**Penetration Testing Report**

**Full Name: Karthikaa. R & Dharanidharan. V  
Program: HCS**

**Date:26/02/2024**

**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** | **HTML Injection, Clickjacking** |

**3. Summary**

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

**Total number of Sub-labs: 8 Sub-labs**

|  |  |  |
| --- | --- | --- |
| **High** | **Medium** | **Low** |
| **1** | **3** | **4** |

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

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

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

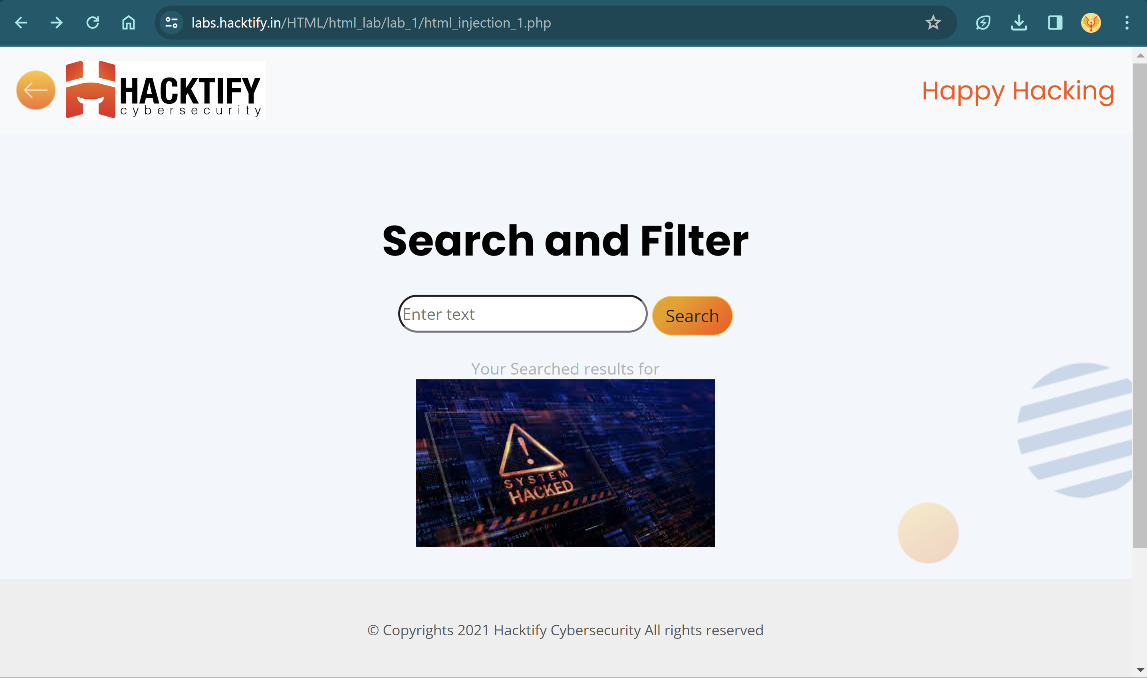
# 1. HTML Injection

# 1.1. HTML's Are Easy!

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| HTML's Are Easy! | **Low** |
| **Tools Used** | |
| 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://medium.com/@whitehatcyber404/how-i-earned-150-in-2-minutes-html-injection-in-email-3f26f27d3822>  <https://anithaana3.medium.com/easy-ways-to-exploit-html-injection-d06a594b9577> | |

# 

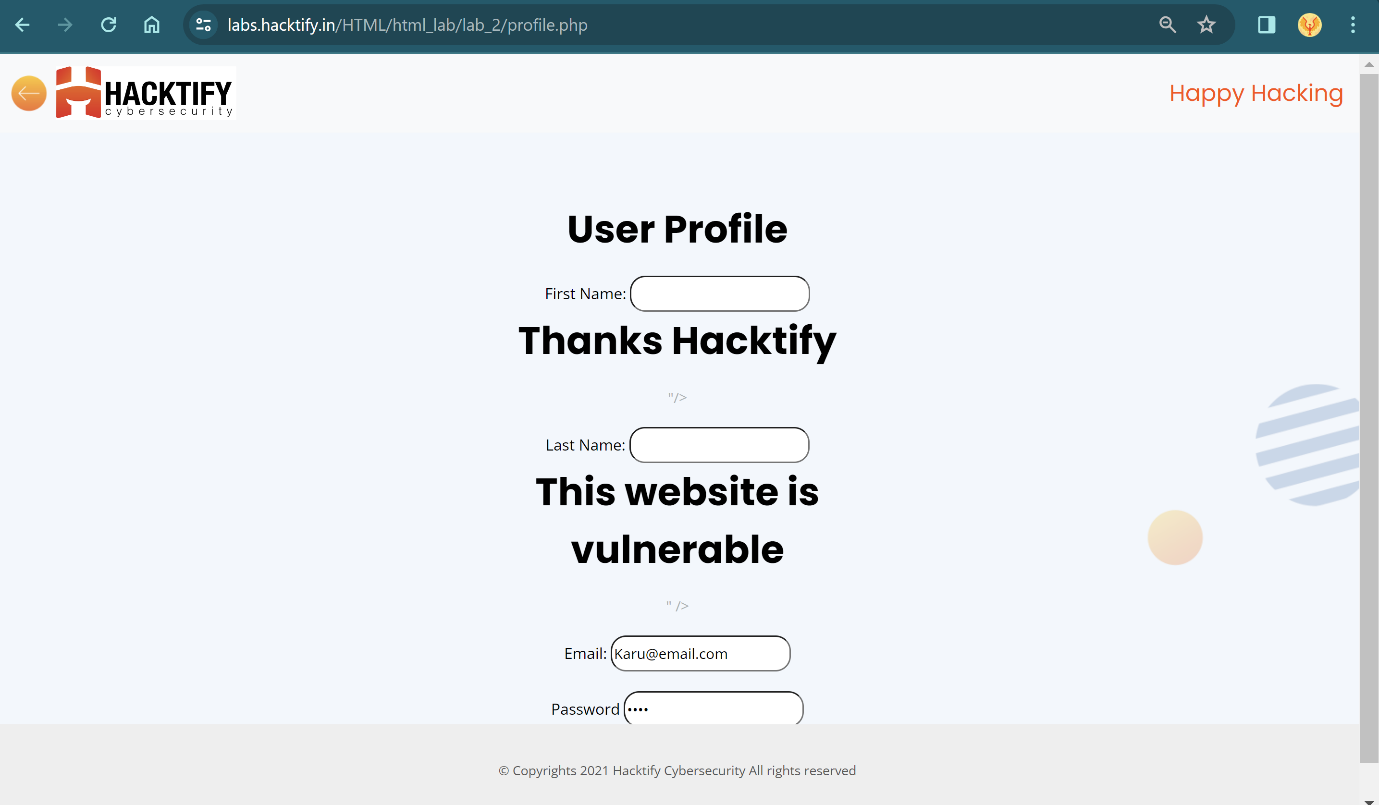
# Proof of Concept



# 1.2. Let Me Store Them!

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| Let Me Store Them! | **Low** |
| **Tools Used** | |
| Manual analysis without specific tools. | |
| **Vulnerability Description** | |
| Through manual analysis of input fields and subsequent observation of 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://portswigger.net/web-security/file-upload> | |

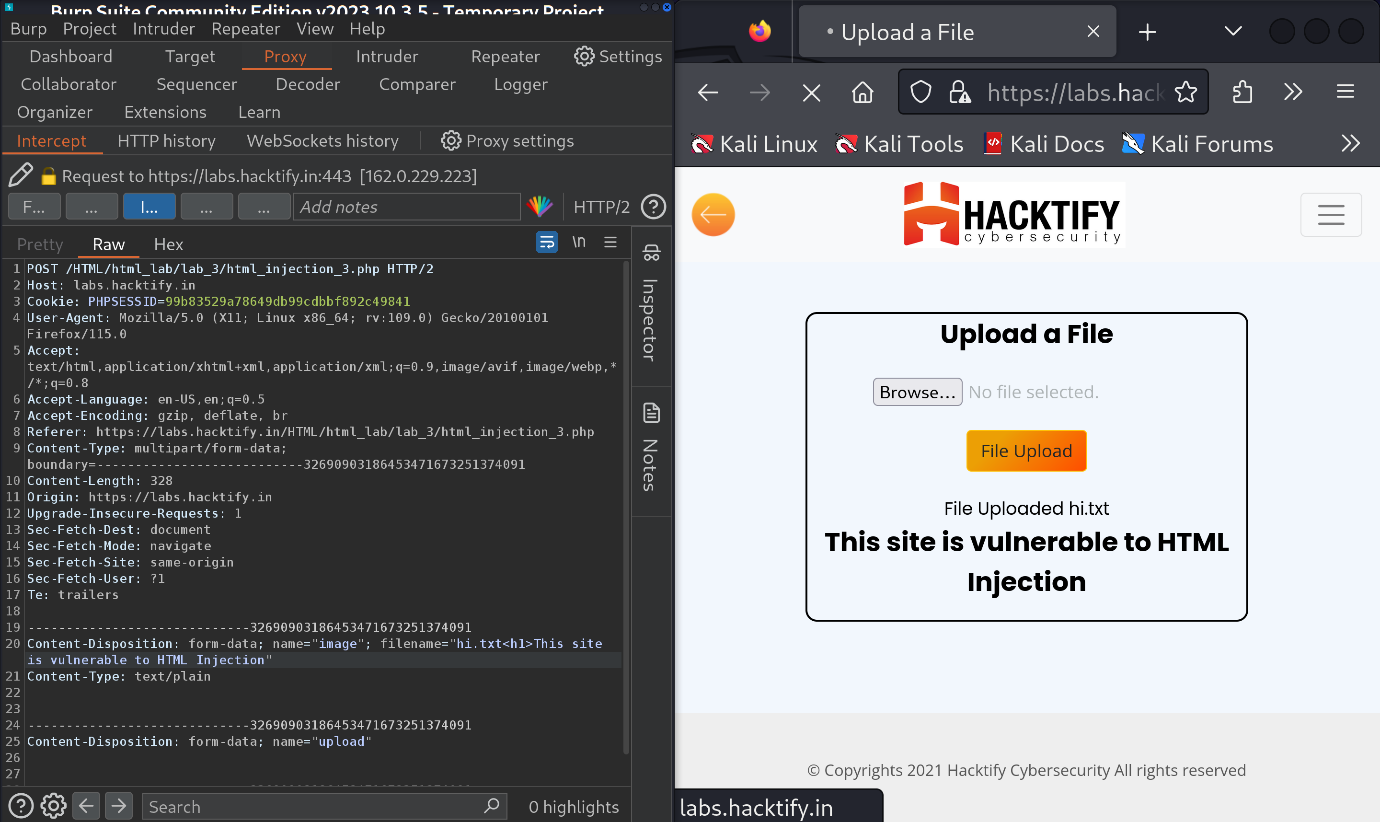
# Proof of Concept

****

# 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. | |
| **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** | |
| Discovered through 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> | |

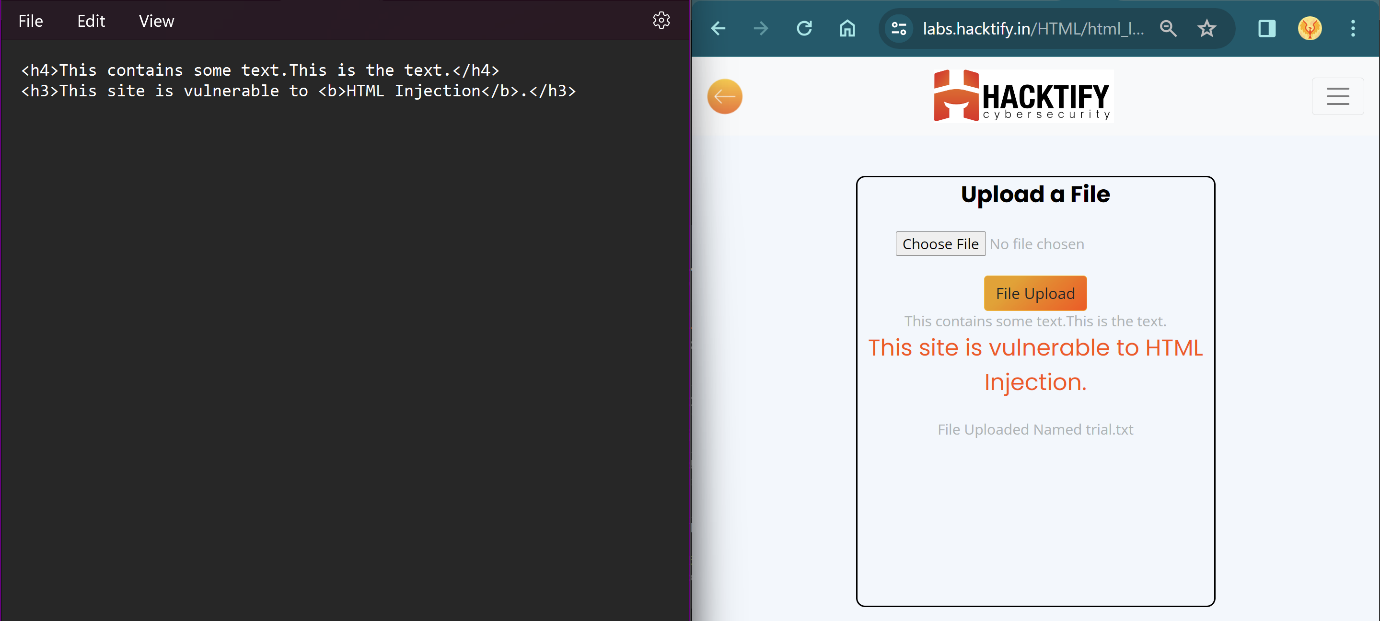
# Proof of Concept



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

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| File Content and HTML Injection a perfect pair! | **Medium** |
| **Tools Used** | |
| No specialized tools were used; manual testing was conducted. | |
| **Vulnerability Description** | |
| The vulnerability allows for HTML injection through file upload functionality, enabling the insertion of malicious code into the web page. | |
| **How It Was Discovered** | |
| Discovered through 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** | |
| -nil- | |

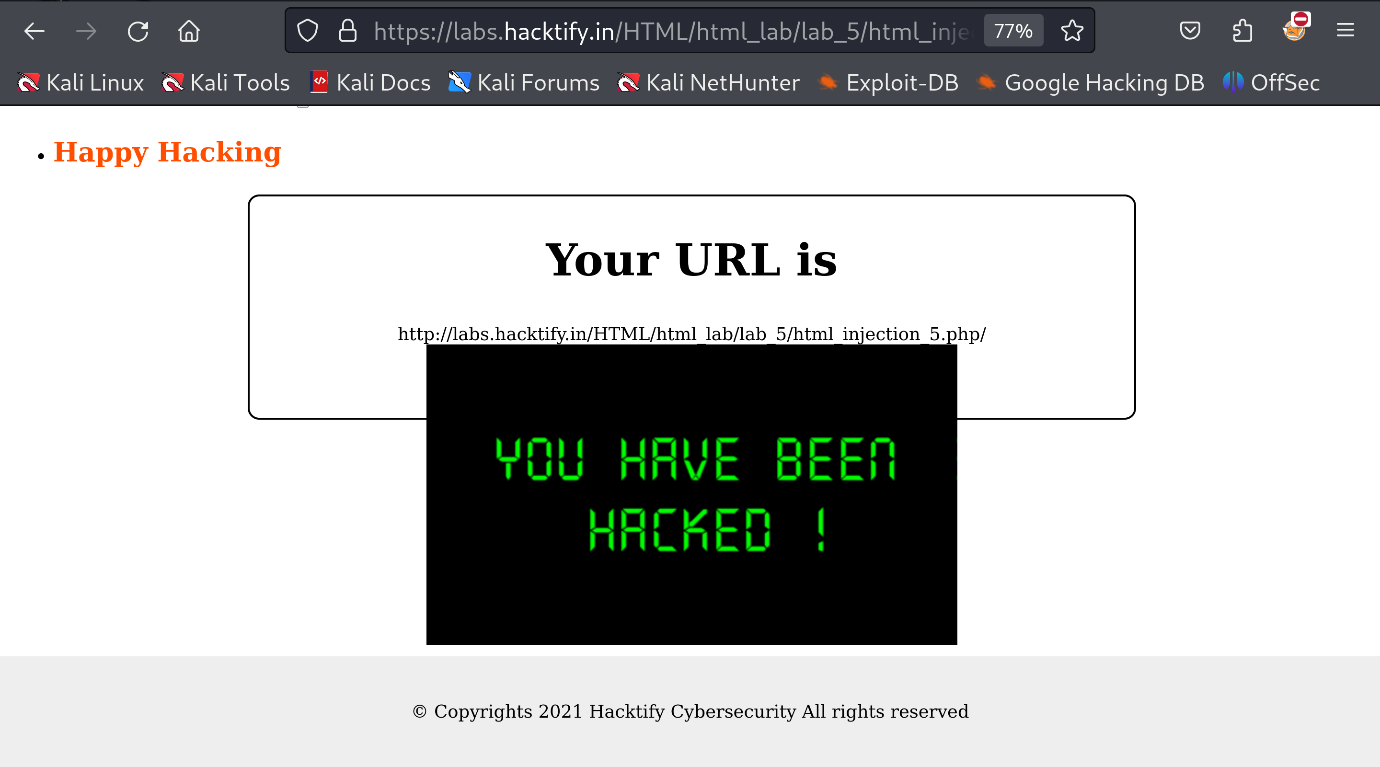
# Proof of Concept



# 1.5. Injecting HTML using URL

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| Injecting HTML using URL | **Medium** |
| **Tools Used** | |
| No Tools used. | |
| **Vulnerability Description** | |
| Exploiting lack of input validation to inject HTML code, resulting in arbitrary code execution. | |
| **How It Was Discovered** | |
| Manual analysis 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> (Image HTML tag reference code)  <https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/03-Testing_for_HTML_Injection> | |

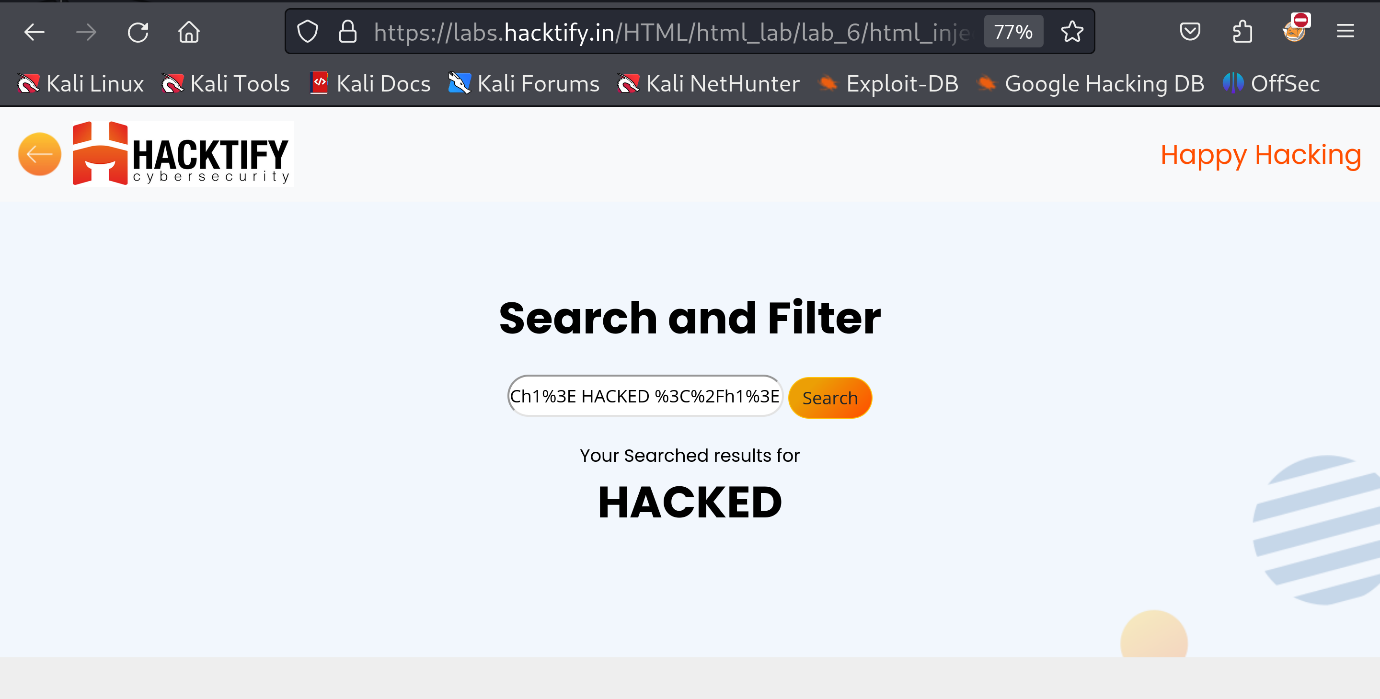
# Proof of Concept

****

# 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 | |

# Proof of Concept

****

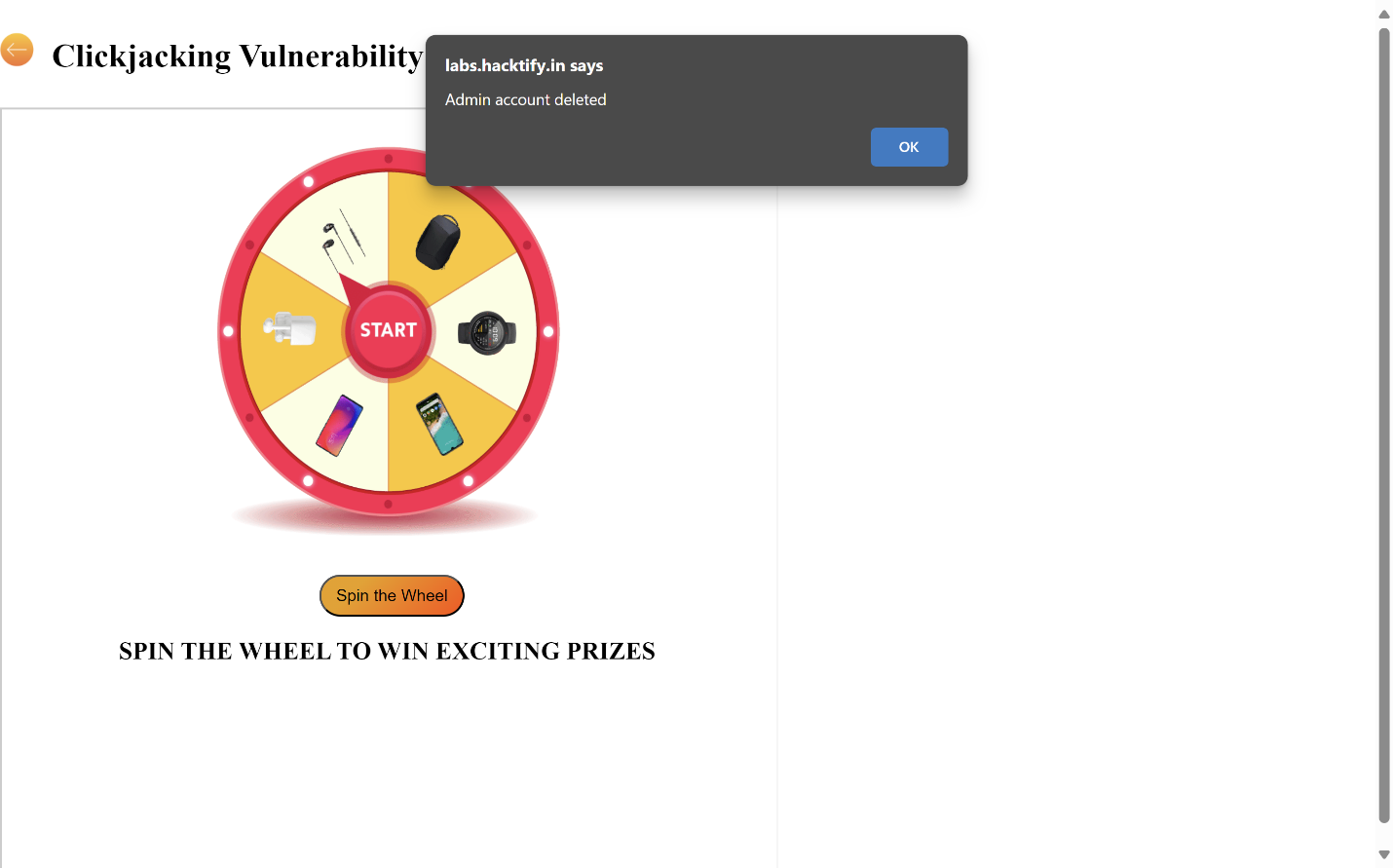
# 2. Clickjacking

# 2.1. Let's Hijack!

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| Let's Hijack! | **Low** |
| **Tools Used** | |
| None; the vulnerability was discovered through manual analysis. | |
| **Vulnerability Description** | |
| Clickjacking vulnerability; attackers can trick users into performing unintended actions by overlaying deceptive elements atop legitimate ones. | |
| **How It Was Discovered** | |
| Manual analysis during lab exercise. | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/clickjacking\_lab/lab\_1/testclickjacking.php | |
| **Consequences of not Fixing the Issue** | |
| Risk of unauthorized actions performed on behalf of users, leading to data loss, financial damage, or compromised accounts. | |
| **Suggested Countermeasures** | |
| Implement frame-busting scripts, use X-Frame-Options header, and employ content security policy (CSP) to mitigate clickjacking attacks. | |
| **References** | |
| https://portswigger.net/web-security/clickjacking | |

# 

# Proof of Concept

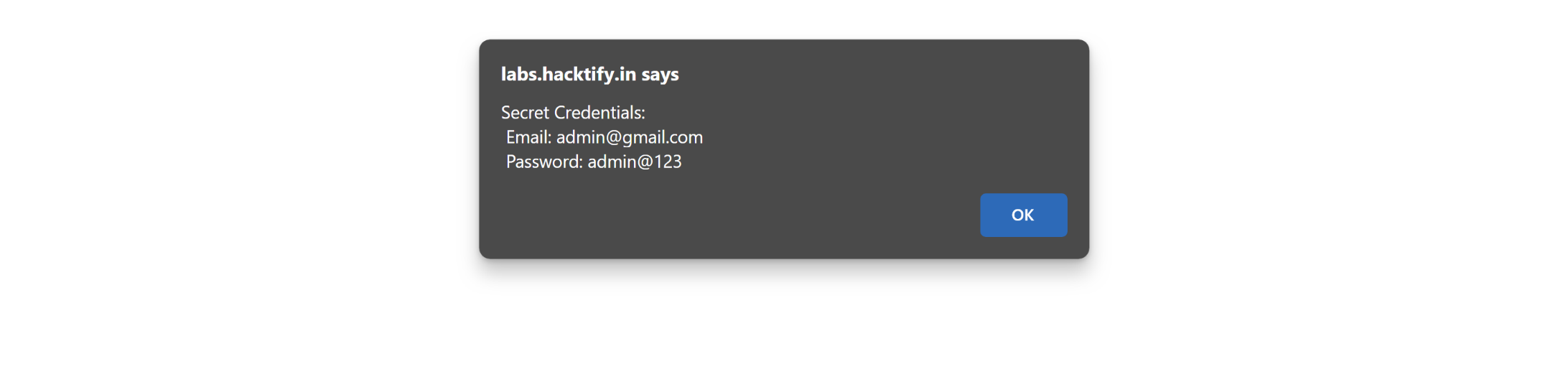


# 2.2. Re-Hijack!

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| Re-Hijack! | **Medium** |
| **Tools Used** | |
| No specific tools were used; the vulnerability was discovered through manual testing. | |
| **Vulnerability Description** | |
| Clickjacking vulnerability allows attackers to overlay deceptive elements over legitimate web content, tricking users into unintended actions like revealing sensitive information. | |
| **How It Was Discovered** | |
| The vulnerability was identified through manual analysis of the web page's structure and behavior. | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/clickjacking\_lab/lab\_2/testclickjacking.php | |
| **Consequences of not Fixing the Issue** | |
| Failure to patch the clickjacking vulnerability could result in unauthorized disclosure of sensitive information, compromised user accounts, and reputational damage to the website owner. | |
| **Suggested Countermeasures** | |
| Implement X-Frame-Options header with 'deny' or 'sameorigin' value, employ frame-busting JavaScript techniques, and regularly educate users about potential risks of clickjacking. | |
| **References** | |
| https://ehteshamulhaq198.medium.com/clickjacking-to-obtain-login-credentials-abee3ae9825e | |

# 

# Proof of Concept

****