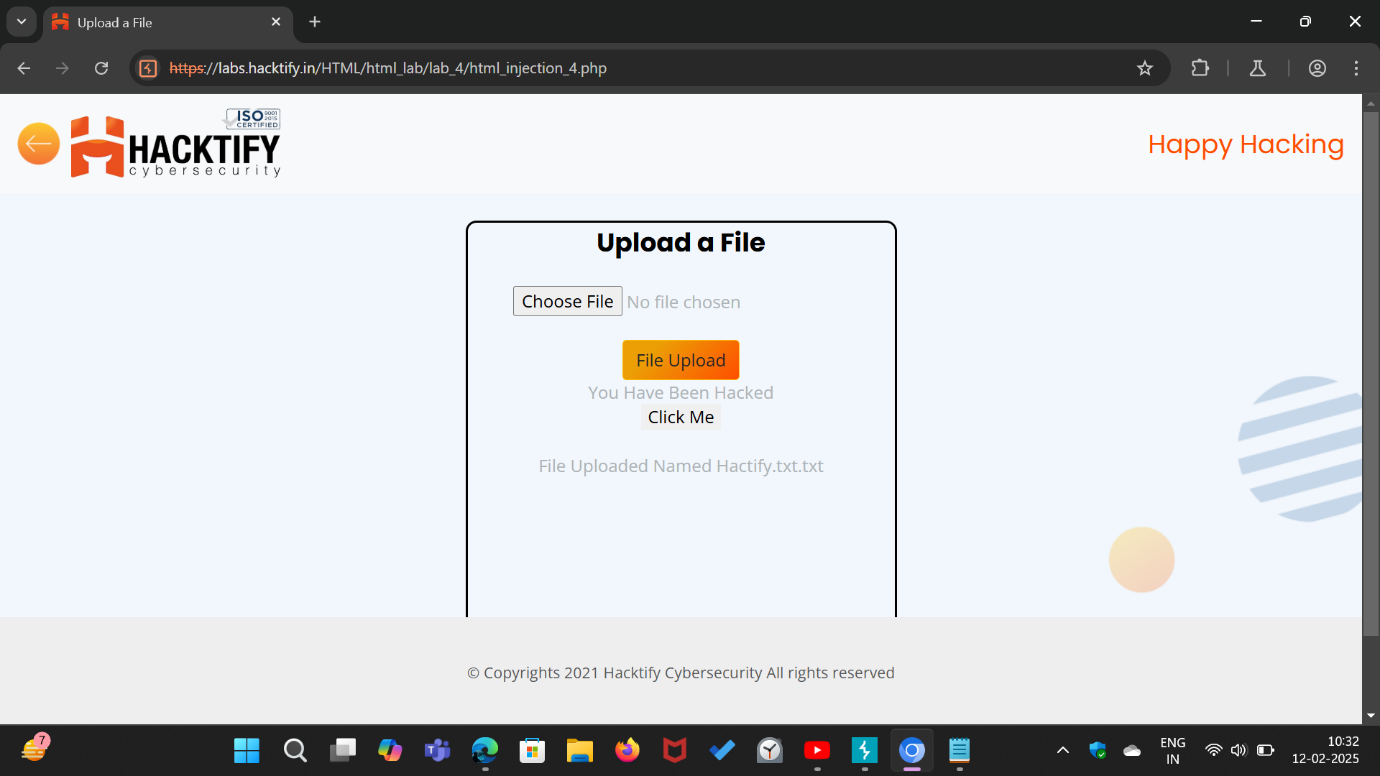
# Proof of Concept



# 1.4. File Content and HTML Injection a perfect pair!

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| File Content and HTML Injection a perfect pair! | **Medium** |
| **Tools Used** | |
| Manual testing was conducted. | |
| **Vulnerability Description** | |
| The vulnerability allows for HTML injection through file upload functionality, enabling the insertion of malicious code (inside the file) into the web page. | |
| **How It Was Discovered** | |
| Manual analysis of the file upload functionality. | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/html\_lab/lab\_4/html\_injection\_4.php | |
| **Consequences of not Fixing the Issue** | |
| Failure to patch the vulnerability could result in malicious code execution, data theft, and compromise of user confidentiality. | |
| **Suggested Countermeasures** | |
| Implement strict input validation, file type verification, and output encoding to prevent HTML injection via file uploads. | |
| **References** | |
| <https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/03-Testing_for_HTML_Injection> | |

# Proof of Concept

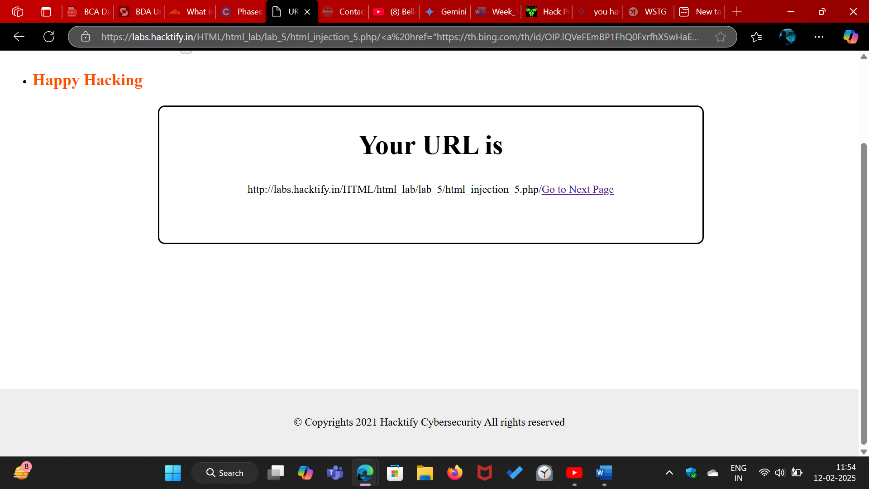


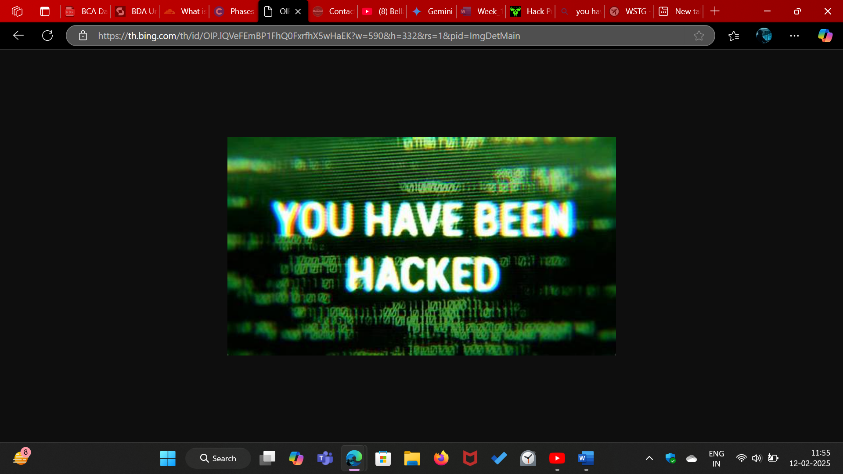
# 

# 1.5. Injecting HTML using URL

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| Injecting HTML using URL | **Medium** |
| **Tools Used** | |
| Manual Testing and Exploiting | |
| **Vulnerability Description** | |
| Exploiting lack of input validation to inject HTML code, resulting in arbitrary code execution. | |
| **How It Was Discovered** | |
| By altering the URL parameters. | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/html\_lab/lab\_5/html\_injection\_5.php | |
| **Consequences of not Fixing the Issue** | |
| Risk of unauthorized content injection, XSS attacks, data compromise, and reputational damage. | |
| **Suggested Countermeasures** | |
| Implement robust input validation, output encoding, and regular security updates. | |
| **References** | |
| <https://www.w3schools.com/tags/tag_img.asp>  <https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/03-Testing_for_HTML_Injection> | |

# Proof of Concept

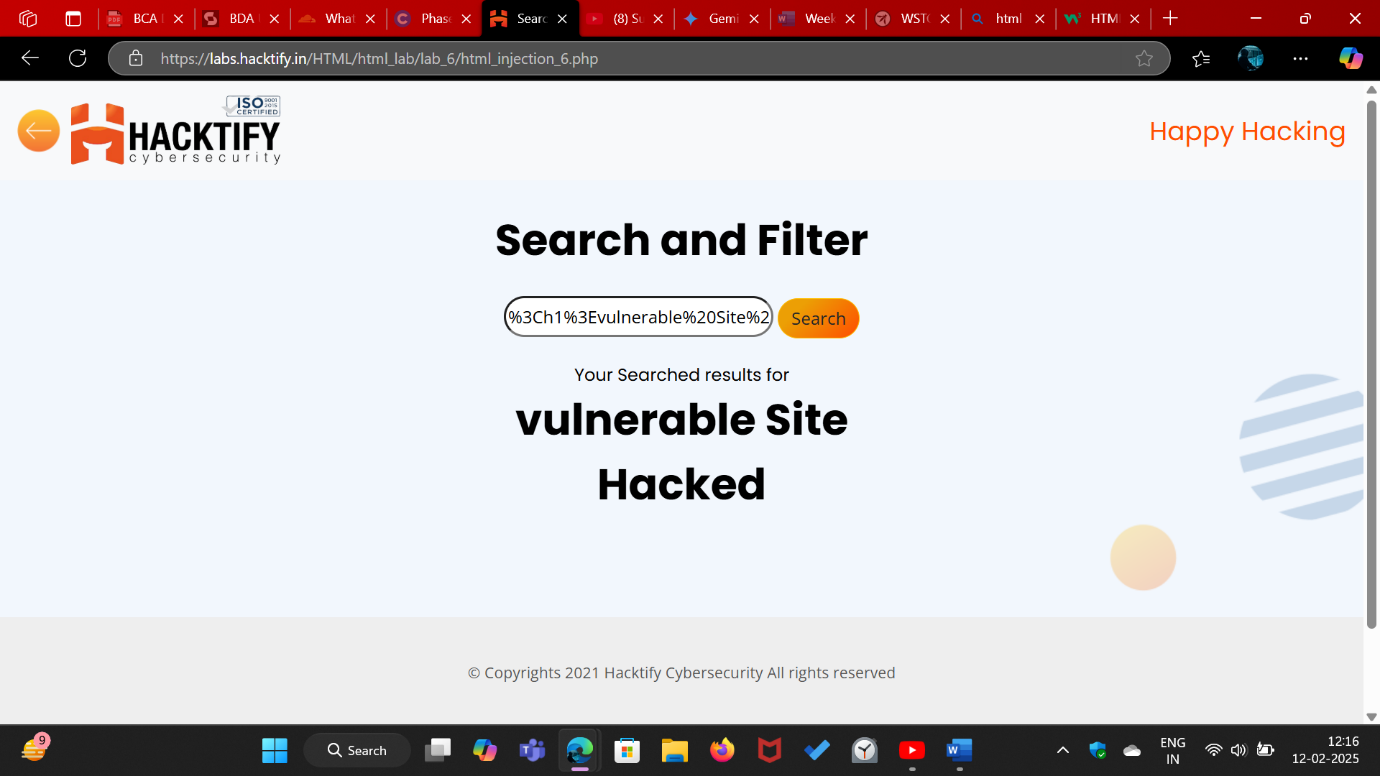
****

****

# 1.6. Encode IT!

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| Encode IT! | **High** |
| **Tools Used** | |
| Manual analysis and a website for converting HTML tags to hex code. | |
| **Vulnerability Description** | |
| HTML injection vulnerability exploited by converting HTML tags to hex code to bypass input validation. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/html\_lab/lab\_6/html\_injection\_6.php | |
| **Consequences of not Fixing the Issue** | |
| Risk of unauthorized data manipulation, XSS attacks, and reputational damage if left unpatched. | |
| **Suggested Countermeasures** | |
| Implement robust input validation and output strong encoding to prevent HTML injection attacks, and regularly update security measures. | |
| **References** | |
| <https://www.w3schools.com/tags/ref_urlencode.ASP>  <https://www.w3docs.com/tools/html-encoder/> | |

# Proof of Concept

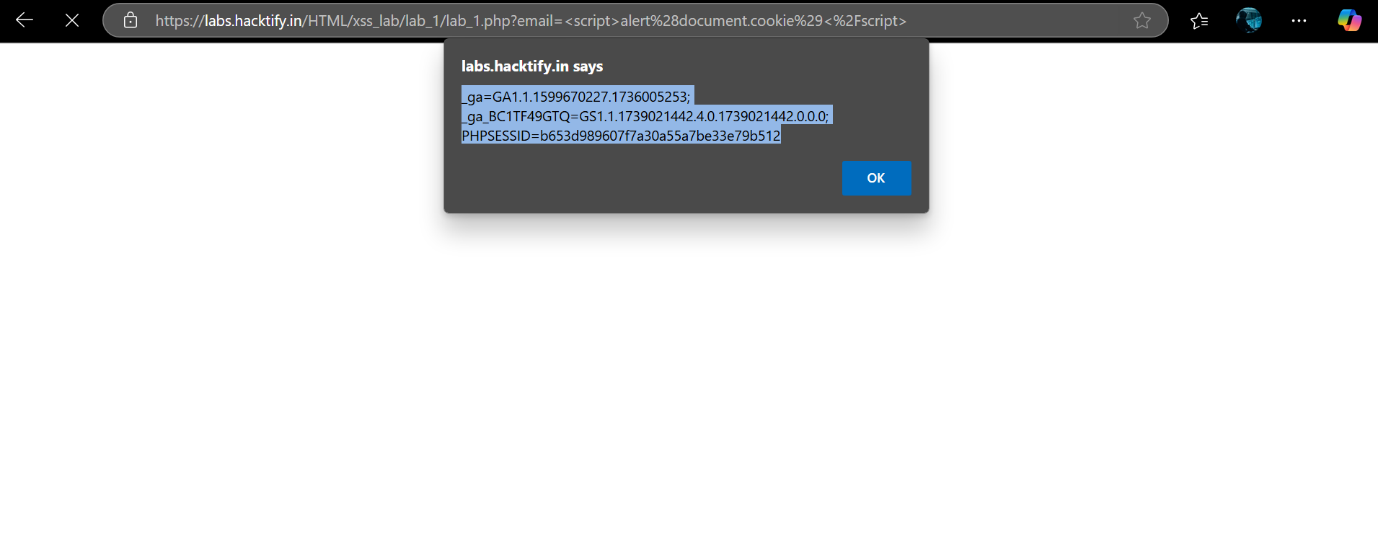
****

# 2. Cross Site Scripting

# 2.1. Let’s Do IT!

|  |  |
| --- | --- |
| Reference | Risk Rating |
| Let’s Do IT! | Low |
| Tools Used | |
| Manually Exploiting a straightforward XSS method via the email input functionality. | |
| Vulnerability Description | |
| The XSS (Cross-Site Scripting) vulnerability in the email subscription input box of a website occurs when the application fails to properly sanitize user-supplied input. Specifically, when an attacker inserts a malicious script into the input box, the website's server reflects this input back to the user's browser without proper validation or encoding. | |
| How It Was Discovered | |
| Manual Analysis | |
| Vulnerable URLs | |
| https://labs.hacktify.in/HTML/xss\_lab/lab\_1/index.php | |
| Consequences of not Fixing the Issue | |
| Failure to patch this vulnerability can lead the attackers to steal sensitive user information (such as cookies or credentials), execution of arbitrary scripts in the context of other users' browsers, leading to unauthorized actions, compromise of user sessions, allowing attackers to impersonate legitimate users and perform malicious activities. | |
| Suggested Countermeasures | |
| Implement strict input validation and output encoding to sanitize user-supplied data and prevent script injection. | |
| References | |
| <https://owasp.org/www-project-web-security-testing-guide/latest/>  <https://portswigger.net/web-security/cross-site-scripting/cheat-sheet> | |

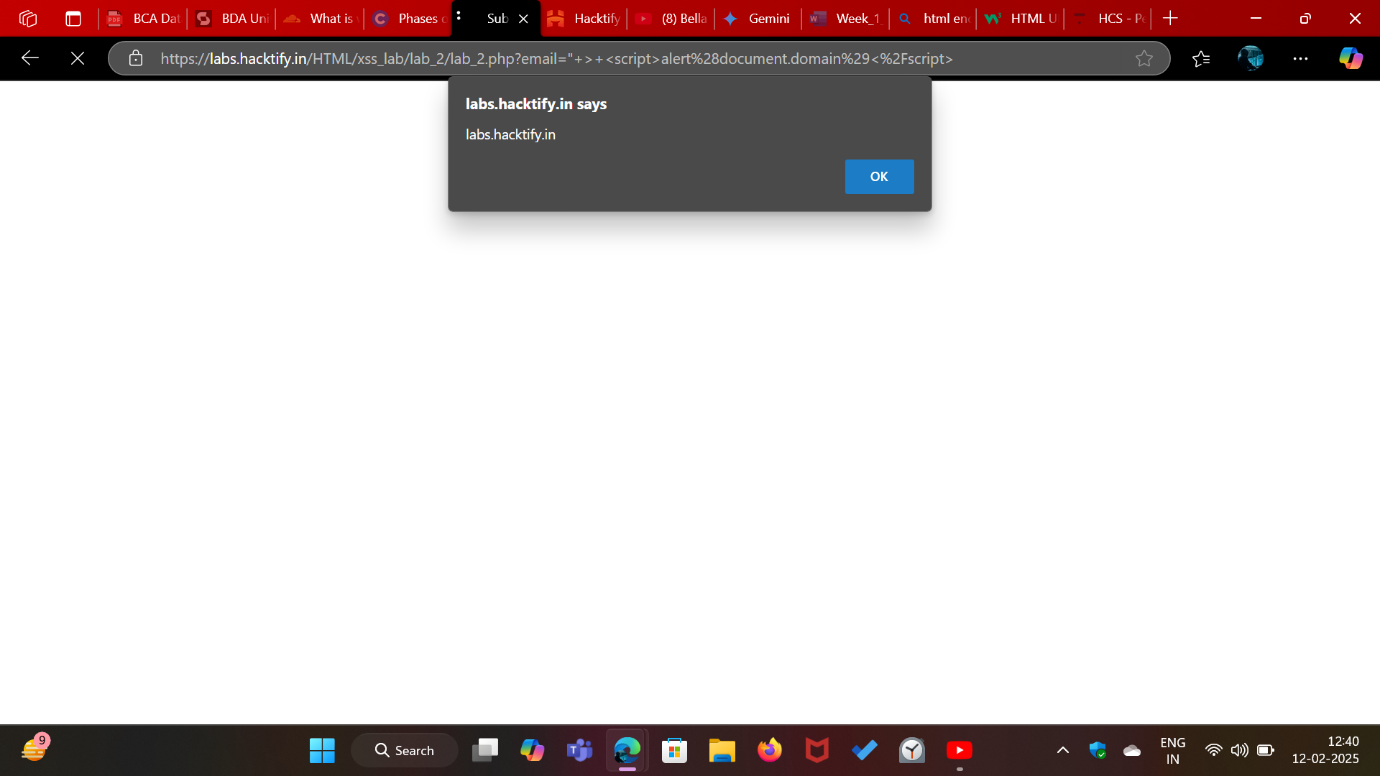
# Proof of Concept



# 2.2. Balancing Is Important in Life!

|  |  |
| --- | --- |
| Reference | Risk Rating |
| Balancing Is Important in Life! | Low |
| Tools Used | |
| Manual analysis and exploited | |
| Vulnerability Description | |
| The XSS (Cross-Site Scripting) vulnerability in the email subscription input box of a website occurs when the application fails to properly sanitize user-supplied input. Specifically, when an attacker inserts a malicious script into the input box, the website's server reflects this input back to the user's browser without proper validation or encoding. | |
| How It Was Discovered | |
| Manual Analysis | |
| Vulnerable URLs | |
| https://labs.hacktify.in/HTML/xss\_lab/lab\_2/index.php | |
| Consequences of not Fixing the Issue | |
| Failure to patch this vulnerability can lead the attackers to steal sensitive user information (such as cookies or credentials), execution of arbitrary scripts in the context of other users' browsers, leading to unauthorized actions, compromise of user sessions, allowing attackers to impersonate legitimate users and perform malicious activities. | |
| Suggested Countermeasures | |
| Implement strict input validation and output encoding to sanitize user-supplied data and prevent script injection. | |
| References | |
| <https://owasp.org/www-project-web-security-testing-guide/latest/> | |

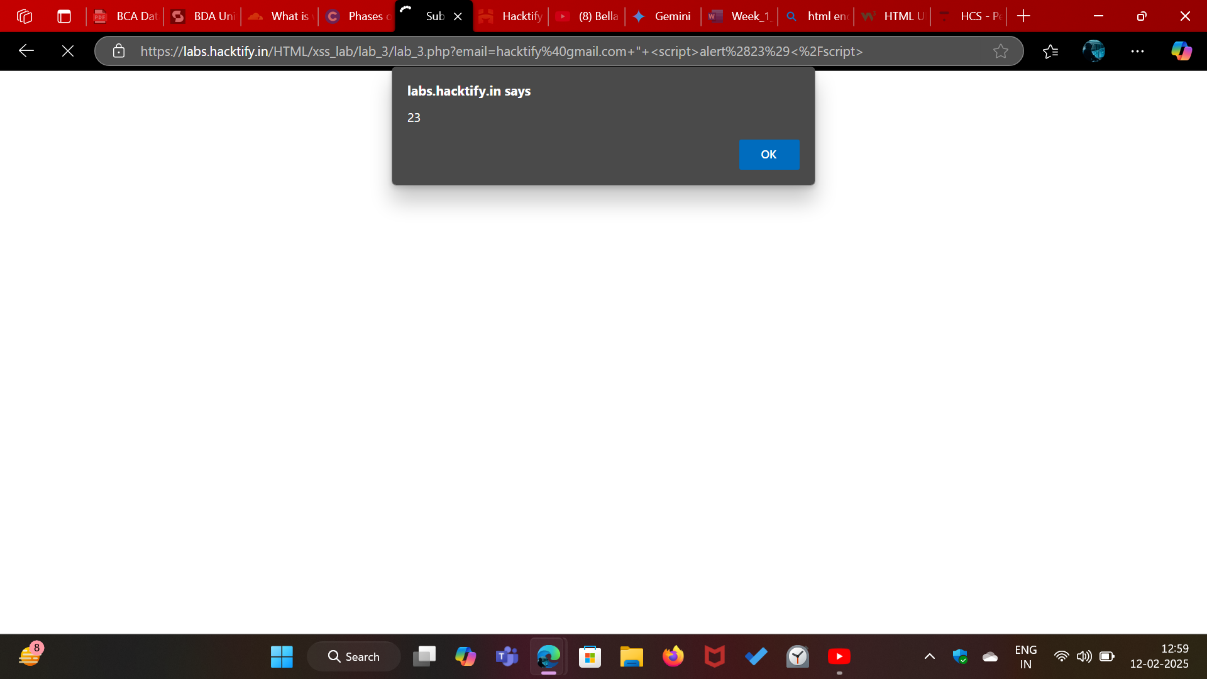
# Proof of Concept

****

# 2.3. XSS is Everywhere!

|  |  |
| --- | --- |
| Reference | Risk Rating |
| XSS is Everywhere! | Low |
| Tools Used | |
| Manual analysis without specific tools. | |
| Vulnerability Description | |
| The vulnerability in the email subscription input box of a website persists despite the addition of email validation. In this scenario, the website employs email validation to ensure that the input conforms to an email address format. However, the validation mechanism fails to adequately sanitize the input, allowing for XSS (Cross-Site Scripting) attacks. | |
| How It Was Discovered | |
| Manual Analysis | |
| Vulnerable URLs | |
| https://labs.hacktify.in/HTML/xss\_lab/lab\_3/index.php | |
| Consequences of not Fixing the Issue | |
| Failure to patch this vulnerability can lead the attackers to steal sensitive user information (such as cookies or credentials), execution of arbitrary scripts in the context of other users' browsers, leading to unauthorized actions, compromise of user sessions, allowing attackers to impersonate legitimate users and perform malicious activities. | |
| Suggested Countermeasures | |
| Enforce strict input validation and output encoding to sanitize user-supplied data and prevent script injection. Utilize server-side input validation to reject or sanitize potentially malicious input before processing it. | |
| References | |
| <https://portswigger.net/web-security/cross-site-scripting/cheat-sheet> | |

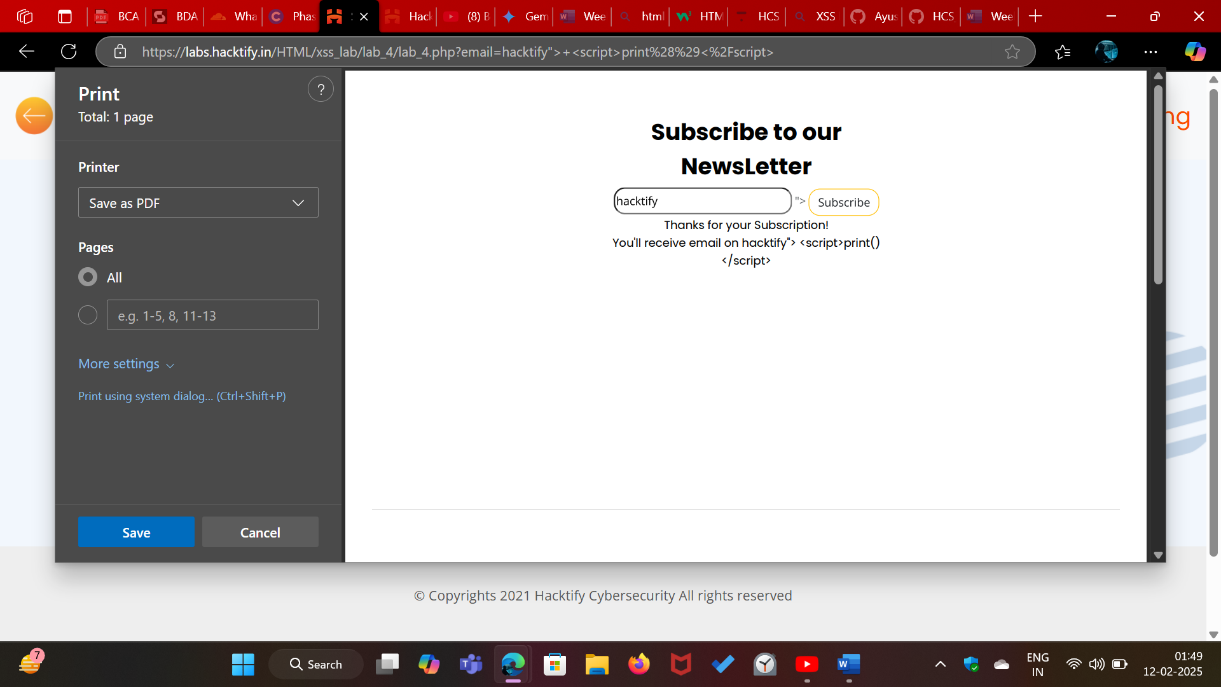
# Proof of Concept



# 2.4. Alternatives are Must!

|  |  |
| --- | --- |
| Reference | Risk Rating |
| Alternatives are Must! | Medium |
| Tools Used | |
| Manually Performed | |
| Vulnerability Description | |
| Despite the addition of alert obfuscation techniques, the vulnerability in the email subscription input box of a website persists. In this scenario, the website attempts to mitigate XSS (Cross-Site Scripting) attacks by employing alert obfuscation to obscure the malicious payload. However, the obfuscation mechanism fails to adequately sanitize the input, allowing for the execution of alternative payloads. | |
| How It Was Discovered | |
| Manual Analysis by applying different function with script tag | |
| Vulnerable URLs | |
| https://labs.hacktify.in/HTML/xss\_lab/lab\_4/index.php | |
| Consequences of not Fixing the Issue | |
| Exploitation by attackers to steal sensitive user information, such as cookies or credentials.  Execution of arbitrary scripts or alternative payloads within the context of other users' browsers, enabling unauthorized actions. | |
| Suggested Countermeasures | |
| Enforce strict input validation and output encoding to sanitize user-supplied data and prevent script injection. Utilize server-side input validation to reject or sanitize potentially malicious input before processing it. | |
| References | |
| <https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html>  <https://portswigger.net/web-security/cross-site-scripting/cheat-sheet> | |

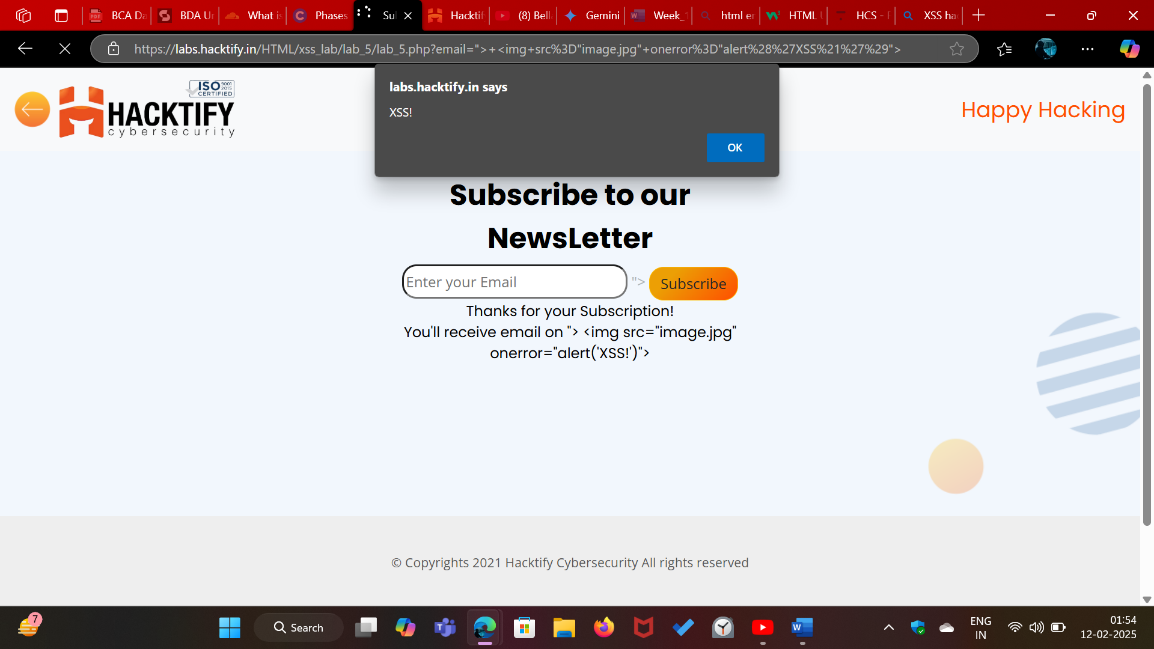
# Proof of Concept



# 2.5. Developer Hates Scripts!

|  |  |
| --- | --- |
| Reference | Risk Rating |
| Developer Hates Scripts! | High |
| Tools Used | |
| Manually Performed | |
| Vulnerability Description | |
| Despite the addition of script obfuscation techniques, the vulnerability in the email subscription input box of a website persists. In this scenario, the website attempts to mitigate XSS (Cross-Site Scripting) attacks by employing script obfuscation to obscure the malicious payload. However, the obfuscation mechanism fails to adequately sanitize the input, allowing for the execution of alternative payloads | |
| How It Was Discovered | |
| Manual Analysis | |
| Vulnerable URLs | |
| https://labs.hacktify.in/HTML/xss\_lab/lab\_5/index.php | |
| Consequences of not Fixing the Issue | |
| Exploitation by attackers to steal sensitive user information, such as cookies or credentials.  Execution of arbitrary scripts or alternative payloads within the context of other users' browsers, enabling unauthorized actions. | |
| Suggested Countermeasures | |
| Enforce strict input validation and output encoding to sanitize user-supplied data and prevent script injection. Utilize server-side input validation to reject or sanitize potentially malicious input before processing it. | |
| References | |
| <https://portswigger.net/web-security/cross-site-scripting/cheat-sheet> | |

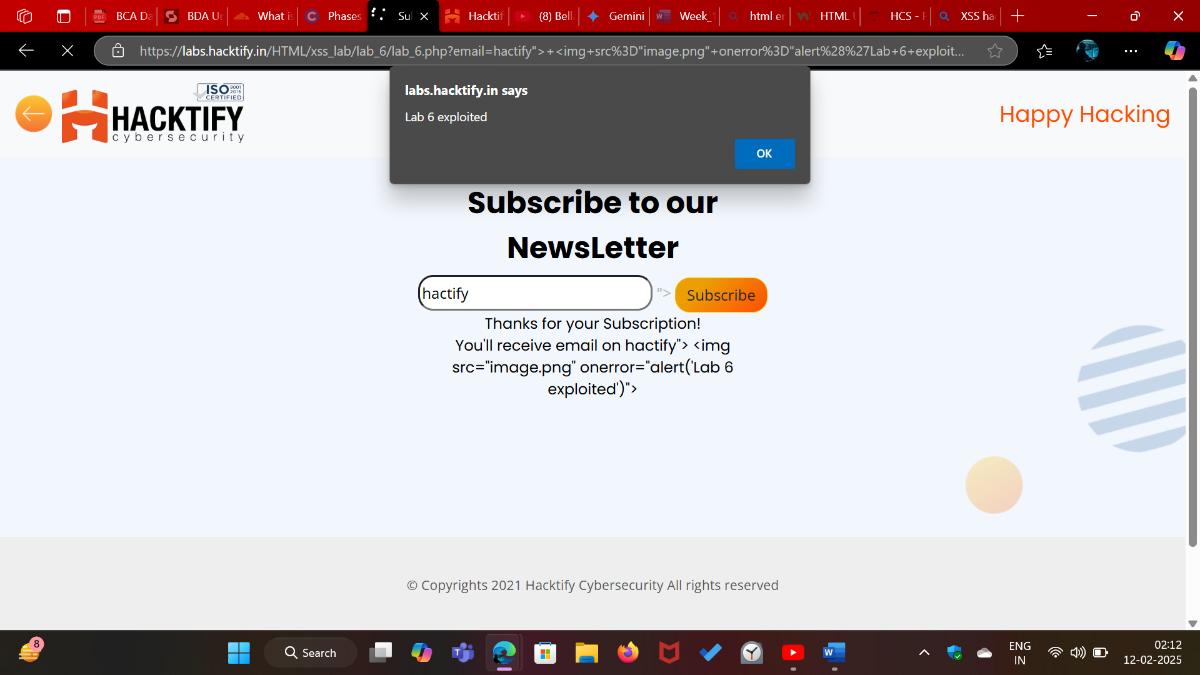
# Proof of Concept



# 2.6. Change the Variation!

|  |  |
| --- | --- |
| Reference | Risk Rating |
| Change the Variation! | High |
| Tools Used | |
| Manually Performed. | |
| Vulnerability Description | |
| Despite the addition of script sanitization techniques, the vulnerability in the email subscription input box of a website persists. In this scenario, the website attempts to mitigate XSS (Cross-Site Scripting) attacks by employing script sanitization to remove or neutralize potentially malicious payloads. However, the sanitization mechanism fails to adequately handle all input variations, allowing for the execution of alternative payloads. | |
| How It Was Discovered | |
| Manual Analysis | |
| Vulnerable URLs | |
| https://labs.hacktify.in/HTML/xss\_lab/lab\_6/index.php | |
| Consequences of not Fixing the Issue | |
| Exploitation by attackers to steal sensitive user information, such as cookies or credentials.  Execution of arbitrary scripts or alternative payloads within the context of other users' browsers, enabling unauthorized actions. | |
| Suggested Countermeasures | |
| Enforce strict input validation and output encoding to sanitize user-supplied data and prevent script injection. Utilize server-side input validation to reject or sanitize potentially malicious input before processing it. | |
| References | |
| <https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html> | |

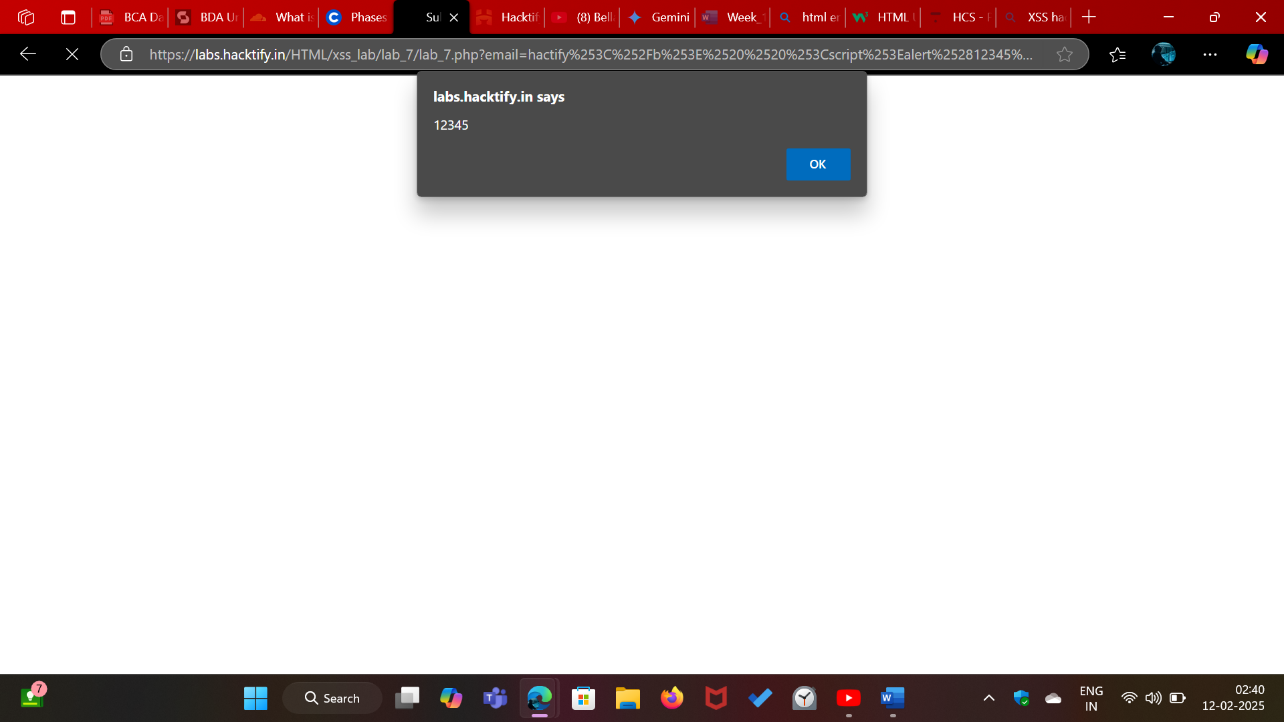
# Proof of Concept



# 2.7. Encoding Is the Key?

|  |  |
| --- | --- |
| Reference | Risk Rating |
| Encoding Is the Key? | Medium |
| Tools Used | |
| No tools are used. | |
| Vulnerability Description | |
| Despite the addition of encoding for symbols in tags, the vulnerability in the email subscription input box of a website persists. In this scenario, the website attempts to mitigate XSS (Cross-Site Scripting) attacks by encoding special characters within HTML tags. However, the encoding mechanism fails to adequately handle all input variations, allowing for the execution of alternative payloads. | |
| How It Was Discovered | |
| Manual Analysis | |
| Vulnerable URLs | |
| https://labs.hacktify.in/HTML/xss\_lab/lab\_7/index.php | |
| Consequences of not Fixing the Issue | |
| Exploitation by attackers to steal sensitive user information, such as cookies or credentials.  Execution of arbitrary scripts or alternative payloads within the context of other users' browsers, enabling unauthorized actions | |
| Suggested Countermeasures | |
| Enforce strict input validation and output encoding to sanitize user-supplied data and prevent script injection. Utilize server-side input validation to reject or sanitize potentially malicious input before processing it. | |
| References | |
| <https://owasp.org/www-project-web-security-testing-guide/latest/>  <https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html> | |

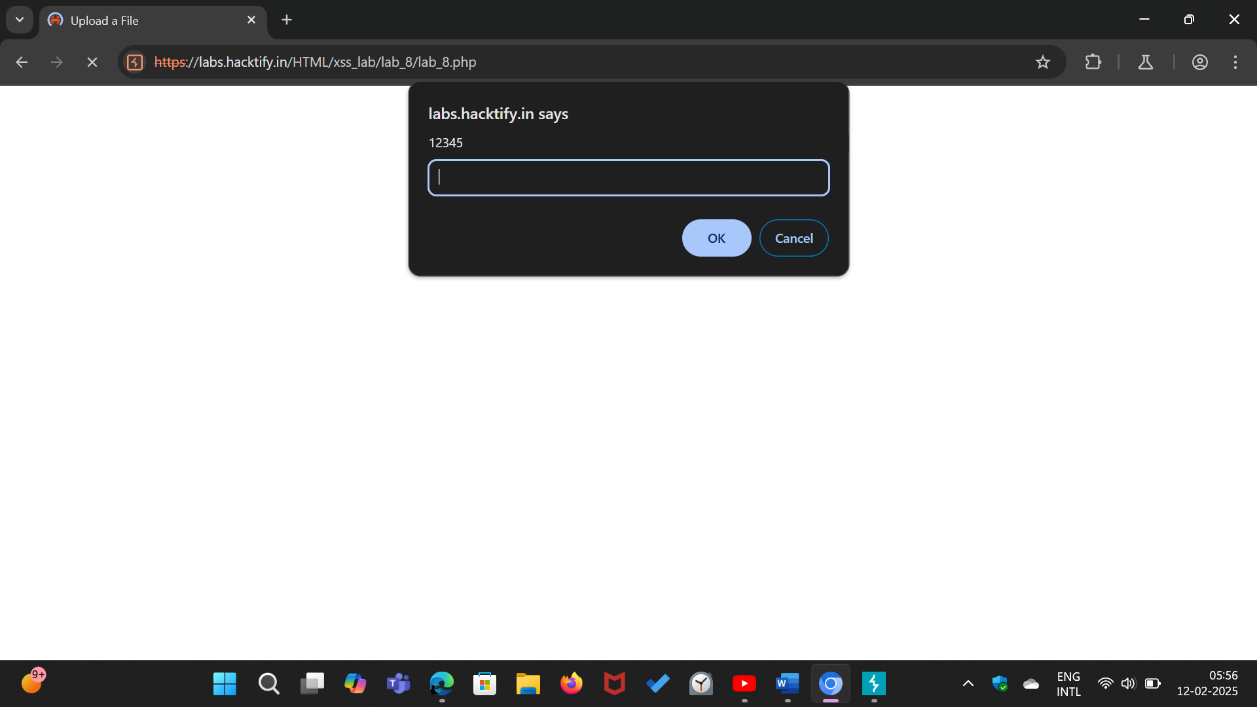
# Proof of Concept



# 2.8. XSS with File Upload (File Name)

|  |  |
| --- | --- |
| Reference | Risk Rating |
| XSS with File Upload (File Name) | Low |
| Tools Used | |
| Burp Suite for intercepting and modifying requests. | |
| Vulnerability Description | |
| Despite the attempt to mitigate XSS (Cross-Site Scripting) attacks through file upload, the vulnerability in the email subscription input box of a website persists. In this scenario, the website allows users to upload files, but the filename input is not properly sanitized or validated. Attackers exploit this vulnerability by intercepting the file upload request using tools like Burp Suite and modifying the filename to include a malicious payload | |
| How It Was Discovered | |
| Discovered through manual analysis by intercepting and modifying requests using Burp Suite. | |
| Vulnerable URLs | |
| https://labs.hacktify.in/HTML/xss\_lab/lab\_8/index.php | |
| Consequences of not Fixing the Issue | |
| Exploitation by attackers to steal sensitive user information, such as cookies or credentials.  Execution of arbitrary scripts or alternative payloads within the context of other users' browsers, enabling unauthorized actions | |
| Suggested Countermeasures | |
| Enforce strict input validation and output encoding to sanitize user-supplied data and prevent script injection. Utilize server-side input validation to reject or sanitize potentially malicious input before processing it. | |
| References | |
| <https://owasp.org/www-project-web-security-testing-guide/latest/>  <https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html> | |

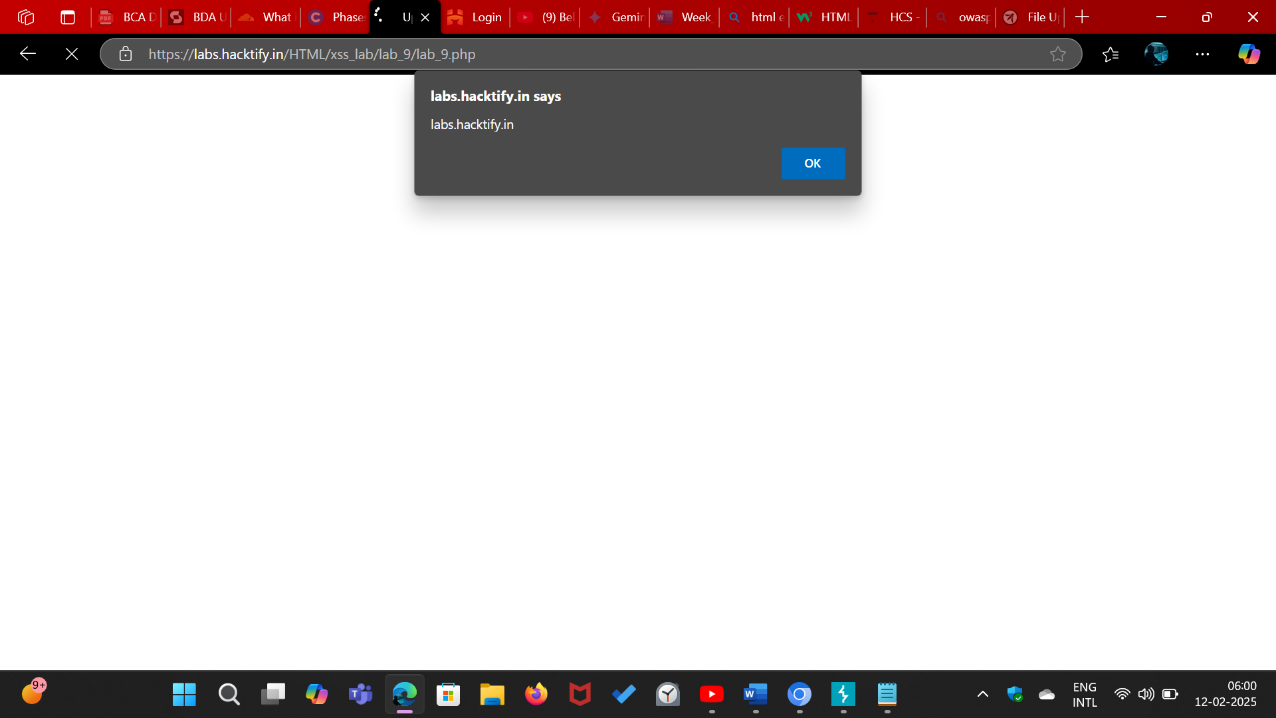
# Proof of Concept



# 2.9. XSS with File Upload (File Content)

|  |  |
| --- | --- |
| Reference | Risk Rating |
| XSS with File with File Upload (File Content) | Medium |
| Tools Used | |
| Manual Testing. | |
| Vulnerability Description | |
| Despite the attempt to mitigate XSS (Cross-Site Scripting) attacks through file upload, the vulnerability in the email subscription input box of a website persists. In this scenario, the website allows users to upload files, but the content of the uploaded files is not properly sanitized or validated. Attackers exploit this vulnerability by uploading a file containing malicious script content | |
| How It Was Discovered | |
| Manual Analysis | |
| Vulnerable URLs | |
| https://labs.hacktify.in/HTML/xss\_lab/lab\_9/index.php | |
| Consequences of not Fixing the Issue | |
| Exploitation by attackers to steal sensitive user information, such as cookies or credentials.  Execution of arbitrary scripts or alternative payloads within the context of other users' browsers, enabling unauthorized actions | |
| Suggested Countermeasures | |
| Enforce strict input validation and output encoding to sanitize user-supplied data and prevent script injection. Utilize server-side input validation to reject or sanitize potentially malicious input before processing it. | |
| References | |
| <https://owasp.org/www-project-web-security-testing-guide/latest/>  <https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html> | |

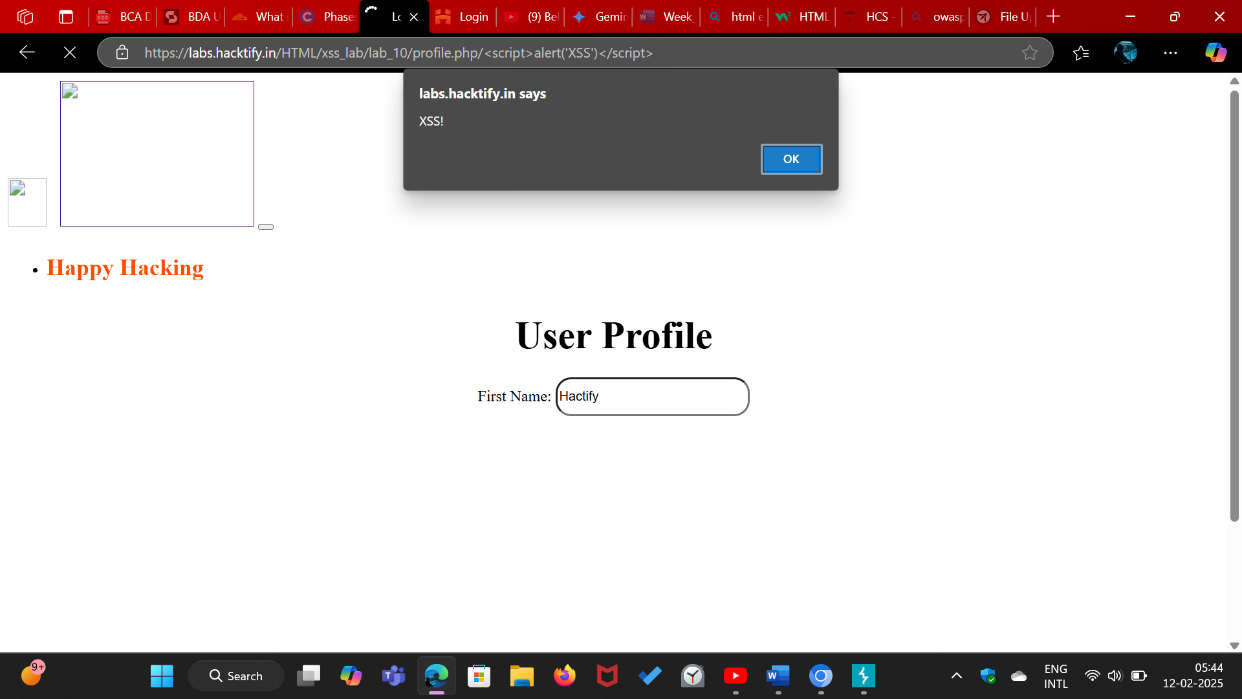
# Proof of Concept



# 2.10. Stored Everywhere!

|  |  |
| --- | --- |
| Reference | Risk Rating |
| Stored Everywhere! | Low |
| Tools Used | |
| Manual Testing. | |
| Vulnerability Description | |
| In the user login page of a website, an XSS (Cross-Site Scripting) vulnerability exists due to inadequate input sanitization in the profile upload page's first name section. We exploit this vulnerability by inserting a malicious script payload. | |
| How It Was Discovered | |
| Manual Analysis | |
| Vulnerable URLs | |
| https://labs.hacktify.in/HTML/xss\_lab/lab\_10/index.php | |
| Consequences of not Fixing the Issue | |
| Exploitation by attackers to steal sensitive user information, such as cookies or credentials.  Execution of arbitrary scripts or alternative payloads within the context of other users' browsers, enabling unauthorized actions. | |
| Suggested Countermeasures | |
| Enforce strict input validation and output encoding to sanitize user-supplied data and prevent script injection. Utilize server-side input validation to reject or sanitize potentially malicious input before processing it. | |
| References | |
| <https://owasp.org/www-project-web-security-testing-guide/latest/>  <https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html> | |

# Proof of Concept



# 2.11. DOM’S Are Love!

|  |  |
| --- | --- |
| Reference | Risk Rating |
| DOM’S Are Love! | High |
| Tools Used | |
| Manual testing was conducted. | |
| Vulnerability Description | |
| In the website, a DOM-based XSS (Cross-Site Scripting) vulnerability exists, allowing attackers to execute malicious scripts within the context of other users' browsers. This vulnerability arises due to improper handling of user-controlled data in the URL parameters, specifically the "name" parameter. We exploit this vulnerability by injecting a malicious payload. | |
| How It Was Discovered | |
| Manual Analysis | |
| Vulnerable URLs | |
| https://labs.hacktify.in/HTML/xss\_lab/lab\_11/index.php | |
| Consequences of not Fixing the Issue | |
| Exploitation by attackers to steal sensitive user information, such as cookies or credentials.  Execution of arbitrary scripts or alternative payloads within the context of other users' browsers, enabling unauthorized actions | |
| Suggested Countermeasures | |
| Enforce strict input validation and output encoding to sanitize user-supplied data and prevent script injection. Utilize server-side input validation to reject or sanitize potentially malicious input before processing it. | |
| References | |
| <https://owasp.org/www-project-web-security-testing-guide/latest/>  <https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html> | |

# Proof of Concept

