

Penetration Testing Report

Week 1

Full Name: **Rayen Gaied**
Program: **HCPT**
Date: **21/02/2024**

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 | <ul style="list-style-type: none">Lab1 - <u>Clickjacking</u> <p>Known as UI Redressing is a deceptive technique used by attackers to trick users into clicking on hidden or disguised elements on a webpage. Attackers achieve this by overlaying transparent or opaque layers over legitimate content, allowing them to manipulate user interactions and perform actions without the user's knowledge or consent. Clickjacking attacks exploit the trust users have in familiar websites and interfaces, deceiving them into inadvertently executing malicious actions.</p> |
| | <ul style="list-style-type: none">Lab2 - <u>HTML Injection</u> <p>Known as Code Injection, this type of attack occurs when an attacker exploits vulnerabilities in a web application's input fields to insert malicious HTML or JavaScript code. The injected code can alter the appearance, behavior, or functionality of the webpage, leading to various security risks such as phishing attacks, session hijacking, or data theft. An example of this is an attacker injecting a script that redirects users to a malicious website or steals their session cookies.</p> |

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 |
|------|--------|-----|
| 4 | 3 | 1 |

High - Number of Sub-labs with hard difficulty level

Medium - Number of Sub-labs with Medium difficulty level

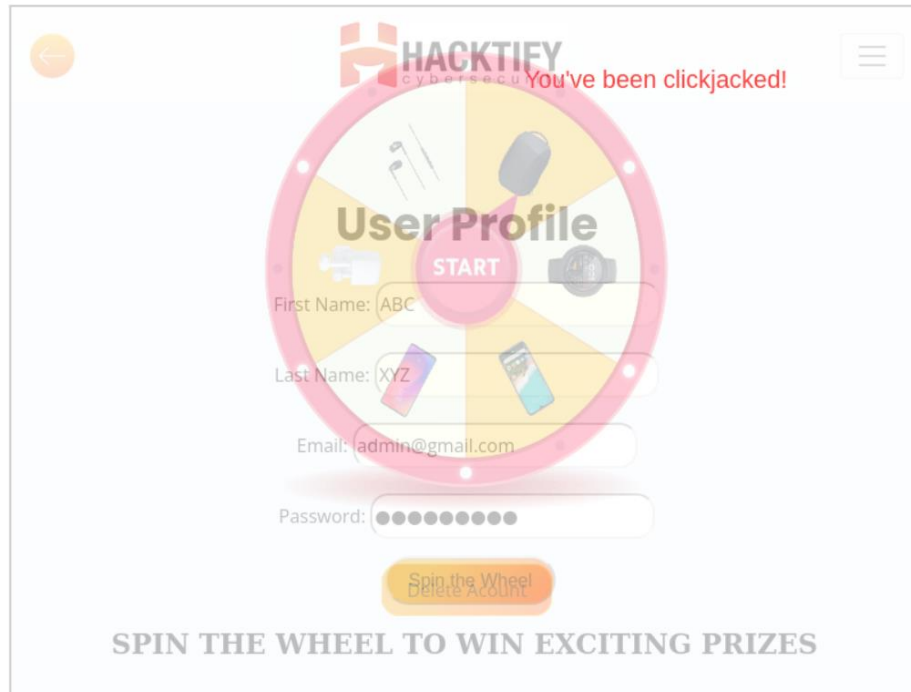
Low - Number of Sub-labs with Easy difficulty level

1. Clickjacking Lab:

1.1. Let's Hijack!

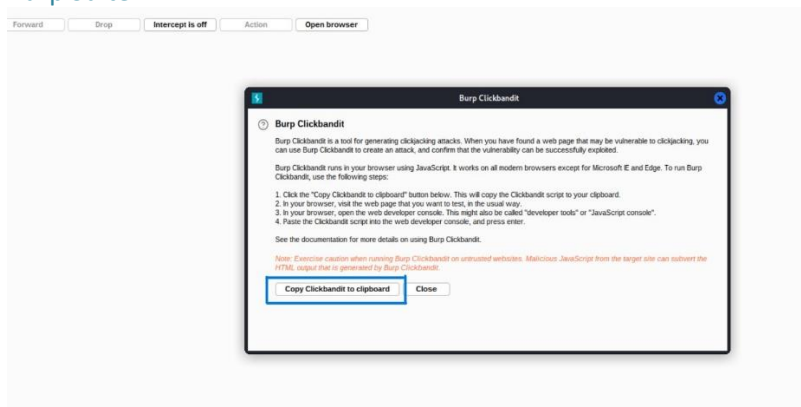
| Reference | Risk Rating |
|---|-------------|
| Let's Hijack! | Low |
| Tools Used | |
| <ul style="list-style-type: none">• PortSwigger's Burp Suite: a powerful suite of integrated tools for performing security testing of web applications. It consists of several components that cater to different aspects of web app security assessment like:<ul style="list-style-type: none">- Burp Proxy: A proxy server that intercepts all HTTP(S) traffic between your browser or tool and the target application. This allows you to inspect and modify requests and responses before sending them to the target.- Spider: An automated web spider that discovers new URLs within the target application based on links found in existing pages.- Intruder: A tool for conducting automated fuzz testing and brute force attacks against specific parameters in requests sent to the target application. | |
| Vulnerability Description | |
| <ul style="list-style-type: none">• The vulnerability appears to be primarily associated with clickjacking, which manipulates the user interface to achieve undesired outcomes. This technique relies on layering elements, such as an <iframe>, over another element to disguise the true nature of the interaction. In this case, the <iframe> covers the "delete account" button, making it seem like clicking anywhere inside the <iframe> will trigger the deletion function when it executes the "spin the wheel" functionality. | |

← Clickjacking Vulnerability



How It Was Discovered

- Burp Suite

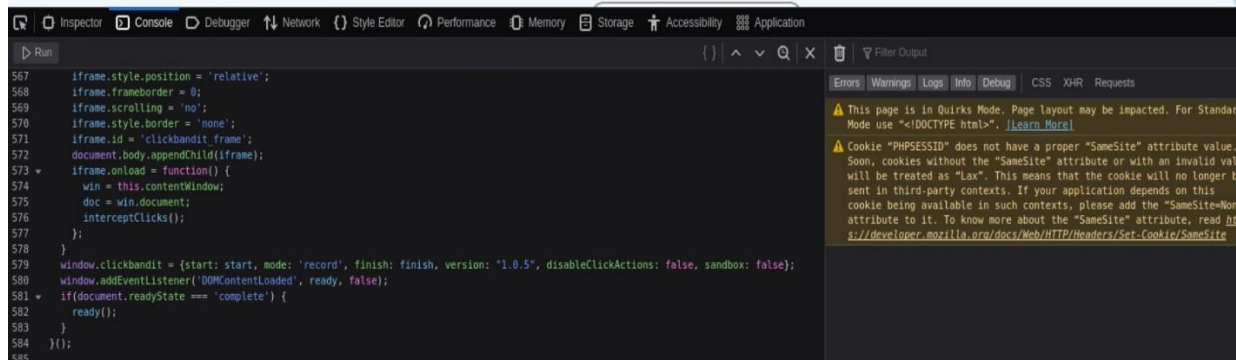


User Profile

First Name:

Last Name:

Email:



Vulnerable URLs

- https://labs.hacktify.in/HTML/clickjacking_lab/lab_1/lab_1.php

Consequences of not Fixing the Issue

- **Lost Data, Leakage, and Theft:** The vulnerability could result in the loss, leakage, or theft of data, compromising the confidentiality, integrity, and availability of the system and its data.

Suggested Countermeasures

- **Access Control:** Implement strong access control mechanisms, such as multi-factor authentication, role-based access control, and least privilege principles, to minimize the impact of successful attacks.
- **Content Security Policy (CSP):** Utilize CSP headers to prevent the execution of untrusted scripts and block the loading of external resources from unverified sources.

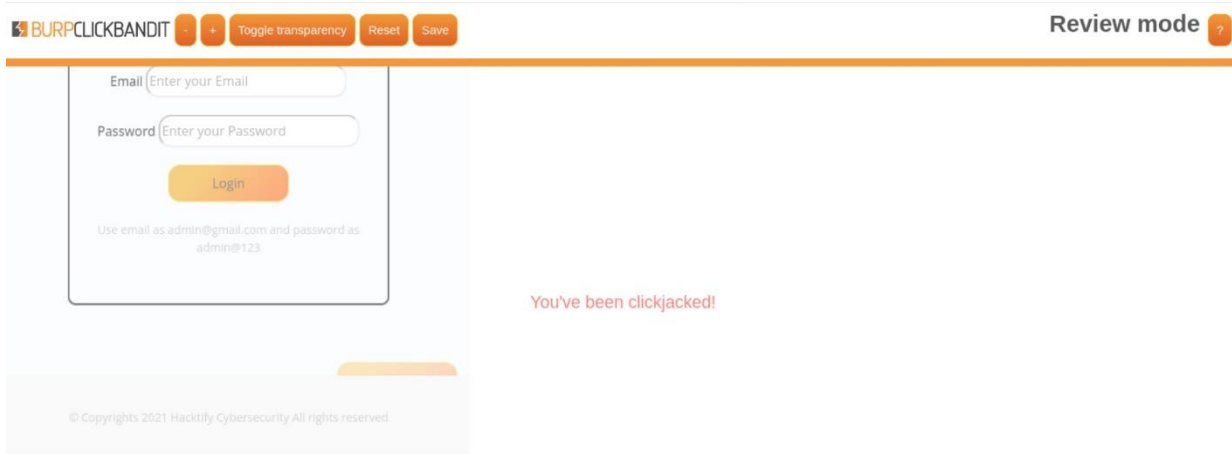
References

- <https://auth0.com/blog/preventing-clickjacking-attacks/>
- https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html

Proof of Concept

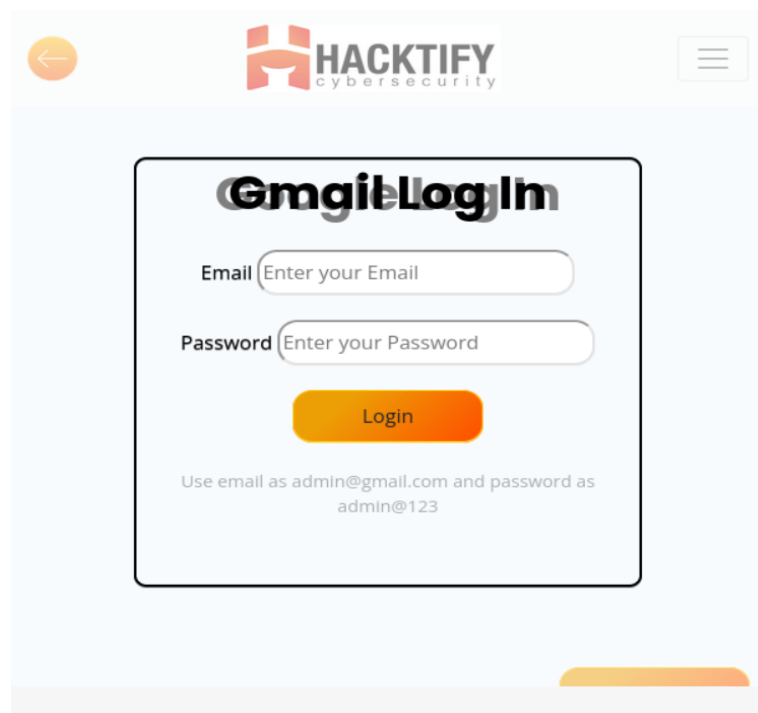
This section contains the proof of the above vulnerabilities as the screenshot of the vulnerability of the lab.

1.2. Re-Hijack!

| Reference | Risk Rating |
|---|-------------|
| Re-Hijack! | Medium |
| Tools Used | |
| <ul style="list-style-type: none">• PortSwigger's Burp Suite: a powerful suite of integrated tools for performing security testing of web applications. It consists of several components that cater to different aspects of web app security assessment like:<ul style="list-style-type: none">- Burp Proxy: A proxy server that intercepts all HTTP(S) traffic between your browser or tool and the target application. This allows you to inspect and modify requests and responses before sending them to the target.- Spider: An automated web spider that discovers new URLs within the target application based on links found in existing pages.- Intruder: A tool for conducting automated fuzz testing and brute force attacks against specific parameters in requests sent to the target application. | |
| Vulnerability Description | |
| <ul style="list-style-type: none">• Two nearly indistinguishable login forms are crafted, differing slightly in their responses upon submission. One form displays the standard "Login Successful" message, while the second form exposes the entered username and password to the attacker.<ul style="list-style-type: none">- The attacker embeds the malicious login form within a convincing email or website, enticing the target to enter their credentials. Since the form closely mimics the genuine login form, the target is unlikely to notice the discrepancies.- Once the target enters their credentials, the attacker gains access to the exposed username and password, allowing them to impersonate the target or utilize the stolen credentials for future attacks. | |
|  <p>The screenshot shows a web application interface with a login form. The form has fields for 'Email' and 'Password', both with placeholder text 'Enter your Email' and 'Enter your Password' respectively. Below the fields is a 'Login' button. A message below the button reads: 'Use email as admin@gmail.com and password as admin@123.' To the right of the form, a red message says 'You've been clickjacked!'. At the bottom left, a copyright notice reads: '© Copyrights 2021 Hacktify Cybersecurity All rights reserved'. The top of the interface has a navigation bar with the text 'BURPCLICKBANDIT', a '+' icon, buttons for 'Toggle transparency', 'Reset', and 'Save', and a 'Review mode' button with a question mark icon.</p> | |

How It Was Discovered

- Burp Suite



Vulnerable URLs

- https://labs.hacktify.in/HTML/clickjacking_lab/lab_2/lab_2.php

Consequences of not Fixing the Issue

- **Compromised User Credentials:** Users who enter their credentials into the malicious login form will have their credentials exposed to the attacker, enabling the attacker to impersonate the user or utilize the stolen credentials for future attacks.
- **Legal Liability:** Organizations that fail to adequately address vulnerabilities may face legal liability for data breaches, privacy violations, and other damages resulting from the exposure of sensitive information.
- **Decreased Operational Efficiency:** Unresolved vulnerabilities can lead to performance degradation, stability issues, and decreased system availability, affecting overall operational efficiency.

Suggested Countermeasures

- **Strong Authentication Mechanisms:** Enforce multi-factor authentication (MFA) to add an extra layer of security beyond traditional username and password combinations.
- **Captcha Verification:** Integrate Captchas or other verification tools to deter automated bot attacks and improve the accuracy of distinguishing between humans and bots.

References

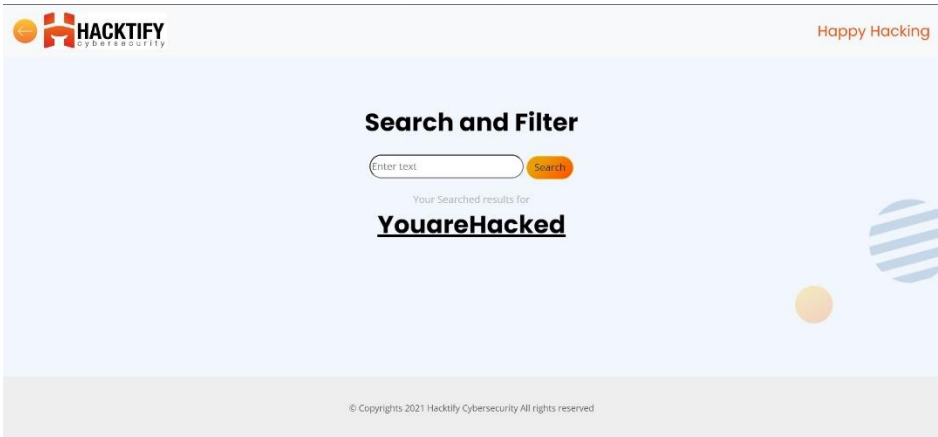
- <https://pratum.com/blog/328-sources-for-vulnerability-and-threat-information>

Proof of Concept

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

2. HTML Injection Lab:

2.1. HTML's Are Easy!

| Reference | Risk Rating |
|--|-------------|
| HTML's Are Easy! | Low |
| Tools Used | |
| HTML Payload | |
| Vulnerability Description | |
| <ul style="list-style-type: none">The addition of <code><h1>YouareHacked</h1></code> to a web page is an example of HTML injection. HTML injection occurs when untrusted user input is incorrectly included in a web page's output. In this case, the added code will be rendered as a level 1 heading ("YouareHacked") on the web page.If this input is not properly validated or sanitized, it can create a security vulnerability. | |
| How It Was Discovered | |
| <ul style="list-style-type: none">The addition of <code><h1>YouareHacked</h1></code> to a web page  | |
| Vulnerable URLs | |
| <ul style="list-style-type: none">https://labs.hacktify.in/HTML/html_lab/lab_1/html_injection_1.php | |
| Consequences of not Fixing the Issue | |
| <ul style="list-style-type: none">Attackers can exploit this vulnerability to inject malicious code into a web page, steal sensitive information, or compromise the security of a web application. This can result | |

| |
|--|
| in data breaches, loss of sensitive information, and damage to the reputation of the organization. |
| Suggested Countermeasures |
| <ul style="list-style-type: none"> Implement strict input validation: Validate user inputs to ensure that they meet the expected format and type. Use whitelisting approaches whenever possible to allow only trusted characters and values. Perform output encoding: Before rendering user inputs in the final HTML output, encode special characters to prevent them from being misinterpreted as executable code. Conduct regular security tests: Test your web applications for injection vulnerabilities during development, staging, and production phases. Automate security testing wherever possible to catch vulnerabilities early and often. |
| References |
| <ul style="list-style-type: none"> https://owasp.org/www-community/attacks/HTML_Injection https://www.acunetix.com/blog/articles/html-injection/ |

2.2. Let Me Store Them!

| Reference | Risk Rating |
|--|-------------|
| Let Me Store Them! | Low |
| Tools Used | |
| HTML Payload | |
| Vulnerability Description | |
| <ul style="list-style-type: none"> Properly is the same as the first one. | |
| How It Was Discovered | |
| <ul style="list-style-type: none"> create a new user using the following values for each field, First Name: " "> h1> user < /h1> " Last Name: ""> h1> go < /h1> " Email-id: "user@mail.com" Password: "123" | |

User Profile

First Name:

user

" />

Last Name:

go

" />

- The HTML injection performed in lab 1 is called reflected HTML injection, as the payload entered in the search box persists only in the current browser session. In contrast, a stored HTML injection would persist even if the user logged out of their account. To test this vulnerability, you can create an account in one browser, such as Firefox, and then close the labs. Next, log in to the labs in another browser, such as Chrome, go to lab 2, and enter the same email ID and password ("user@mail.com" and "123"). You can observe that the payload persists as stored HTML injection. Stored HTML injection allows the attacker to store the payload at the server-side and is more impactful than reflected HTML injection.

Vulnerable URLs

- https://labs.hacktify.in/HTML/html_lab/lab_2/register.php

Consequences of not Fixing the Issue

- Data breaches: Hackers can steal sensitive information, leading to data leaks and potential legal liabilities.
- Long-term infections: Exploiting unpatched vulnerabilities enables attackers to maintain persistent access to infected systems, potentially allowing them to launch further attacks

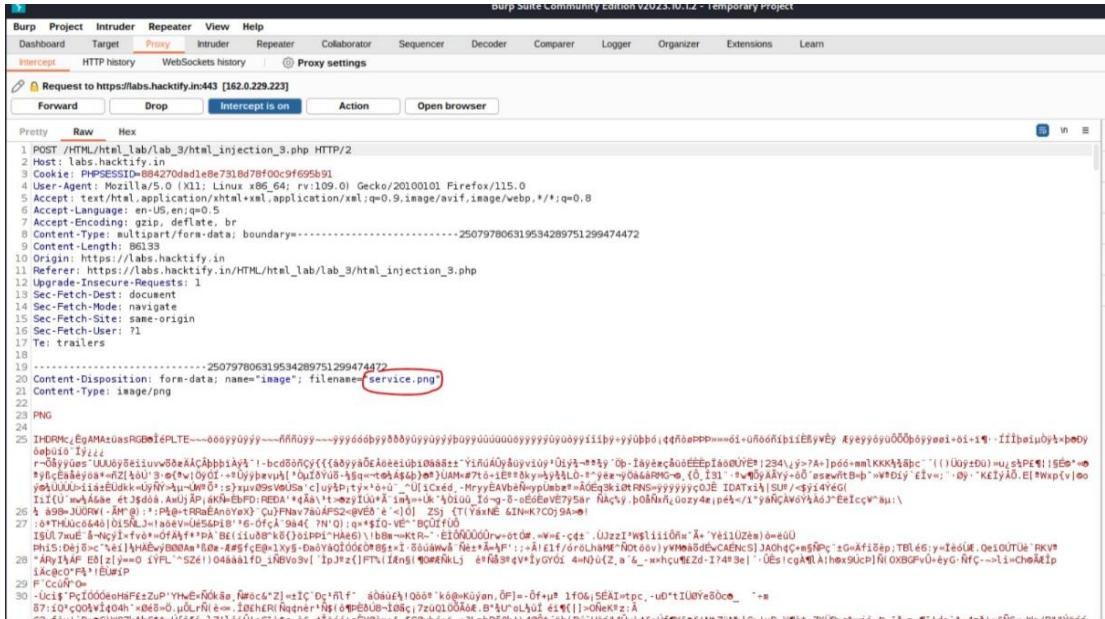
Suggested Countermeasures

- Implement strict input validation: Validate user inputs to ensure that they meet the expected format and type. Use whitelisting approaches whenever possible to allow only trusted characters and values.
- Conduct regular security tests: Test your web applications for injection vulnerabilities during development, staging, and production phases. Automate security testing wherever possible to catch vulnerabilities early and often.
- Monitor and log activities: Log user interactions and events related to input validation and output encoding. Review logs periodically to detect anomalous behavior and investigate potential injection attempts.

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
- <https://www.acunetix.com/vulnerabilities/web/html-injection/>

2.3. File Names Are Also Vulnerable!

| Reference | Risk Rating |
|--|-------------|
| File Names Are Also Vulnerable! | Low |
| Tools Used | |
| Burp Suite | |
| Vulnerability Description | |
| <ul style="list-style-type: none">• Insertion of client-side scripts into the code<ul style="list-style-type: none">- These vulnerabilities occur when untrusted user inputs are incorrectly included in a web page's output, enabling attackers to inject malicious JavaScript or other client-side scripts into the web application. | |
| How It Was Discovered | |
|   | |

```
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Te: trailers
```

```
-----361876572029674706573734986705
Content-Disposition: form-data; name="image"; filename="<h1><u>service.png"
Content-Type: image/png
```

PNG



- This observation suggests that the "filename" parameter, used to display the name of the uploaded file, lacks proper sanitization. Consequently, this allows the insertion of client-side scripts into the code.

Vulnerable URLs

- https://labs.hacktify.in/HTML/html_lab/lab_3/html_injection_3.php

Consequences of not Fixing the Issue

- Cross-Site Scripting (XSS) Attacks: Attackers can exploit the vulnerability to inject malicious scripts into web pages, potentially leading to the theft of sensitive user data, session hijacking, or defacement of the website.
- Cross-Site Scripting (XSS) Attacks: Attackers can exploit the vulnerability to inject malicious scripts into web pages, potentially leading to the theft of sensitive user data, session hijacking, or defacement of the website.

Suggested Countermeasures

- Input Validation: Implement strict input validation to filter out malicious script tags and other unwanted characters.
- Parameterized Queries: Use parameterized queries instead of direct string interpolation to prevent SQL injection and XSS attacks.
- Access Control: Restrict access to sensitive areas of the application to authorized personnel only.

References

2.4. File Content And HTML Injection A Perfect Pair!

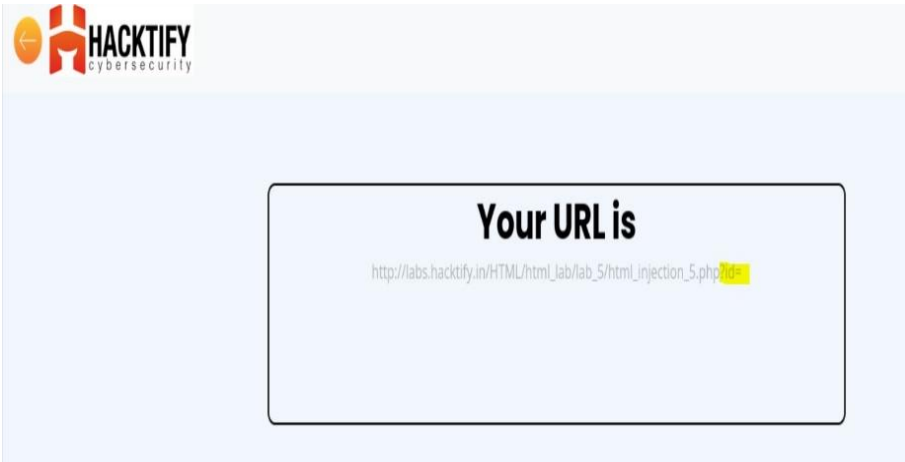
| Reference | Risk Rating |
|---|-------------|
| File Content And HTML Injection A Perfect Pair! | Medium |
| Tools Used | |
| Basic HTML code & Upload | |
| Vulnerability Description | |
| <ul style="list-style-type: none">The backend currently lacks a mechanism to verify the file type during the upload process, and additionally allows parsing and execution of files without proper validation. In analogous real-world scenarios, this vulnerability could potentially facilitate a shell upload, consequently granting unauthorized access to the server for the attacker. | |
| How It Was Discovered | |
| <div><pre>1 <html> 2 <head> 3 <title> Hacked </title> 4 </head> 5 <body> 6 <h1> <u>Hello Hacktify</u></h1> 7 <h2> Stop NOW Hacking!!! </h2> 8 </body> 9 </html> 10 11 12 </pre></div> <div></div> | |
| Vulnerable URLs | |
| <ul style="list-style-type: none">https://labs.hacktify.in/HTML/html_lab/lab_4/html_injection_4.php | |
| Consequences of not Fixing the Issue | |
| <ul style="list-style-type: none">Unauthorized Access: Exploiting the identified vulnerability may grant unauthorized individuals access to sensitive areas of the system or confidential data.Data Compromise: The absence of proper file type validation and execution control could result in the compromise of critical data, leading to data breaches or leakage.Malicious Code Execution: Allowing unverified files to be parsed and executed opens the door to the injection of malicious code, potentially leading to the manipulation or disruption of system functionalities. | |

Suggested Countermeasures

- Implement robust file type validation mechanisms during the upload process to ensure that only permitted file types are accepted.
- Utilize file signature checks or MIME type verification to enhance the accuracy of file type validation.
- Restrict the execution of uploaded files to only essential and safe functionalities. Avoid allowing arbitrary execution of uploaded files.

References

2.5. Injecting HTML Using URL

| Reference | Risk Rating |
|---|-------------|
| Injecting HTML Using URL | Medium |
| Tools Used | |
| HTML Payload | |
| Vulnerability Description | |
| <ul style="list-style-type: none">• The queries in the URL are not being filtered/sanitized which is the cause for our URL to be susceptible to HTML injection attacks. | |
| How It Was Discovered | |
| <p>1- Check the source code for the lab.</p> <pre><section class="pager-section"> <div class="container"> <center> <div class="containers"> <div class="contain"> <h1> Your URL is</h1> http://labs.hacktify.in/HTML/html_lab/lab_5/html_injection_5.php </div> </div> </center> </div> </div></pre> <p>2- Let's add a parameter to our URL, Let's say "?id="</p>  | |

- That gives us the most important information that our UI is getting fetched from the server-side script to the URL we search for!!!

3- Now let's use the following payload: “?id=<h1>GET it</h1>”



Vulnerable URLs

- https://labs.hacktify.in/HTML/html_lab/lab_5/html_injection_5.php

Consequences of not Fixing the Issue

- Cross-Site Scripting (XSS) Attacks: Attackers can exploit the vulnerability to inject malicious scripts into web pages, potentially leading to the theft of sensitive user data, session hijacking, or defacement of the website.
- Compromised User Trust: Security incidents resulting from XSS attacks can damage the trust that users have in the affected organization's website and services.
- Data Breaches: Unfixed XSS vulnerabilities can result in the compromise of sensitive information, such as user credentials, personal data, and financial details.

Suggested Countermeasures

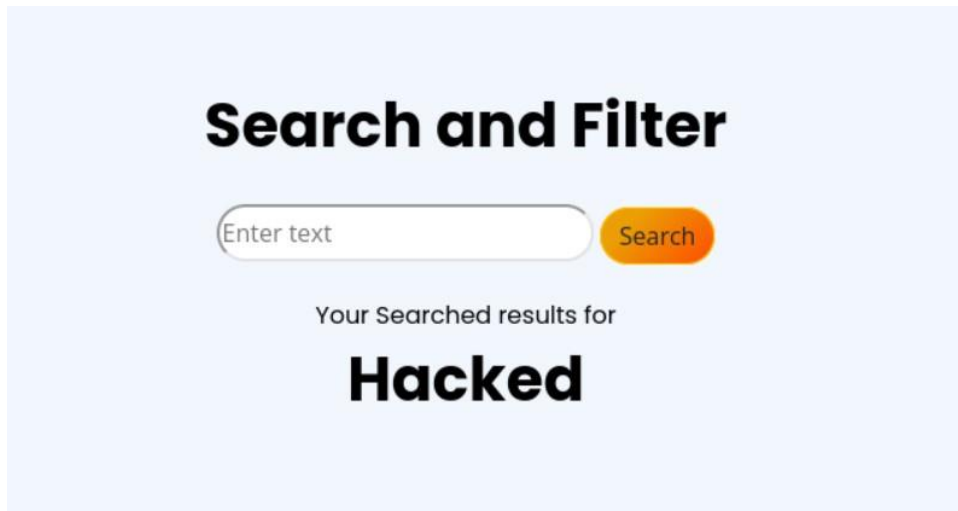
- Whitelist acceptable input: Only accept input that falls within predefined criteria, such as alphanumeric characters, spaces, and certain punctuation marks. Reject any input that contains special characters or scripts.
- Encode special characters: Convert special characters to their corresponding HTML entities to prevent them from being interpreted as executable code.
- Implement Content Security Policy (CSP): CSP is a powerful tool that helps prevent XSS attacks by specifying allowed resources and blocking unsafe execution of scripts.

References

2.6. Encode IT!

| Reference | Risk Rating |
|--|-------------|
| Encode IT! | Hard |
| Tools Used | |
| HTML Payload & URL encoding | |
| Vulnerability Description | |
| <ul style="list-style-type: none">Blacklisting: creating a list of known malicious HTML tags, attributes, or other content and then blocking any input that matches items on the list. While blacklisting can be used as a temporary measure to mitigate HTML injection vulnerabilities, it is generally considered less effective than whitelisting. | |
| How It Was Discovered | |
| <div><h3>Search and Filter</h3><div><input type="text" value="Enter text"/><input type="button" value="Search"/></div><p>Your Searched results for ;h!;Hacked;/h!;</p></div> <p>- So we just need to not use angular bracket, and the name for our lab is also encode it! so why not encode our angular brackets using URL encoding, "<" => "%3c" ">" => "%3e"</p> <div><h3>Search and Filter</h3><div><input type="text" value="%3ch1%3eHacked%3c/h1%3e"/><input type="button" value="Search"/></div></div> | |

- The response to our new payload:



Vulnerable URLs

- https://labs.hacktify.in/HTML/html_lab/lab_6/html_injection_6.php

Consequences of not Fixing the Issue

- Unsafe URLs: Malicious links can be inserted into the webpage, exposing users to various forms of attacks, such as phishing scams or drive-by downloads.
- False positives: Blacklisted items might be legitimate in certain scenarios, leading to false positives where valid content gets rejected.
- Circumvention: Determined attackers can find ways around blacklists, either by obfuscating the malicious content or finding alternative techniques.

Suggested Countermeasures

- Avoid overreliance on blacklisting and combine it with other security measures.
- Use libraries and frameworks that have built-in security features to prevent HTML injection attacks.
- Use Content Security Policy (CSP) to restrict the types of content that can be loaded on the webpage.

References

- <https://www.invicti.com/learn/html-injection/>

**** ** In conclusion, both **clickjacking** and **HTML injection** vulnerabilities pose serious threats to internet users and can have severe consequences for individuals and organizations. Clickjacking exploits visual tricks to deceive website visitors into unintentionally clicking on UI elements, leading to unauthorized actions and potential exposure of sensitive data. The outcomes of a clickjacking attack can range from inconvenience to significant damage to reputation, finances, and personal safety.**

Similarly, HTML injection vulnerabilities carry serious risks, including data breaches, compromised user trust, legal and regulatory consequences, financial losses, impact on brand reputation, disruption of operations, and long-term security risks. To mitigate these risks, organizations should proactively implement preventive measures such as input validation, output encoding, and other security best practices, relying on whitelists rather than blacklists to enhance protection against HTML injection attacks.