

# Week 3 Penetration Testing Report

**Full Name:** Chirag Suthar

**Program:** HCPT

**Date:** 03-03-2025

## Introduction

This report hereby describes the proceedings and results of a Black Box security assessment conducted against **Week 3 Labs**. The report hereby lists the findings and corresponding best practice mitigation actions and recommendations.

### 1. Objective

The objective of the assessment was to uncover vulnerabilities in the **Week 3 Labs** and provide a final security assessment report comprising vulnerabilities, remediation strategy, and recommendation guidelines to help mitigate the identified vulnerabilities and risks during the activity.

### 2. Scope

This section defines the scope and boundaries of the project.

<b>Application Name</b>	Cross Origin Resource Sharing, Cross-Site Request Forgery
-------------------------	---

### 3. Summary

Outlined is a Black Box Application Security assessment for **Week 3 Labs**.

**Total number of Sub-labs: 16 Sub-labs**

High	Medium	Low
5	4	4

**High** - Number of Sub-labs with hard difficulty level

**Medium** - Number of Sub-labs with medium difficulty level,

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

## 1. Cross Origin Resource Sharing (CORS)

### 1.1. CORS With Arbitrary Origin

Reference	Risk Rating
CORS With Arbitrary Origin	Low
<b>Tools Used</b>	
Google Chrome and Burp Suite	
<b>Vulnerability Description</b>	
Cross-origin resource sharing (CORS) is a browser mechanism which enables controlled access to resources located outside of a given domain. It extends and adds flexibility to the same-origin policy. However, it also provides potential for cross-domain based attacks, if a website's CORS policy is poorly configured and implemented. The CORS protocol uses some HTTP headers that define trusted web origins and associated properties such as whether authenticated access is permitted. Many modern websites use CORS to allow access from subdomains and trusted third parties. Sometimes because of mistakes of developers' attackers can use misconfiguration to exploit vulnerability.	
<b>How It Was Discovered</b>	
I found this vulnerability by manually writing Origin payload by intercepting it in the Burp Suite.	
<b>Vulnerable URLs</b>	
<a href="https://labs.hackify.in/HTML/cors_lab/lab_1/index.php">https://labs.hackify.in/HTML/cors_lab/lab_1/index.php</a>	
<b>Consequences of not Fixing the Issue</b>	
Attackers would treat many victims to visit the attacker's website, if victim is logged in, then his personal information is recorded on attacker's server. Attacker can perform any action in the user's account, bypassing CSRF tokens.	
<b>Suggested Countermeasures</b>	
<ol style="list-style-type: none"><li><b>Proper configuration of cross-domain requests</b> : If a web resource contains sensitive information, the origin should be properly specified in the Access-Control-Allow-Origin header.</li><li><b>Only allow trusted sites</b> : Dynamically reflecting origins from cross-domain requests without validation is readily exploitable and should be avoided.</li><li><b>Avoid whitelisting null</b> : Avoid using the header Access-Control-Allow-Origin: null . Cross-domain resource calls from internal documents and sandboxed requests can specify the null origin. CORS headers should be properly defined in respect of trusted origins for private and public servers.</li><li><b>Avoid wildcards in internal networks</b> : Avoid using wildcards in internal networks. Trusting network configuration alone to protect internal resources is not sufficient when internal browsers can access untrusted external domains.</li></ol>	
<b>References</b>	
<ol style="list-style-type: none"><li><a href="https://portswigger.net/web-security/cors">https://portswigger.net/web-security/cors</a></li><li><a href="https://owasp.org/www-community/attacks/CORS_OriginHeaderScrutiny">https://owasp.org/www-community/attacks/CORS_OriginHeaderScrutiny</a></li><li><a href="https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS">https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS</a></li></ol>	
<b>Payload Used</b>	
origin:attacker.com	

## Proof of Concept

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

The screenshot shows the Burp Suite Community Edition interface with the following details:

**Request:**

```
1 GET /HML/cors_lab/lab_1/cors_1.php HTTP/1.1
2 Host: labs.hackify.in
3 Origin: attacker.com
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:78.0) Gecko/20100101 Firefox/78.0
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
8 Referer: http://labs.hackify.in/HML/cors_lab/lab_1/index.php
9 DNT: 1
10 Connection: close
11 Cookie: PHPSESSID=de7eaa4f365b273bf4ce97905c21085
12 Upgrade-Insecure-Requests: 1
13 Sec-GRC: 
14 Cache-Control: max-age=0
15
16
```

**Response:**

```
1 HTTP/1.1 200 OK
2 keep-alive: timeout=5, max=100
3 x-powered-by: PHP/7.4.33
4 expires: Thu, 19 Nov 1988 08:52:00 GMT
5 cache-control: no-store, no-cache, must-revalidate
6 pragma: no-cache
7 set-cookie: PHPSESSID=de7eaa4f365b273bf4ce97905c21085
8 access-control-allow-credentials: true
9 access-control-allow-origin: attacker.com
10 content-type: text/html; charset=UTF-8
11 Content-Length: 3109
12 vary: Accept-Encoding,User-Agent
13 date: Sat, 01 Mar 2025 17:20:55 GMT
14 server: LiteSpeed
15 x-turbo-charged-by: LiteSpeed
16 connection: close
17
18 <html>
19 <head>
20 <meta charset="UTF-8" />
21 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
22 <meta name="keywords" content="" />
23 <link rel="icon" href=".../assets/img/favicon.png" />
24 <link rel="stylesheet" type="text/css" href="assets/css/animate.css" />
25 <link rel="stylesheet" type="text/css" href=".../assets/css/bootstrap.min.css" />
26 <link rel="stylesheet" type="text/css" href=".../assets/css/font-awesome.min.css" />
27 <link rel="stylesheet" type="text/css" href=".../assets/css/main.css" />
28 <link rel="stylesheet" type="text/css" href=".../assets/css/responsive.css" />
29 <title>
30 <url>
31 </title>
32 <div>
33 .containers{
34   margin:0;
35   height:480px;
36   width:400px;
37 }
```

**Bottom Status Bar:**

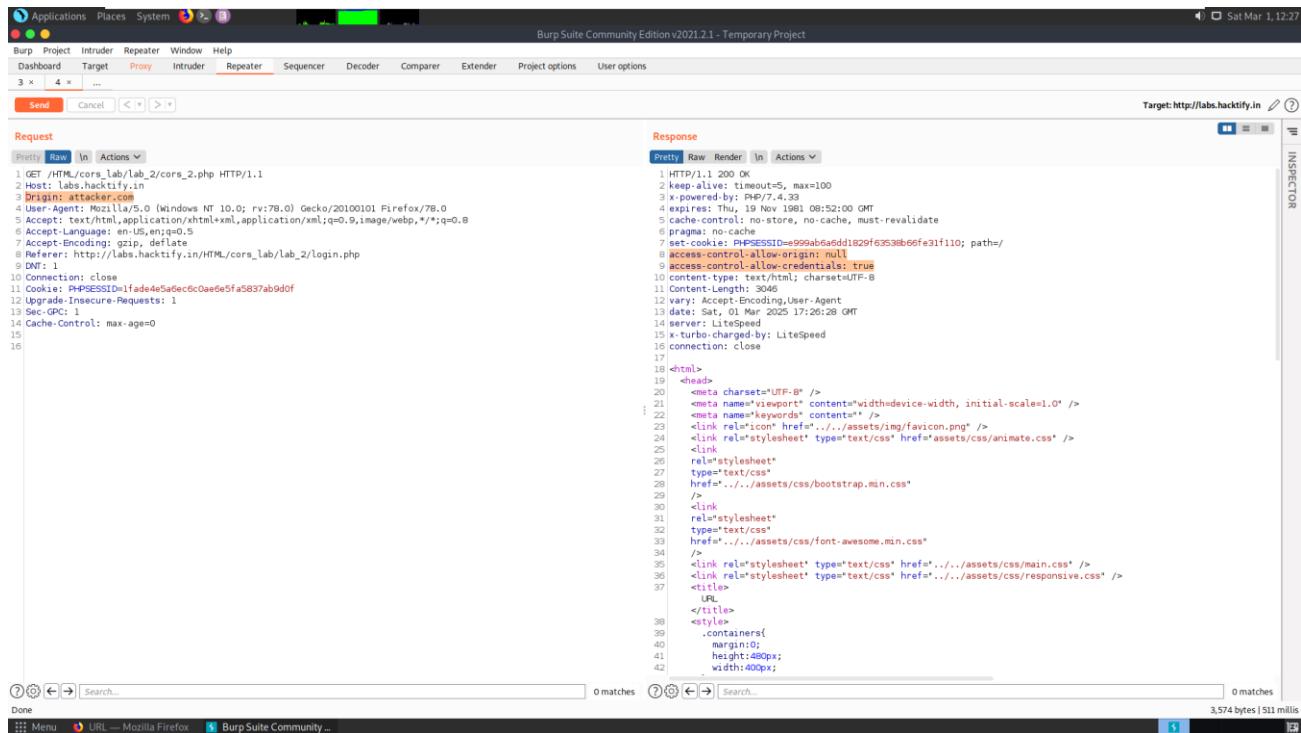
Done URL — Mozilla Firefox Burp Suite Community ... 0 matches 3,638 bytes | 943 millis

## 1.2. CORS with Null Origin

Reference	Risk Rating
CORS With Null Origin	Low
Tools Used	
Google Chrome and Burp Suite	
Vulnerability Description	
<p>Cross-origin resource sharing (CORS) is a browser mechanism which enables controlled access to resources located outside of a given domain. It extends and adds flexibility to the same-origin policy. However, it also provides potential for cross-domain based attacks, if a website's CORS policy is poorly configured and implemented. The CORS protocol uses some HTTP headers that define trusted web origins and associated properties such as whether authenticated access is permitted. Many modern websites use CORS to allow access from subdomains and trusted third parties. Sometimes because of mistakes of developers' attackers can use misconfiguration to exploit vulnerability.</p>	
How It Was Discovered	
I found this vulnerability by manually writing Origin payload by intercepting it in the Burp Suite.	
Vulnerable URLs	
<a href="https://labs.hackify.in/HTML/cors_lab/lab_2/index.php">https://labs.hackify.in/HTML/cors_lab/lab_2/index.php</a>	
Consequences of not Fixing the Issue	
<p>Attackers would treat many victims to visit the attacker's website, if victim is logged in, then his personal information is recorded on attacker's server. Attacker can perform any action in the user's account, bypassing CSRF tokens.</p>	
Suggested Countermeasures	
<ol style="list-style-type: none"><li><b>Proper configuration of cross-domain requests :</b> If a web resource contains sensitive information, the origin should be properly specified in the Access-Control-Allow-Origin header.</li><li><b>Only allow trusted sites :</b> Dynamically reflecting origins from cross-domain requests without validation is readily exploitable and should be avoided.</li><li><b>Avoid whitelisting null :</b> Avoid using the header Access-Control-Allow-Origin: null . Cross-domain resource calls from internal documents and sandboxed requests can specify the null origin. CORS headers should be properly defined in respect of trusted origins for private and public servers.</li><li><b>Avoid wildcards in internal networks :</b> Avoid using wildcards in internal networks. Trusting network configuration alone to protect internal resources is not sufficient when internal browsers can access untrusted external domains.</li></ol>	
References	
<ol style="list-style-type: none"><li><a href="https://portswigger.net/web-security/cors">https://portswigger.net/web-security/cors</a></li><li><a href="https://owasp.org/www-community/attacks/CORS_OriginHeaderScrutiny">https://owasp.org/www-community/attacks/CORS_OriginHeaderScrutiny</a></li><li><a href="https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS">https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS</a></li></ol>	
Payload Used	
origin:attacker.com	

## Proof of Concept

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



The screenshot shows the Burp Suite Community Edition interface. The top bar displays "Burm Suite Community Edition v2021.2.1 - Temporary Project" and the date "Sat Mar 1, 12:27". The menu bar includes "Applications", "Places", "System", "File", "Edit", "Project", "Intruder", "Repeater", "Sequencer", "Decoder", "Comparer", "Extender", "Project options", and "User options". Below the menu is a toolbar with "Send", "Cancel", and navigation icons. The main window is divided into two panes: "Request" and "Response". The "Request" pane shows a raw HTTP message with several headers and a cookie. The "Response" pane displays the captured HTML response, which includes meta tags, links to CSS files like "animate.css", "bootstrap.min.css", and "font-awesome.min.css", and a script tag for "main.js". The bottom status bar shows "Done", "URL — Mozilla Firefox", and "Burp Suite Community ...".

```
1 GET /cors_lab/lab_2/cors_2.php HTTP/1.1
2 Host: labs.hackify.in
3 Origin: attacker.com
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:78.0) Gecko/20100101 Firefox/78.0
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
8 Referer: http://labs.hackify.in/HML/cors_lab/lab_2/login.php
9 DNT: 1
10 Connection: close
11 Cookie: PHPSESSID=1faade45a6ec6c0ae6e5fa5837ab9d0f
12 Upgrade-Insecure-Requests: 1
13 Strict-GRC: 1
14 Cache-Control: max-age=0
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
```

### 1.3. CORS with prefix match

Reference	Risk Rating
CORS With prefix match	Medium
Tools Used	
Google Chrome and Burp Suite	
Vulnerability Description	
<p>Cross-origin resource sharing (CORS) is a browser mechanism which enables controlled access to resources located outside of a given domain. It extends and adds flexibility to the same-origin policy. However, it also provides potential for cross-domain based attacks, if a website's CORS policy is poorly configured and implemented. The CORS protocol uses some HTTP headers that define trusted web origins and associated properties such as whether authenticated access is permitted. Many modern websites use CORS to allow access from subdomains and trusted third parties. Sometimes because of mistakes of developers' attackers can use misconfiguration to exploit vulnerability.</p>	
How It Was Discovered	
I found this vulnerability by manually writing Origin payload by intercepting it in the Burp Suite.	
Vulnerable URLs	
<a href="https://labs.hacktify.in/HTML/cors_lab/lab_3/index.php">https://labs.hacktify.in/HTML/cors_lab/lab_3/index.php</a>	
Consequences of not Fixing the Issue	
<p>Attackers would treat many victims to visit the attacker's website, if victim is logged in, then his personal information is recorded on attacker's server. Attacker can perform any action in the user's account, bypassing CSRF tokens.</p>	
Suggested Countermeasures	
<ol style="list-style-type: none"><li><b>Proper configuration of cross-domain requests</b> : If a web resource contains sensitive information, the origin should be properly specified in the Access-Control-Allow-Origin header.</li><li><b>Only allow trusted sites</b> : Dynamically reflecting origins from cross-domain requests without validation is readily exploitable and should be voided.</li><li><b>Avoid whitelisting null</b> : Avoid using the header Access-Control-Allow-Origin: null . Cross-domain resource calls from internal documents and sandboxed requests can specify the null origin. CORS headers should be properly defined in respect of trusted origins for private and public servers.</li><li><b>Avoid wildcards in internal networks</b> : Avoid using wildcards in internal networks. Trusting network configuration alone to protect internal resources is not sufficient when internal browsers can access untrusted external domains.</li></ol>	
References	
<ol style="list-style-type: none"><li><a href="https://portswigger.net/web-security/cors">https://portswigger.net/web-security/cors</a></li><li><a href="https://owasp.org/www-community/attacks/CORS_OriginHeaderScrutiny">https://owasp.org/www-community/attacks/CORS_OriginHeaderScrutiny</a></li><li><a href="https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS">https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS</a></li></ol>	
Payload Used	
origin:hacktify.in,attacker.com	

## Proof of Concept

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

The screenshot shows the Burp Suite Community Edition interface. The 'Proxy' tab is selected. A request is shown in the 'Request' pane:

```
1 GET /cors_lab/lab_3/cors_3.php HTTP/1.1
2 Host: labs.hackify.in
3 Origin: hackify.in,attacker.com
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:78.0) Gecko/20100101 Firefox/78.0
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
8 Referer: http://labs.hackify.in/HML/cors_lab/lab_3/login.php
9 DNT: 1
10 Connection: close
11 Cookie: PHPSESSID=a4dd9c43e4f46db7dc2b855af1944479
12 Upgrade-Insecure-Requests: 1
13 Sec-GRC: 
14 Cache-Control: max-age=0
15
16
```

The response is shown in the 'Response' pane:

```
1 HTTP/1.1 200 OK
2 keep-alive: timeout=5, max=100
3 x-powered-by: PHP/7.4.33
4 expires: Thu, 19 Nov 1981 08:52:00 GMT
5 cache-control: no-store, no-cache, must-revalidate
6 pragma: no-cache
7 set-cookie: PHPSESSID=565b20adcb81fb49a1697cb7be7907a6; path=/
8 access-control-allow-credentials: true
9 access-control-allow-origin: hackify.in,attacker.com
10 content-type: text/html; charset=UTF-8
11 Content-Length: 3047
12 vary: Accept-Encoding,User-Agent
13 date: Sat, 01 Mar 2025 17:30:30 GMT
14 server: LiteSpeed
15 x-turbo-charged-by: LiteSpeed
16 connection: close
17
18 <html>
19   <head>
20     <meta charset="UTF-8" />
21     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
22     <meta name="keywords" content="" />
23     <link rel="icon" href="../../assets/img/favicon.png" />
24     <link rel="stylesheet" type="text/css" href="assets/css/animate.css" />
25     <link rel="stylesheet" type="text/css" href="assets/css/bootstrap.min.css" />
26     <link rel="stylesheet" type="text/css" href="assets/css/font-awesome.min.css" />
27     <link rel="stylesheet" type="text/css" href="assets/css/main.css" />
28     <link rel="stylesheet" type="text/css" href="assets/css/responsive.css" />
29   </head>
30   <body>
31     <div class="container" style="margin: 0; height: 500px; width: 400px;">
32       <div style="border: 1px solid black; padding: 10px; text-align: center; font-size: 14px; color: #333; background-color: #f9f9f9; border-radius: 5px; margin-bottom: 10px;">
33         <h3>Welcome to the CORS Lab!</h3>
34         <p>This is a simple CORS configuration test page. You can see that the response is being correctly handled by the server, even though the request was made from a different origin.</p>
35         <a href="#" style="color: #007bff; text-decoration: none; font-weight: bold; font-size: 1em; margin-top: 10px;">Go Back</a>
36       </div>
37       <div style="border: 1px solid black; padding: 10px; text-align: center; font-size: 14px; color: #333; background-color: #f9f9f9; border-radius: 5px; margin-bottom: 10px;">
38         <h3>CORS Configuration Test</h3>
39         <p>The CORS configuration is working correctly. The response is being properly handled and returned to the client, even though it originated from a different domain. This demonstrates that the server is correctly setting the appropriate CORS headers (Access-Control-Allow-Origin, Access-Control-Allow-Credentials, etc.) to allow requests from the specified origin.</p>
40         <a href="#" style="color: #007bff; text-decoration: none; font-weight: bold; font-size: 1em; margin-top: 10px;">Go Back</a>
41     </div>
42   </body>
43 </html>
```

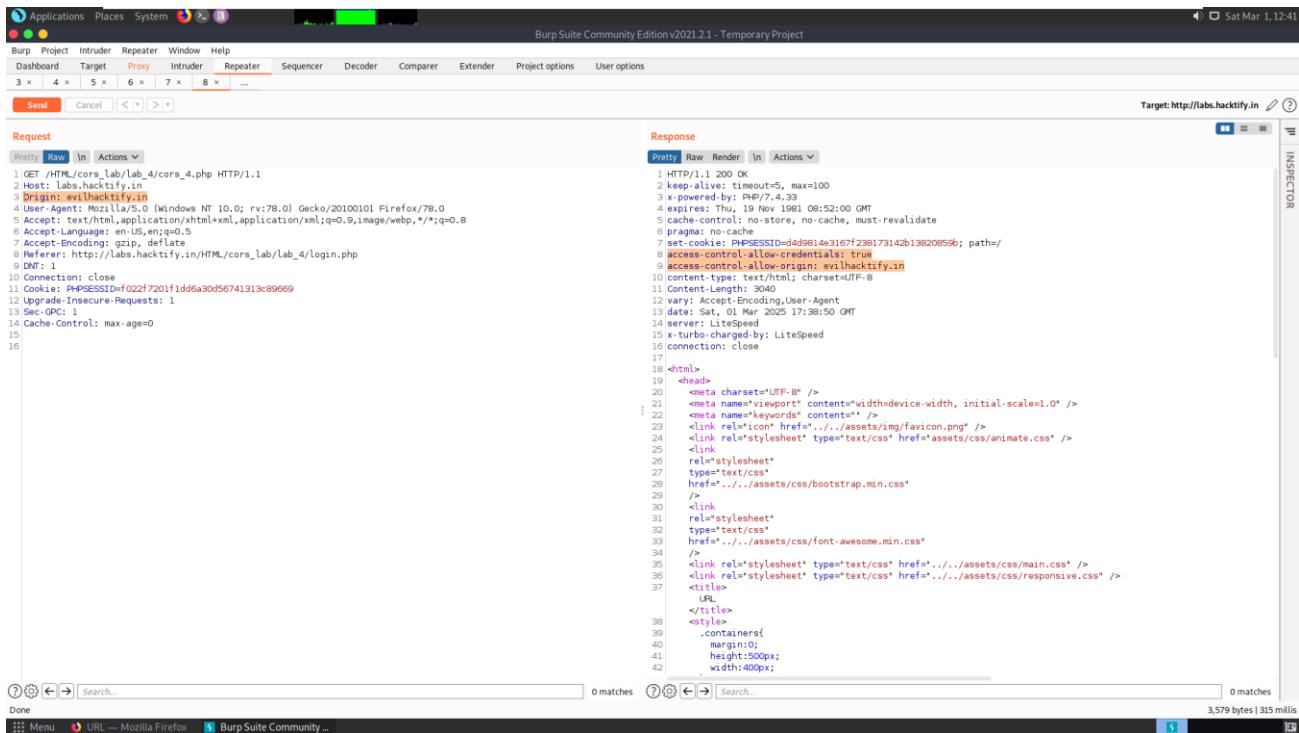
At the bottom, there are search bars for 'Search...' and 'URL'.

## 1.4. CORS with suffix match

Reference	Risk Rating
CORS With suffix match	Medium
Tools Used	
Google Chrome and Burp Suite	
Vulnerability Description	
<p>Cross-origin resource sharing (CORS) is a browser mechanism which enables controlled access to resources located outside of a given domain. It extends and adds flexibility to the same-origin policy. However, it also provides potential for cross-domain based attacks, if a website's CORS policy is poorly configured and implemented. The CORS protocol uses some HTTP headers that define trusted web origins and associated properties such as whether authenticated access is permitted. Many modern websites use CORS to allow access from subdomains and trusted third parties. Sometimes because of mistakes of developers' attackers can use misconfiguration to exploit vulnerability.</p>	
How It Was Discovered	
I found this vulnerability by manually writing Origin payload by intercepting it in the Burp Suite.	
Vulnerable URLs	
<a href="https://labs.hackify.in/HTML/cors_lab/lab_4/index.php">https://labs.hackify.in/HTML/cors_lab/lab_4/index.php</a>	
Consequences of not Fixing the Issue	
<p>Attackers would treat many victims to visit the attacker's website, if victim is logged in, then his personal information is recorded on attacker's server. Attacker can perform any action in the user's account, bypassing CSRF tokens.</p>	
Suggested Countermeasures	
<ol style="list-style-type: none"><li><b>Proper configuration of cross-domain requests :</b> If a web resource contains sensitive information, the origin should be properly specified in the Access-Control-Allow-Origin header.</li><li><b>Only allow trusted sites :</b> Dynamically reflecting origins from cross-domain requests without validation is readily exploitable and should be avoided.</li><li><b>Avoid whitelisting null :</b> Avoid using the header Access-Control-Allow-Origin: null . Cross-domain resource calls from internal documents and sandboxed requests can specify the null origin. CORS headers should be properly defined in respect of trusted origins for private and public servers.</li><li><b>Avoid wildcards in internal networks :</b> Avoid using wildcards in internal networks. Trusting network configuration alone to protect internal resources is not sufficient when internal browsers can access untrusted external domains.</li></ol>	
References	
<ol style="list-style-type: none"><li><a href="https://portswigger.net/web-security/cors">https://portswigger.net/web-security/cors</a></li><li><a href="https://owasp.org/www-community/attacks/CORS_OriginHeaderScrutiny">https://owasp.org/www-community/attacks/CORS_OriginHeaderScrutiny</a></li><li><a href="https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS">https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS</a></li></ol>	
Payload Used	
origin:evilhackify.com	

## Proof of Concept

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



The screenshot shows the Burp Suite Community Edition interface. The 'Repeater' tab is selected. The 'Request' pane shows a captured GET request to `/HML/cors_lab/lab_4/cors_4.php`. The 'Response' pane displays the received HTML content, which includes several `<link rel="stylesheet"` tags pointing to external assets like `animate.css`, `bootstrap.min.css`, and `font-awesome.min.css`. The status bar at the bottom indicates the target is `http://labs.hackify.in`.

```
1 GET /HML/cors_lab/lab_4/cors_4.php HTTP/1.1
2 Host: labs.hackify.in
3 Origin: evilhackify.in
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:78.0) Gecko/20100101 Firefox/78.0
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
8 Referer: http://labs.hackify.in/HML/cors_lab/lab_4/login.php
9 DNT: 1
10 Connection: close
11 Cookie: PHPSESSID=f022f7201f1dd6a30d56741313c89669
12 Upgrade-Insecure-Requests: 1
13 Sec-GRC: 1
14 Cache-Control: max-age=0
15
16
17
18 <html>
19   <head>
20     <meta charset="UTF-8" />
21     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
22     <meta name="keywords" content="" />
23     <link rel="icon" href="../../assets/img/favicon.png" />
24     <link rel="stylesheet" type="text/css" href="assets/css/animate.css" />
25     <link rel="stylesheet" type="text/css" href="assets/css/bootstrap.min.css" />
26     <link rel="stylesheet" type="text/css" href="assets/css/font-awesome.min.css" />
27     <link rel="stylesheet" type="text/css" href="assets/css/main.css" />
28     <link rel="stylesheet" type="text/css" href="assets/css/responsive.css" />
29   </head>
30   <body>
31     <div class="container" style="text-align: center; margin-top: 100px;">
32       <h1>Welcome to the CORS Lab!</h1>
33       <p>This is a simple CORS Lab for testing Cross-Origin Resource Sharing (CORS).</p>
34       <p>You can test your browser's CORS implementation by sending requests to this server from different origins.</p>
35       <p>For example, you can try sending a request to this endpoint from a different domain or port.</p>
36       <p>You can also test specific CORS headers by modifying the request headers in the Repeater tab.</p>
37     </div>
38   </body>
39 </html>
```

## 1.5. CORS with Escape dot

Reference	Risk Rating
CORS with Escape dot	High
Tools Used	
Google Chrome and Burp Suite	
Vulnerability Description	
Cross-origin resource sharing (CORS) is a browser mechanism which enables controlled access to resources located outside of a given domain. It extends and adds flexibility to the same-origin policy. However, it also provides potential for cross-domain based attacks, if a website's CORS policy is poorly configured and implemented. The CORS protocol uses some HTTP headers that define trusted web origins and associated properties such as whether authenticated access is permitted. Many modern websites use CORS to allow access from subdomains and trusted third parties. Sometimes because of mistakes of developers' attackers can use misconfiguration to exploit vulnerability.	
How It Was Discovered	
I found this vulnerability by manually writing Origin payload by intercepting it in the Burp Suite.	
Vulnerable URLs	
<a href="https://labs.hackify.in/HTML/cors_lab/lab_5/index.php">https://labs.hackify.in/HTML/cors_lab/lab_5/index.php</a>	
Consequences of not Fixing the Issue	
Attackers would treat many victims to visit the attacker's website, if victim is logged in, then his personal information is recorded on attacker's server. Attacker can perform any action in the user's account, bypassing CSRF tokens.	
Suggested Countermeasures	
<ol style="list-style-type: none"><li><b>Proper configuration of cross-domain requests</b> : If a web resource contains sensitive information, the origin should be properly specified in the Access-Control-Allow-Origin header.</li><li><b>Only allow trusted sites</b> : Dynamically reflecting origins from cross-domain requests without validation is readily exploitable and should be avoided.</li><li><b>Avoid whitelisting null</b> : Avoid using the header Access-Control-Allow-Origin: null . Cross-domain resource calls from internal documents and sandboxed requests can specify the null origin. CORS headers should be properly defined in respect of trusted origins for private and public servers.</li><li><b>Avoid wildcards in internal networks</b> : Avoid using wildcards in internal networks. Trusting network configuration alone to protect internal resources is not sufficient when internal browsers can access untrusted external domains.</li></ol>	
References	
<ol style="list-style-type: none"><li><a href="https://portswigger.net/web-security/cors">https://portswigger.net/web-security/cors</a></li><li><a href="https://owasp.org/www-community/attacks/CORS_OriginHeaderScrutiny">https://owasp.org/www-community/attacks/CORS_OriginHeaderScrutiny</a></li><li><a href="https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS">https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS</a></li></ol>	
Payload Used	
origin:www.hackify.in	

## Proof of Concept

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

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

**Request:**

```
1 GET /HML/cors_lab/lab_5/cors_5.php HTTP/1.1
2 Host: labs.hackify.in
3 Origin: www.hackify.in
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:78.0) Gecko/20100101 Firefox/78.0
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
8 Referer: http://labs.hackify.in/HML/cors_lab/lab_5/login.php
9 DNT: 1
10 Connection: close
11 Cookie: PHPSESSID=d42diabc2c27f3a2ca5f0daa4d4fae3
12 Upgrade-Insecure-Requests: 1
13 Sec-GRC: 
14 Cache-Control: max-age=0
15
16
```

**Response:**

```
1 HTTP/1.1 200 OK
2 keep-alive: timeout=5, max=100
3 x-powered-by: PHP/7.4.33
4 expires: Thu, 19 Nov 1989 08:52:00 GMT
5 cache-control: no-store, no-cache, must-revalidate
6 pragma: no-cache
7 set-cookie: PHPSESSID=7ec0cd9dd1e32a5ed8e580dd5d371c4; path=/
8 access-control-allow-credentials: true
9 access-control-allow-origin: www.hackify.in
10 content-type: text/html; charset=UTF-8
11 Content-Length: 306
12 vary: Accept-Encoding,User-Agent
13 date: Sat, 01 Mar 2025 17:47:22 GMT
14 server: LiteSpeed
15 x-turbo-charged-by: LiteSpeed
16 connection: close
17
18 <html>
19 <head>
20 <meta charset="UTF-8" />
21 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
22 <meta name="keywords" content="" />
23 <link rel="icon" href=".../assets/img/favicon.png" />
24 <link rel="stylesheet" type="text/css" href="assets/css/animate.css" />
25 <link rel="stylesheet" type="text/css" href=".../assets/css/bootstrap.min.css" />
26 <link rel="stylesheet" type="text/css" href=".../assets/css/font-awesome.min.css" />
27 <link rel="stylesheet" type="text/css" href=".../assets/css/main.css" />
28 <link rel="stylesheet" type="text/css" href=".../assets/css/responsive.css" />
29 <title>
30 <url>
31 </title>
32 </url>
33 <div class="containers" style="margin:0; height:480px; width:400px;">
34 </div>
35 <script>
36 </script>
37 <script>
38 </script>
39 <script>
40 </script>
41 <script>
42 </script>
```

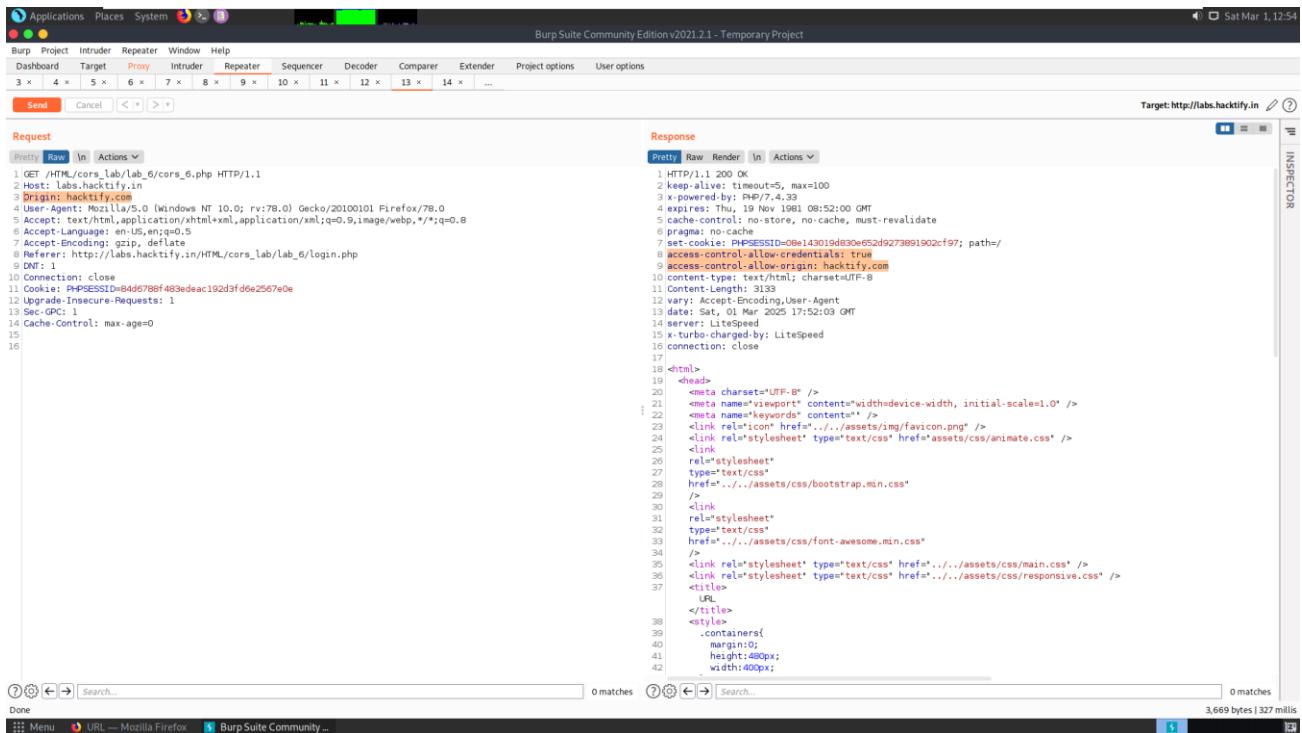
At the bottom, there are search fields for "Search..." and "URL", both showing "0 matches". The status bar at the bottom right indicates "3,625 bytes | 1,310 millis".

## 1.6. CORS with Substring match

Reference	Risk Rating
CORS with Substring match	High
Tools Used	
Google Chrome and Burp Suite	
Vulnerability Description	
Cross-origin resource sharing (CORS) is a browser mechanism which enables controlled access to resources located outside of a given domain. It extends and adds flexibility to the same-origin policy. However, it also provides potential for cross-domain based attacks, if a website's CORS policy is poorly configured and implemented. The CORS protocol uses some HTTP headers that define trusted web origins and associated properties such as whether authenticated access is permitted. Many modern websites use CORS to allow access from subdomains and trusted third parties. Sometimes because of mistakes of developers' attackers can use misconfiguration to exploit vulnerability.	
How It Was Discovered	
I found this vulnerability by manually writing Origin payload by intercepting it in the Burp Suite.	
Vulnerable URLs	
<a href="https://labs.hackify.in/HTML/cors_lab/lab_6/index.php">https://labs.hackify.in/HTML/cors_lab/lab_6/index.php</a>	
Consequences of not Fixing the Issue	
Attackers would treat many victims to visit the attacker's website, if victim is logged in, then his personal information is recorded on attacker's server. Attacker can perform any action in the user's account, bypassing CSRF tokens.	
Suggested Countermeasures	
<ol style="list-style-type: none"><li><b>Proper configuration of cross-domain requests</b> : If a web resource contains sensitive information, the origin should be properly specified in the Access-Control-Allow-Origin header.</li><li><b>Only allow trusted sites</b> : Dynamically reflecting origins from cross-domain requests without validation is readily exploitable and should be avoided.</li><li><b>Avoid whitelisting null</b> : Avoid using the header Access-Control-Allow-Origin: null . Cross-domain resource calls from internal documents and sandboxed requests can specify the null origin. CORS headers should be properly defined in respect of trusted origins for private and public servers.</li><li><b>Avoid wildcards in internal networks</b> : Avoid using wildcards in internal networks. Trusting network configuration alone to protect internal resources is not sufficient when internal browsers can access untrusted external domains.</li></ol>	
References	
<ol style="list-style-type: none"><li><a href="https://portswigger.net/web-security/cors">https://portswigger.net/web-security/cors</a></li><li><a href="https://owasp.org/www-community/attacks/CORS_OriginHeaderScrutiny">https://owasp.org/www-community/attacks/CORS_OriginHeaderScrutiny</a></li><li><a href="https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS">https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS</a></li></ol>	
Payload Used	
origin:hackify.com	

## Proof of Concept

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



The screenshot shows the Burp Suite Community Edition interface. The title bar reads "Burp Suite Community Edition v2021.2.1 - Temporary Project". The menu bar includes "Applications", "Places", "System", and "Help". The toolbar has icons for "Send", "Cancel", and navigation arrows. The main window is divided into two panes: "Request" and "Response".

**Request:**

```
1 GET /HML/cors_lab/lab_6/cors_6.php HTTP/1.1
2 Host: labs.hackify.in
3 Origin: hackify.com
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:78.0) Gecko/20100101 Firefox/78.0
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
8 Referer: http://labs.hackify.in/HML/cors_lab/lab_6/login.php
9 DNT: 1
10 Connection: close
11 Cookie: PHPSESSID=b4d678f483edeadc192d3fd6e2567e0e
12 Upgrade-Insecure-Requests: 1
13 Sec-GRC: 
14 Cache-Control: max-age=0
15 
16
```

**Response:**

```
1 HTTP/1.1 200 OK
2 keep-alive: timeout=5, max=100
3 x-powered-by: PHP/7.4.33
4 expires: Thu, 19 Nov 1989 08:52:00 GMT
5 cache-control: no-store, no-cache, must-revalidate
6 pragma: no-cache
7 set-cookie: PHPSESSID=09e1a3019d830e652d9273891902cf97; path=/
8 access-control-allow-credentials: true
9 access-control-allow-origin: hackify.com
10 content-type: text/html; charset=UTF-8
11 Content-Length: 313
12 vary: Accept-Encoding,User-Agent
13 date: Sat, 01 Mar 2025 17:52:03 GMT
14 server: LiteSpeed
15 x-turbo-charged-by: LiteSpeed
16 connection: close
17
18 <html>
19   <head>
20     <meta charset="UTF-8" />
21     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
22     <meta name="keywords" content="" />
23     <link rel="icon" href="../../assets/img/favicon.png" />
24     <link rel="stylesheet" type="text/css" href="assets/css/animate.css" />
25     <link rel="stylesheet" type="text/css" href="assets/css/bootstrap.min.css" />
26     <link rel="stylesheet" type="text/css" href="assets/css/font-awesome.min.css" />
27     <link rel="stylesheet" type="text/css" href="assets/css/main.css" />
28     <link rel="stylesheet" type="text/css" href="assets/css/responsive.css" />
29   </head>
30   <body>
31     <div class="container" style="margin: 0; height: 400px; width: 100%;>
32       <div style="height: 100%; width: 100%; background-color: #ccc; position: relative; overflow: hidden;>
33         <div style="position: absolute; top: 0; left: 0; width: 100%; height: 100%; background-color: black; opacity: 0.5; z-index: 1;>
34           <div style="position: absolute; top: 50%; left: 50%; transform: translate(-50%, -50%); color: white; font-size: 2em; z-index: 2;>
35             URL
36           </div>
37         </div>
38       </div>
39     </div>
40   </body>
41 </html>
42
```

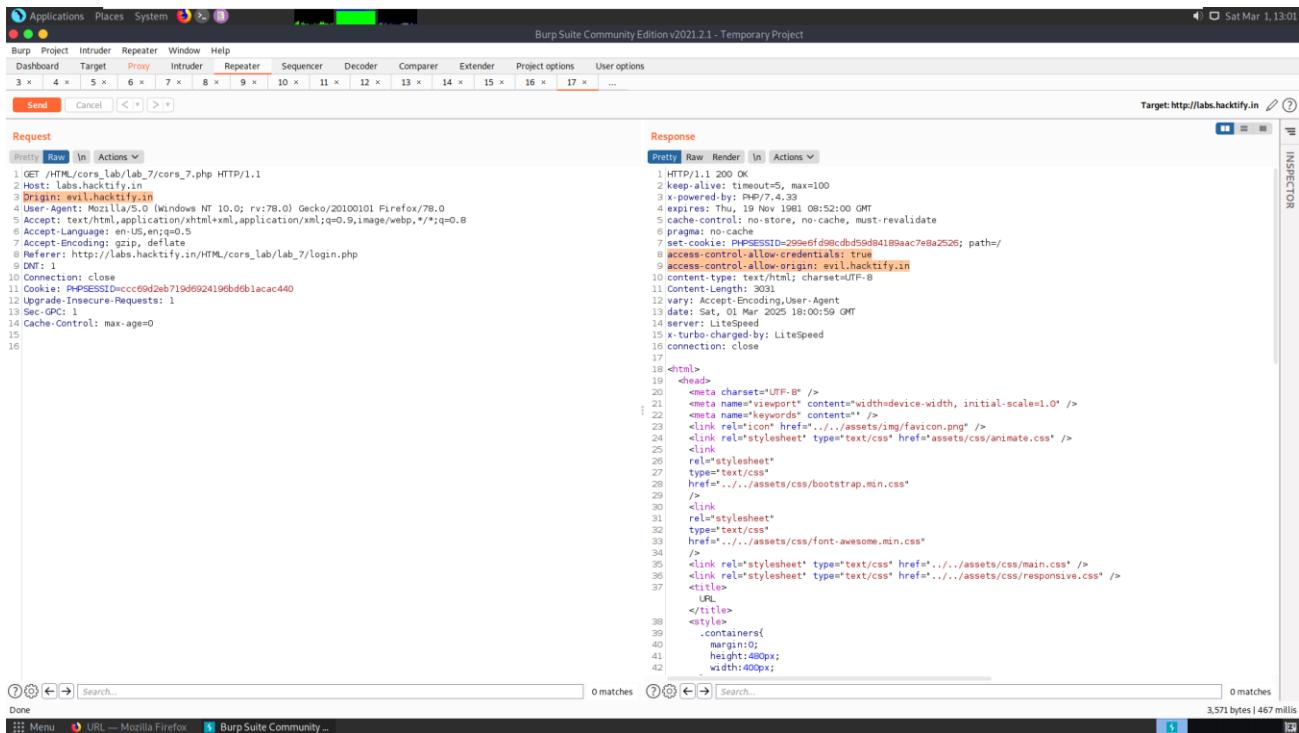
At the bottom, there are search fields for "Search..." and "URL", and status information: "0 matches", "3,669 bytes | 327 millis".

## 1.7. CORS with Arbitrary Subdomain

Reference	Risk Rating
CORS with Arbitrary Subdomain	High
Tools Used	
Google Chrome and Burp Suite	
Vulnerability Description	
<p>Cross-origin resource sharing (CORS) is a browser mechanism which enables controlled access to resources located outside of a given domain. It extends and adds flexibility to the same-origin policy. However, it also provides potential for cross-domain based attacks, if a website's CORS policy is poorly configured and implemented. The CORS protocol uses some HTTP headers that define trusted web origins and associated properties such as whether authenticated access is permitted. Many modern websites use CORS to allow access from subdomains and trusted third parties. Sometimes because of mistakes of developers' attackers can use misconfiguration to exploit vulnerability.</p>	
How It Was Discovered	
I found this vulnerability by manually writing Origin payload by intercepting it in the Burp Suite.	
Vulnerable URLs	
<a href="https://labs.hacktify.in/HTML/cors_lab/lab_7/index.php">https://labs.hacktify.in/HTML/cors_lab/lab_7/index.php</a>	
Consequences of not Fixing the Issue	
<p>Attackers would treat many victims to visit the attacker's website, if victim is logged in, then his personal information is recorded on attacker's server. Attacker can perform any action in the user's account, bypassing CSRF tokens.</p>	
Suggested Countermeasures	
<ol style="list-style-type: none"><li><b>Proper configuration of cross-domain requests :</b> If a web resource contains sensitive information, the origin should be properly specified in the Access-Control-Allow-Origin header.</li><li><b>Only allow trusted sites :</b> Dynamically reflecting origins from cross-domain requests without validation is readily exploitable and should be avoided.</li><li><b>Avoid whitelisting null :</b> Avoid using the header Access-Control-Allow-Origin: null . Cross-domain resource calls from internal documents and sandboxed requests can specify the null origin. CORS headers should be properly defined in respect of trusted origins for private and public servers.</li><li><b>Avoid wildcards in internal networks :</b> Avoid using wildcards in internal networks. Trusting network configuration alone to protect internal resources is not sufficient when internal browsers can access untrusted external domains.</li></ol>	
References	
<ol style="list-style-type: none"><li><a href="https://portswigger.net/web-security/cors">https://portswigger.net/web-security/cors</a></li><li><a href="https://owasp.org/www-community/attacks/CORS_OriginHeaderScrutiny">https://owasp.org/www-community/attacks/CORS_OriginHeaderScrutiny</a></li><li><a href="https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS">https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS</a></li></ol>	
Payload Used	
origin:evil.hacktify.in	

## Proof of Concept

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



The screenshot shows the Burp Suite Community Edition interface. The 'Request' tab displays a captured HTTP request:

```
1 GET /cors_lab/lab_7/cors_7.php HTTP/1.1
2 Host: labs.hackify.in
3 Origin: evil.hackify.in
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:78.0) Gecko/20100101 Firefox/78.0
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
8 Referer: http://labs.hackify.in/HML/cors_lab/lab_7/login.php
9 DNT: 1
10 Connection: close
11 Cookie: PHPSESSID=cc69d2eb71bd6924196bddb1acac440
12 Upgrade-Insecure-Requests: 1
13 Sec-GRC: 
14 Cache-Control: max-age=0
15 
16
```

The 'Response' tab shows the captured HTML response, which is very long and contains many script tags, indicating a reflected XSS payload.

```
1 HTTP/1.1 200 OK
2 keep-alive: timeout=5, max=100
3 x-powered-by: PHP/7.4.33
4 expires: Thu, 19 Nov 1981 08:52:00 GMT
5 cache-control: no-store, no-cache, must-revalidate
6 pragma: no-cache
7 set-cookie: PHPSESSID=299e6f199cddbd5d9d4189aac7e8a2526; path=/
8 access-control-allow-credentials: true
9 access-control-allow-origin: evil.hackify.in
10 content-type: text/html; charset=UTF-8
11 Content-Length: 303
12 vary: Accept-Encoding,User-Agent
13 date: Sat, 01 Mar 2025 18:00:59 GMT
14 server: LiteSpeed
15 x-turbo-charged-by: LiteSpeed
16 connection: close
17
18 <html>
19   <head>
20     <meta charset="UTF-8" />
21     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
22     <meta name="keywords" content="" />
23     <link rel="icon" href="../../assets/img/favicon.png" />
24     <link rel="stylesheet" type="text/css" href="../../assets/css/animate.css" />
25     <link rel="stylesheet" type="text/css" href="../../assets/css/bootstrap.min.css" />
26     <link rel="stylesheet" type="text/css" href="../../assets/css/font-awesome.min.css" />
27     <link rel="stylesheet" type="text/css" href="../../assets/css/main.css" />
28     <link rel="stylesheet" type="text/css" href="../../assets/css/responsive.css" />
29   </head>
30   <body>
31     <div>
32       <h1>Welcome to the CORS Lab</h1>
33       <p>This page demonstrates various CORS-related attacks.</p>
34       <ul>
35         <li>Cross-Origin Resource Sharing (CORS)</li>
36         <li>Cross-Site Scripting (XSS)</li>
37         <li>Cross-Site Request Forgery (CSRF)</li>
38         <li>Clickjacking</li>
39         <li>Content-Type Sniffing</li>
40         <li>Origin Tracing</li>
41         <li>HTTP Response Headers Manipulation</li>
42       </ul>
43     </div>
44   </body>
45 </html>
```

The bottom status bar indicates the target is `http://labs.hackify.in`, and the message bar says "Done".

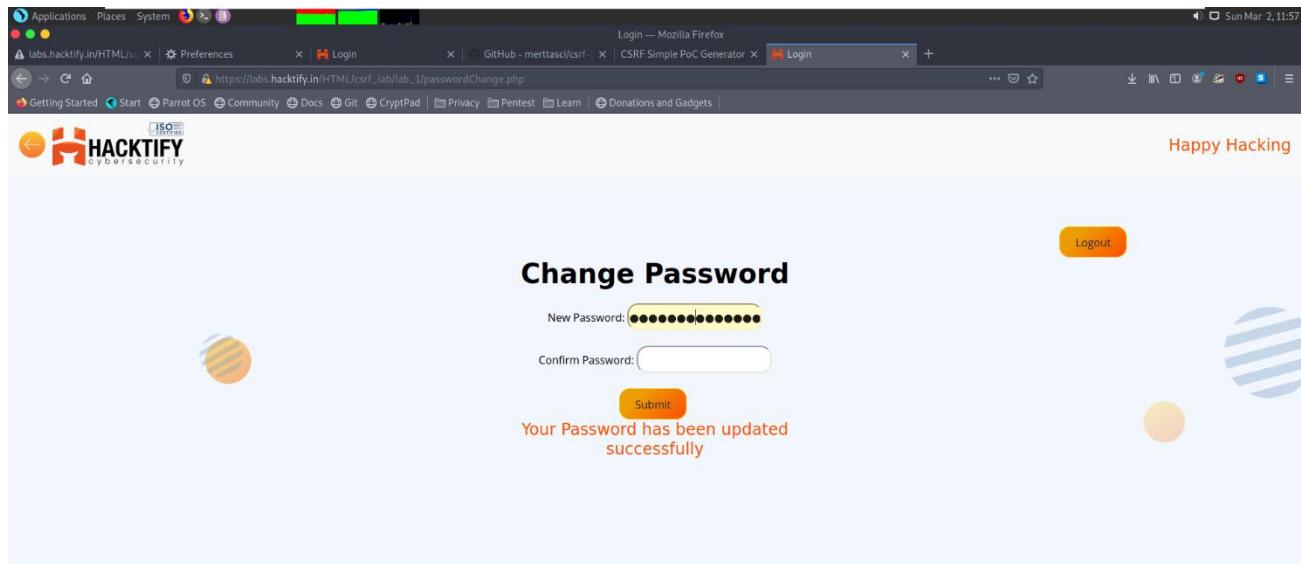
## 2. Cross-Site Request Forgery (CSRF)

### 2.1. Eassy CSRF

Reference	Risk Rating
Eassy CSRF	Low
<b>Tools Used</b>	
Google chrome and Burp Suite	
<b>Vulnerability Description</b>	
Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. With a little help of social engineering an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.	
<b>How It Was Discovered</b>	
I found this vulnerability by generating CSRF POCs of forged URL and sending it to the victim.	
<b>Vulnerable URLs</b>	
<a href="https://labs.hackify.in/HTML/csrf_lab/lab_1/passwordChange.php">https://labs.hackify.in/HTML/csrf_lab/lab_1/passwordChange.php</a>	
<b>Consequences of not Fixing the Issue</b>	
In a successful CSRF attack, the attacker causes the victim user to carry out an action unintentionally. Depending on the nature of the action, the attacker might be able to gain full control over the user's account. If the compromised user has a privileged role within the application, then the attacker might be able to take full control of all the application's data and functionality	
<b>Suggested Countermeasures</b>	
<ol style="list-style-type: none"><li><b>Anti-CSRF Tokens:</b> Use a token that is associated with a particular user and can be found as a hidden value in every state changing form which is present on the web application. This token, called a CSRF Token or a Synchronizer Token.</li><li><b>Same Site Cookies:</b> CSRF attacks are only possible since Cookies are always sent with any requests that are sent to a particular origin, which is related to that Cookie. Due to the nature of a CSRF attack, a flag can be set against a Cookie, turning it into a same-site Cookie. A same-site Cookie is a Cookie which can only be sent, if the request is being made from the same origin that is related to the Cookie being sent.</li></ol>	
<b>References</b>	
<ol style="list-style-type: none"><li><a href="https://portswigger.net/web-security/csrf">https://portswigger.net/web-security/csrf</a></li><li><a href="https://owasp.org/www-community/attacks/csrf">https://owasp.org/www-community/attacks/csrf</a></li><li><a href="https://www.acunetix.com/websitetecurity/csrf-attacks/">https://www.acunetix.com/websitetecurity/csrf-attacks/</a></li></ol>	
<b>Payload Used</b>	
Generated CSRF POC using Burp Suite and changed the password to "abcd" followed by sending the URL to victim user. Logged in into the system as victim user, followed by hitting submit button from it.	

## Proof of Concept

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

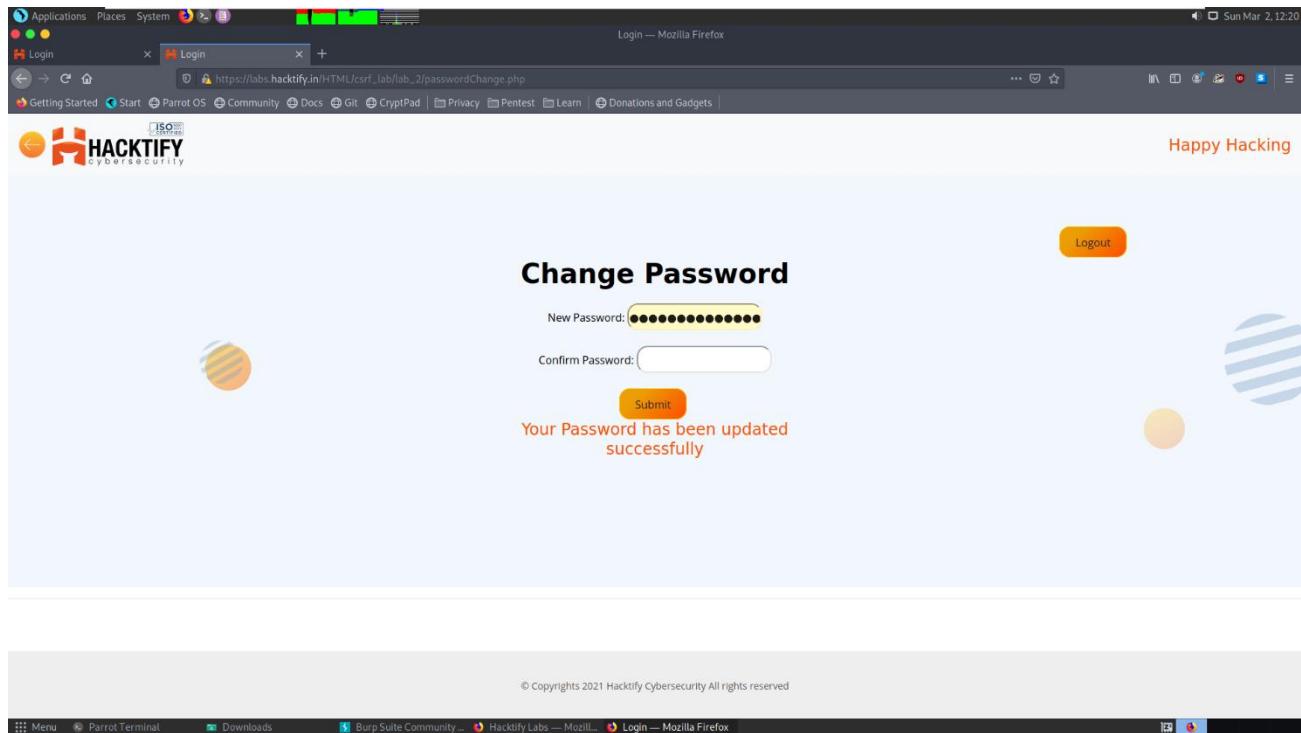


## 2.2. Always Validate Tokens

Reference	Risk Rating
Always Validate Tokens	Medium
<b>Tools Used</b>	
Google Chrome and Burp Suite	
<b>Vulnerability Description</b>	
Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. With a little help of social engineering an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.	
<b>How It Was Discovered</b>	
I found this vulnerability by generating CSRF POCs of forged URL and sending it to the victim.	
<b>Vulnerable URLs</b>	
<a href="https://labs.hackify.in/HTML/csrf_lab/lab_2/passwordChange.php">https://labs.hackify.in/HTML/csrf_lab/lab_2/passwordChange.php</a>	
<b>Consequences of not Fixing the Issue</b>	
In a successful CSRF attack, the attacker causes the victim user to carry out an action unintentionally. Depending on the nature of the action, the attacker might be able to gain full control over the user's account. If the compromised user has a privileged role within the application, then the attacker might be able to take full control of all the application's data and functionality	
<b>Suggested Countermeasures</b>	
<ol style="list-style-type: none"><li><b>Anti-CSRF Tokens:</b> Use a token that is associated with a particular user and can be found as a hidden value in every state changing form which is present on the web application. This token, called a CSRF Token or a Synchronizer Token.</li><li><b>Same Site Cookies:</b> CSRF attacks are only possible since Cookies are always sent with any requests that are sent to a particular origin, which is related to that Cookie. Due to the nature of a CSRF attack, a flag can be set against a Cookie, turning it into a same-site Cookie. A same-site Cookie is a Cookie which can only be sent, if the request is being made from the same origin that is related to the Cookie being sent.</li></ol>	
<b>References</b>	
<ol style="list-style-type: none"><li><a href="https://portswigger.net/web-security/csrf">https://portswigger.net/web-security/csrf</a></li><li><a href="https://owasp.org/www-community/attacks/csrf">https://owasp.org/www-community/attacks/csrf</a></li><li><a href="https://www.acunetix.com/websitetecurity/csrf-attacks/">https://www.acunetix.com/websitetecurity/csrf-attacks/</a></li></ol>	
<b>Payload Used</b>	
Generated CSRF POC using Burp Suite and changed the password to "abcd" followed by sending the URL to victim user. Logged in into the system as victim user, followed by hitting submit button from it.	

## Proof of Concept

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

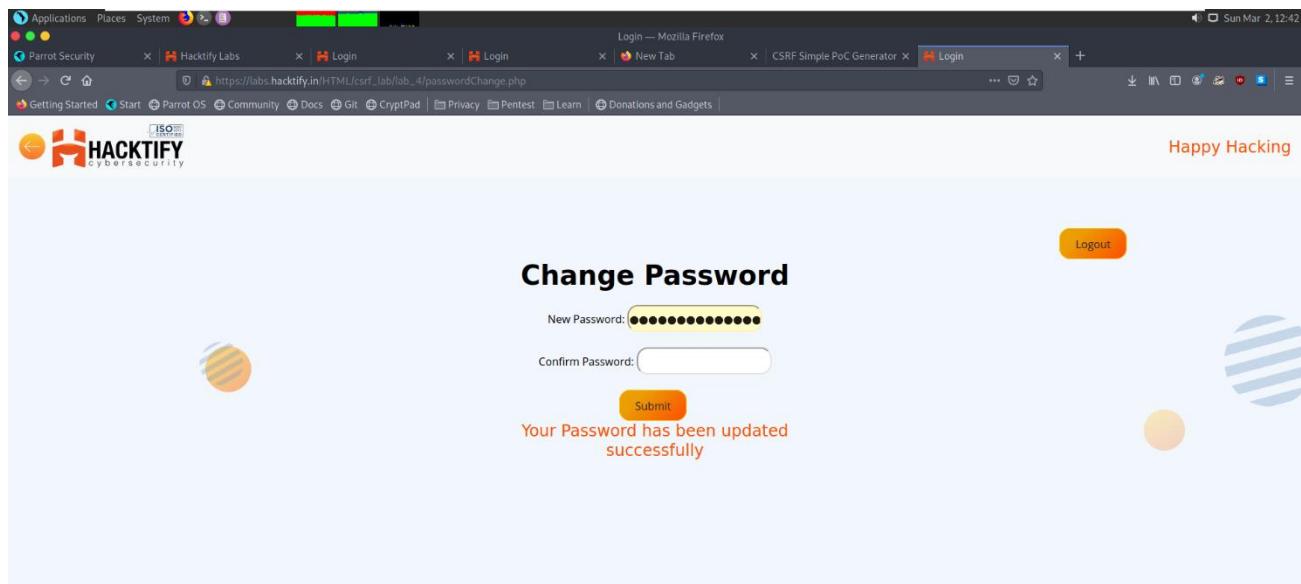


## 2.3. I hate when someone uses my tokens!

Reference	Risk Rating
I hate when someone uses my tokens!	Medium
<b>Tools Used</b>	
Google Chrome and Burp Suite	
<b>Vulnerability Description</b>	
Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. With a little help of social engineering an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.	
<b>How It Was Discovered</b>	
I found this vulnerability by generating CSRF POCs of forged URL and sending it to the victim.	
<b>Vulnerable URLs</b>	
<a href="https://labs.hackify.in/HTML/csrf_lab/lab_4/passwordChange.php">https://labs.hackify.in/HTML/csrf_lab/lab_4/passwordChange.php</a>	
<b>Consequences of not Fixing the Issue</b>	
In a successful CSRF attack, the attacker causes the victim user to carry out an action unintentionally. Depending on the nature of the action, the attacker might be able to gain full control over the user's account. If the compromised user has a privileged role within the application, then the attacker might be able to take full control of all the application's data and functionality.	
<b>Suggested Countermeasures</b>	
1. <b>Anti-CSRF Tokens:</b> Use a token that is associated with a particular user and can be found as a hidden value in every state changing form which is present on the web application. This token, called a CSRF Token or a Synchronizer Token. 2. <b>Same Site Cookies:</b> CSRF attacks are only possible since Cookies are always sent with any requests that are sent to a particular origin, which is related to that Cookie. Due to the nature of a CSRF attack, a flag can be set against a Cookie, turning it into a same-site Cookie. A same-site Cookie is a Cookie which can only be sent, if the request is being made from the same origin that is related to the Cookie being sent.	
<b>References</b>	
1. <a href="https://portswigger.net/web-security/csrf">https://portswigger.net/web-security/csrf</a> 2. <a href="https://owasp.org/www-community/attacks/csrf">https://owasp.org/www-community/attacks/csrf</a> 3. <a href="https://www.acunetix.com/websitedevelopment/csrf-attacks/">https://www.acunetix.com/websitedevelopment/csrf-attacks/</a>	
<b>Payload Used</b>	
Generated CSRF POC using Burp Suite and changed the password to "abcd" followed by sending the URL to victim user. Logged in into the system as victim user, followed by hitting submit button from it.	

## Proof of Concept

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

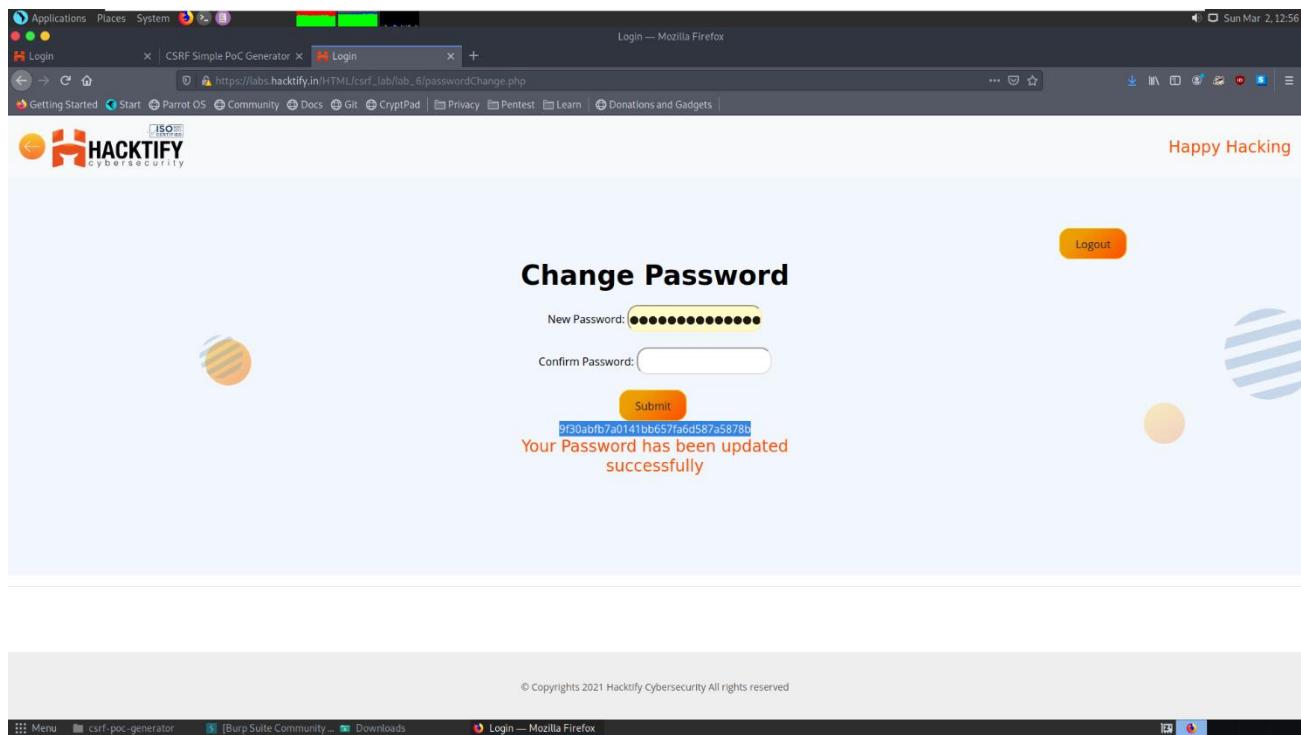


## 2.4. GET Me or POST ME

Reference	Risk Rating
GET Me or POST ME	Low
<b>Tools Used</b>	
Google Chrome and Burp Suite	
<b>Vulnerability Description</b>	
Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. With a little help of social engineering an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.	
<b>How It Was Discovered</b>	
I found this vulnerability by generating CSRF POCs of forged URL and sending it to the victim.	
<b>Vulnerable URLs</b>	
<a href="https://labs.hackify.in/HTML/csrf_lab/lab_6/passwordChange.php">https://labs.hackify.in/HTML/csrf_lab/lab_6/passwordChange.php</a>	
<b>Consequences of not Fixing the Issue</b>	
In a successful CSRF attack, the attacker causes the victim user to carry out an action unintentionally. Depending on the nature of the action, the attacker might be able to gain full control over the user's account. If the compromised user has a privileged role within the application, then the attacker might be able to take full control of all the application's data and functionality	
<b>Suggested Countermeasures</b>	
<ol style="list-style-type: none"><li><b>Anti-CSRF Tokens:</b> Use a token that is associated with a particular user and can be found as a hidden value in every state changing form which is present on the web application. This token, called a CSRF Token or a Synchronizer Token.</li><li><b>Same Site Cookies:</b> CSRF attacks are only possible since Cookies are always sent with any requests that are sent to a particular origin, which is related to that Cookie. Due to the nature of a CSRF attack, a flag can be set against a Cookie, turning it into a same-site Cookie. A same-site Cookie is a Cookie which can only be sent, if the request is being made from the same origin that is related to the Cookie being sent.</li></ol>	
<b>References</b>	
<ol style="list-style-type: none"><li><a href="https://portswigger.net/web-security/csrf">https://portswigger.net/web-security/csrf</a></li><li><a href="https://owasp.org/www-community/attacks/csrf">https://owasp.org/www-community/attacks/csrf</a></li><li><a href="https://www.acunetix.com/websitedevelopment/csrf-attacks/">https://www.acunetix.com/websitedevelopment/csrf-attacks/</a></li></ol>	
<b>Payload Used</b>	
Generated CSRF POC using Burp Suite and changed the password to "abcd" followed by sending the URL to victim user. Logged in into the system as victim user, followed by hitting submit button from it.	

## Proof of Concept

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

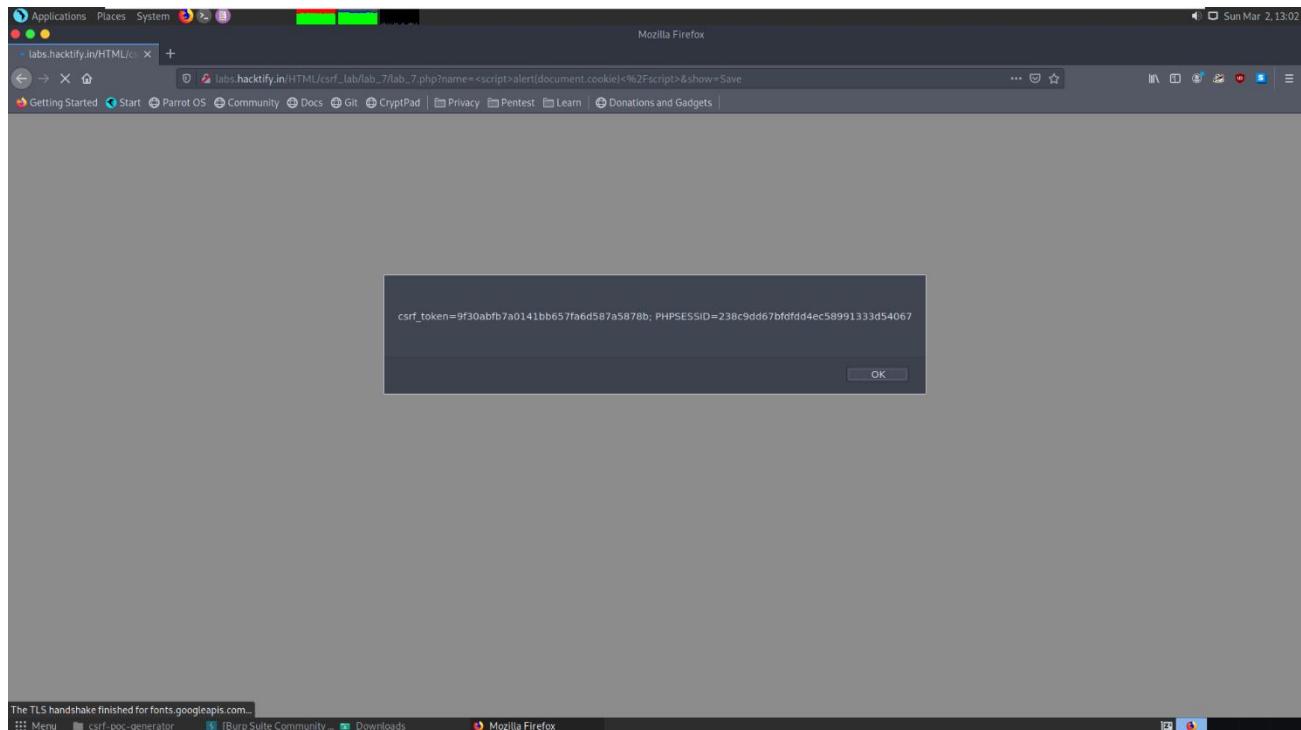


## 2.5. XSS the saviour

Reference	Risk Rating
XSS the saviour	Low
<b>Tools Used</b>	
Google Chrome and Burp Suite	
<b>Vulnerability Description</b>	
Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. With a little help of social engineering an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.	
<b>How It Was Discovered</b>	
I found this vulnerability by generating CSRF POCs of forged URL and sending it to the victim.	
<b>Vulnerable URLs</b>	
<a href="http://labs.hacktify.in/HTML/csrf_lab/lab_7/lab_7.php?name=%3Cscript%3Ealert%28document.cookie%29%3C%2Fscript%3E&amp;show=Save">http://labs.hacktify.in/HTML/csrf_lab/lab_7/lab_7.php?name=%3Cscript%3Ealert%28document.cookie%29%3C%2Fscript%3E&amp;show=Save</a>	
<b>Consequences of not Fixing the Issue</b>	
In a successful CSRF attack, the attacker causes the victim user to carry out an action unintentionally. Depending on the nature of the action, the attacker might be able to gain full control over the user's account. If the compromised user has a privileged role within the application, then the attacker might be able to take full control of all the application's data and functionality	
<b>Suggested Countermeasures</b>	
<ol style="list-style-type: none"><li><b>Anti-CSRF Tokens:</b> Use a token that is associated with a particular user and can be found as a hidden value in every state changing form which is present on the web application. This token, called a CSRF Token or a Synchronizer Token.</li><li><b>Same Site Cookies:</b> CSRF attacks are only possible since Cookies are always sent with any requests that are sent to a particular origin, which is related to that Cookie. Due to the nature of a CSRF attack, a flag can be set against a Cookie, turning it into a same-site Cookie. A same-site Cookie is a Cookie which can only be sent, if the request is being made from the same origin that is related to the Cookie being sent.</li></ol>	
<b>References</b>	
<ol style="list-style-type: none"><li><a href="https://portswigger.net/web-security/csrf">https://portswigger.net/web-security/csrf</a></li><li><a href="https://owasp.org/www-community/attacks/csrf">https://owasp.org/www-community/attacks/csrf</a></li><li><a href="https://www.acunetix.com/websitedevelopment/csrf-attacks/">https://www.acunetix.com/websitedevelopment/csrf-attacks/</a></li></ol>	
<b>Payload Used</b>	
<script>alert(document.cookie)</script>	

## Proof of Concept

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



## 2.6. rm -rf token

Reference	Risk Rating
rm -rf token	Low
<b>Tools Used</b>	
Google Chrome and Burp Suite	
<b>Vulnerability Description</b>	
Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. With a little help of social engineering an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.	
<b>How It Was Discovered</b>	
I found this vulnerability by generating CSRF POCs of forged URL and sending it to the victim.	
<b>Vulnerable URLs</b>	
<a href="https://labs.hackify.in/HTML/csrf_lab/lab_8/passwordChange.php">https://labs.hackify.in/HTML/csrf_lab/lab_8/passwordChange.php</a>	
<b>Consequences of not Fixing the Issue</b>	
In a successful CSRF attack, the attacker causes the victim user to carry out an action unintentionally. Depending on the nature of the action, the attacker might be able to gain full control over the user's account. If the compromised user has a privileged role within the application, then the attacker might be able to take full control of all the application's data and functionality	
<b>Suggested Countermeasures</b>	
<ol style="list-style-type: none"><li><b>Anti-CSRF Tokens:</b> Use a token that is associated with a particular user and can be found as a hidden value in every state changing form which is present on the web application. This token, called a CSRF Token or a Synchronizer Token.</li><li><b>Same Site Cookies:</b> CSRF attacks are only possible since Cookies are always sent with any requests that are sent to a particular origin, which is related to that Cookie. Due to the nature of a CSRF attack, a flag can be set against a Cookie, turning it into a same-site Cookie. A same-site Cookie is a Cookie which can only be sent, if the request is being made from the same origin that is related to the Cookie being sent.</li></ol>	
<b>References</b>	
<ol style="list-style-type: none"><li><a href="https://portswigger.net/web-security/csrf">https://portswigger.net/web-security/csrf</a></li><li><a href="https://owasp.org/www-community/attacks/csrf">https://owasp.org/www-community/attacks/csrf</a></li><li><a href="https://www.acunetix.com/websitedevelopment/csrf-attacks/">https://www.acunetix.com/websitedevelopment/csrf-attacks/</a></li></ol>	
<b>Payload Used</b>	
Generated CSRF POC using Burp Suite and changed the password to "abcd" followed by sending the URL to victim user. Logged in into the system as victim user, followed by hitting submit button from it.	

## Proof of Concept

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

