# Penetration Testing Report

**Full Name: Ritwik Gupta**
**Program: HCPT**
**Date: February 21, 2024**

## 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** | **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** | |
| Chrome Browser | |
| **Vulnerability Description** | |
| In the provided HTML's-Are-Easy lab, we observed a “Search and Filter” input form text field which is susceptible to HTML injection vulnerability. A threat actor can leverage this flaw to inject their own content or malicious code into the website. The input text field acts as an entry point for the injection payload.<br>Injection code/ Payload :<br><h1>Ritwik Gupta: This page is susceptible to HTML injection!</h1> | |
| **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** | |
| If not patched, the vulnerability can be leveraged by threat actors to make this website a potential phishing site that can be further used to target users. | |
| **Suggested Countermeasures** | |
| 1) Input validation : Use input validation libraries or frameworks to sanitize user input automatically.<br>2) Output encoding: Use HTML entity encoding or JavaScript escaping to neutralize special characters | |
| **References** | |
| 1)https://hacktify.thinkific.com/courses/take/cyber-security-internship-week-1/pdfs/52738677-web-security-fundamentals-guide<br>2) https://www.acunetix.com/vulnerabilities/web/html-injection/ | |

# Proof of Concept

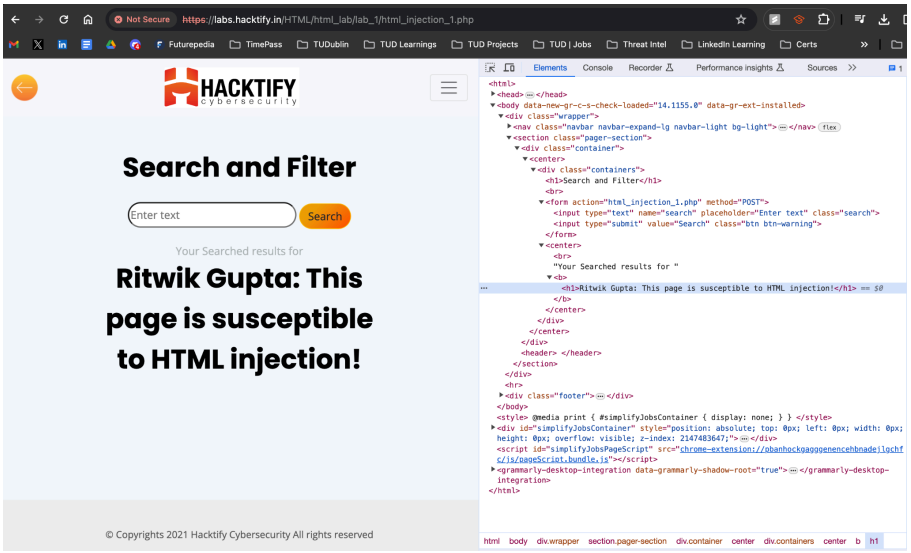


Figure 1. HTML's Are Easy lab exploitation successful

## 1.2. Let Me Store Them!

| Reference | Risk Rating |
|---|---|
| Let-Me-Store-Them | **Low** |
| **Tools Used** | |
| Chrome Browser | |
| **Vulnerability Description** | |
| In the provided Let-Me-Store-Them lab, we observed a “User Log In” input form with email and password fields. A user has two options either login with already registered credentials or register. After registering and logging in with the credentials we are displayed with ‘User Profile’ page with all the user data collected in the registration phase. Here, the user data value is pre-loaded in the text fields. These text fields are susceptible to HTML injection vulnerability. A threat actor can leverage this flaw to inject their own content or malicious code into the website. The input text field acts as an entry point for the injection payload.<br>Injection code/ Payload : "> <h1>Ritwik</h1> | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/html_lab/lab_2/html_injection_2.php | |
| **Consequences of not Fixing the Issue** | |
| If not patched, the vulnerability can be leveraged by threat actors to transform this website a potential phishing site that can be further used to target users as well as carry out account takeover attacks. | |
| **Suggested Countermeasures** | |
| 1) Input validation: Use input validation libraries or frameworks to sanitize user input automatically.<br>2) Output encoding: Use HTML entity encoding or JavaScript escaping to neutralize special characters | |
| **References** | |
| 1)https://hacktify.thinkific.com/courses/take/cyber-security-internship-week-1/pdfs/52738677-web-security-fundamentals-guide<br>2) https://www.acunetix.com/vulnerabilities/web/html-injection/ | |

## Proof of Concept

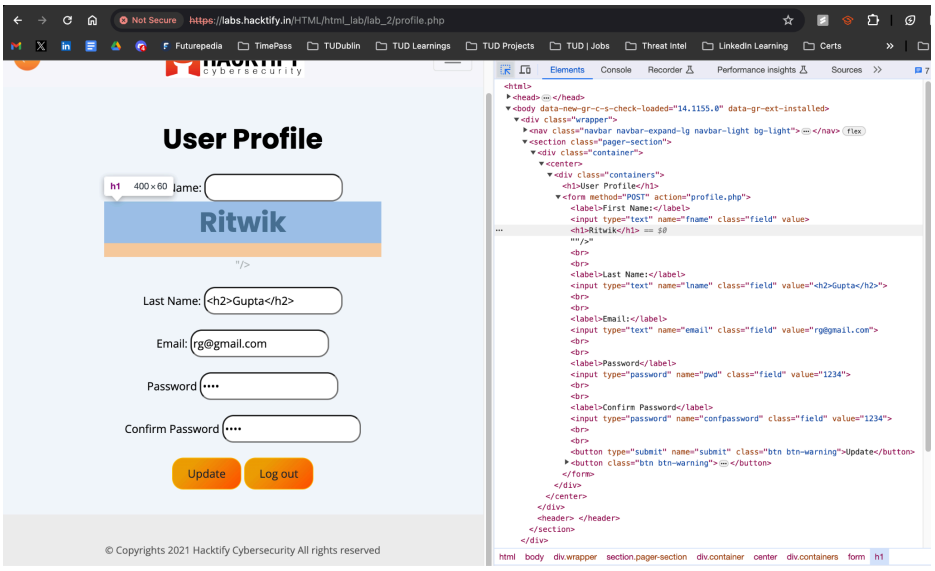


Figure 2. Let Me Store Them lab exploitation successful

## 1.3. File Names Are Also Vulnerable!

| Reference | Risk Rating |
|---|---|
| File-Names-Are-Also-Vulnerable | **Medium** |
| **Tools Used** | |
| Chrome Browser | |
| **Vulnerability Description** | |
| In the provided Let-Me-Store-Them lab, we observed a file upload form with an option to choose a file. On successful upload of file supplied, an affirmation message appears with the filename of uploaded file. If we rename the file with html tags within it, it gets parsed at the client side from the server response. In this way, the file input field is susceptible to HTML injection vulnerability. A threat actor can leverage this flaw to inject their own content or malicious code into the website. The file input field acts as an entry point for the injection payload.<br>Injection code/ Payload :<br><h1>Ritwik_Gupta-ExploitSuccessful.jpg</h1><br><h1>Ritwik_Gupta-ExploitSuccessful.jpg | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/html_lab/lab_3/html_injection_3.php | |
| **Consequences of not Fixing the Issue** | |
| If not patched, the vulnerability can be leveraged by threat actors to modify the appearence and content of the webpage. | |
| **Suggested Countermeasures** | |
| 1) Output encoding: Use HTML entity encoding or JavaScript escaping to neutralize special characters<br>2) Response/output sanitization before parsing | |
| **References** | |
| 1)https://hacktify.thinkific.com/courses/take/cyber-security-internship-week-1/pdfs/52738677-web-security-fundamentals-guide<br>2) https://www.acunetix.com/vulnerabilities/web/html-injection/ | |

# Proof of Concept

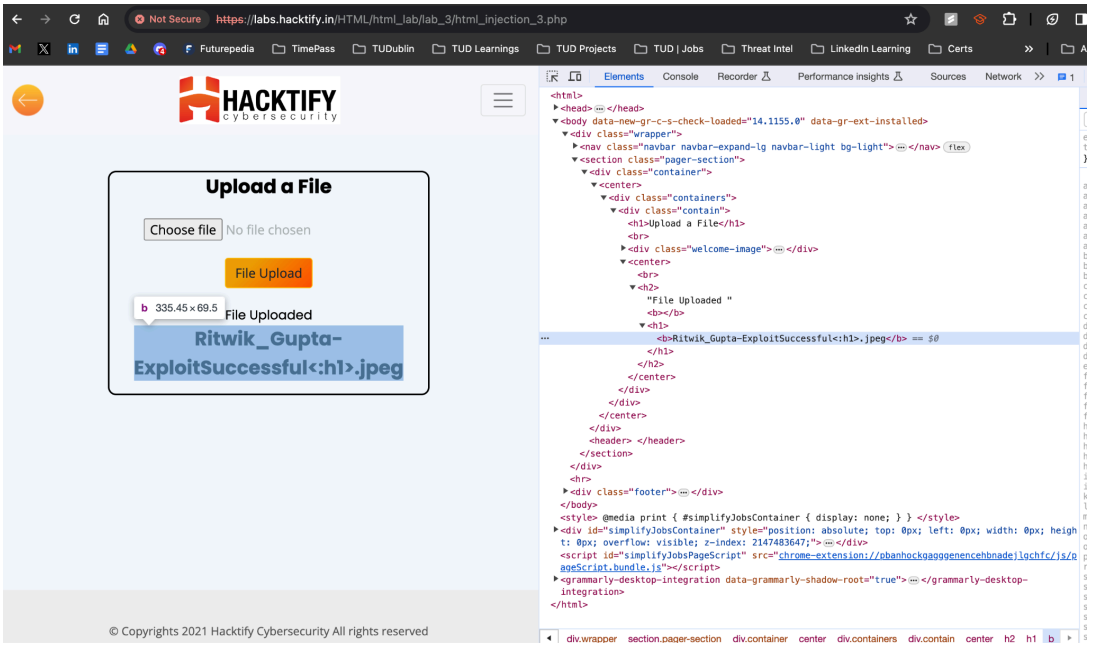


Figure 3. lab exploitation successful with header end tag </h1>

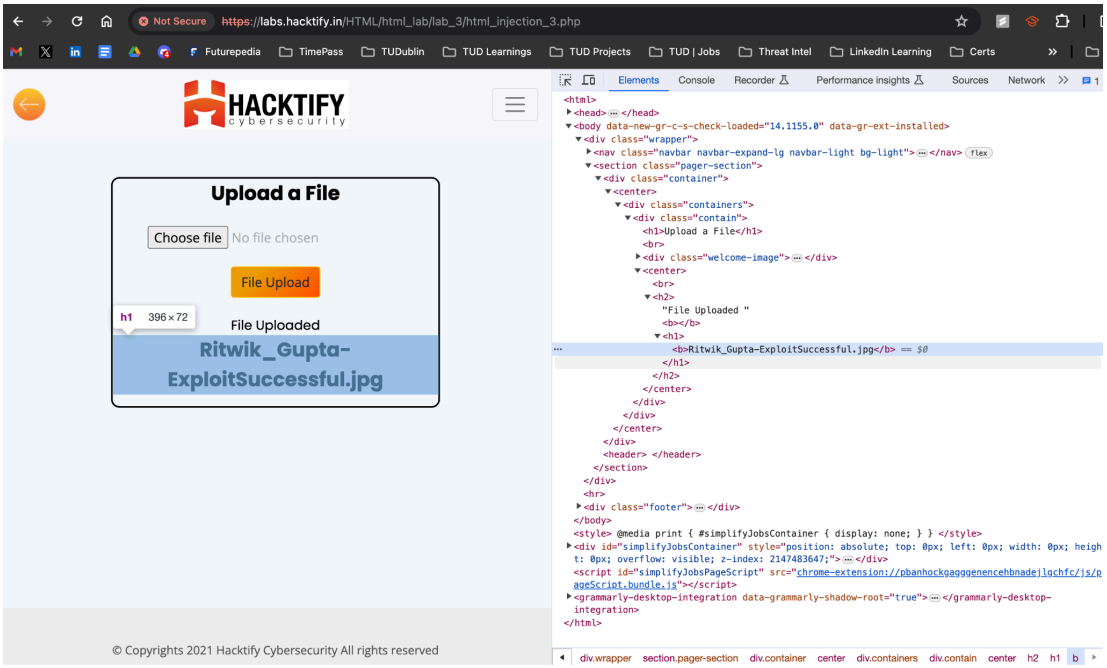


Figure 4. lab exploitation successful without header end tag

# 1.4. File Content And HTML Injection A Perfect Pair!

| Reference | Risk Rating |
|---|---|
| File-Content-And-HTML-Injection | Medium |
| **Tools Used** | |
| Chrome Browser | |
| **Vulnerability Description** | |

In the provided File-Content-And-HTML-Injection lab, we observed a file upload form with an option to choose a file. On successful upload of the file supplied, an affirmation message appears with the filename of the uploaded file along with the file contents. If we input a specially crafted file with HTML tags within it, it gets parsed on the client side and is displayed right after pressing the File Upload button. In this way, the file input field is susceptible to HTML injection vulnerability. A threat actor can leverage this flaw to inject their own content or malicious code into the website. The file input field acts as an entry point for the injection payload.

**Injection code/ Payload :**

<h1>Ritwik Gupta</h1>

<BUTTON TYPE="button" ONCLICK="alert('Exploitation Successful!')">Click me!</BUTTON>

**How It Was Discovered**

Manual Analysis

**Vulnerable URLs**

https://labs.hacktify.in/HTML/html_lab/lab_4/html_injection_4.php

**Consequences of not Fixing the Issue**

The consequences of unrestricted file upload can vary, including complete system takeover, an overloaded file system or database, forwarding attacks to back-end systems, client-side attacks, or simple defacement.

**Suggested Countermeasures**

HTML encoding: HTML encoding is the process of converting HTML characters, such as < and >, into their corresponding character entities, such as < and >. This can help prevent HTML injection attacks by ensuring that user-submitted data is treated as plain text rather than HTML code.

**References**

1) https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload
2)https://hacktify.thinkific.com/courses/take/cyber-security-internship-week-1/pdfs/52738677-web-security-fundamentals-guide
3) https://www.acunetix.com/vulnerabilities/web/html-injection/

## Proof of Concept

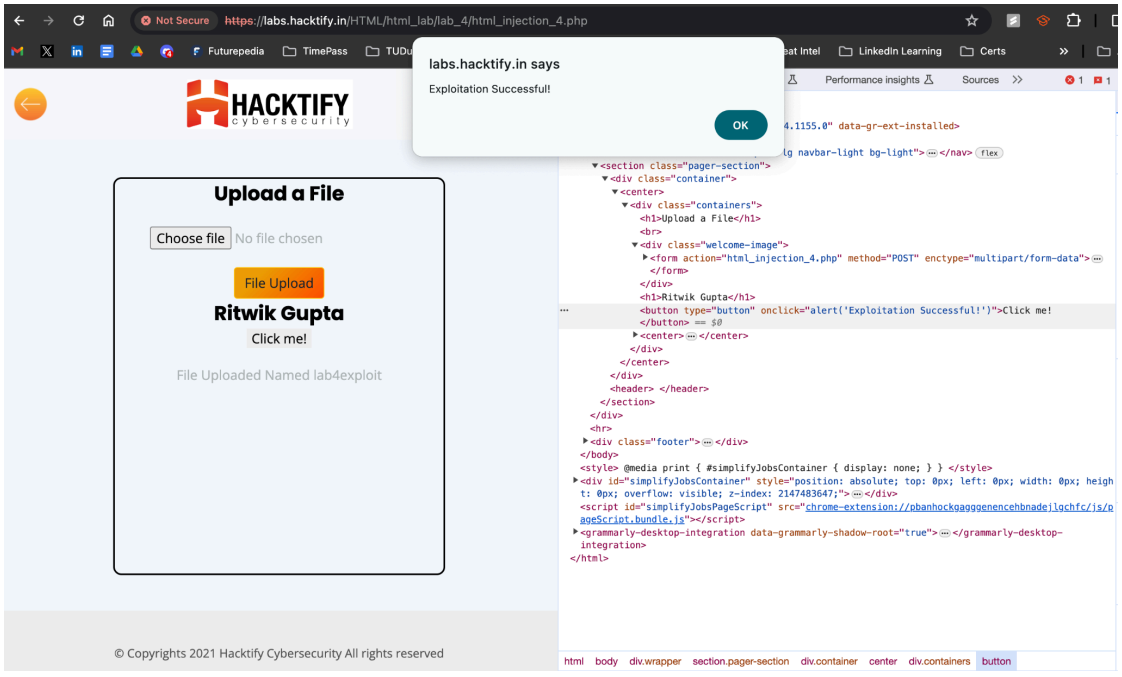


Figure 5. File Content And HTML Injection A Perfect Pair lab exploitation successful

## 1.5. Injecting HTML Using URL

| Reference | Risk Rating |
|---|---|
| Injecting-HTML-Using-URL | **Medium** |
| **Tools Used** | |
| Chrome Browser | |
| **Vulnerability Description** | |
| In the provided Injecting-HTML-Using-URL lab, there are no input fields available to test/inject. On careful analysis, we observed that the URL is printed as it is in the HTML body. Attempting to add HTML tags in the query string gets passed fetched from the server to print it in the webpage. Making changes to the URL query strings leads to identical changes in the URL printed on screen. If we input a specially crafted URL with desired HTML tags within it, it gets parsed on the client side and is displayed. A threat actor can leverage this flaw to inject their own content or malicious code into the website.<br>**Injection code/ Payload :**<br><h1>Ritwik gupta</h1><br><h3>Injection Vulnerable</h3><br>%3Ch1%3ERitwik%20gupta%3C/h1%3E%3Cbr%3E%3Ch3%3EInjection%20Vulnerable%3C/h3%3E | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/html_lab/lab_5/html_injection_5.php | |
| **Consequences of not Fixing the Issue** | |
| If not patched, the vulnerability can be leveraged by threat actors to modify the appearance and content of the webpage and even used for website defacement. | |
| **Suggested Countermeasures** | |
| 1) Output encoding: Use HTML entity encoding or JavaScript escaping to neutralize special characters<br>2) Response/output sanitization before parsing by browser.<br>3) All the control characters and Unicode ones should be removed from the filenames and their extensions without any exception. Also, the special characters such as “;”, “:”, “>”, “<”, “/” ,”\”, additional “.”, “*”, “%”, “$”, and so on should be discarded as well. | |
| **References** | |
| 1) https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload<br>2)https://hacktify.thinkific.com/courses/take/cyber-security-internship-week-1/pdfs/52738677-web-security-fundamentals-guide<br>3) https://www.acunetix.com/vulnerabilities/web/html-injection/<br>4) https://github.com/InfoSecWarrior/Offensive-Payloads/tree/main | |

## Proof of Concept

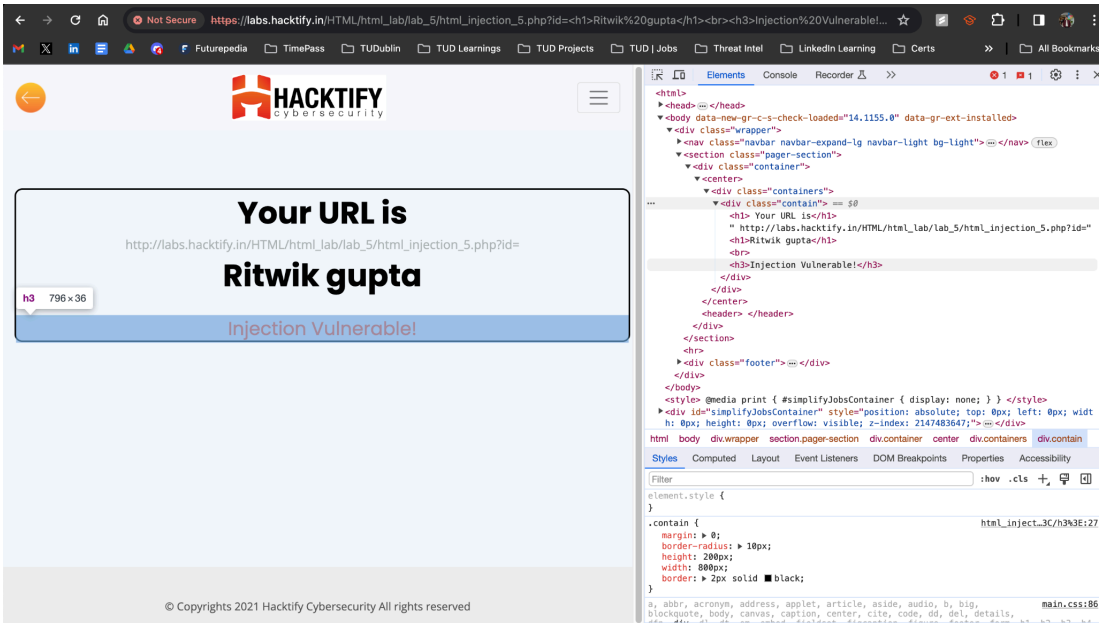


Figure 6. Injecting HTML Using URL lab exploitation successful

## 1.6. Encode IT!

| Reference | Risk Rating |
|---|---|
| Encode-IT | Low |
| **Tools Used** | |
| Chrome Browser | |
| **Vulnerability Description** | |
| In the provided Encode-IT lab, we observed a “Search and Filter” input form text field which is susceptible to HTML injection vulnerability. A threat actor can leverage this flaw to inject their own content or malicious code to the website. The input text field acts as an entry point for the injection payload. However, the input field in this instance is sanitizing the presence of ‘<’ or ‘>’ brackets. Thus we can supply the HTML injection payload in URL encoded format to bypass this check.<br>Injection code/ Payload :<br>%3Ch1%3ERitwik%20Gupta%3C/h1%3E%3Cbr%3E%3Ch2%3EInjection%20Vulnerable!%3C/h2%3E | |
| **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** | |
| If not patched, the vulnerability can be leveraged by threat actors to make this website a potential phishing site that can be further used to target users. | |
| **Suggested Countermeasures** | |
| 1) Input validation : Use input validation libraries or frameworks to sanitize user input automatically.<br>2) Output encoding: Use HTML entity encoding or JavaScript escaping to neutralize special characters | |
| **References** | |
| 1)https://hacktify.thinkific.com/courses/take/cyber-security-internship-week-1/pdfs/52738677-web-security-fundamentals-guide<br>2) https://www.acunetix.com/vulnerabilities/web/html-injection/<br>3) https://gchq.github.io/CyberChef/ | |

## Proof of Concept

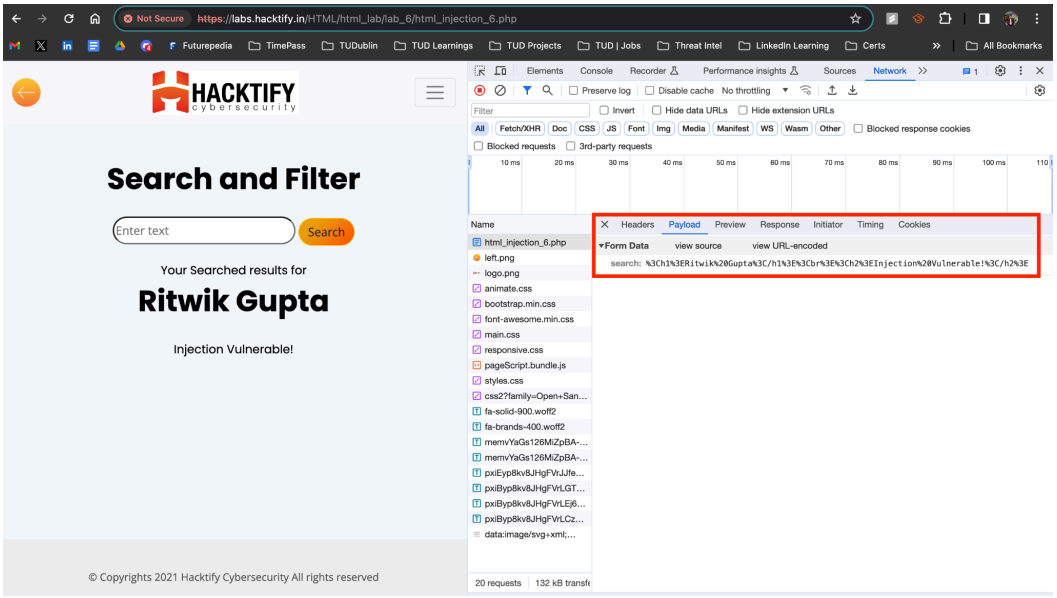


Figure 7. Injecting URL encoded HTML payload

# 2. Clickjacking

## 2.1. Let's Hijack!

| Reference | Risk Rating |
|---|---|
| Lets-Hijack | **Low** |
| **Tools Used** | |
| To check HTTP Security headers: https://securityheaders.com/<br>Chrome Browser | |
| **Vulnerability Description** | |
| In the provided Lets-Hijack lab, we observed a User profile form pre-loaded with admin user details and a button to delete the account details. Surfing through the Test button, we could see a SPIN THE WHEEL button with a little blurred User profile form as seen on the previous page. On inspecting the page on browser DevTools we identify an iframe HTML tag that loads the clickjacking lab1 webpage. On pressing the spin button, it eventually submits the 'Delete Account' button and admin account gets deleted. This proves the susceptibility of Clickjacking in the provided lab. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/clickjacking_lab/lab_1/testclickjacking.php | |
| **Consequences of not Fixing the Issue** | |
| If not patched, the vulnerability can be leveraged by threat actors to carry out Database operations / actions on behalf of victim user without them even knowing about it. | |
| **Suggested Countermeasures** | |
| 1. Content-Security-Policy: frame-ancestors <source>; header must be implemented at web server.<br>2. X-Frame-Options: SAMEORIGIN / DENY must be set to restrict the framing. | |
| **References** | |
| 1. https://securityheaders.com/<br>2. https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/11-Client_Side_Testing/09-Testing_for_Clickjacking<br>3. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options | |

## Proof of Concept

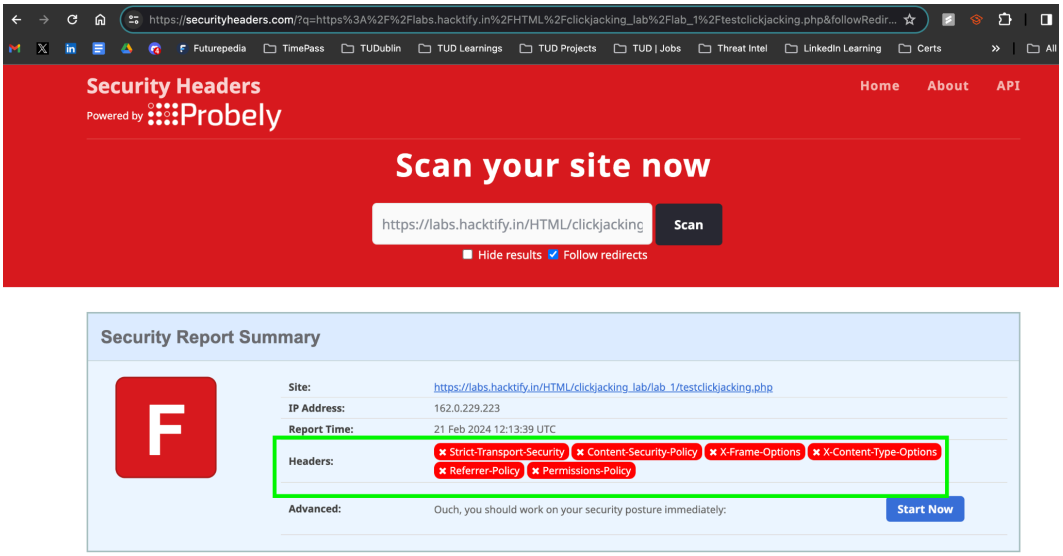


Figure 8. Checking HTTP security headers for the lab URL

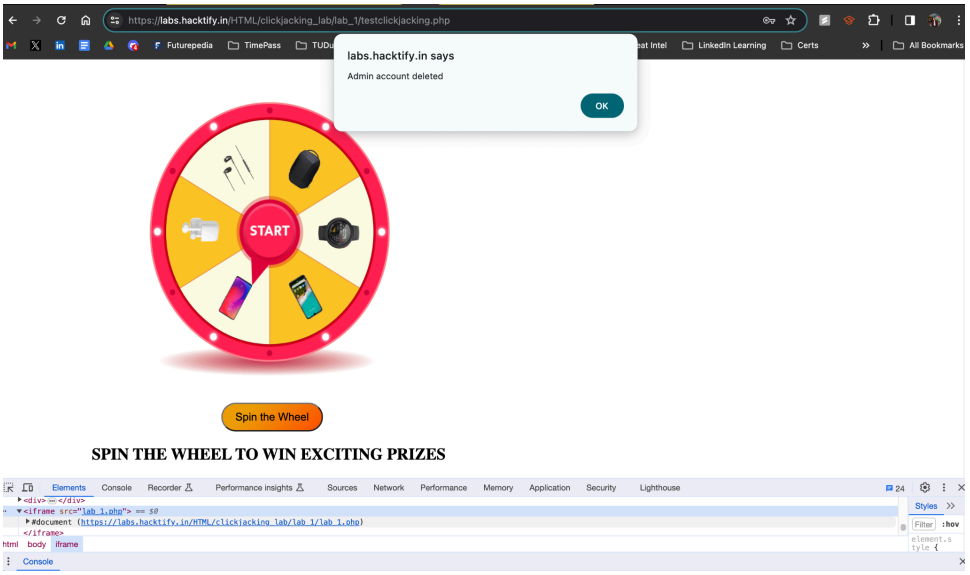


Figure 9. Clickjacking using iframe for User Profile webpage

## 2.2. Re-Hijack!

| Reference | Risk Rating |
|---|---|
| Re-Hijack | High |
| **Tools Used** | |
| To check HTTP Security headers: https://securityheaders.com/<br>https://hacktify.in/clickjacking/<br>Chrome Browser | |
| **Vulnerability Description** | |

| |
|---|
| In the provided Re-Hijack lab, we observed a login form. On checking the webpage at https://securityheaders.com/ we identify that it is missing HTTP security headers such as CSP and X-frame headers. This indicates a potential susceptibility to Clickjacking in the provided lab. Next, we use the Hactify Clickjacking PoC tool to transform the vulnerable lab webpage for capturing credentials. Now on entering the credentials provided on this modified Login page, we see a popup with a message that the credentials have been captured. |
| **How It Was Discovered** |
| Manual Analysis |
| **Vulnerable URLs** |
| https://labs.hacktify.in/HTML/clickjacking_lab/lab_2/lab_2.php |
| **Consequences of not Fixing the Issue** |
| If not patched, the vulnerability can be leveraged by threat actors to harvest credentials and eventually account takeovers. |
| **Suggested Countermeasures** |
| 1. Content-Security-Policy: frame-ancestors <source>; header must be implemented at the web server.<br>2. X-Frame-Options: SAMEORIGIN / DENY must be set to restrict the framing.<br>3. Implementing JavaScript code in the page to attempt to prevent it being loaded in a frame (known as a "frame-buster"). |
| **References** |
| 1. https://securityheaders.com/<br>2. https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/11-Client_Side_Testing/09-Testing_for_Clickjacking |

## Proof of Concept

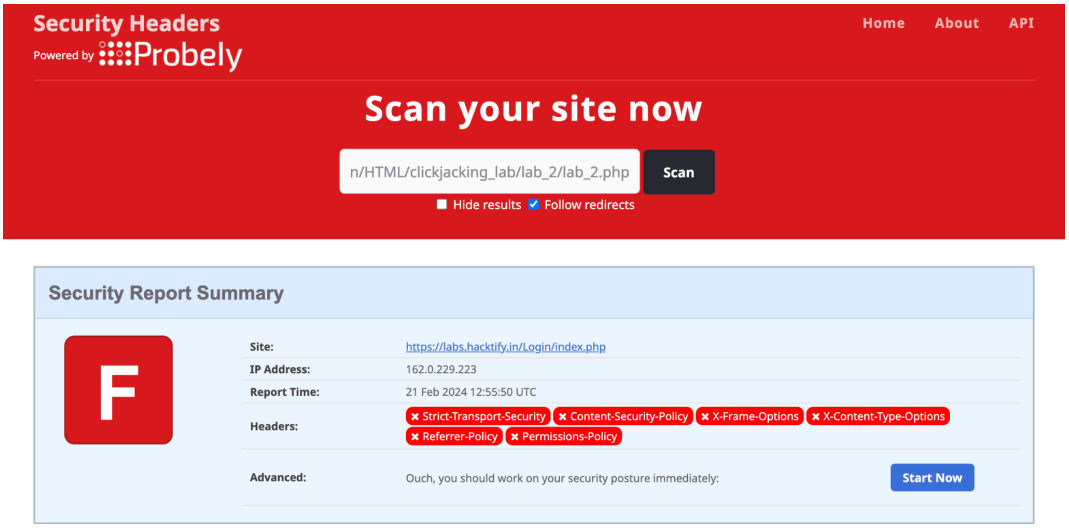


Figure 10. Checking HTTP security headers for the lab URL

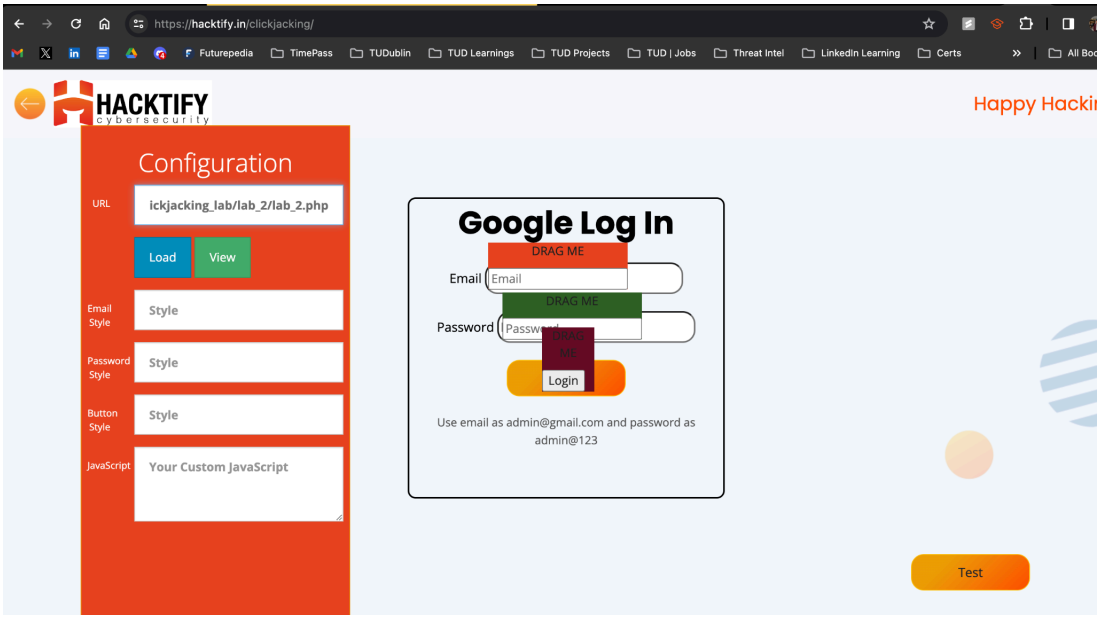


Figure 11. Hactify Clickjacking PoC tool to transform the vulnerable lab webpage for capturing credentials

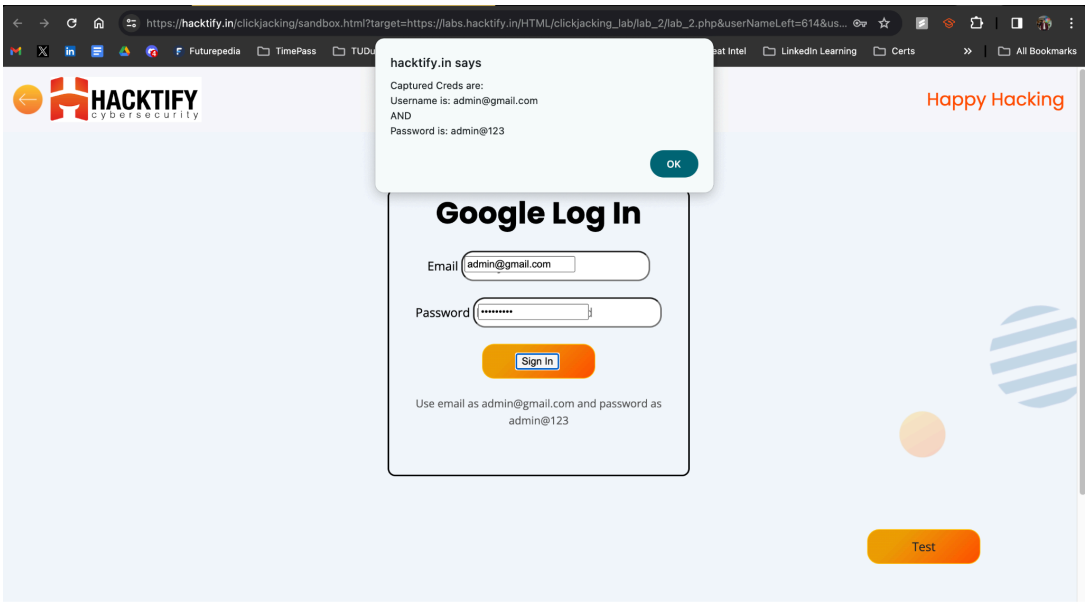


Figure 12. Clickjacking using iframes for email and password successful