Week 2 Penetration Testing Report

Introduction

This report document hereby describes the proceedings and results of a Black Box security assessment conducted against the **Week 2 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 2 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 2 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,**

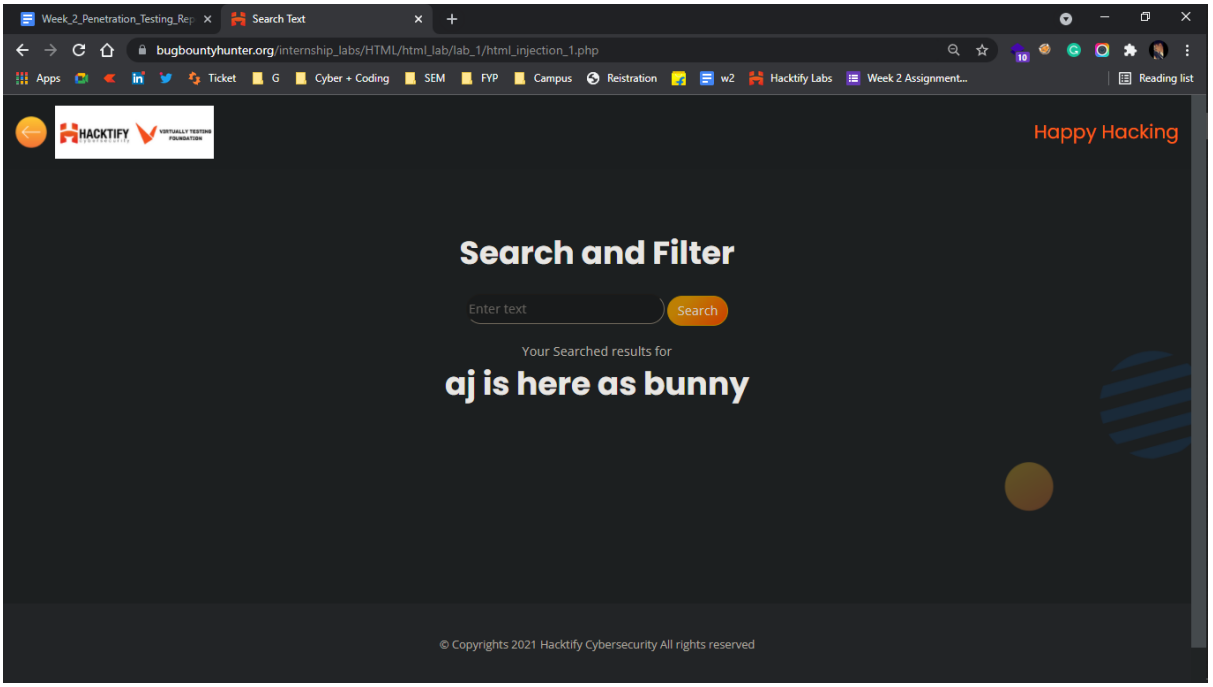
Low - **Number of Sub-labs with Easy difficulty level**

1. {HTML Injection}

1.1. {HTML's are easy!}

Reference	Risk Rating
HTML's are easy!	**Low**
Tools Used	
Google chrome.	
Vulnerability Description	
I found this vulnerability by using basic HTML tags i.e., h1 tag.	
How It Was Discovered	
Manual Analysis.	
Vulnerable URLs	
https://www.bugbountyhunter.org/internship_labs/HTML/html_lab/lab_1/html_injection_1.php	
Consequences of not Fixing the Issue	
This can lead to managing and accessing other web pages and it can lead to phishing other users.	
Suggested Countermeasures	
Sanitize the input by validating canonical tags (<>) and slash (/). or < > ( ) ' " / \ * ; : = { }.	
References	
I used basic knowledge of HTML to discover this vulnerability.	

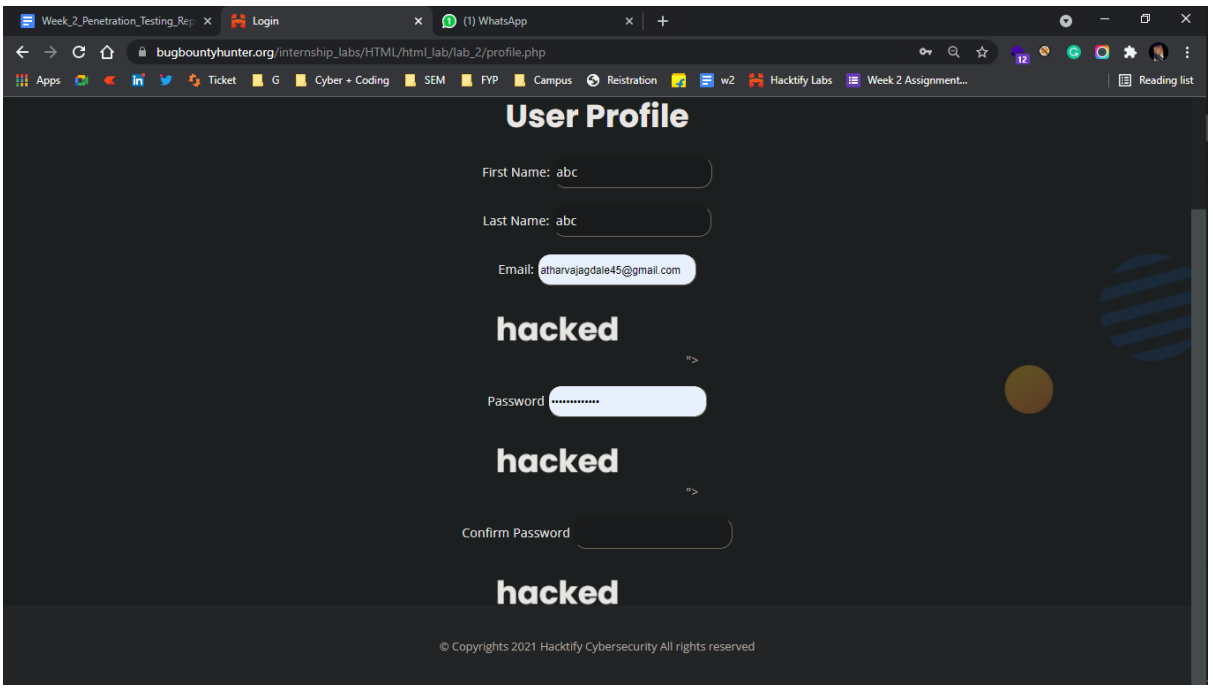
Proof of Concept



1.2. {Let me store them}

Reference	Risk Rating
Let me store them	**Low**
Tools Used	
Google chrome.	
Vulnerability Description	
I found this vulnerability by trying multiple HTML tags on different input fields. Here I am able to run every HTML valid code to inject malicious code.	
How It Was Discovered	
Manual Analysis.	
Vulnerable URLs	
https://www.bugbountyhunter.org/internship_labs/HTML/html_lab/lab_2/html_injection_2.php	
Consequences of not Fixing the Issue	
To steal another person's identity.	
Suggested Countermeasures	
Sanitize the input by validating canonical tags (<>) and slash (/). or < > ( ) ‘ “ / \ * ; : = { }.	
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	

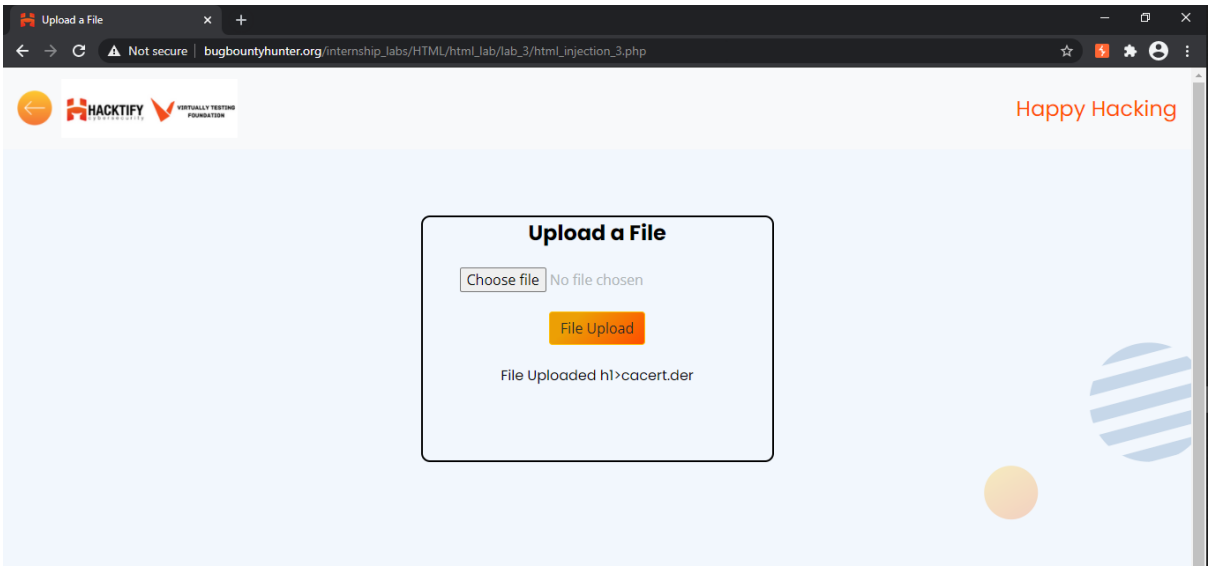
Proof of Concept



1.3. {File names are also vulnerable}

Reference	Risk Rating
File names are vulnerable	**Low**
Tools Used	
Google chrome and BurpSuite	
Vulnerability Description	
I found this vulnerability by intercepting a web app on burpsuite and changing the filename on it to forward back on the web to a successful run.	
How It Was Discovered	
Manual Analysis as well as automated analysis	
Vulnerable URLs	
https://www.bugbountyhunter.org/internship_labs/HTML/html_lab/lab_3/html_injection_3.php	
Consequences of not Fixing the Issue	
Malicious file names may distract the user and also it can cause the user to do something illegal or wrong on the web.	
Suggested Countermeasures	
Sanitize the input by validating canonical tags (<>) and slash (/). or < > ( ) ‘ “ / \ * ; : = { } and it should not accept any of these tags by external resources also.	
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	

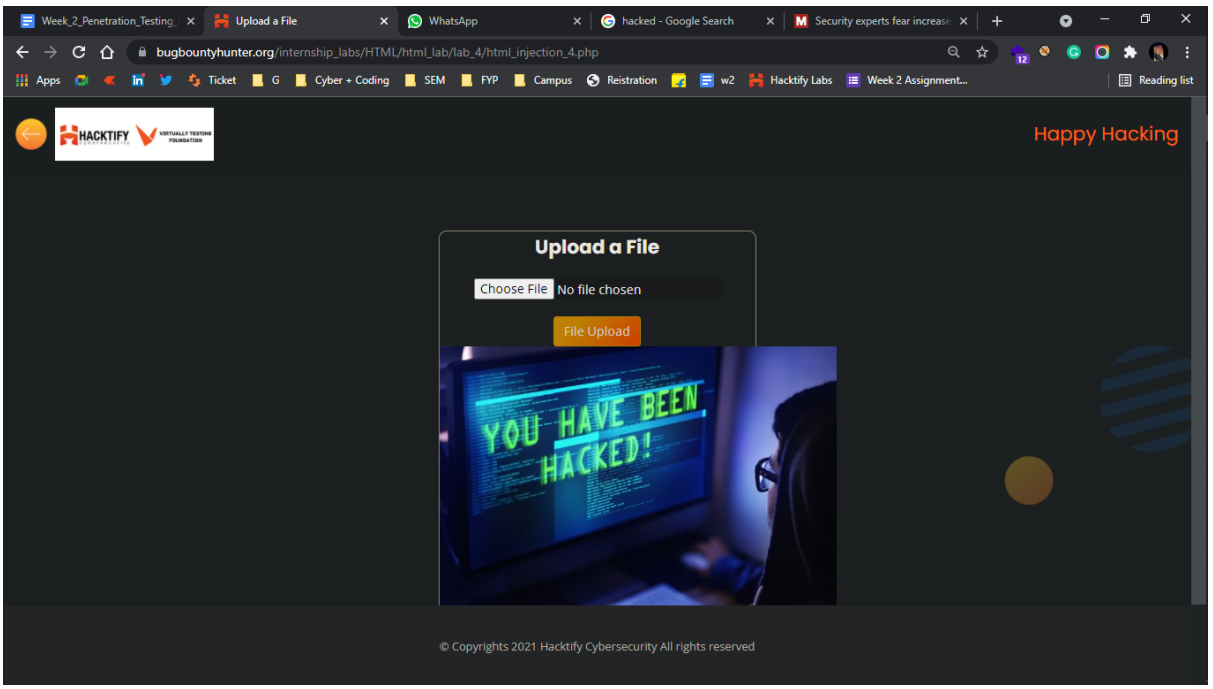
Proof of Concept



1.4. {File content and HTML Injection a perfect pair}

Reference	Risk Rating
File content and HTML Injection are a perfect pair	**Medium**
Tools Used	
Google chrome	
Vulnerability Description	
I found this vulnerability by just uploading malicious file in the file uploading section with different file extensions and containing malicious code in it.	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://www.bugbountyhunter.org/internship_labs/HTML/html_lab/lab_4/html_injection_4.php	
Consequences of not Fixing the Issue	
It can lead to taking down an entire website and the website will show wrong data content on the website.	
Suggested Countermeasures	
Sanitization of file extensions that it should accept only required files with valid and specified file extensions	
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	

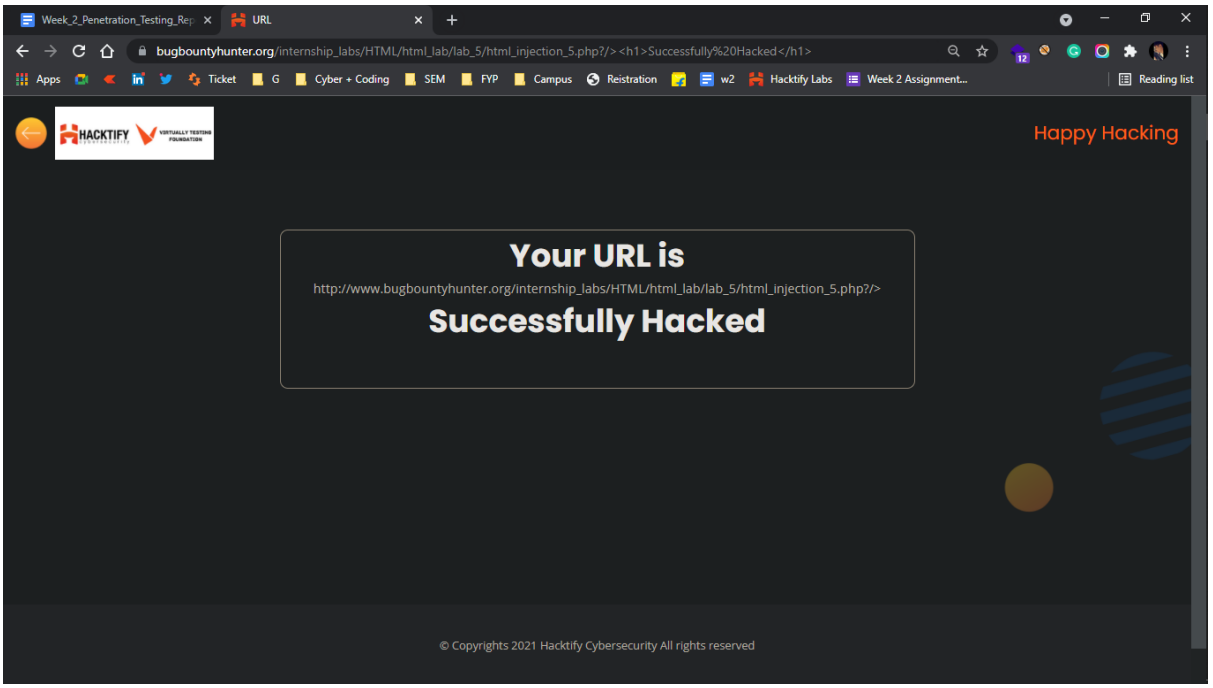
Proof of Concept



1.5. {Injecting HTML using URL}

Reference	Risk Rating
Injecting HTML using URL	**Medium**
Tools Used	
Google chrome	
Vulnerability Description	
I found this vulnerability by tampering actual website of the code and injecting valid HTML code as malicious cod or payload	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://www.bugbountyhunter.org/internship_labs/HTML/html_lab/lab_5/html_injection_5.php	
Consequences of not Fixing the Issue	
It can allow attacker modify the URL and modify the web pages	
Suggested Countermeasures	
Invalid URL should not work or should display errors to the user	
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	

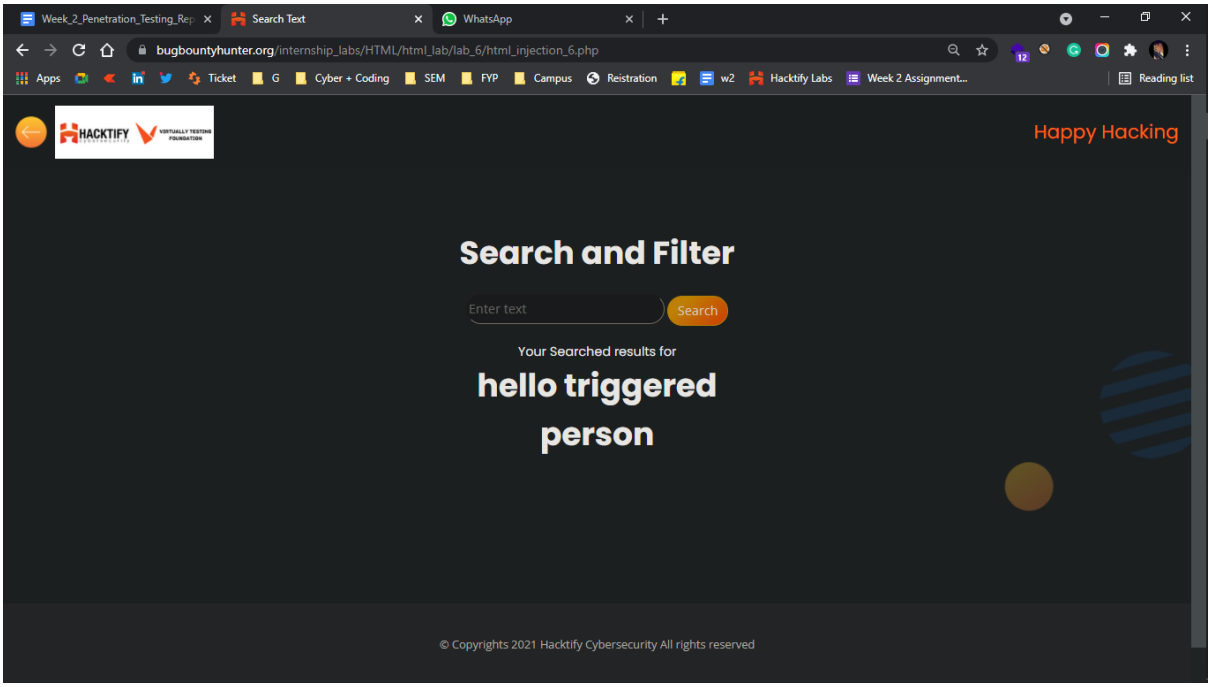
Proof of Concept



1.6. {Encode it}

Reference	Risk Rating
Encode it	**High**
Tools Used	
Google chrome and Burp Suite	
Vulnerability Description	
I found this vulnerability by entering basic HTML valid code by encoding it into URL encoder	
How It Was Discovered	
Manual Analysis and automated analysis	
Vulnerable URLs	
https://www.bugbountyhunter.org/internship_labs/HTML/html_lab/lab_6/html_injection_6.php	
Consequences of not Fixing the Issue	
This can lead to managing and accessing other web pages and it can lead to phishing other users.	
Suggested Countermeasures	
Sanitize the input by validating canonical tags (<>) and slash (/). or < > ( ) ‘ “ / \ * ; : = { } also encoding with the help of other external tools	
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

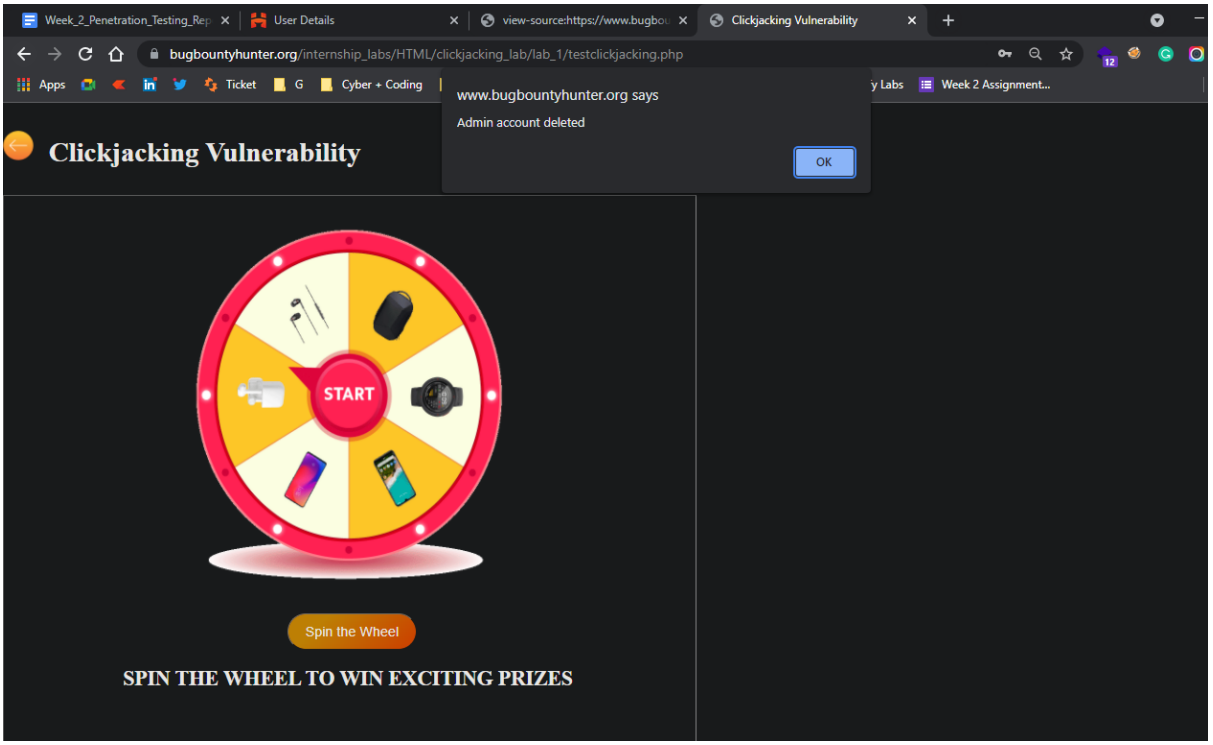


2. {Clickjacking}

2.1. {Let's Hijack}

Reference	Risk Rating
Let's Hijack	**Low / Medium / High**
Tools Used	
Google Chrome	
Vulnerability Description	
I found this vulnerability by clicking on the direct link of the external page link which is directly available on the page source option.	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://www.bugbountyhunter.org/internship_labs/HTML/clickjacking_lab/lab_1/lab_1.php	
Consequences of not Fixing the Issue	
This is an interface-based attack in which a user is tricked into clicking on actionable content on a hidden website by clicking on some other content in a decoy website.	
Suggested Countermeasures	
Important page links should not be easily available on the page source option or it must be in encoded format. and the user must check what they are reaching out after clicking on a particular button.	
References	
https://owasp.org/www-community/attacks/Clickjacking https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html	

Proof of Concept



2.2. {Re-Hijack}

Reference	Risk Rating
Re-Hijack	**Medium**
Tools Used	
Google chrome	
Vulnerability Description	
I found this vulnerability by clicking on the direct link of the external page link which is directly available on the page source option.	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://www.bugbountyhunter.org/internship_labs/HTML/clickjacking_lab/lab_2/lab_2.php	
Consequences of not Fixing the Issue	
This is an interface-based attack in which a user is tricked into clicking on actionable content on a hidden website by clicking on some other content in a decoy website.	
Suggested Countermeasures	
Important page links should not be easily available on the page source option or it must be in encoded format. and the user must check what they are reaching out after clicking on a particular button.	
References	
https://owasp.org/www-community/attacks/Clickjacking https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html	

Proof of Concept