Hacktify Security - Penetration Testing Report

Name- Krish Gupta

CORS Labs Report

Lab Title: CSRF Lab 1- Easy CSRF Lab Report

1. Introduction

This report documents the exploitation of a Cross-Site Request Forgery (CSRF) vulnerability found in the 'Easy CSRF' lab. CSRF tricks authenticated users into performing unintended actions on a web application without their consent.

2. Vulnerability Details

- **Vulnerability Type:** Cross-Site Request Forgery (CSRF)
- **Lab Name:** Easy CSRF
- **Objective:** Change the user's email address without their knowledge.
- **Severity:** High

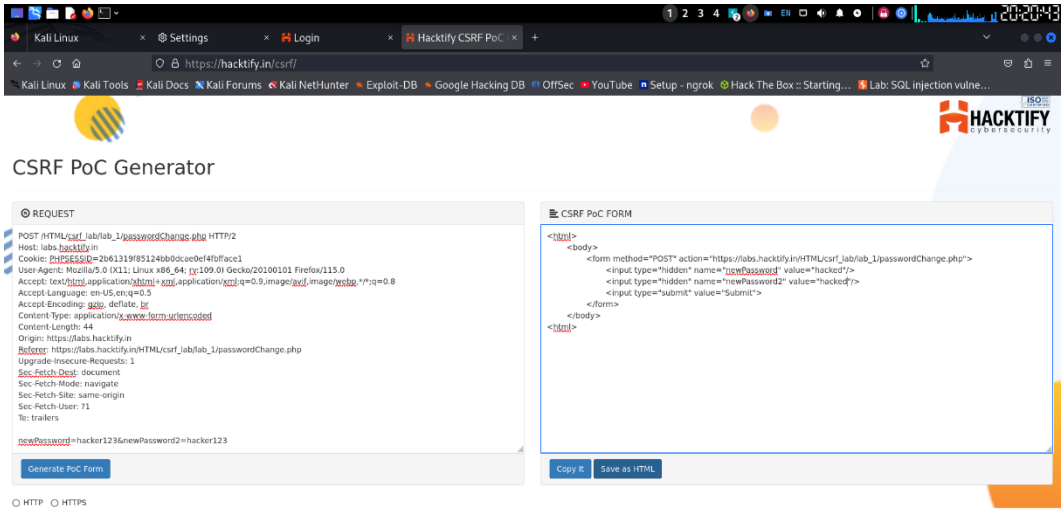
3. Steps to Reproduce

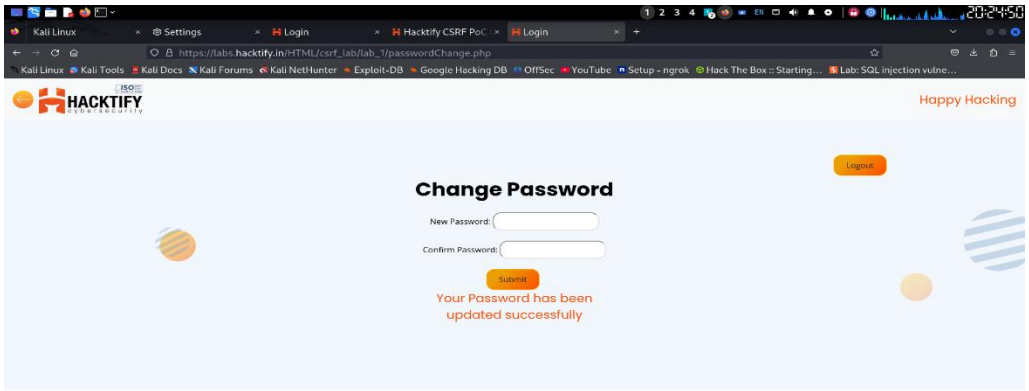
1. **Capture the Request:**
   - Intercept the email change request using Burp Suite.
2. **Create Malicious HTML:**
   - Crafted an HTML form that sends an email change request on form submission.
3. **Host and Execute:**
   - Hosted the HTML on a local server and triggered the request by visiting the page while authenticated.

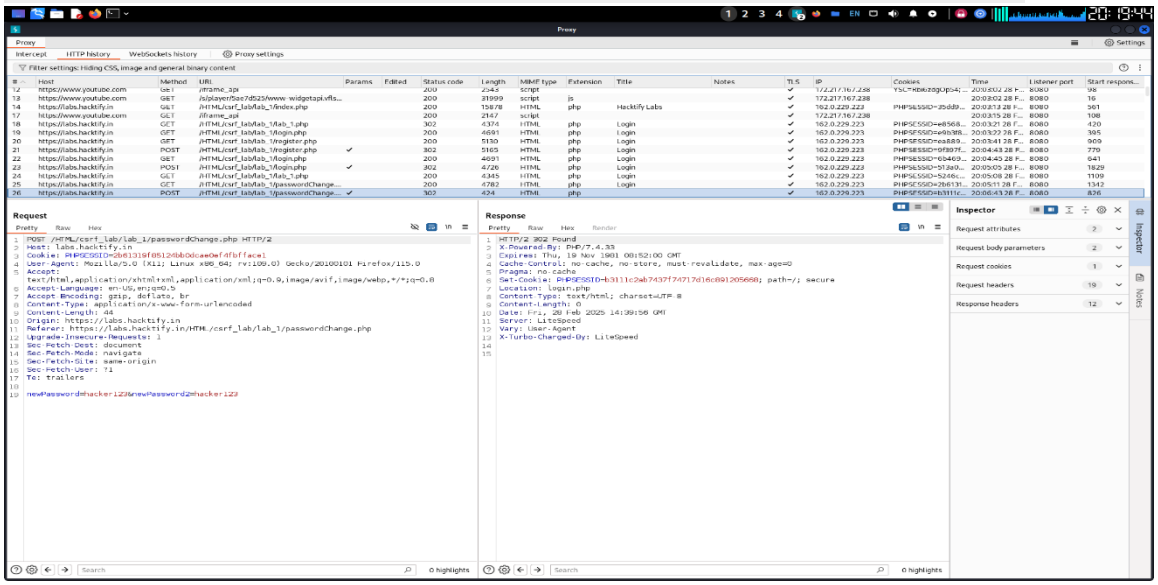
4. Proof of Concept (PoC)

- **Captured Request:**
- **Crafted HTML Form:**
- <form action="http://vulnerable-website.com/change-email" method="POST">
- <input type="hidden" name="email" value="attacker@evil.com">
- <input type="submit" value="Submit">
- </form>

- **Execution Results:**







5. Impact

Exploitation of this CSRF vulnerability allows an attacker to modify sensitive user information without their knowledge, potentially leading to account takeover.

6. Mitigation

- Implement CSRF tokens for state-changing requests.
- Verify the 'Origin' and 'Referer' headers.
- Enforce SameSite cookie attributes.

Lab Title: CSRF Lab 2 - Always Validate Tokens

Description:

This lab demonstrates how Cross-Site Request Forgery (CSRF) vulnerabilities can be exploited using GET-based requests. By tricking an authenticated user into clicking a malicious link, an attacker can perform unauthorized actions on their behalf.

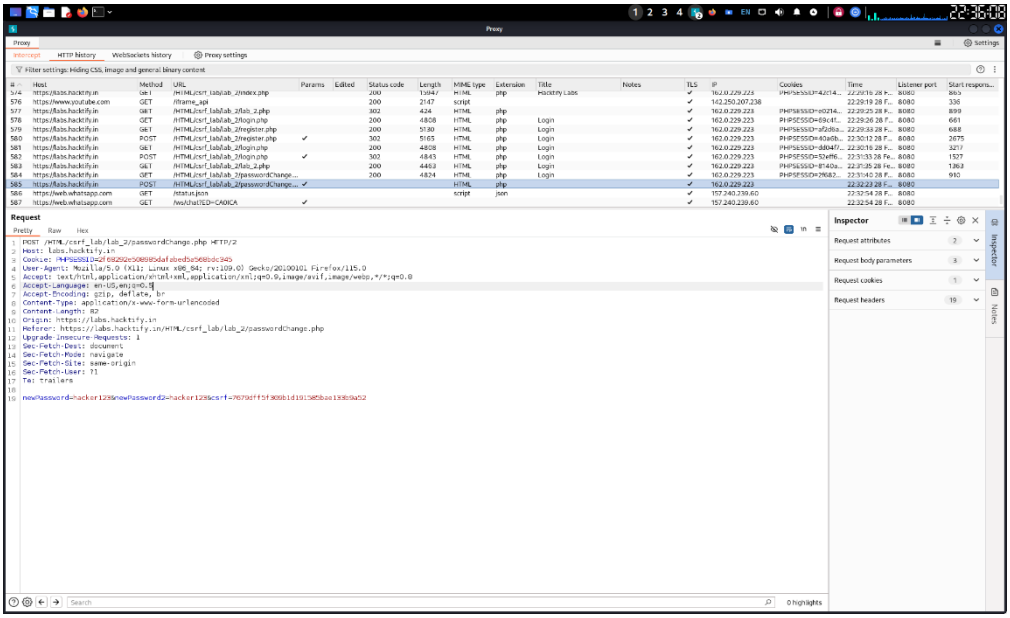
Vulnerability Details:

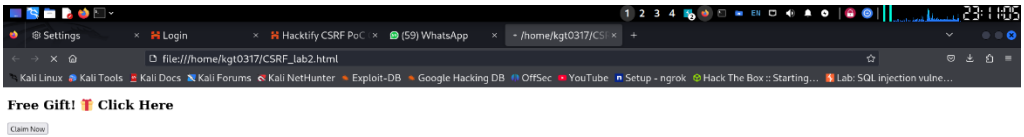
- **Vulnerability Type:** CSRF (Cross-Site Request Forgery)
- **Impact:** Unauthorized actions performed by an attacker on behalf of an authenticated user
- **Severity:** High

Proof of Concept (PoC):

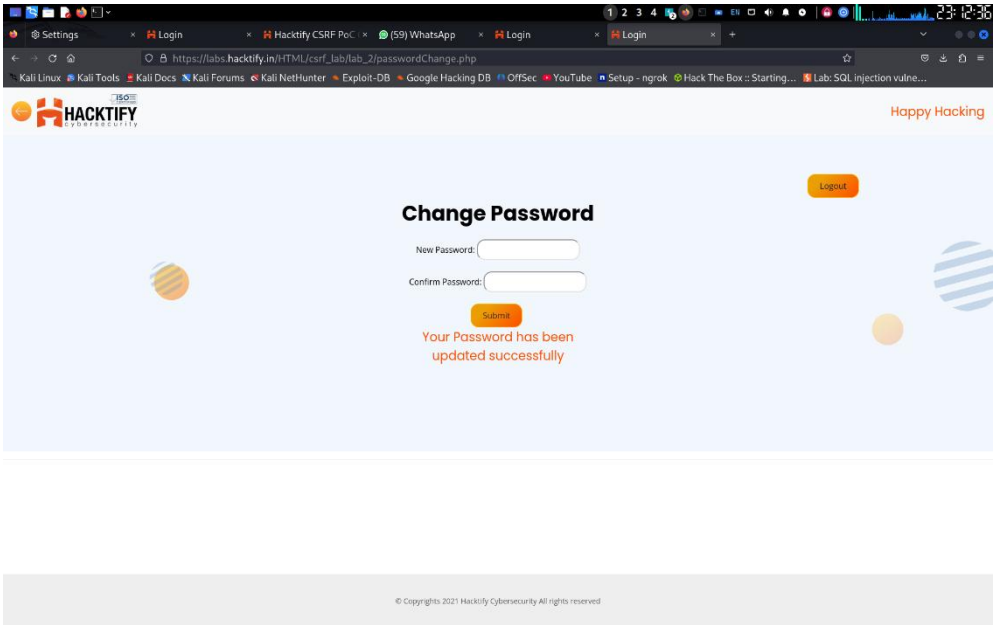
1. **Identify the vulnerable request:**
    - Captured the GET request in Burp Suite where the state-changing action occurs.
    - Refer to the Burp Suite screenshot for request details.
2. **Craft the malicious link:**
    - Constructed a URL mimicking the vulnerable request:
    - http://vulnerable-website.com/change-email?email=attacker%40example.com
3. **Host the malicious link:**
    - Created an HTML page containing the malicious link.
4. <html>
5. <body>
6. Click me!

7. </body>
8. </html>
9. **Execution Proof:**
    - Sent the malicious link to the victim.
    - Upon clicking, the victim's email address was changed without their consent.
    - Refer to the attached screenshots for the captured request and resulting changes:





- 



Mitigation:

- Implement CSRF tokens for state-changing requests.
- Enforce same-site cookie attributes.
- Validate the origin and referer headers.

Conclusion:

This lab illustrates the ease of exploiting GET-based CSRF vulnerabilities and highlights the importance of implementing proper defenses against CSRF attacks.

Lab Title: CSRF Lab 3: I hate when someone uses my tokens!

Vulnerability Type: CSRF (Cross-Site Request Forgery)

Description:

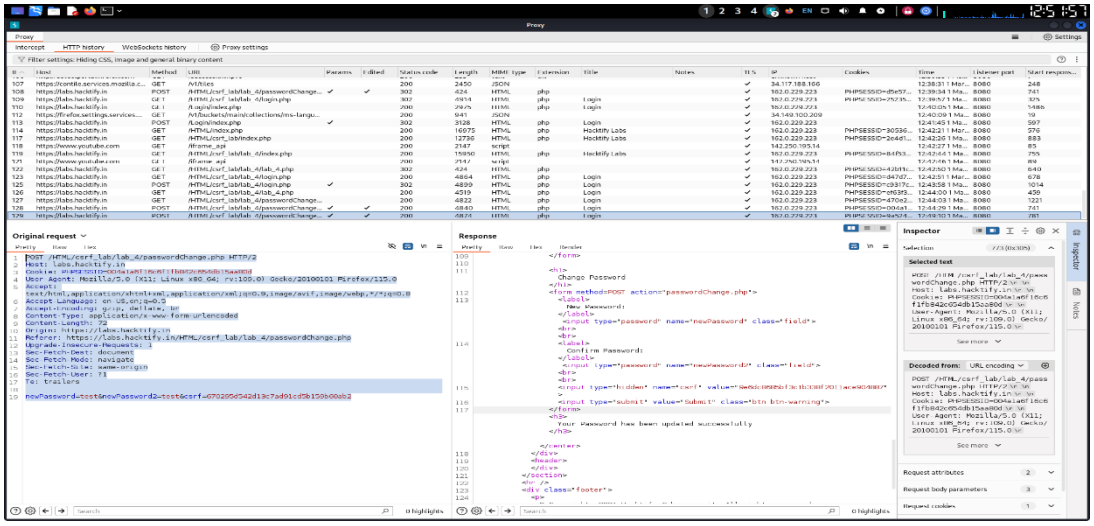
This lab demonstrates a CSRF vulnerability where the application uses anti-CSRF tokens, but the implementation has flaws that allow an attacker to reuse previously issued tokens to perform unauthorized actions. In this case, the password change functionality is vulnerable to CSRF.

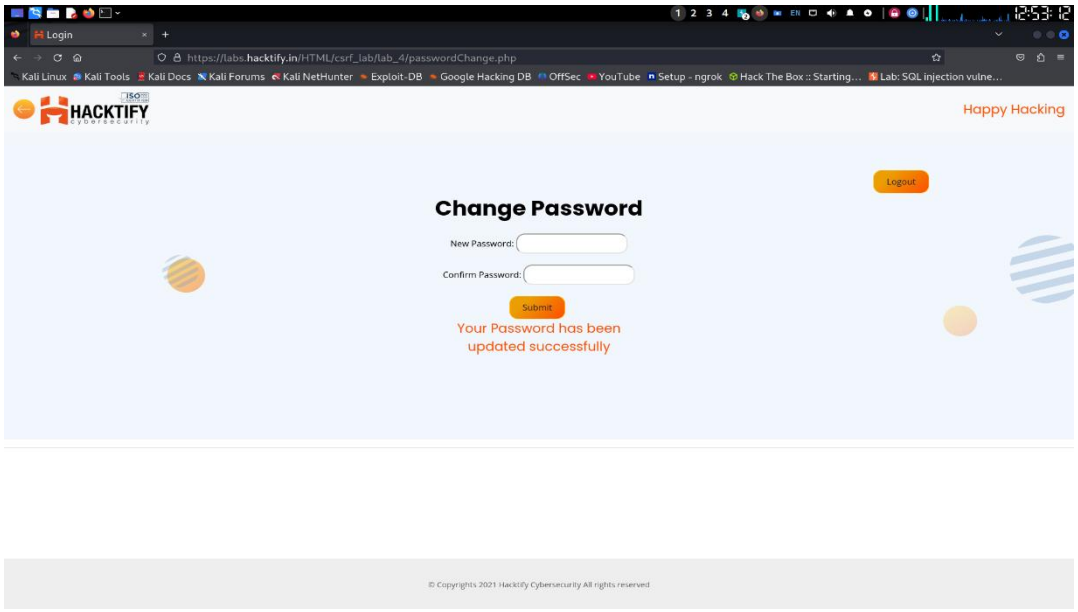
Steps to Reproduce:

1. **Intercept the Request:**
    - Log in to the application and navigate to the password change functionality.
    - Turn on the intercept in Burp Suite and capture the password change request.
2. **Identify the CSRF Token:**
    - In the POST request, locate the 'csrf' parameter.

3. **Modify the Request:**
   - Remove the CSRF token or replace it with an old CSRF token captured from a previous request.
4. **Forward the Request:**
   - Forward the modified request and observe the response.
5. **Check the Response:**
   - If the password change is successful despite using an old CSRF token, it confirms the vulnerability.

Proof of Concept:





Impact:

An attacker can change the victim's password or perform other unauthorized actions on their behalf without their consent.

Severity: High

Mitigation:

- Implement strict CSRF token validation by ensuring tokens are one-time use and expire after a short period.
- Bind CSRF tokens to specific user sessions.
- Validate the 'Referer' and 'Origin' headers for sensitive requests.
- Use SameSite cookies to prevent CSRF attacks.

CSRF Lab 4 Report- GET me or POST me

Vulnerability: Cross-Site Request Forgery (CSRF)

Description:

CSRF (Cross-Site Request Forgery) is a web security vulnerability that forces an authenticated user to execute unwanted actions on a different web application. The attacker tricks the victim into submitting a malicious request, which performs actions on behalf of the victim without their consent.

Impact:

- Unauthorized changes to user data
- Account takeover
- Manipulation of application state

Steps to Reproduce:

1. The victim logs into the target web application.
2. The attacker crafts a malicious HTML form designed to execute an action on behalf of the victim.
3. The victim clicks on a link or visits a page controlled by the attacker, which submits the malicious form.
4. The unauthorized action is executed with the victim’s authenticated session.

Proof of Concept (PoC):

The following HTML payload demonstrates the CSRF attack:

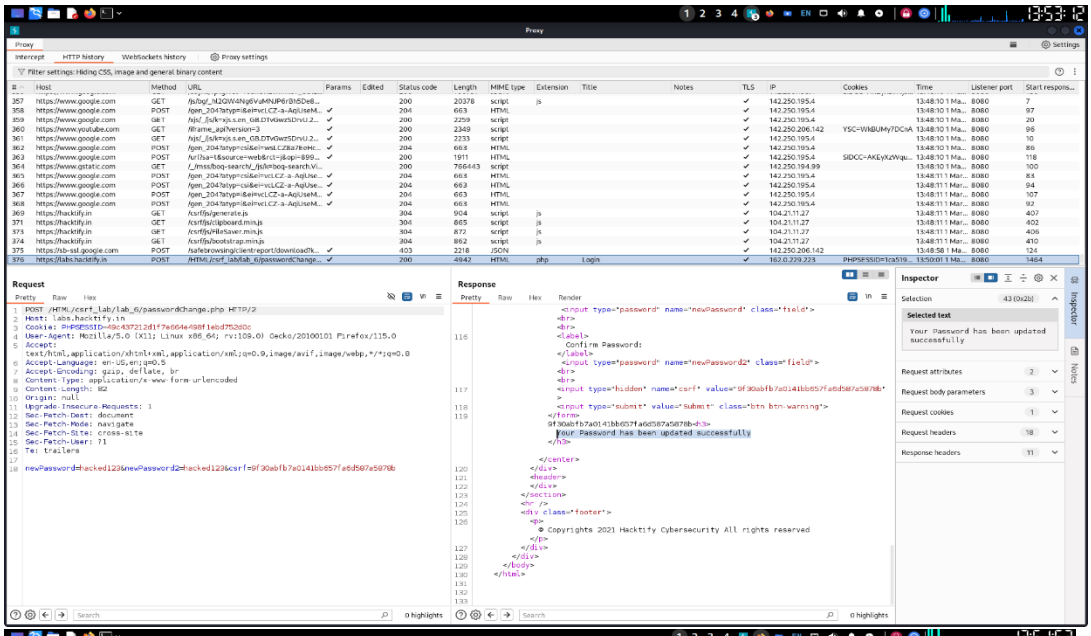
```
<form action="http://vulnerable-website.com/change-email" method="POST">
```

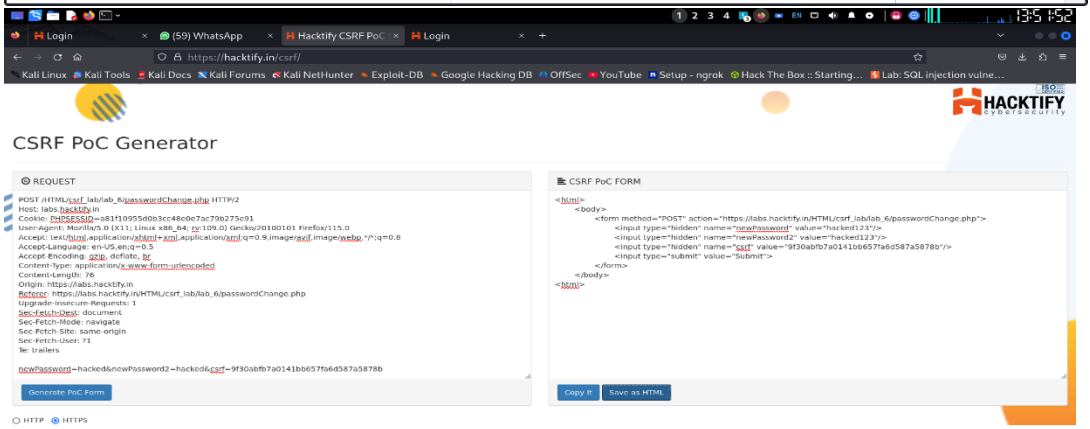
```
  <input type="hidden" name="email" value="attacker@example.com" />
  <input type="submit" value="Submit request" />
</form>
<script>
  document.forms[0].submit();
</script>
```

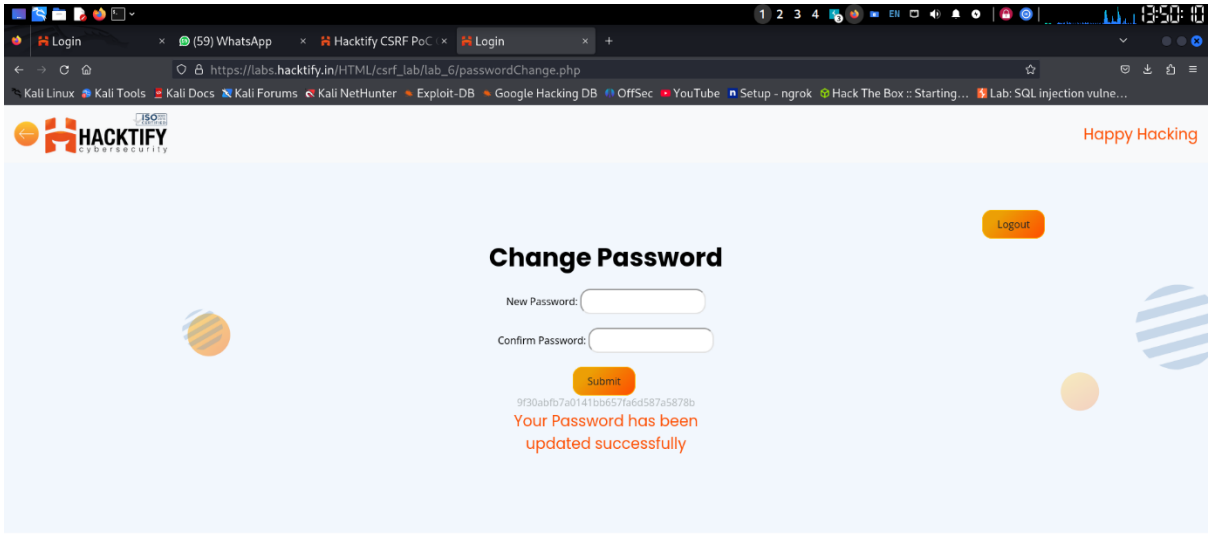
PoC HTML File: csrf-poc-1740817138709.html

Execution Proof:

Screenshots of the CSRF attack execution:







Severity: High

CSRF vulnerabilities can lead to severe security issues like account takeover, data manipulation, and loss of confidentiality. The risk increases when sensitive operations like changing user details, making transactions, or altering permissions are involved.

Lab5 Name: XSS – The Saviour

Vulnerability Type: Reflected Cross-Site Scripting (XSS)

Description:

The application is vulnerable to reflected XSS. User input passed via the URL is not properly sanitized and is directly rendered in the response, enabling the execution of arbitrary JavaScript code.

Steps to Reproduce:

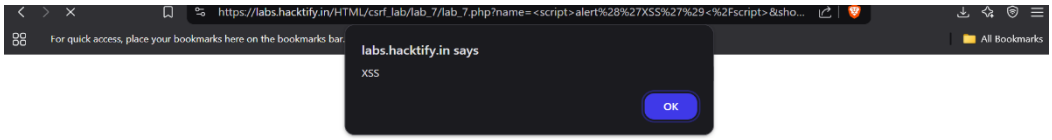
1. Navigate to the vulnerable URL after registering and logging in:
2. https://labs.hacktify.in/HTML/csrf_lab/lab_7/lab_7.php?name=<script>alert('XSS')</script>
3. Observe that the JavaScript executes, showing an alert box with the text 'XSS'.

Proof of Concept (PoC):

Payload used:

<script>alert('XSS')</script>

Execution Proof: (Screenshot attached) — The alert box demonstrating the XSS payload execution.



Impact:

An attacker can execute arbitrary JavaScript in the victim's browser, potentially leading to session hijacking, defacement, and theft of sensitive information.

Severity: High

CSRF Lab 6: rm -rf token

Vulnerability Description

In this lab, the Cross-Site Request Forgery (CSRF) vulnerability was found in the password update functionality. The application does not properly validate the CSRF token, making it possible for an attacker to forge requests on behalf of an authenticated user.

Steps to Reproduce

1. **Register/Login: Access the application and log in using valid credentials.**
2. **Capture Request: Navigate to the 'Change Password' page and capture the HTTP request using a proxy like Burp Suite.**
3. **Generate CSRF PoC: Use the CSRF PoC Generator to create an HTML form that mimics the password change request.**
4. **Exploit: Host the generated HTML form and trigger the request to change the user's password without their consent.**

Proof of Concept (PoC)

Captured Request:

POST /HTML/csrf_lab/lab_8/passwordChange.php HTTP/2

Host: labs.hacktify.in

Cookie: PHPSESSID=54748ef9d6a62c70ae145213b0cdfcf6

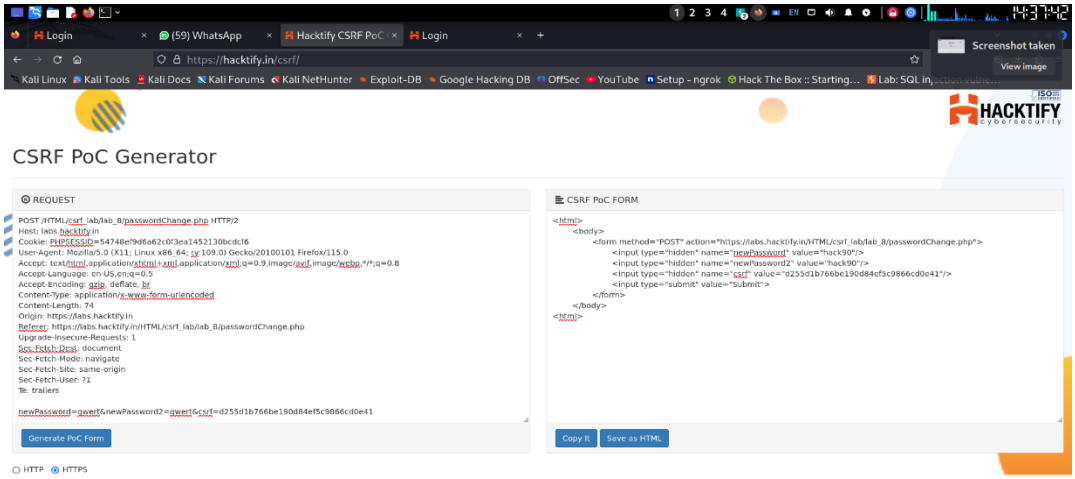
...

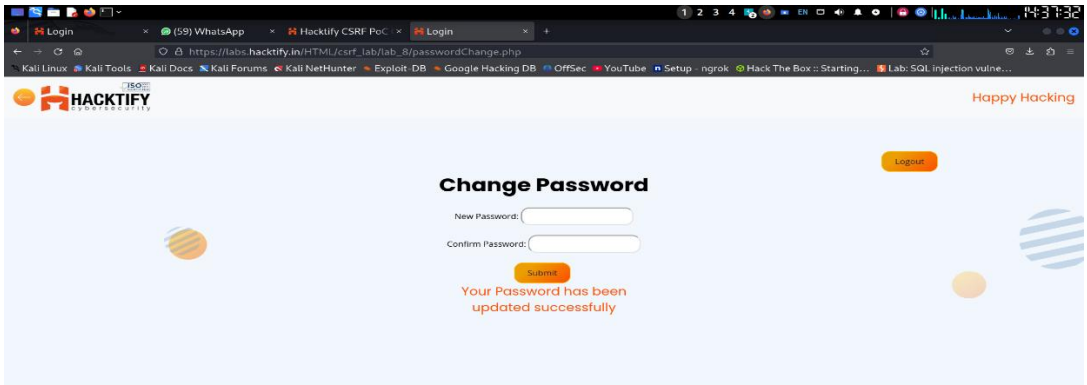
newPassword=guwert&newPassword2=guwert&csrf=d255d1b766eb190d84efc59866cd0e41

Generated CSRF Form:

```
<form method="POST"
action="https://labs.hacktify.in/HTML/csrf_lab/lab_8/passwordChange.php">
   <input type="hidden" name="newPassword" value="hack90"/>
   <input type="hidden" name="newPassword2" value="hack90"/>
   <input type="hidden" name="csrf" value="d255d1b766eb190d84efc59866cd0e41"/>
   <input type="submit" value="Submit"/>
</form>
```

Execution Proof:





Impact

An attacker can change the password of any logged-in user by tricking them into visiting a malicious site, potentially locking them out of their account.

Severity: High

Recommendation

- **Implement proper CSRF token validation.**
- **Ensure CSRF tokens are unique per request and tied to user sessions.**
- **Validate the 'Referer' or 'Origin' headers.**
- **Use SameSite cookies to mitigate CSRF attacks.**

CORS Labs Report

Lab 1: CORS With Arbitrary Origin

Introduction

This report covers the solution and findings for Lab 1: 'CORS With Arbitrary Origin' from the Hacktify CORS labs. The objective of this lab was to identify and exploit misconfigured Cross-Origin Resource Sharing (CORS) settings to access data from an unauthorized origin.

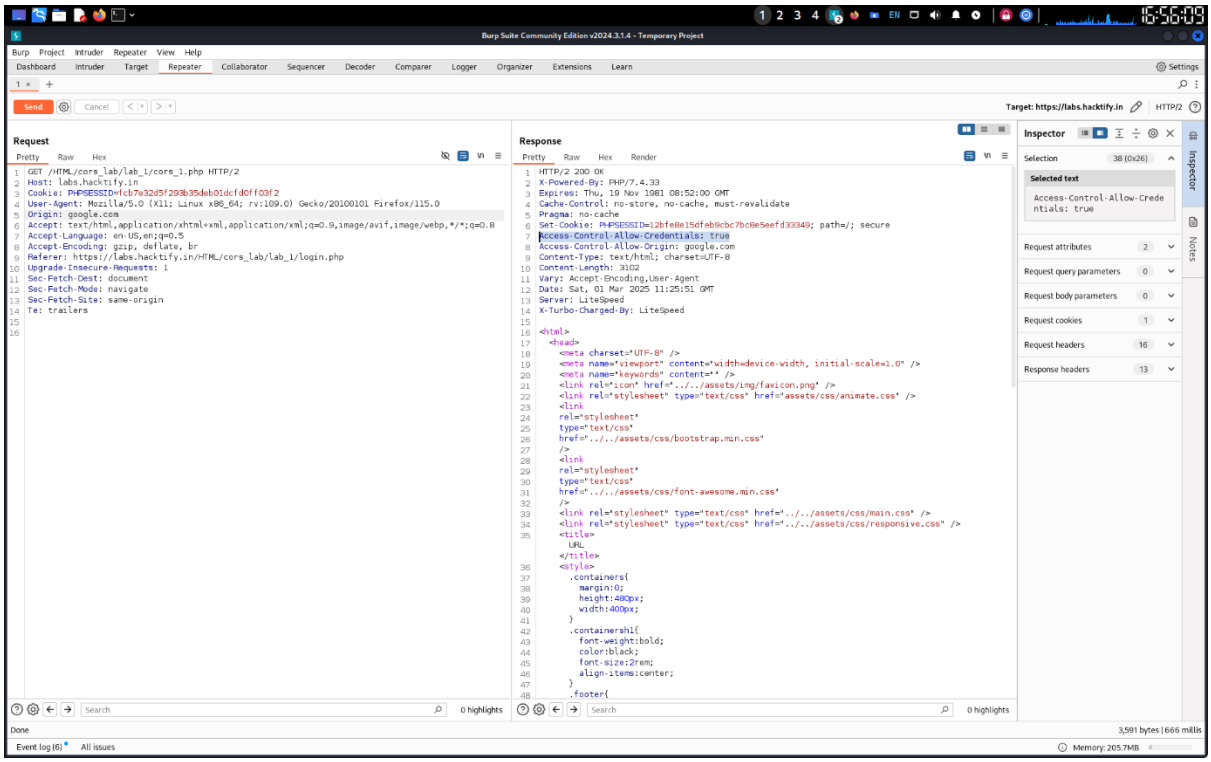
Vulnerability Description

The application accepts requests from any arbitrary origin and reflects the origin in the response without proper validation. This leads to a security misconfiguration that can be exploited to fetch sensitive information from the server.

Steps to Reproduce

1. Captured the login request using Burp Suite.
2. Sent the request to the Repeater tab.
3. Added the Origin header and set its value to https://google.com.
4. Sent the modified request.
5. Observed the response containing the header:
    - Access-Control-Allow-Origin: google.com
    - Access-Control-Allow-Credentials: true

Proof of Concept (PoC)



Impact

This misconfiguration allows an attacker to make authenticated requests from an unauthorized domain and access sensitive user data.

Severity

High

Lab 2: CORS With null Origin

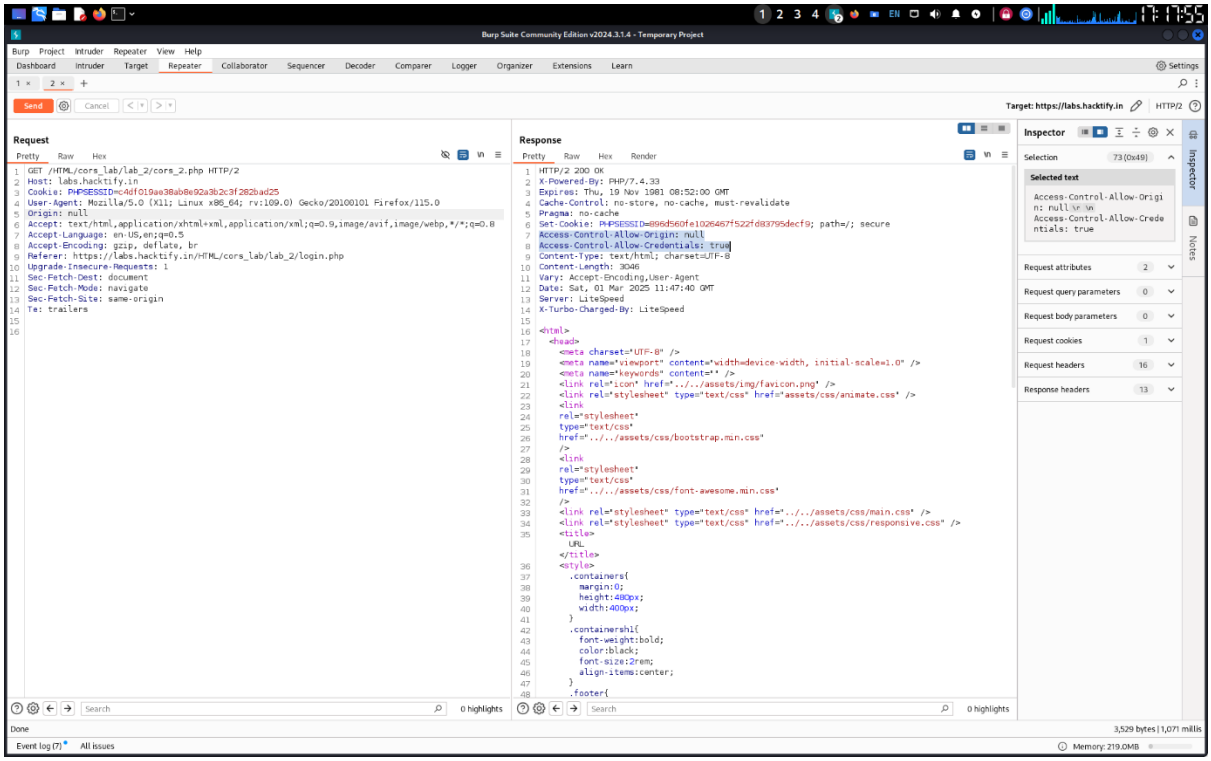
Introduction

This report covers the exploitation of CORS misconfiguration where the server accepts a null origin.

Vulnerability Details

The server accepts requests with a null origin, making it susceptible to exploitation via sandboxed environments.

Proof of Concept (PoC)



Request:

GET /HTML/cors_lab/lab_2/cors_2.php HTTP/2

Host: labs.hacktify.in

Origin: null

...

Response:

Access-Control-Allow-Origin: null

Access-Control-Allow-Credentials: true

...

Execution Proof

The response header includes Access-Control-Allow-Origin: null, confirming the misconfiguration.

Severity

Medium — While the null origin is less common, it can still enable attacks from sandboxed environments.

Lab3 Name: CORS with Prefix Match

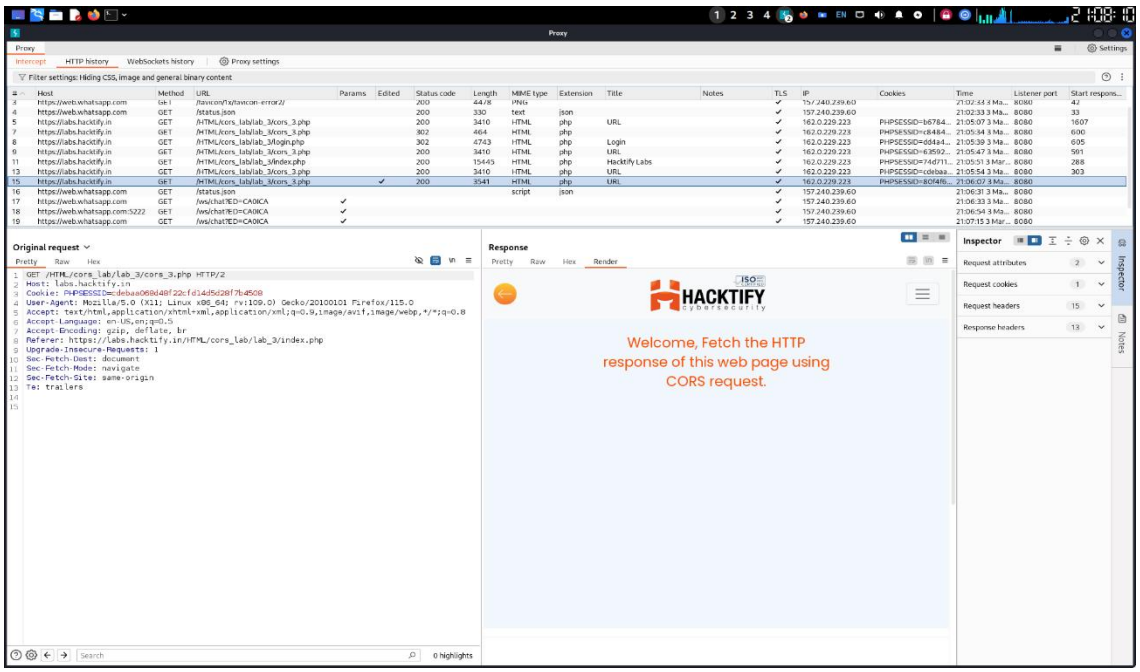
Vulnerability Description:

In this lab, the CORS implementation incorrectly allows any origin that starts with the trusted domain. This results in a security issue where an attacker can craft a malicious domain with a matching prefix and gain unauthorized access to sensitive information.

Steps to Reproduce:

1. **Intercept the Request:**
    - Open Burp Suite and turn Intercept ON.
    - Capture the request to HTML/cors_lab/lab_3/cors_3.php after logging in.
2. **Modify the Origin Header:**
    - Change the Origin header to: https://labs.hacktify.in.hacker.com
3. **Forward the Request:**
    - Forward the modified request and observe the response.
4. **Response Analysis:**
    - Notice the response contains the following headers:
    - Access-Control-Allow-Origin: https://labs.hacktify.in.hacker.com
    - Access-Control-Allow-Credentials: true
    - This confirms the CORS misconfiguration and the acceptance of an arbitrary origin that shares a prefix with the trusted domain.

Proof of Concept (PoC):



Impact:

- An attacker can exploit this misconfiguration to steal sensitive information such as session tokens, user data, and more by using a crafted malicious domain.

Severity: High

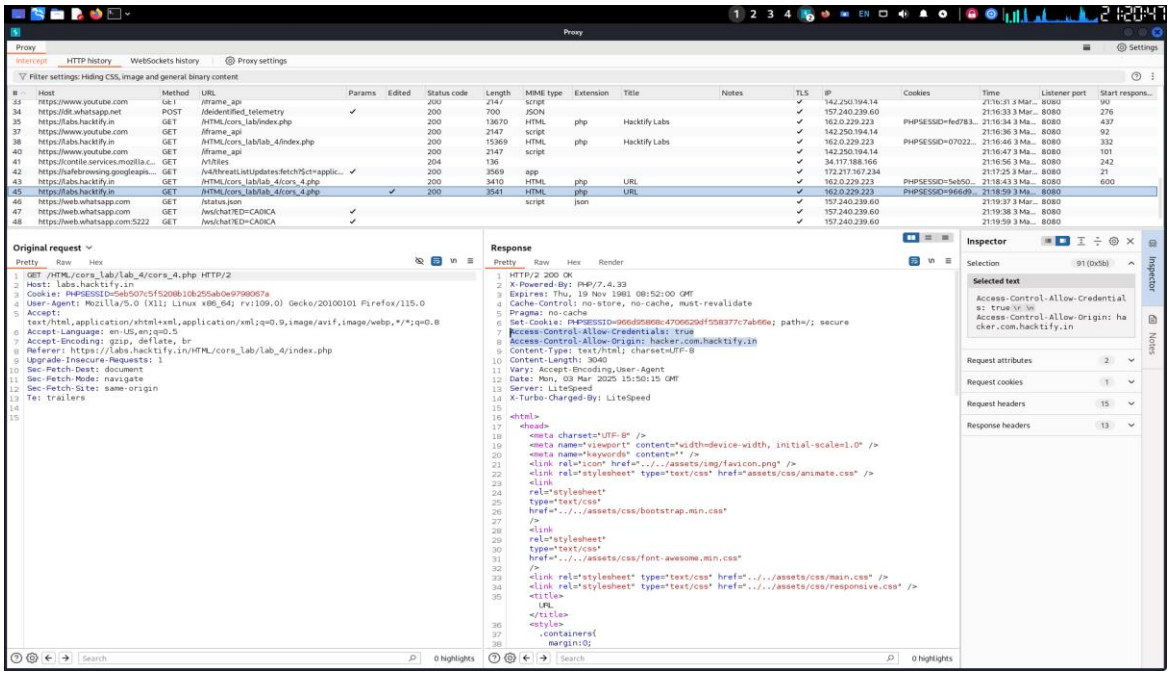
CORS Lab 4: CORS with Suffix Match

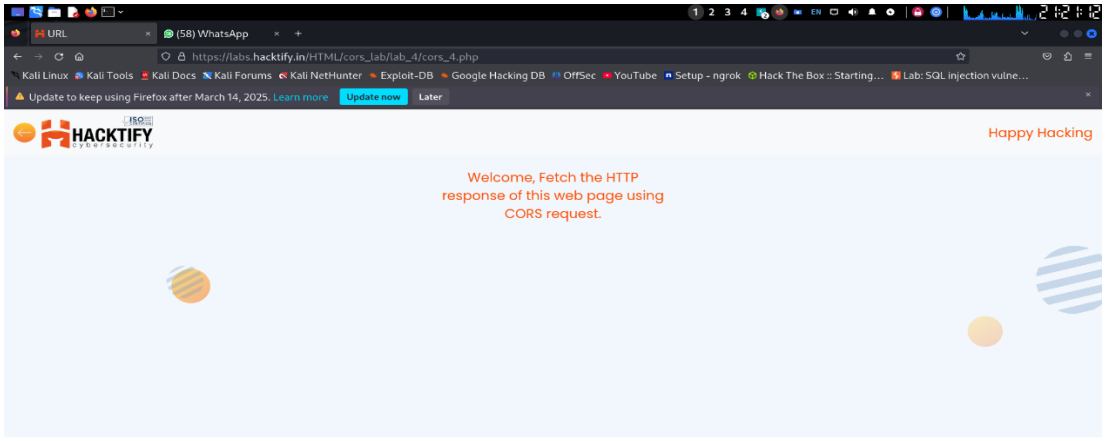
Vulnerability Description: In this lab, the CORS implementation is vulnerable because it uses a suffix match to validate the origin. This allows an attacker to craft an origin like hacker.com.hacktify.in, which bypasses security checks and gains unauthorized access to sensitive data.

Steps to Reproduce:

1. Open the lab URL: https://labs.hacktify.in/HTML/cors_lab/lab_4/cors_4.php
2. Capture the request in Burp Suite and observe the response headers.
3. Notice the Access-Control-Allow-Origin header accepts an origin with a suffix match, e.g., hacker.com.hacktify.in.
4. Confirm that Access-Control-Allow-Credentials is set to true, which allows cookies and session information to be sent.

Proof of Concept:





Impact: An attacker can create a malicious site with a carefully crafted origin to steal sensitive user data, including session cookies and personal information.

Severity: High

CORS Lab 5: CORS with Escape Dot

Vulnerability Name

CORS Misconfiguration - Escape Dot Bypass

Lab Name

CORS with Escape Dot

Description

The Cross-Origin Resource Sharing (CORS) policy implemented on the target application allows requests from an improperly validated origin. In this lab, the vulnerable origin was wwwhacktify.in (missing the dot between "www" and the domain). This indicates that the server accepts untrusted origins by matching non-standard domain patterns, leading to potential exploitation.

Impact

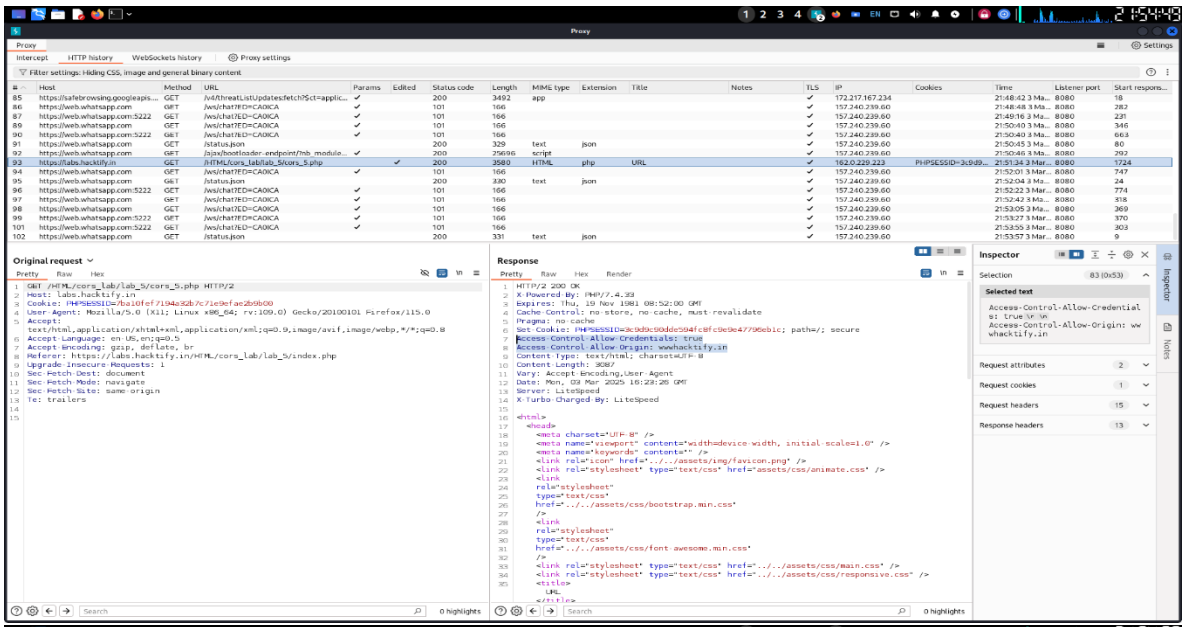
An attacker can craft a malicious website that tricks authenticated users into making requests to the vulnerable site. Since the server accepts the untrusted origin and sends sensitive data back, it can lead to:

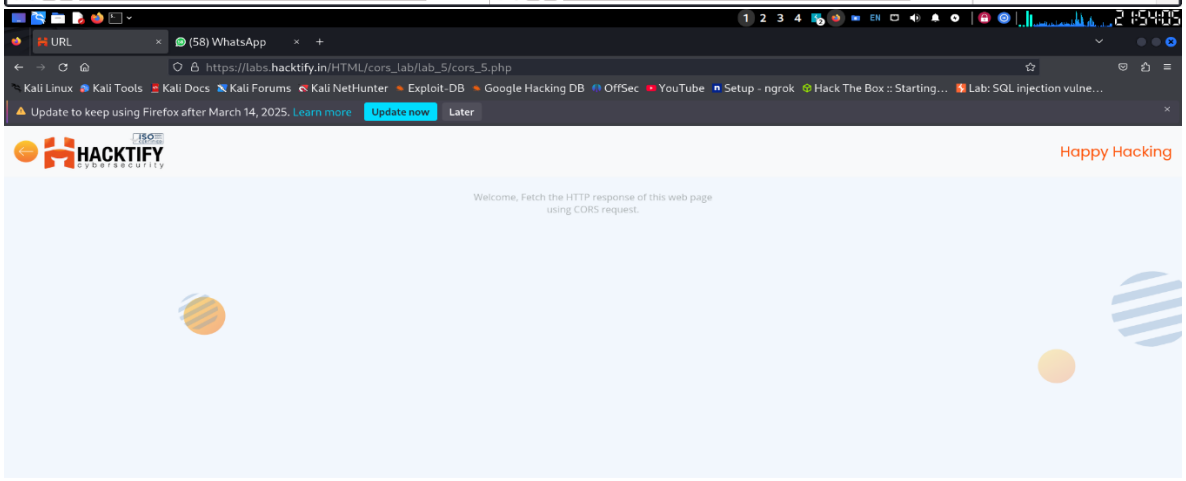
- Account Takeover
- Sensitive Data Exposure
- CSRF-like attacks with enhanced capabilities

Steps to Reproduce

1. Access the vulnerable lab endpoint: https://labs.hacktify.in/HTML/cors_lab/lab_5/cors_5.php
2. Observe the response headers and note the following:
   - Access-Control-Allow-Credentials: true
   - Access-Control-Allow-Origin: wwwhacktify.in (no dot between www and hacktify.in)
3. This demonstrates the acceptance of an arbitrary origin with improper validation.

Proof of Concept (PoC)





Severity

High

Recommendations

- Implement strict CORS policies by validating trusted origins properly.
- Avoid using wildcard or partial domain matching for CORS.
- Consider using an allowlist of trusted origins with exact matches.

CORS Lab 6: CORS with Substring Match

Vulnerability Description

The "CORS with Substring Match" vulnerability arises when a server's Access-Control-Allow-Origin policy uses overly broad string matching, allowing unauthorized origins that contain specific substrings to access sensitive resources. This misconfiguration can lead to unauthorized cross-origin requests and data leaks.

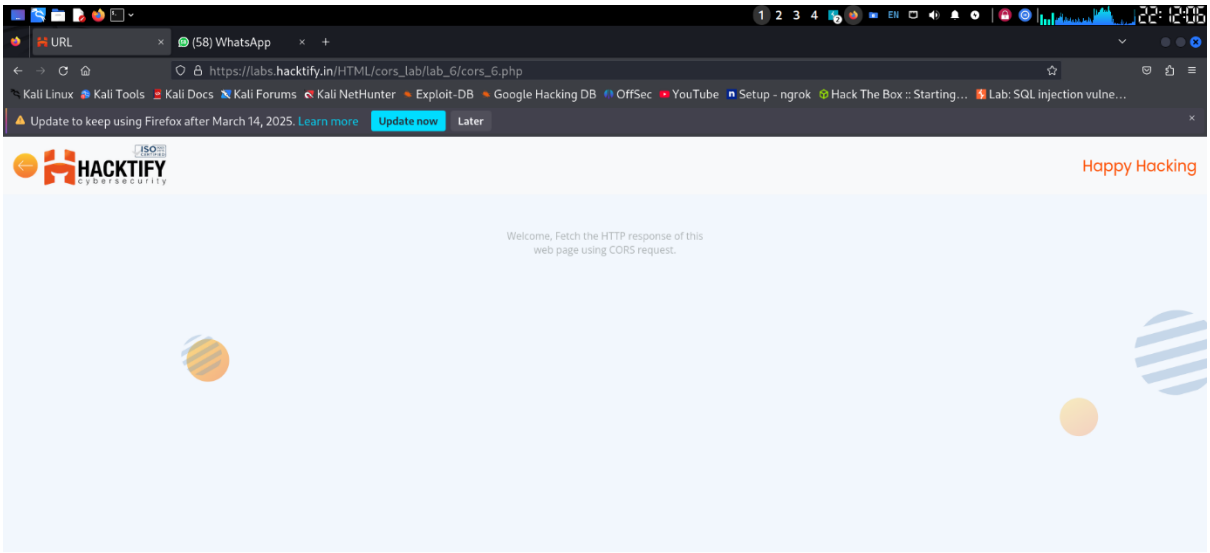
Impact

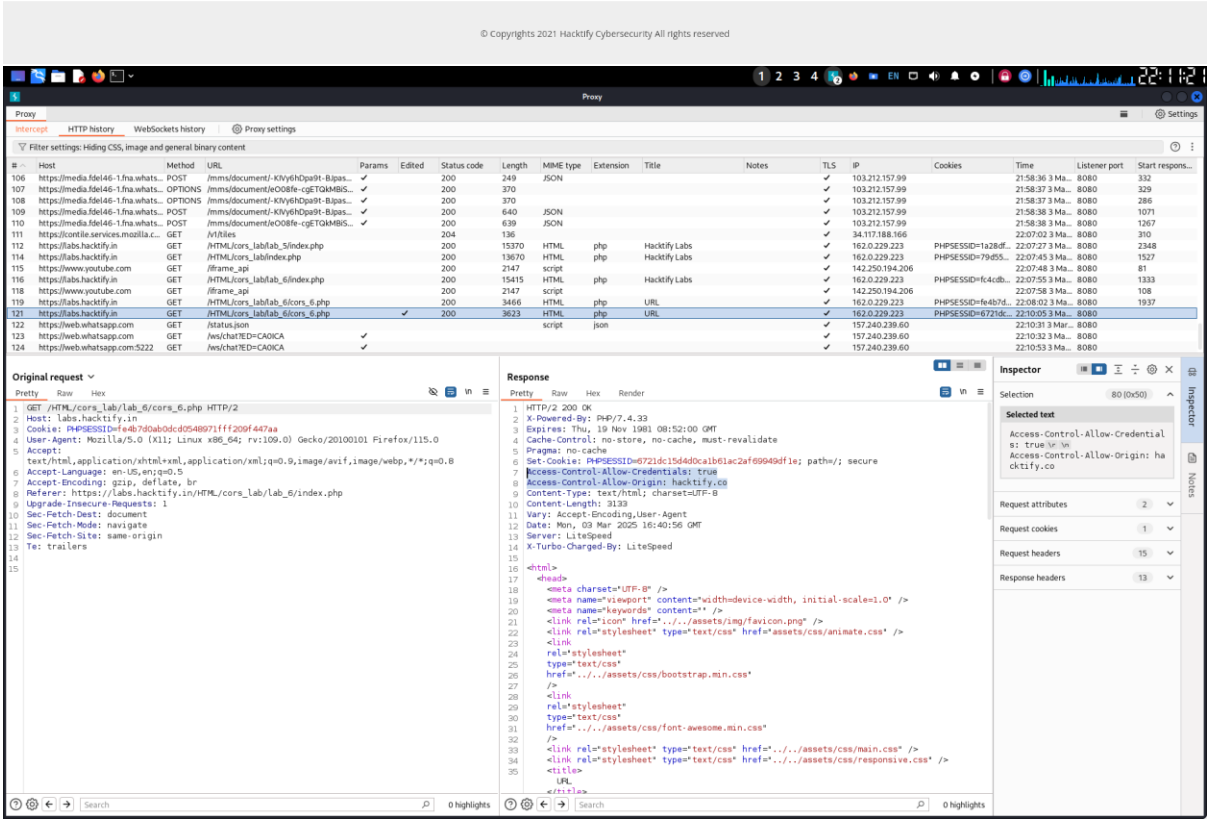
An attacker can exploit this by crafting a malicious website with an origin containing a specific substring that the server trusts. This results in unauthorized access to sensitive data and potential account takeover scenarios.

Steps to Reproduce

1. Navigate to the target lab URL: https://labs.hacktify.in/HTML/cors_lab/lab_6/cors_6.php
2. Intercept the request using Burp Suite.
3. Observe the response headers:
    - Access-Control-Allow-Origin: hacktify.co
    - Access-Control-Allow-Credentials: true
4. Confirm that the server is allowing any origin containing the substring hacktify.co, making it vulnerable to exploitation.

Proof of Concept (PoC)





Severity

High — This vulnerability can lead to unauthorized data access and potential account compromise.

Remediation

- Implement strict origin validation and avoid using substring matches in the Access-Control-Allow-Origin header.
- Consider using an allowlist of trusted domains and ensure proper CORS policy configurations.
- Disable Access-Control-Allow-Credentials: true unless strictly necessary.

Lab Name: CORS with Arbitrary Subdomain (Lab 7)

Vulnerability Description:

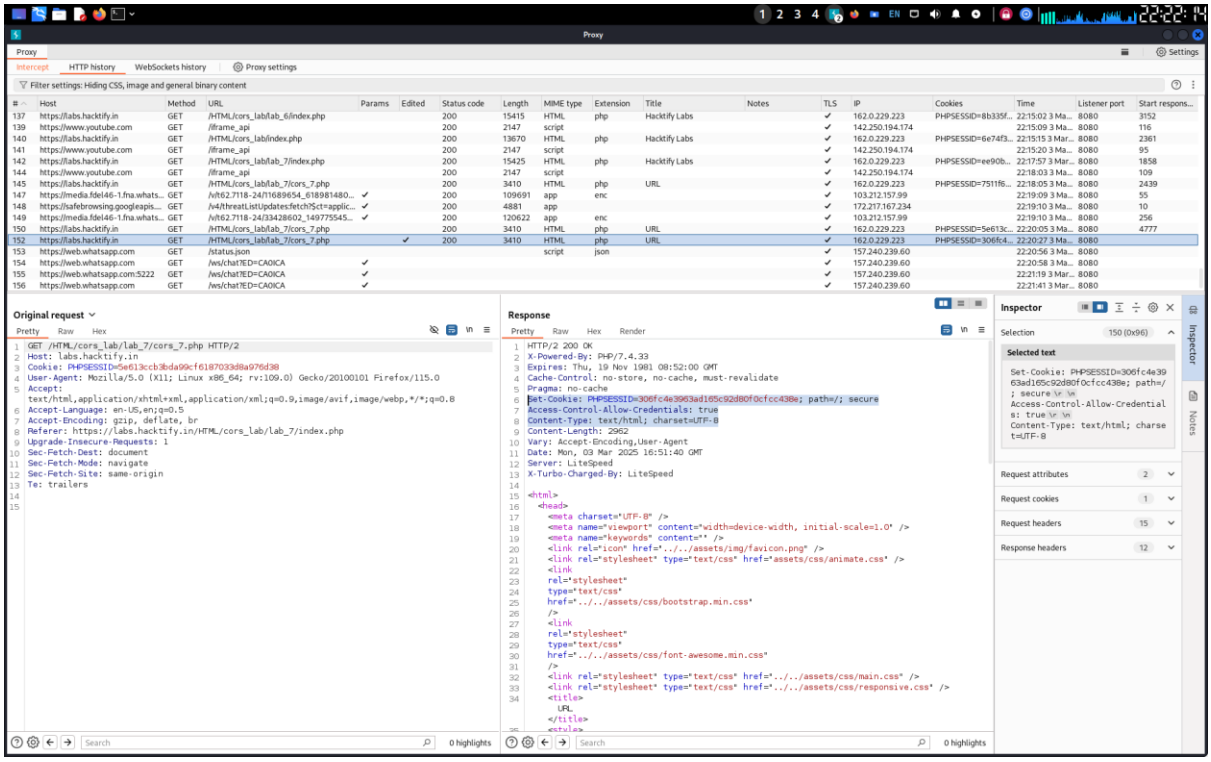
Cross-Origin Resource Sharing (CORS) with arbitrary subdomain vulnerability arises when a web application allows requests from any subdomain without proper validation. This can lead to unauthorized access to sensitive data by malicious subdomains.

Steps to Reproduce:

1. **Navigate to the Target URL:**
    - Access the lab environment at https://labs.hacktify.in/HTML/cors_lab/lab_7/index.php.
2. **Observe Request and Response:**
    - Using Burp Suite, capture the request to the target endpoint cors_7.php.
    - Note the response headers, particularly the Access-Control-Allow-Origin and Access-Control-Allow-Credentials.
3. **Identify the Vulnerability:**
    - Origin is set to http://hacker.hacktify.in (arbitrary subdomain) and is accepted.
    - Access-Control-Allow-Credentials: true is set, allowing credentialed requests from this arbitrary subdomain.

Proof of Concept (PoC)

Execution Proof:



Severity: High

The ability to exploit CORS with an arbitrary subdomain and access sensitive data across origins significantly increases the risk of session hijacking and data theft.