**Penetration Testing Report**

**Full Name:1813\_AYUSH PARKARA  
Program: HCPT**

**Date:08/03/2024**

**Introduction**

This report document hereby describes the proceedings and results of a Black Box security assessment conducted against the **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.

|  |  |
| --- | --- |
| **Application Name** | **{CSRF}, {SSRF }** |

**3. Summary**

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

**Total number of Sub-labs: {15} Sub-lab**

|  |  |  |
| --- | --- | --- |
| **High** | **Medium** | **Low** |
| **{4}** | **{5}** | **{6}** |

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

**Medium - Number of Sub-labs with Medium difficulty leve**

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

# 1. {Cross-site request forgery}

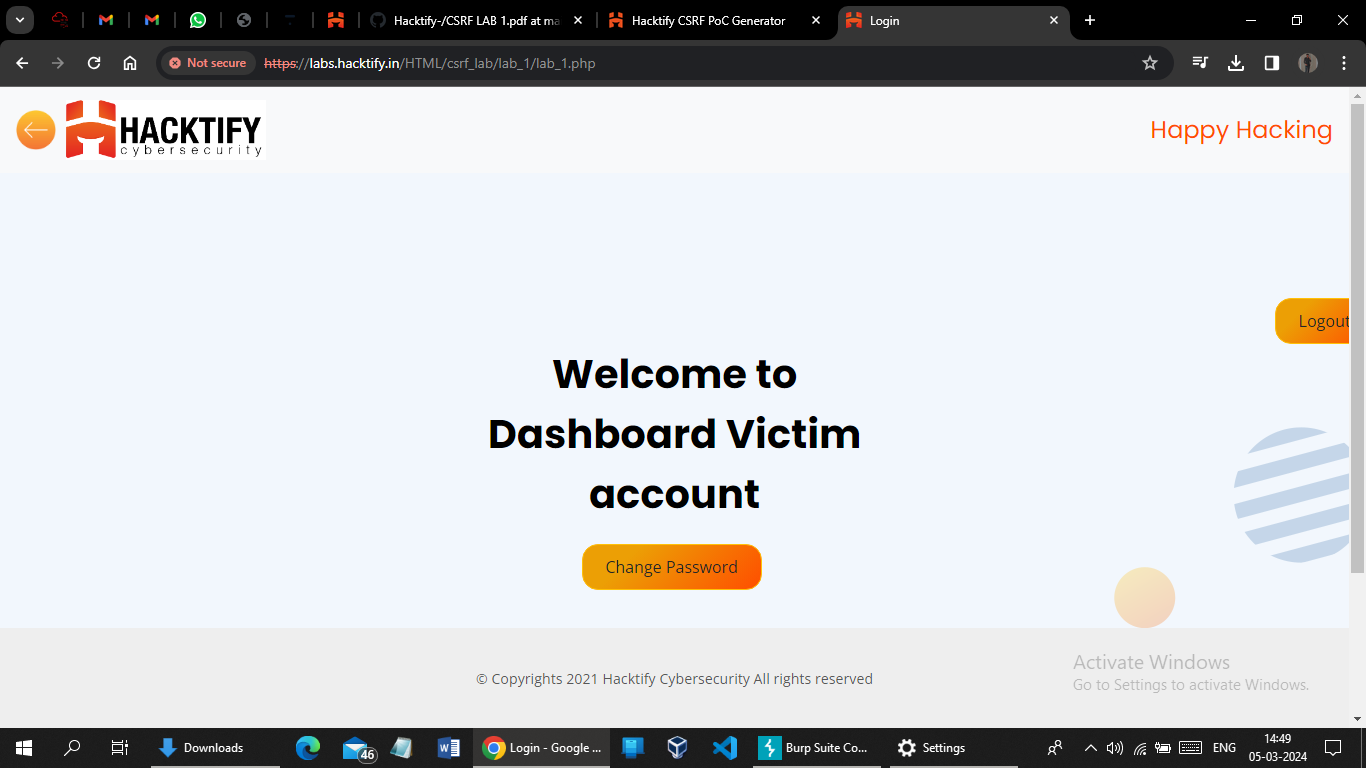
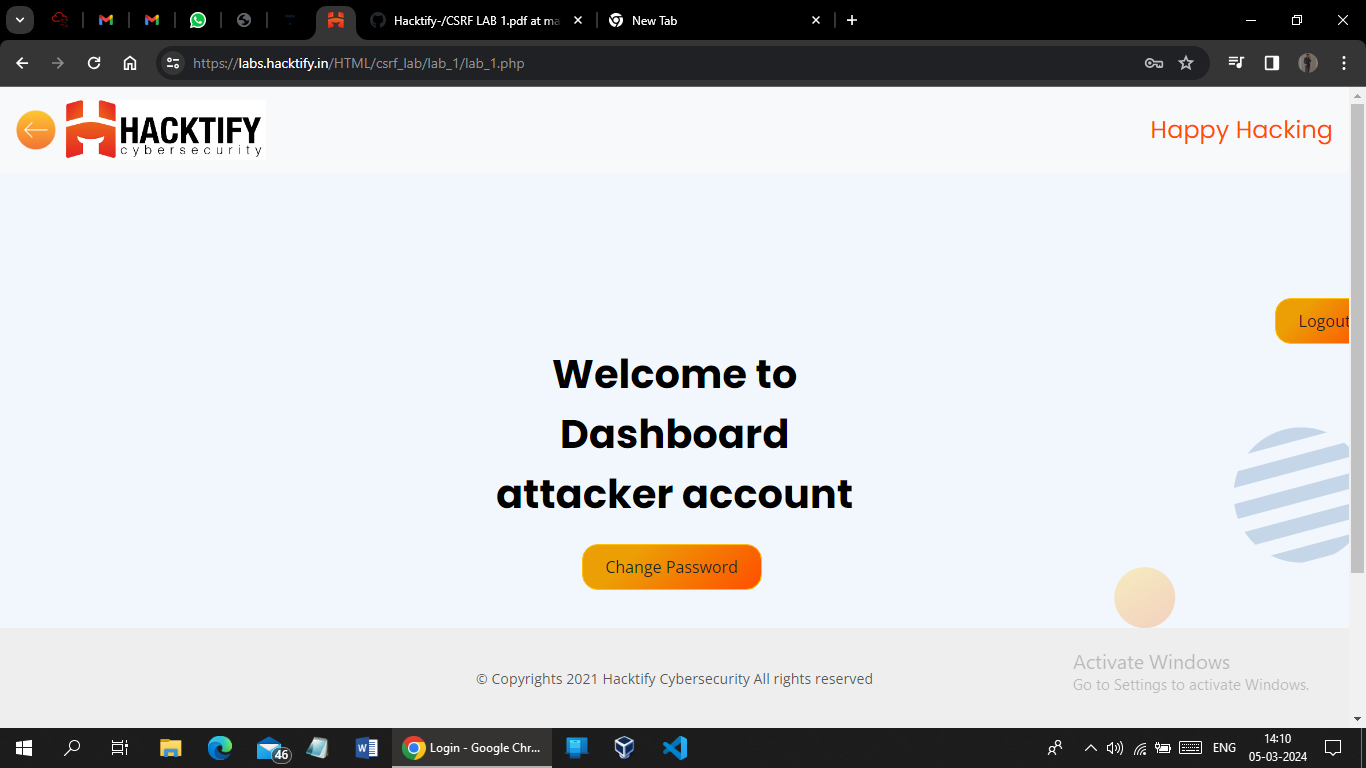
# 1.1. {Eassyy CSRF}

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| { Eassyy CSRF } | **Low** |
| **Tools Used** | |
| Burp suite , CSRF POC Generator | |
| **Vulnerability Description** | |
| Cross-Site Request Forgery 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.  Cross-Site Request Forgery 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. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/csrf\_lab/lab\_1/passwordChange.php | |
| **Consequences of not Fixing the Issue** | |
| * Unauthorized actions * Data theft * Account compromised * Reputation damage * Financial losses | |
| **Suggested Countermeasures** | |
| * Implement security measures such as using :   CSRF tokens  Validate requests  Secure coding practices  Regular auditing  Educate people about CSRF attacks   * Avoid clicking on suspicious links | |
| **References** | |
| <https://portswigger.net/web-security/csrf>  <https://owasp.org/www-community/attacks/csrf>  <https://www.invicti.com/learn/cross-site-request-forgery-csrf/> | |

# 

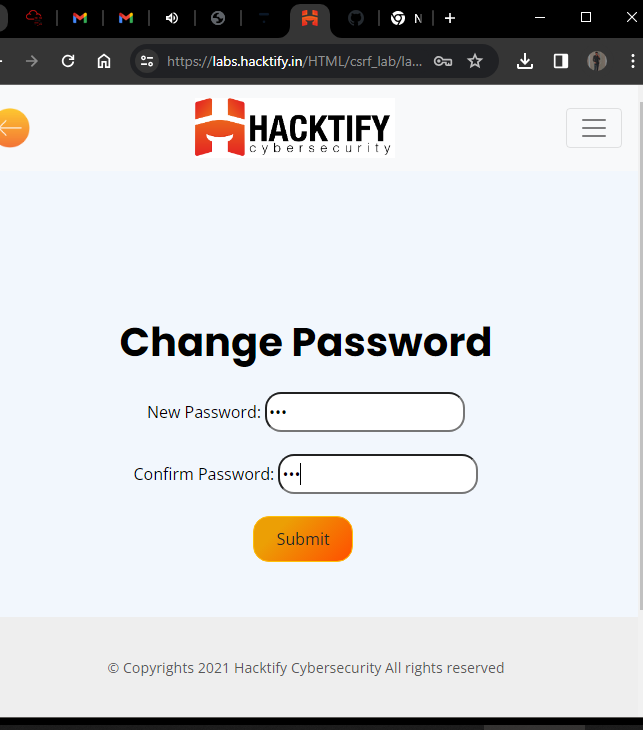
# Proof of Concept

First I create two accounts one is victim and another is attacker.

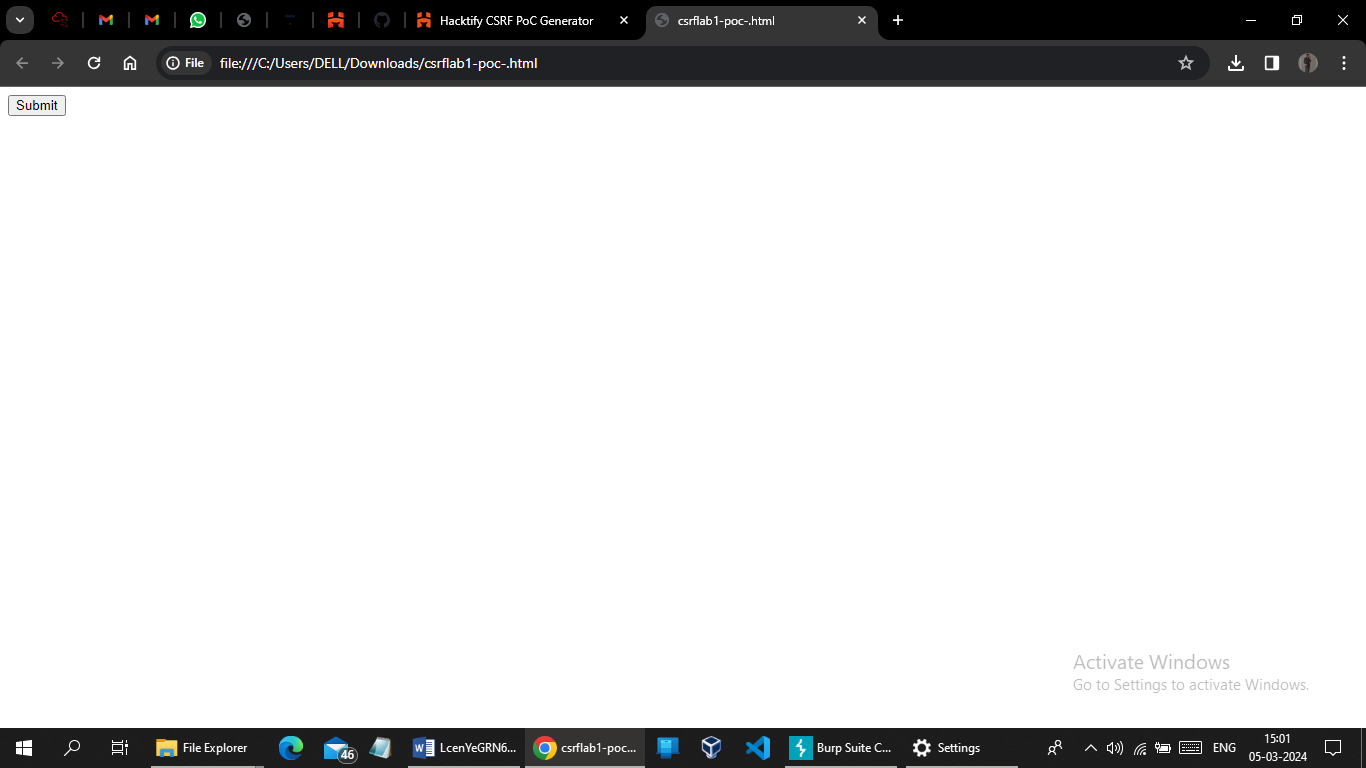
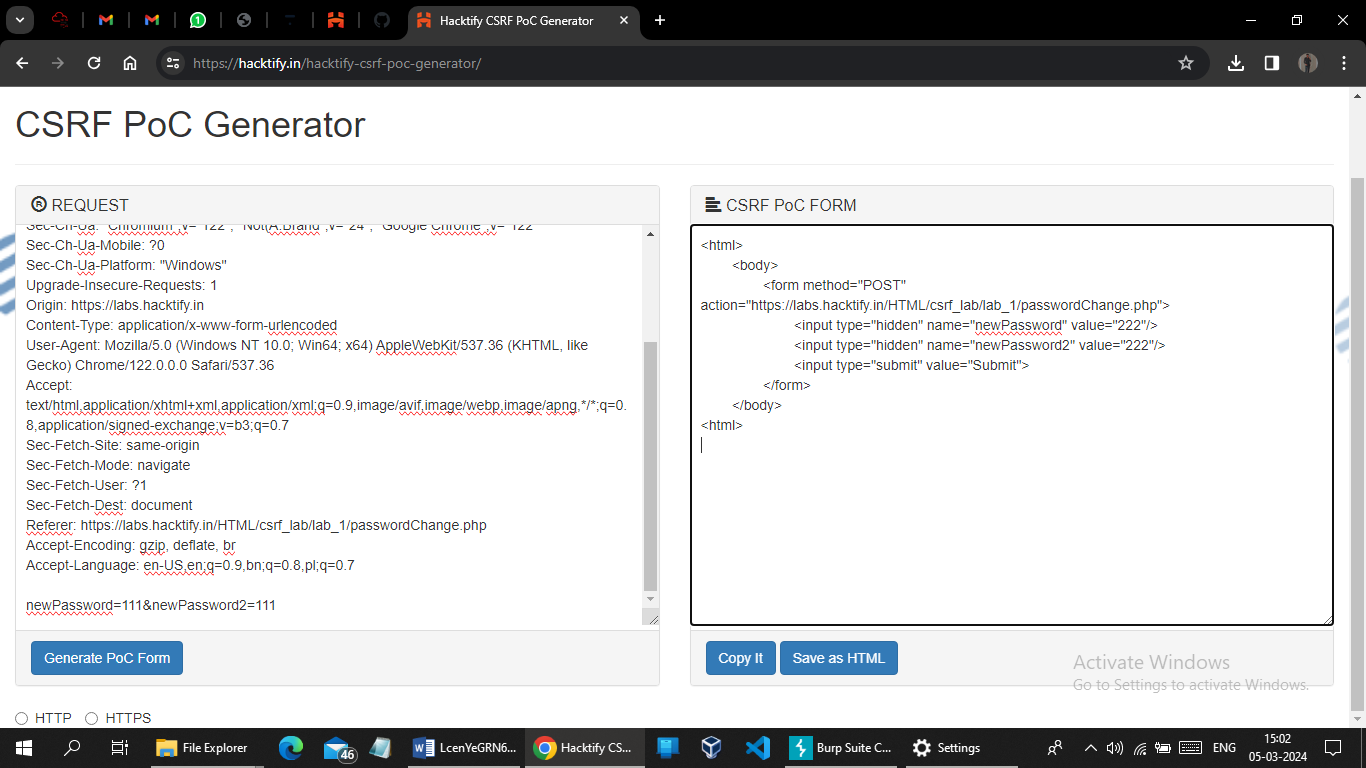


After this I login with attackers credentials into the attacker account

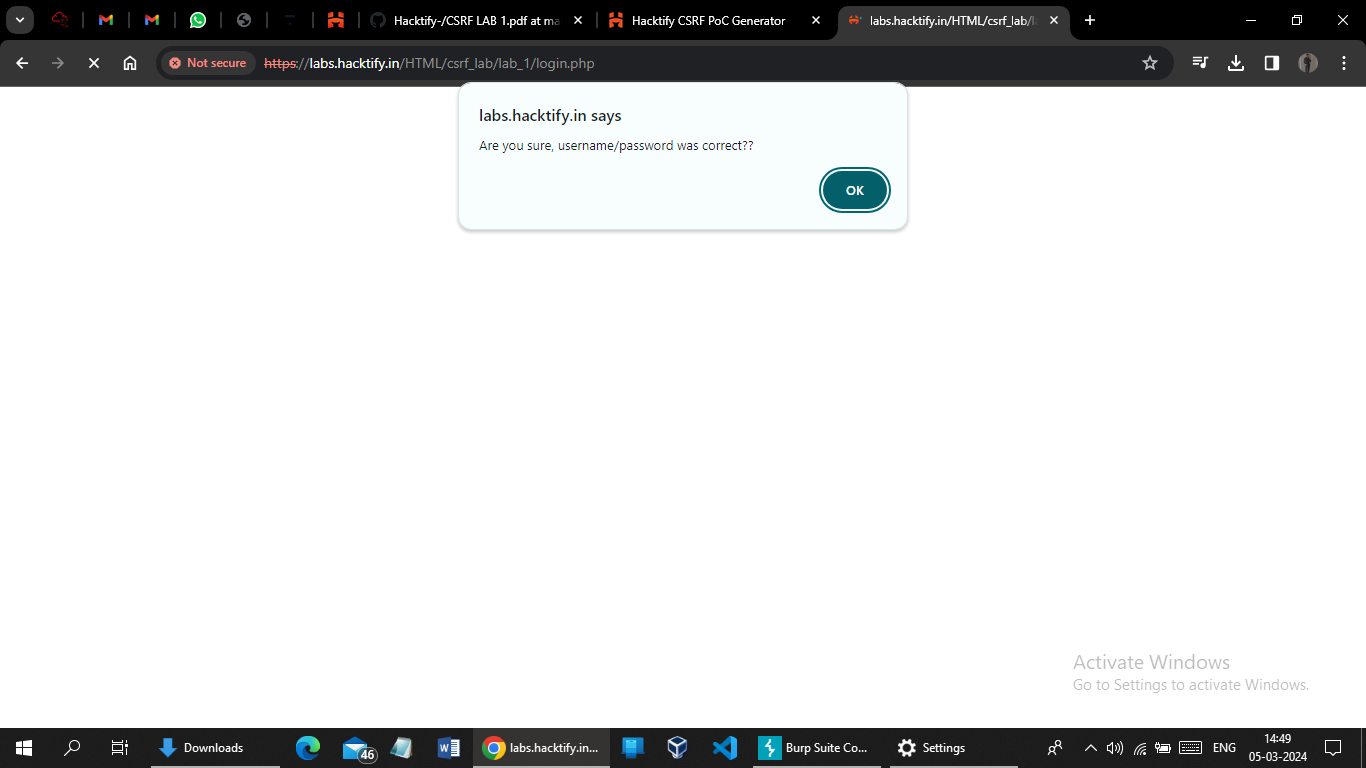
Click on change password and provide a new password .



Now intercepting the request on burp and generate the CSRF POC



After executing POC and try to login with old password and it give invalidpassword



And after I tried to login into victim account with new password which use in html POC file, I logined successfully.

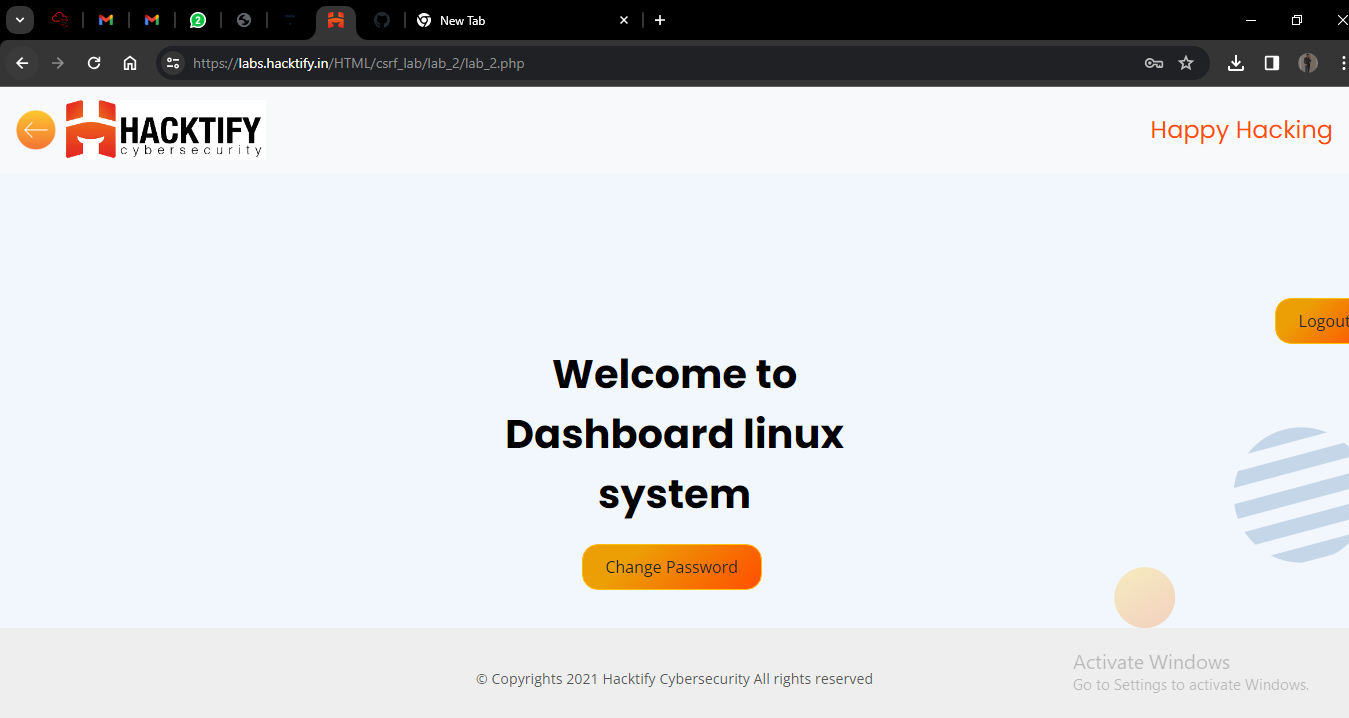
It means that lab1 is vulnerable to CSRF.

# 1.2. {Always Validate Tokens}

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| { Always Validate Tokens } | **Medium** |
| **Tools Used** | |
| Burp suite , CSRF POC Generator | |
| **Vulnerability Description** | |
| Cross-Site Request Forgery 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.  Cross-Site Request Forgery 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. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/csrf\_lab/lab\_2/passwordChange.php | |
| **Consequences of not Fixing the Issue** | |
| * Unauthorized actions * Data theft * Account compromised * Reputation damage * Financial losses | |
| **Suggested Countermeasures** | |
| * Implement security measures such as using :   CSRF tokens  Validate requests  Secure coding practices  Regular auditing  Educate people about CSRF attacks   * Avoid clicking on suspicious links | |
| **References** | |
| <https://portswigger.net/web-security/csrf>  <https://owasp.org/www-community/attacks/csrf>  <https://www.invicti.com/learn/cross-site-request-forgery-csrf/> | |

# Proof of Concept

First I create two account one is window and another one is linux.





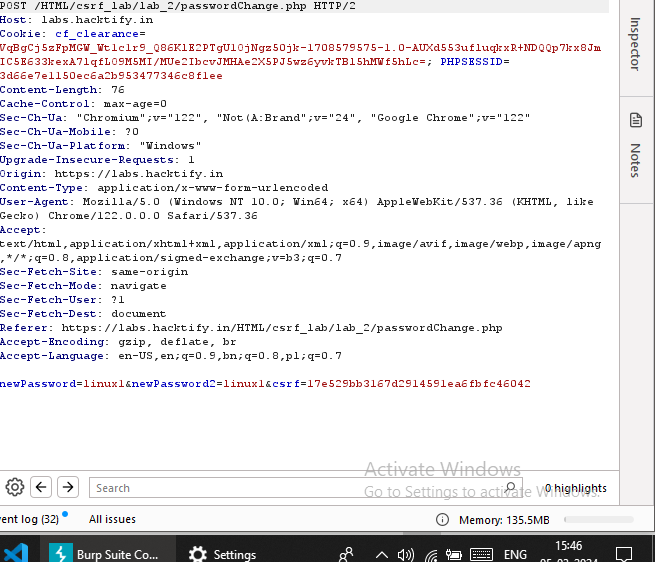
After this I login with linux credentials into the linux account

Click on change password and provide a new password .

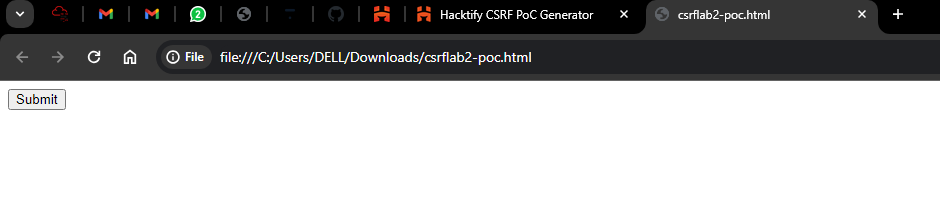
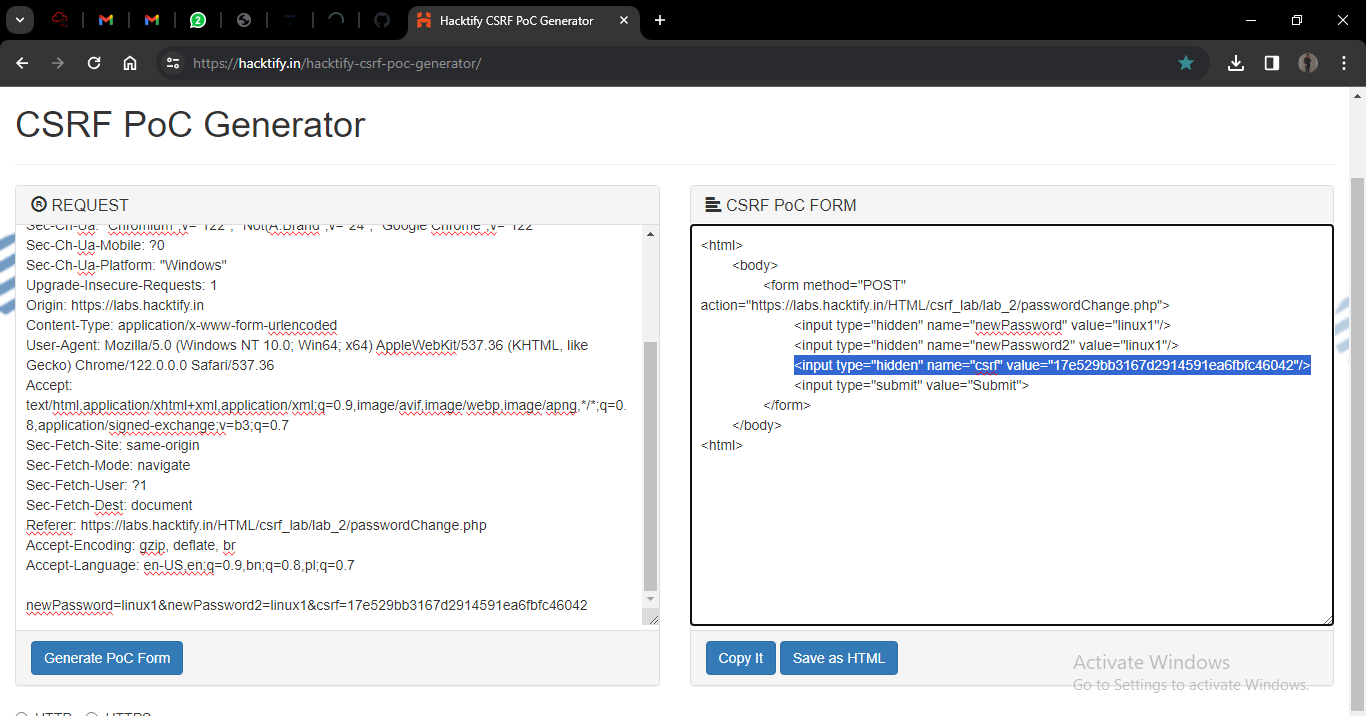


Now intercepting the request on burp and generate the CSRF POC.

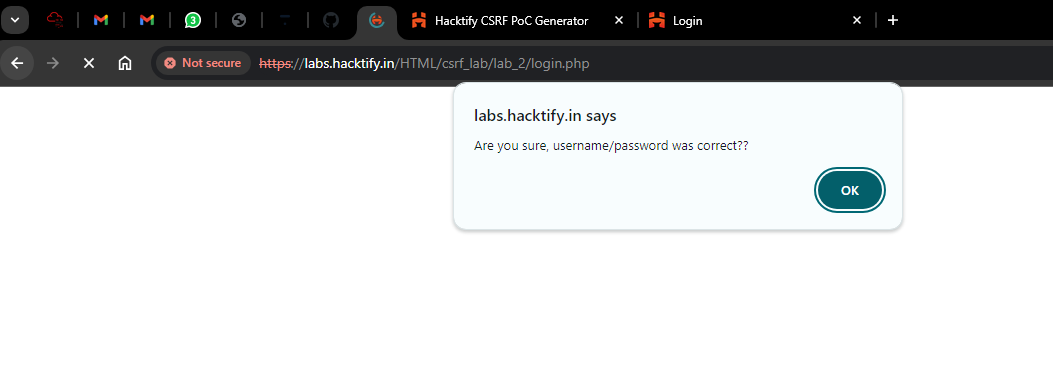
After intercepting the request on burp, we can see that there is a CSRF token



If I remove the token it will say invalid token so we have to put token in POC



click on submit and try to login with old password and it gives that password is invalid.

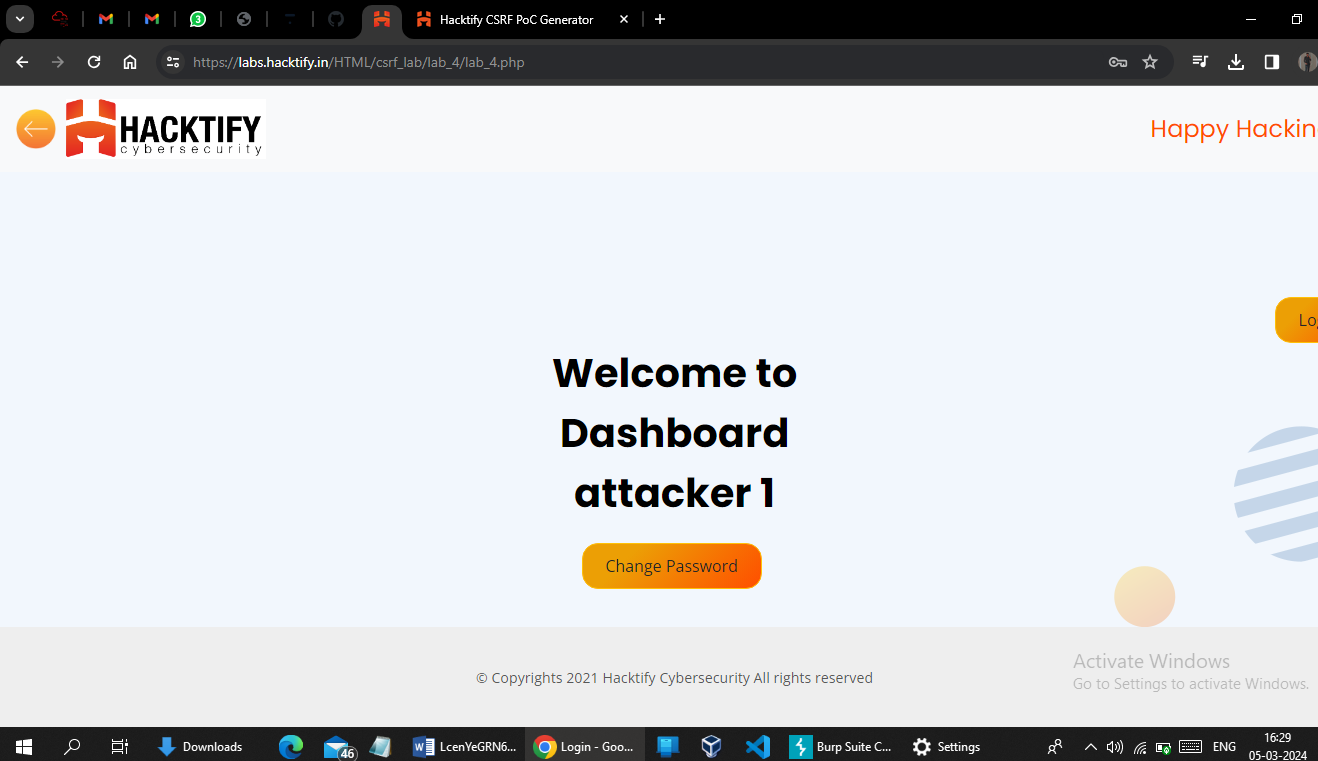
And after I tried to login into linux account with new password which use in html POC file, I logined successfully.

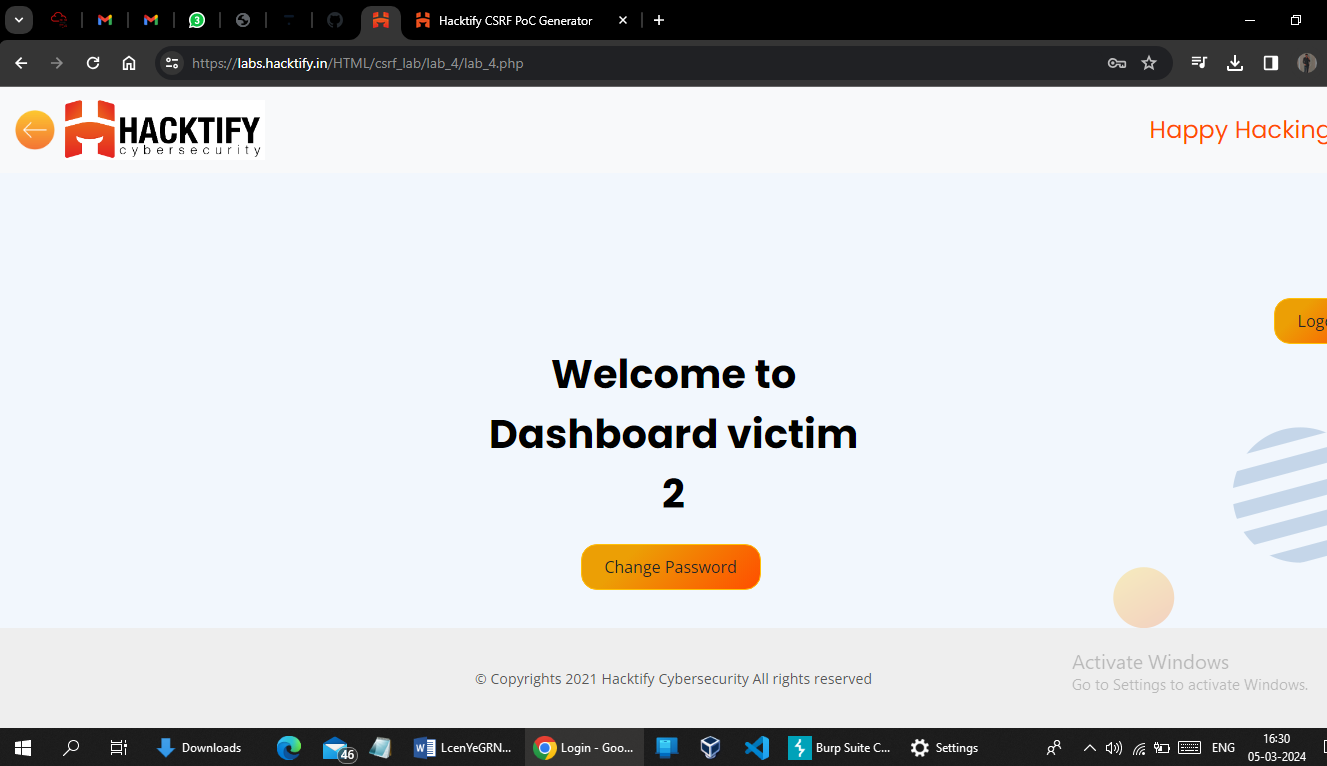
It means that lab2 is vulnerable to CSRF.

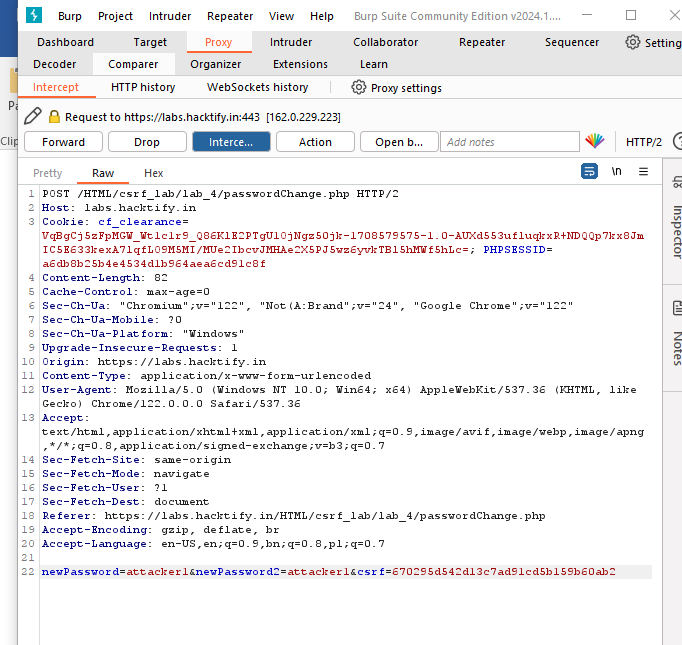
# 1.4. {I Hate When Someone Uses My Tokens!}

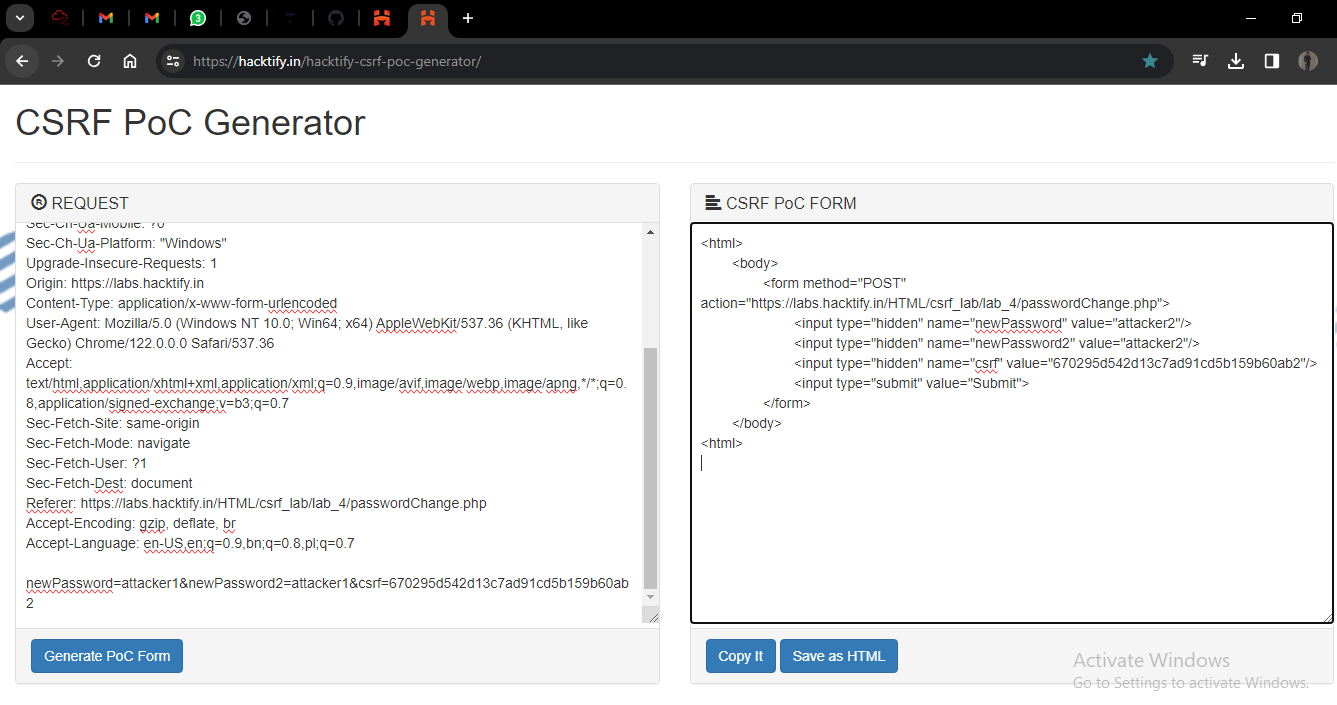
|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| {I Hate When Someone Uses My Tokens!} | **Medium** |
| **Tools Used** | |
| Burp suite , CSRF POC Generator | |
| **Vulnerability Description** | |
| Cross-Site Request Forgery is an attack that forces an end user to execute unwanted actions 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.  Cross-Site Request Forgery 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. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/csrf\_lab/lab\_4/passwordChange.php | |
| **Consequences of not Fixing the Issue** | |
| * Unauthorized actions * Data theft * Account compromised * Reputation damage * Financial losses | |
| **Suggested Countermeasures** | |
| * Implement security measures such as using :   CSRF tokens  Validate requests  Secure coding practices  Regular auditing  Educate people about CSRF attacks   * Avoid clicking on suspicious links | |
| **References** | |
| <https://portswigger.net/web-security/csrf>  <https://owasp.org/www-community/attacks/csrf>  <https://www.invicti.com/learn/cross-site-request-forgery-csrf/> | |

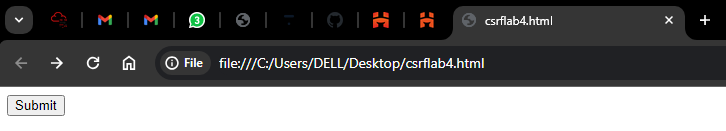
# Proof of Concept

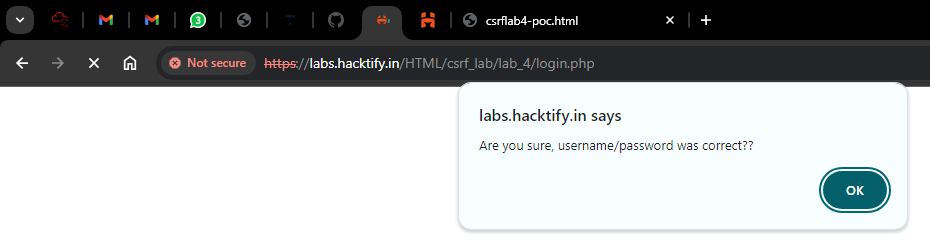








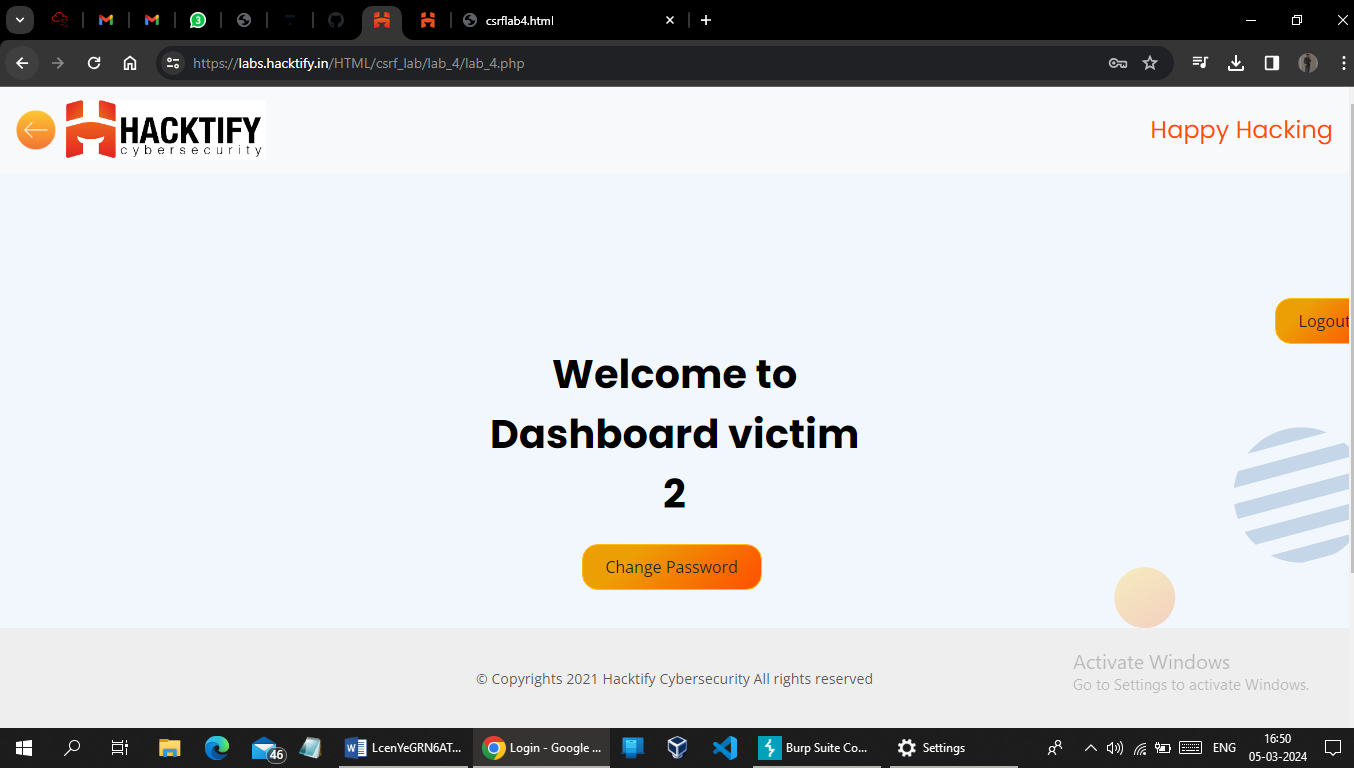




Victim account is not open with old password after the click on submit button.

Try to login with new password and it logined in.

Successfully password has been changed as updated in CSRF POC.

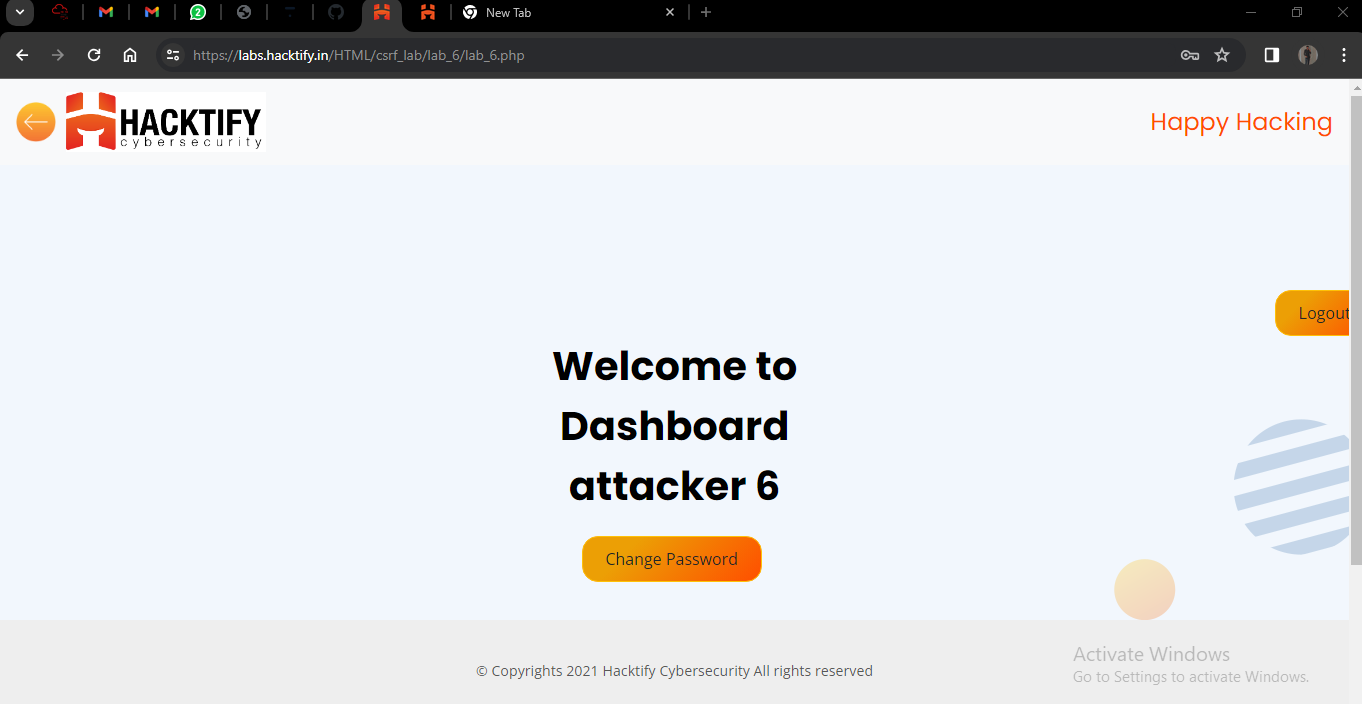


# 1.6. {GET Me Or POST ME}

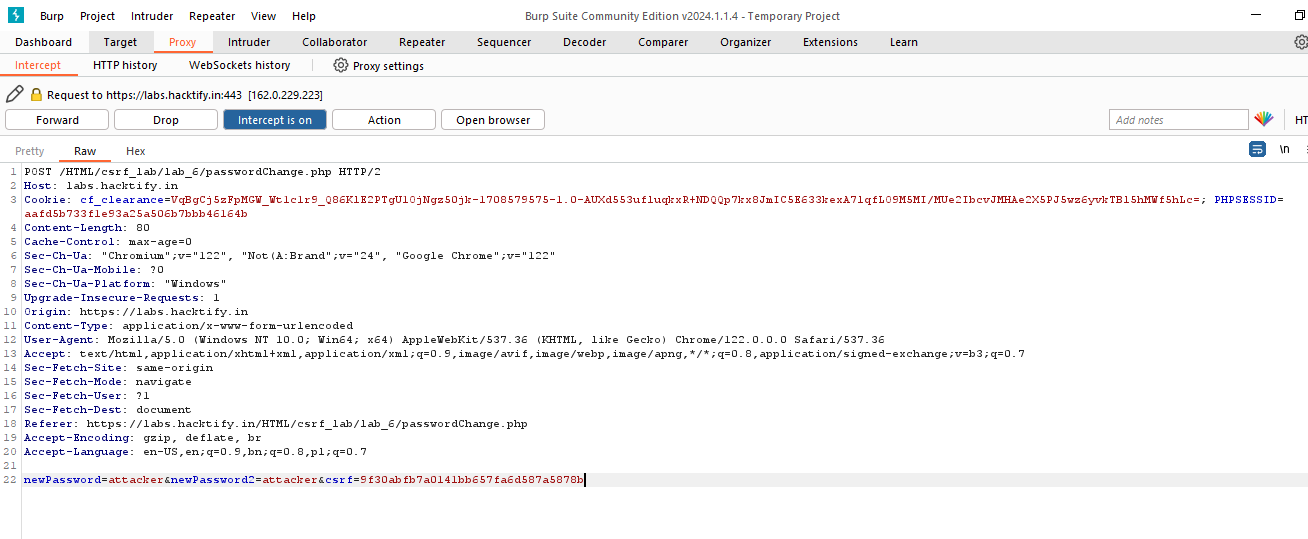
|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| {GET Me Or POST ME} | **Low** |
| **Tools Used** | |
| Burp suite , CSRF POC Generator | |
| **Vulnerability Description** | |
| Cross-Site Request Forgery is an attack that forces an end user to execute unwanted actions on a web application 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.  Cross-Site Request Forgery 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. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/csrf\_lab/lab\_6/passwordChange.php | |
| **Consequences of not Fixing the Issue** | |
| * Unauthorized actions * Data theft * Account compromised * Reputation damage * Financial losses | |
| **Suggested Countermeasures** | |
| * Implement security measures such as using :   CSRF tokens  Validate requests  Secure coding practices  Regular auditing  Educate people about CSRF attacks   * Avoid clicking on suspicious links | |
| **References** | |
| <https://portswigger.net/web-security/csrf>  <https://owasp.org/www-community/attacks/csrf>  <https://www.invicti.com/learn/cross-site-request-forgery-csrf/> | |

# Proof of Concept

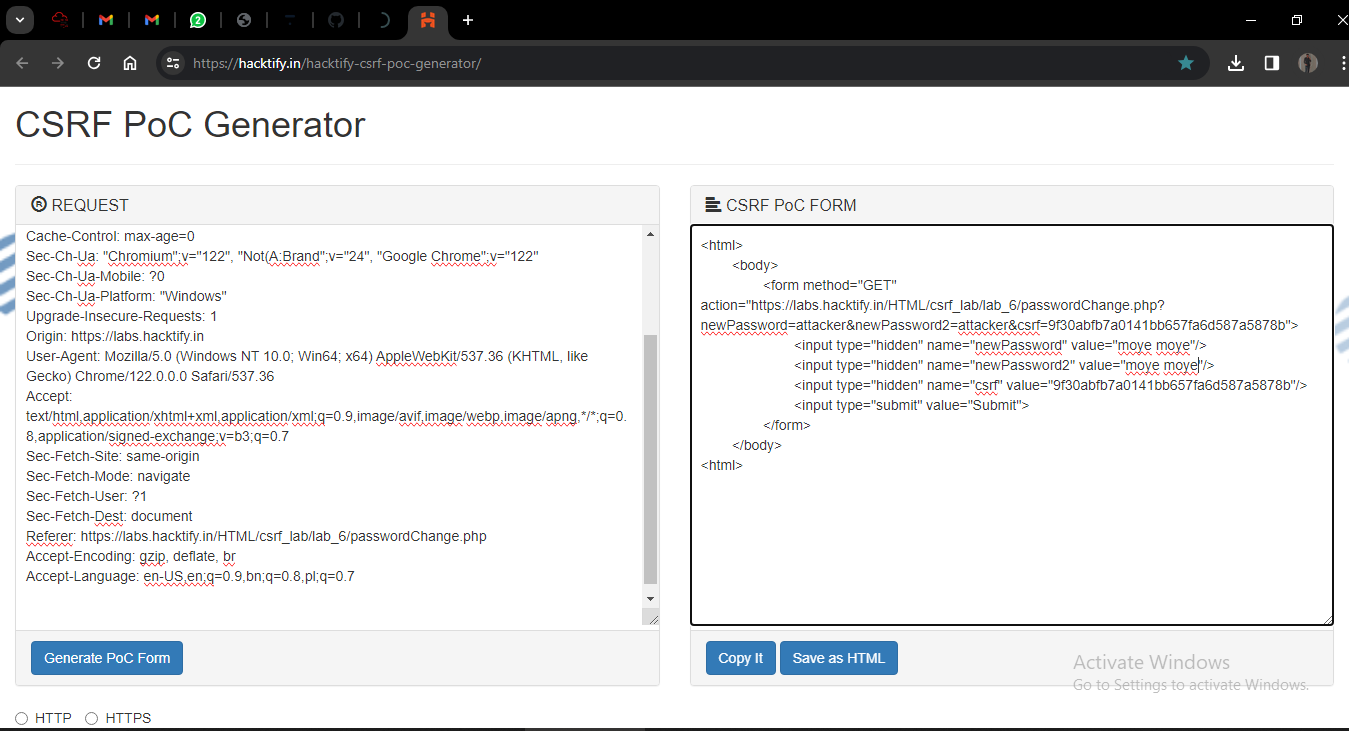
Created two different account.





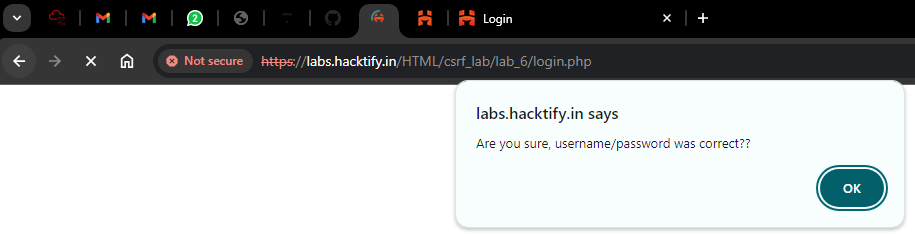
Log into attacker 6 account and change the current password with intercept the request using burp generate CSRF POC

After intercept the request change the method post to get.



Generate CSRF POC and open html file and click submit button(after log into the victim account)

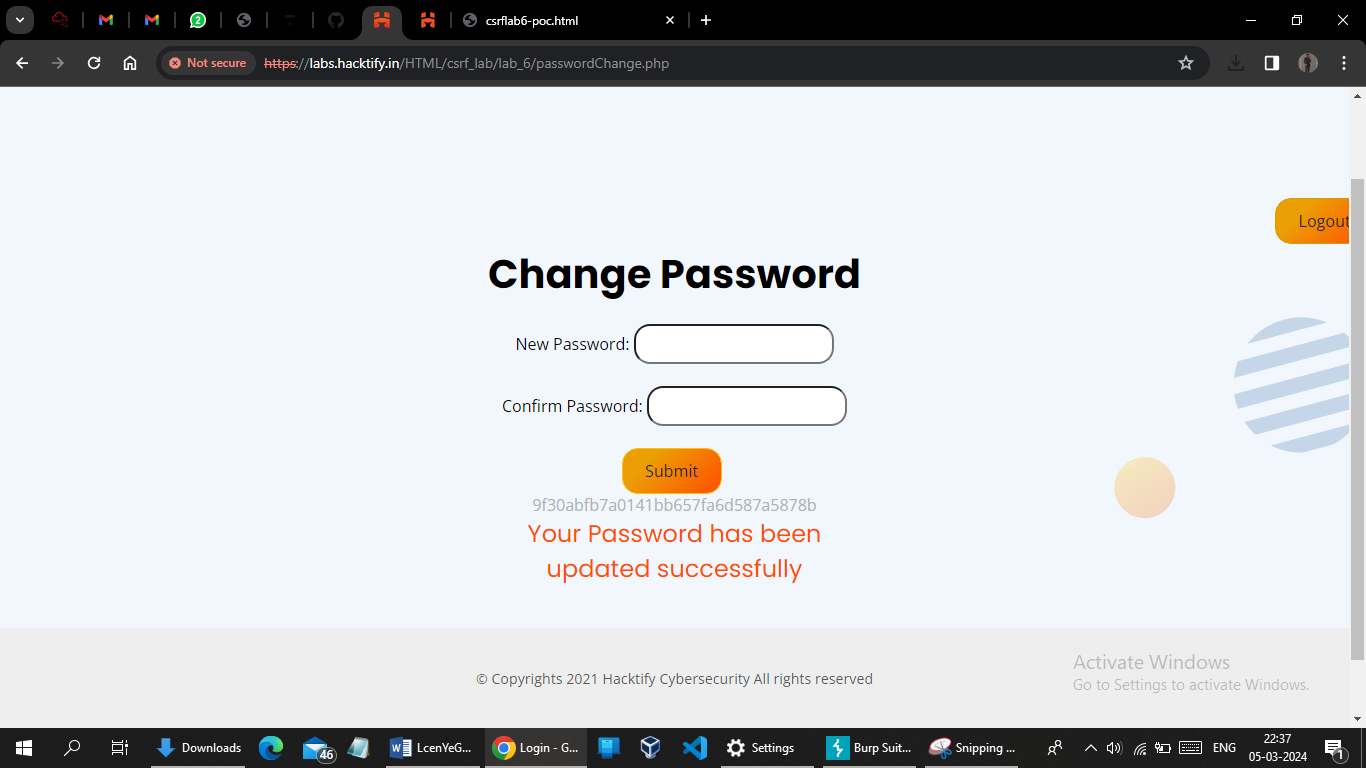
Try to log in into victim account with old password and it gives



When I tried to re-login with new password that are used in CSRF POC it is login successfully



When intercepting the request from attacker account ,As the request send to the server and intercept it in burp suite the CSRF token is also reflected on the web page.

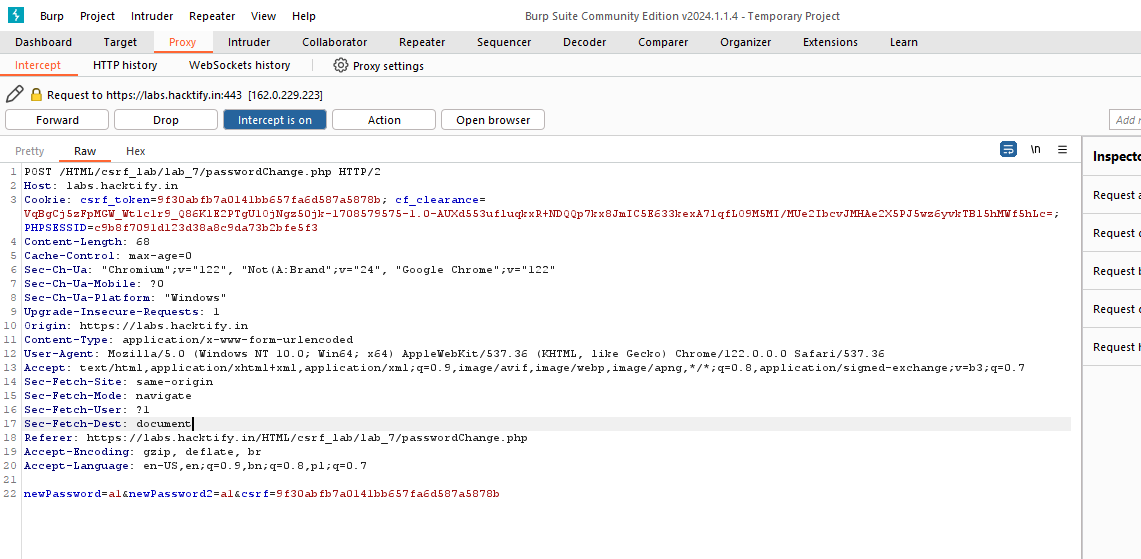


# 1.7. {XSS The Saviour}

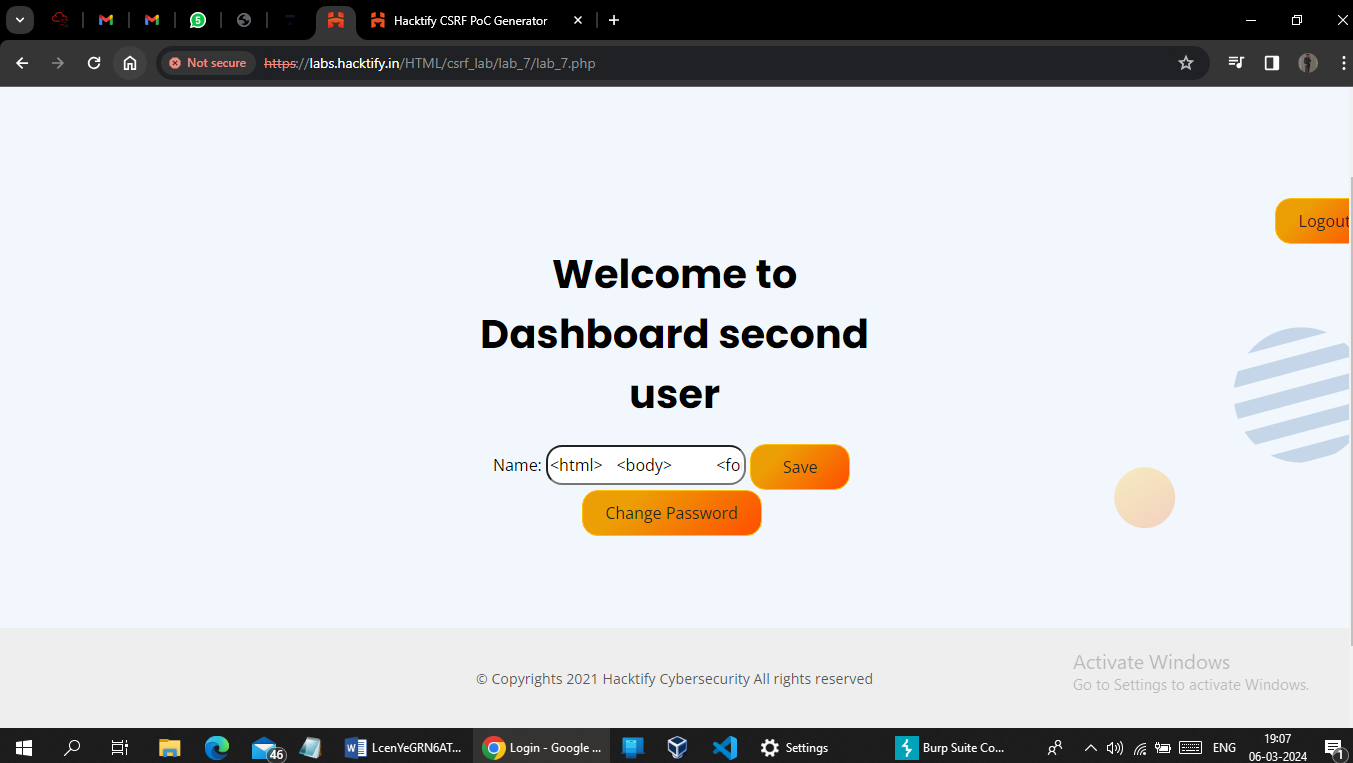
|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| { XSS The Saviour } | **High** |
| **Tools Used** | |
| Burp suite , CSRF POC Generator | |
| **Vulnerability Description** | |
| Cross-Site Request Forgery is an attack that forces an end user to execute unwanted actions on a web application 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.  Cross-Site Request Forgery 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. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/csrf\_lab/lab\_7/passwordChange.php | |
| **Consequences of not Fixing the Issue** | |
| * Unauthorized actions * Data theft * Account compromised * Reputation damage * Financial losses | |
| **Suggested Countermeasures** | |
| * Implement security measures such as using :   CSRF tokens  Validate requests  Secure coding practices  Regular auditing  Educate people about CSRF attacks   * Avoid clicking on suspicious links | |
| **References** | |
| <https://portswigger.net/web-security/csrf>  <https://owasp.org/www-community/attacks/csrf>  <https://www.invicti.com/learn/cross-site-request-forgery-csrf/> | |

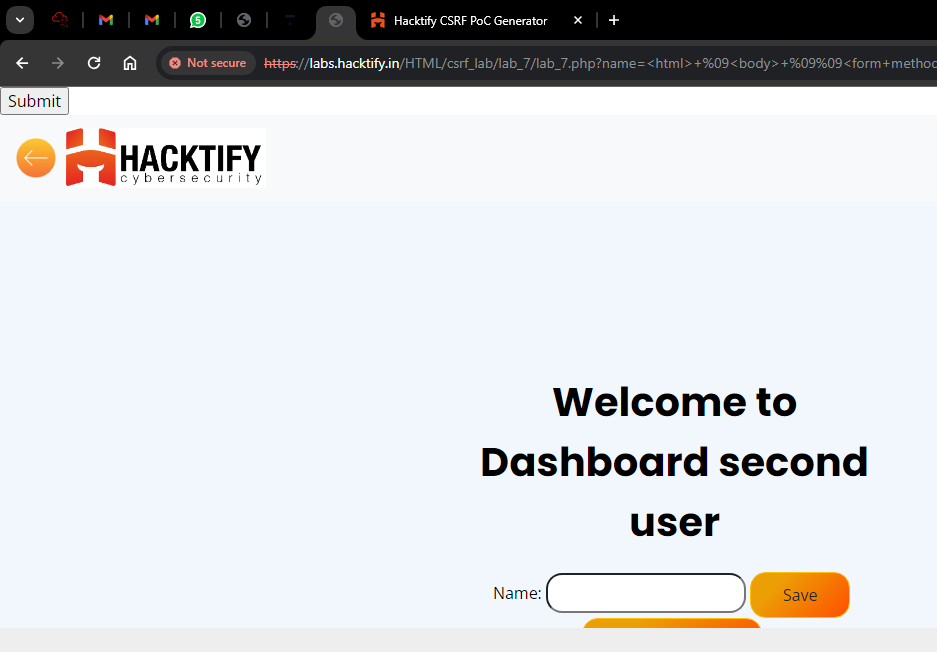
# Proof of Concept

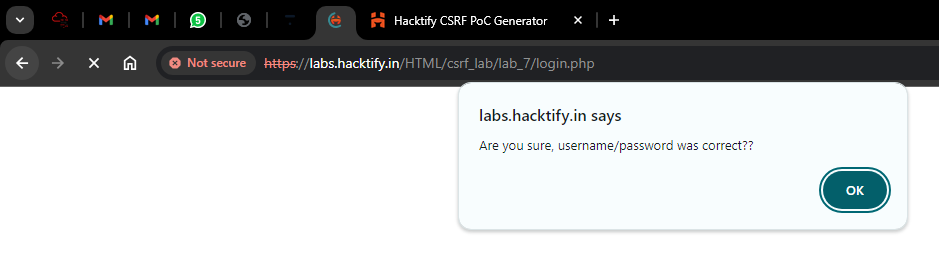
* First I create two account one is first user and another one is second user.
* Now login into the first user account and intercept the request in burp.
* After intercepting the request generate CSRF POC .
* Change the password in generated CSRF POC.
* Login into the second user.
* Copy the generated POC and paste it into the name field of second user.
* Click on save button.
* As we click on save button, POC is execute.
* Log out into the second user account and re-login into the second user account with old password. It give a message that entered password is incorrect.
* Try to login with new password that are changed in the CSRF POC.
* It successfully loged-in.

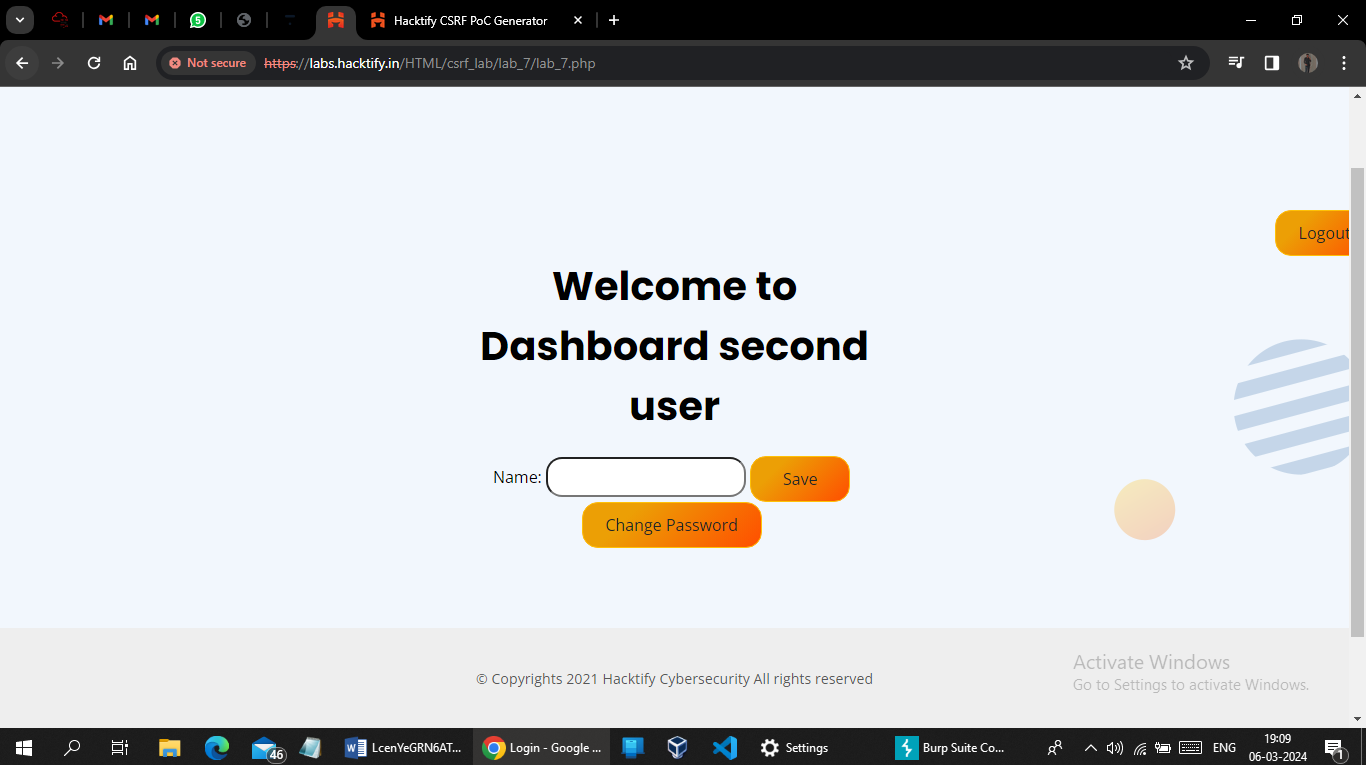










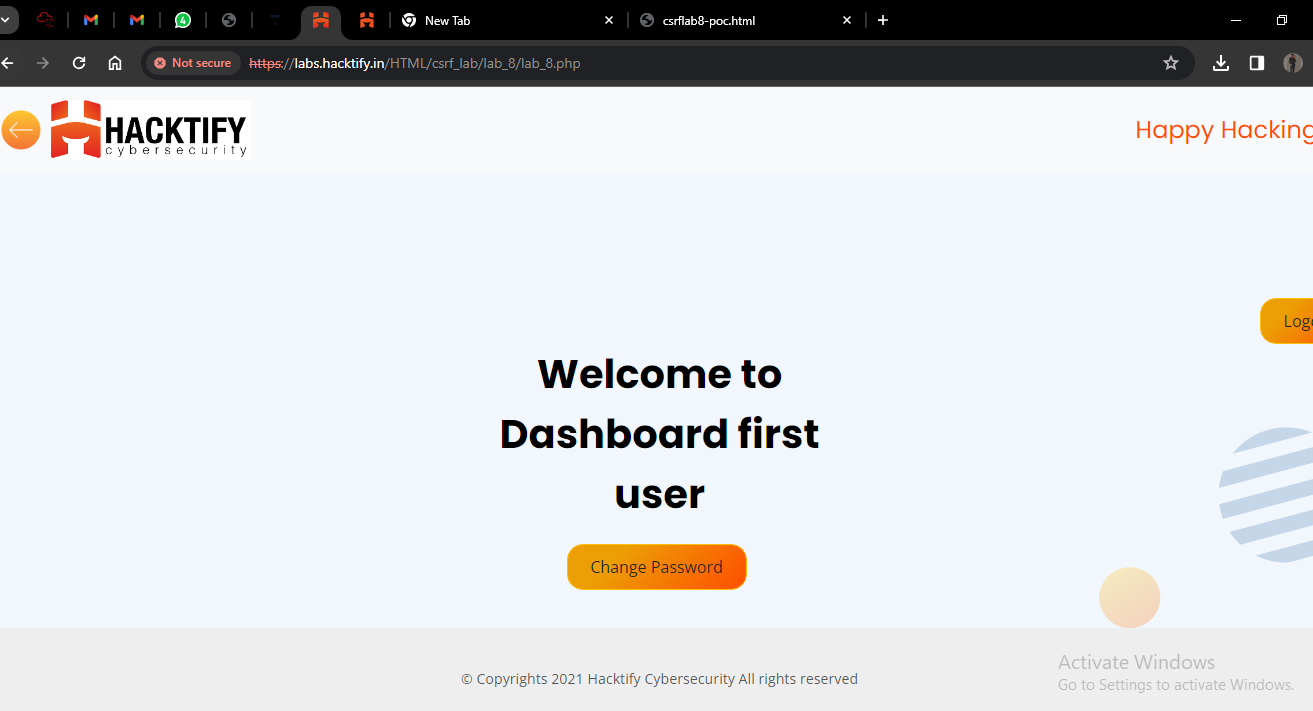


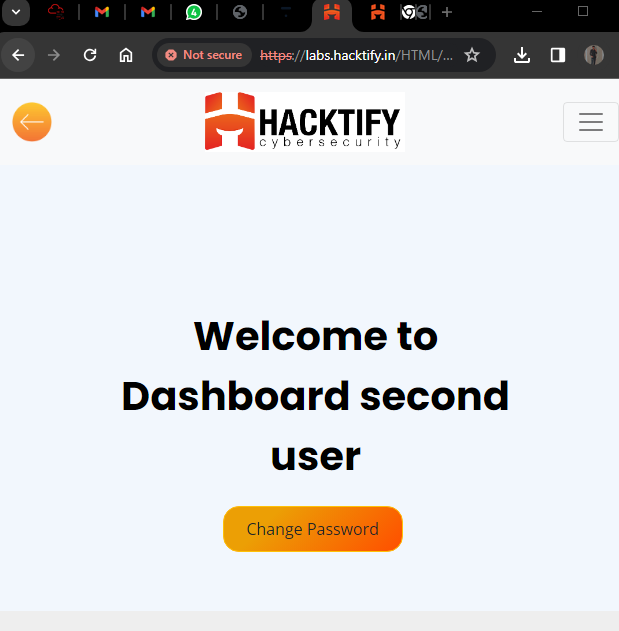
# 1.8. {Rm -Rf Token}

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| { Rm -Rf Token } | **High** |
| **Tools Used** | |
| Burp suite , CSRF POC Generator | |
| **Vulnerability Description** | |
| Cross-Site Request Forgery 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.  Cross-Site Request Forgery 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. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/csrf\_lab/lab\_8/passwordChange.php | |
| **Consequences of not Fixing the Issue** | |
| * Unauthorized actions * Data theft * Account compromised * Reputation damage * Financial losses | |
| **Suggested Countermeasures** | |
| * Implement security measures such as using :   CSRF tokens  Validate requests  Secure coding practices  Regular auditing  Educate people about CSRF attacks   * Avoid clicking on suspicious links | |
| **References** | |
| <https://portswigger.net/web-security/csrf>  <https://owasp.org/www-community/attacks/csrf>  <https://www.invicti.com/learn/cross-site-request-forgery-csrf/> | |

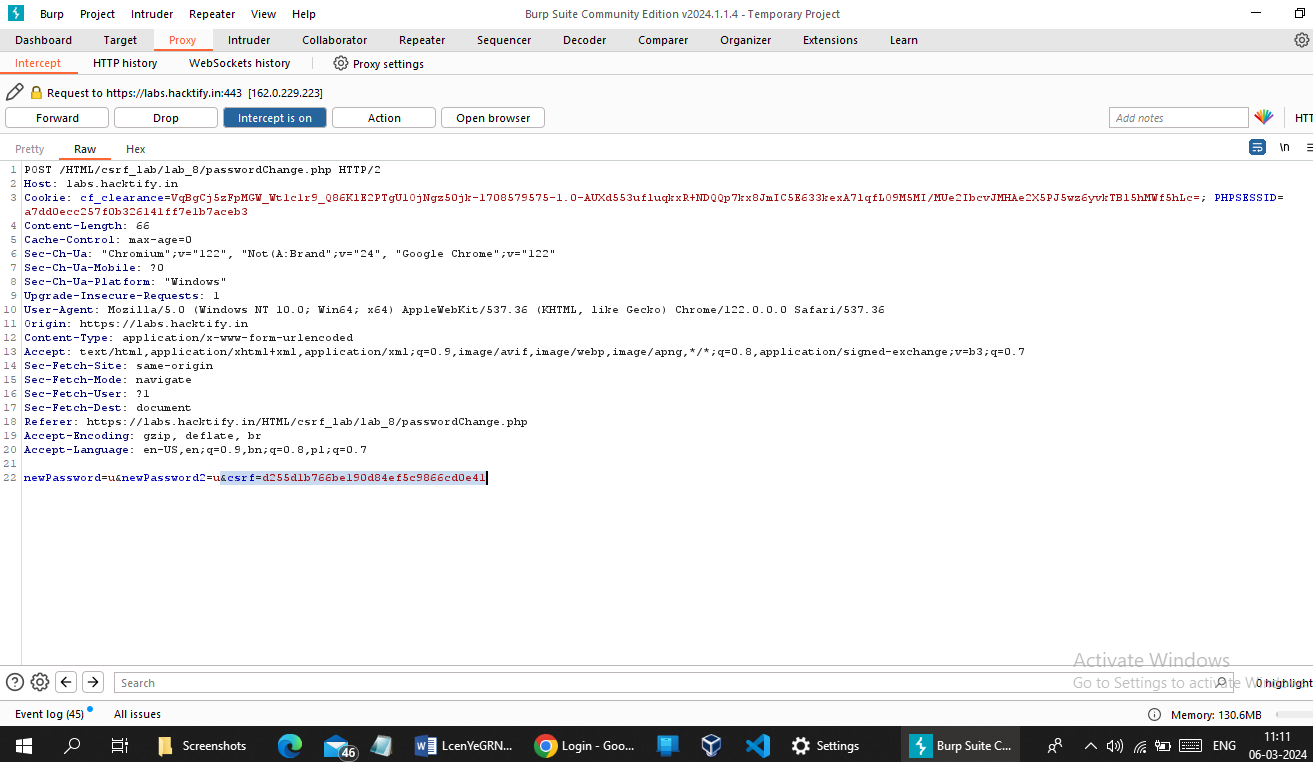
# Proof of Concept

First create two account one is first user and another is second user



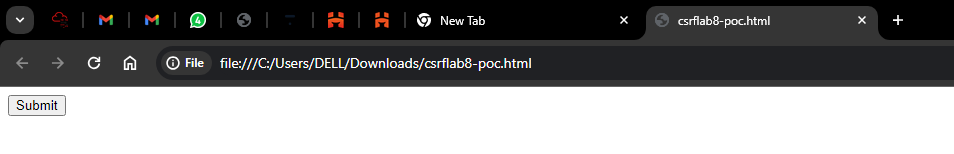


Remove the token and try generate CSRF POC





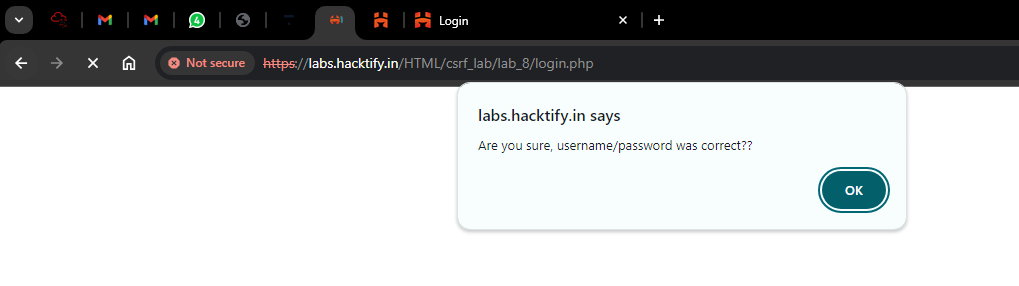
Create submit button using CSRF POC



Send it to the second user account to change their password unethicaly.

After click on submit button and successfully changed the password of second user .

When try to login into the second user account with old password it give



And try to login with new password which are used in CSRF POC it successfully loged-in.

# 2. {SSRF - Server-Side Request Forgery}

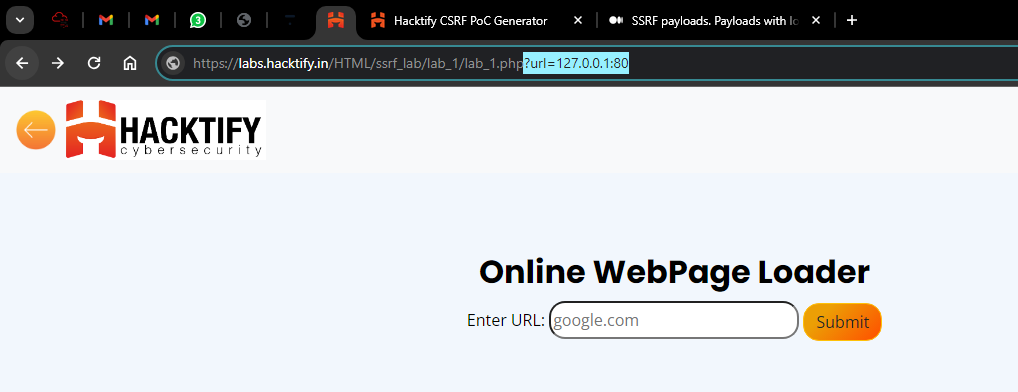
# 2.1. {Get The 127.0.0.1}

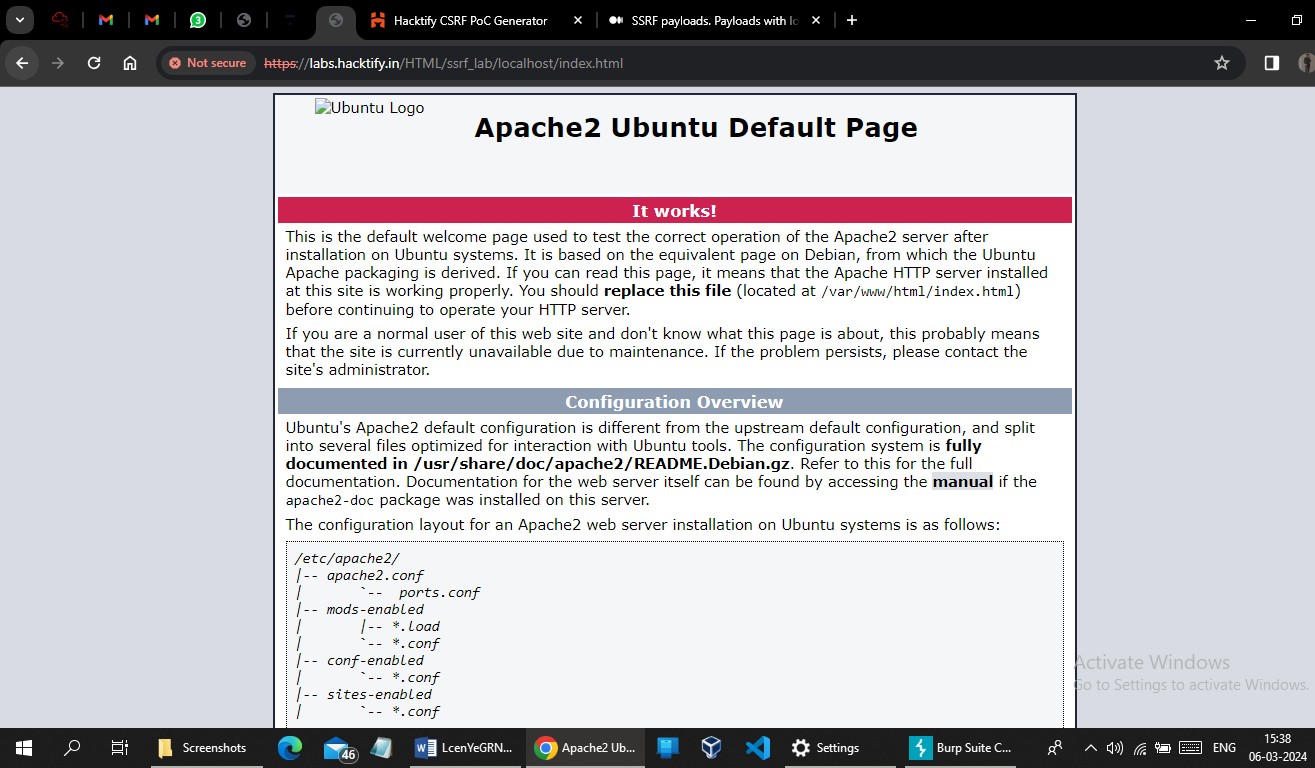
|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| {Get The 127.0.0.1} | **Low** |
| **Tools Used** | |
| SSRF payload | |
| **Vulnerability Description** | |
| Server-side request forgery (also known as SSRF) is a web security vulnerability that allows an attacker to induce the server-side application to make HTTP requests to an arbitrary domain of the attacker's choosing. In a typical SSRF attack, the attacker might cause the server to make a connection to internal-only services within the organization's infrastructure. In other cases, they may be able to force the server to connect to arbitrary external systems, potentially leaking sensitive data such as authorization credentials.  SSRF vulnerabilities let an attacker send crafted requests from the back-end server of a vulnerable application. Criminals usually use SSRF attacks to target internal systems that are behind firewalls and are not accessible from the external network.  SSRF vulnerabilities occur when an attacker has full or partial control of the request sent by the web application. A common example is when an attacker can control the third-party service URL to which the web application makes a request. | |
| **How It Was Discovered** | |
| Manual analysis - using ssrf paylaod | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/ssrf\_lab/lab\_1/lab\_1.php?url=127.0.0.1:80 | |
| **Consequences of not Fixing the Issue** | |
| * Data theft * Server compromised * Server disruption * Unauthorized access * Attacking internal infrastructure | |
| **Suggested Countermeasures** | |
| * Keep software up-to-date * Whitelists and DNS resolution * Response handling * Disable unused URL schemas * Authentication on internal services | |
| **References** | |
| <https://portswigger.net/web-security/ssrf>  <https://owasp.org/www-community/attacks/Server_Side_Request_Forgery>  <https://www.imperva.com/learn/application-security/server-side-request-forgery-ssrf/>  <https://www.geeksforgeeks.org/server-side-request-forgery-ssrf-in-depth/> | |

# 

# Proof of Concept

Payload : 127.0.0.1:80





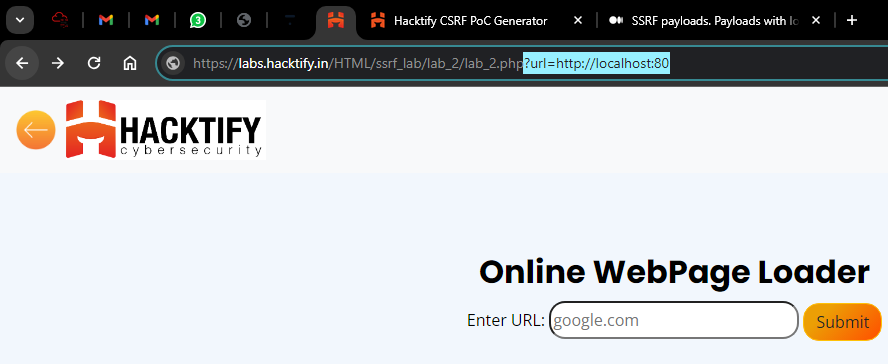
# 2.2. {Http(S)? Nevermind!!}

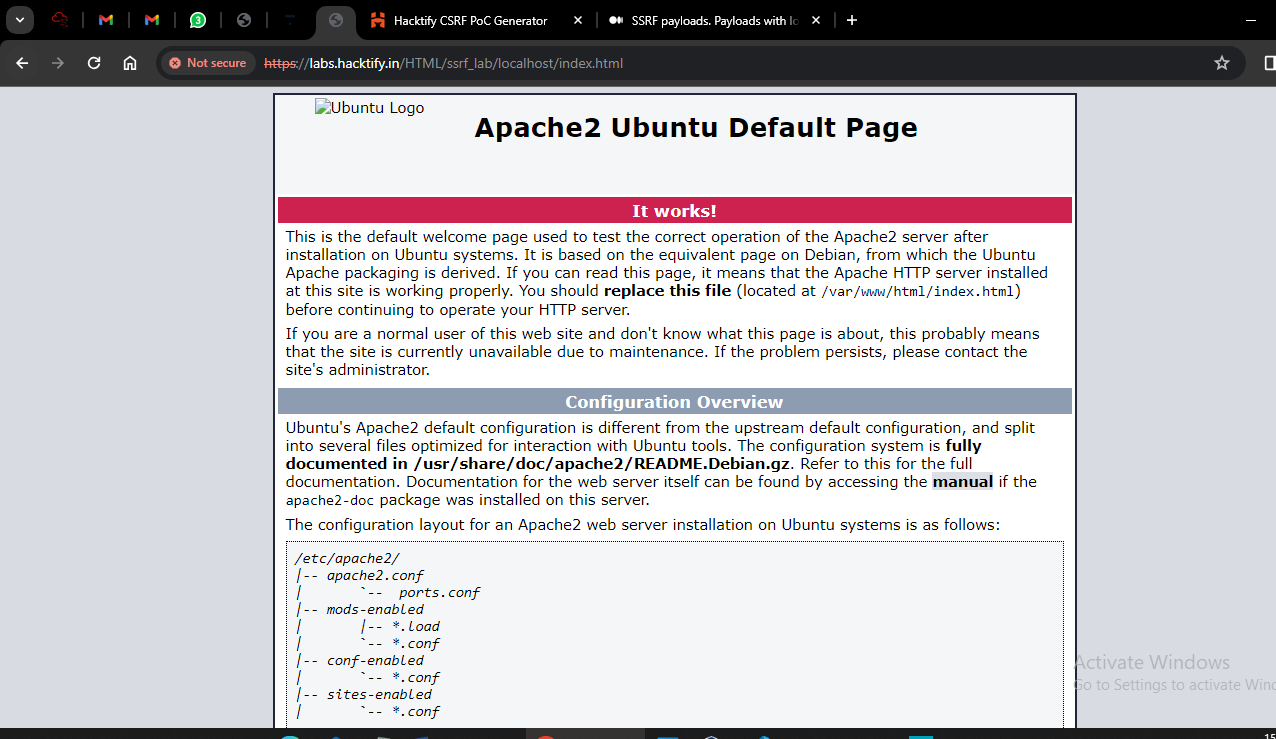
|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| { Http(S)? Nevermind !!} | **Low** |
| **Tools Used** | |
| SSRF payload | |
| **Vulnerability Description** | |
| Server-side request forgery (also known as SSRF) is a web security vulnerability that allows an attacker to induce the server-side application to make HTTP requests to an arbitrary domain of the attacker's choosing. In a typical SSRF attack, the attacker might cause the server to make a connection to internal-only services within the organization's infrastructure. In other cases, they may be able to force the server to connect to arbitrary external systems, potentially leaking sensitive data such as authorization credentials.  SSRF vulnerabilities let an attacker send crafted requests from the back-end server of a vulnerable application. Criminals usually use SSRF attacks to target internal systems that are behind firewalls and are not accessible from the external network.  SSRF vulnerabilities occur when an attacker has full or partial control of the request sent by the web application. A common example is when an attacker can control the third-party service URL to which the web application makes a request. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/ssrf\_lab/lab\_2/lab\_2.php?url=http://127.0.0.1:80 | |
| **Consequences of not Fixing the Issue** | |
| * Data theft * Server compromised * Server disruption * Unauthorized access * Attacking internal infrastructure | |
| **Suggested Countermeasures** | |
| * Keep software up-to-date * Whitelists and DNS resolution * Response handling * Disable unused URL schemas * Authentication on internal services | |
| **References** | |
| <https://portswigger.net/web-security/ssrf>  <https://owasp.org/www-community/attacks/Server_Side_Request_Forgery>  <https://www.imperva.com/learn/application-security/server-side-request-forgery-ssrf/>  <https://www.geeksforgeeks.org/server-side-request-forgery-ssrf-in-depth/> | |

# 

# Proof of Concept

Payload : http://localhost:80



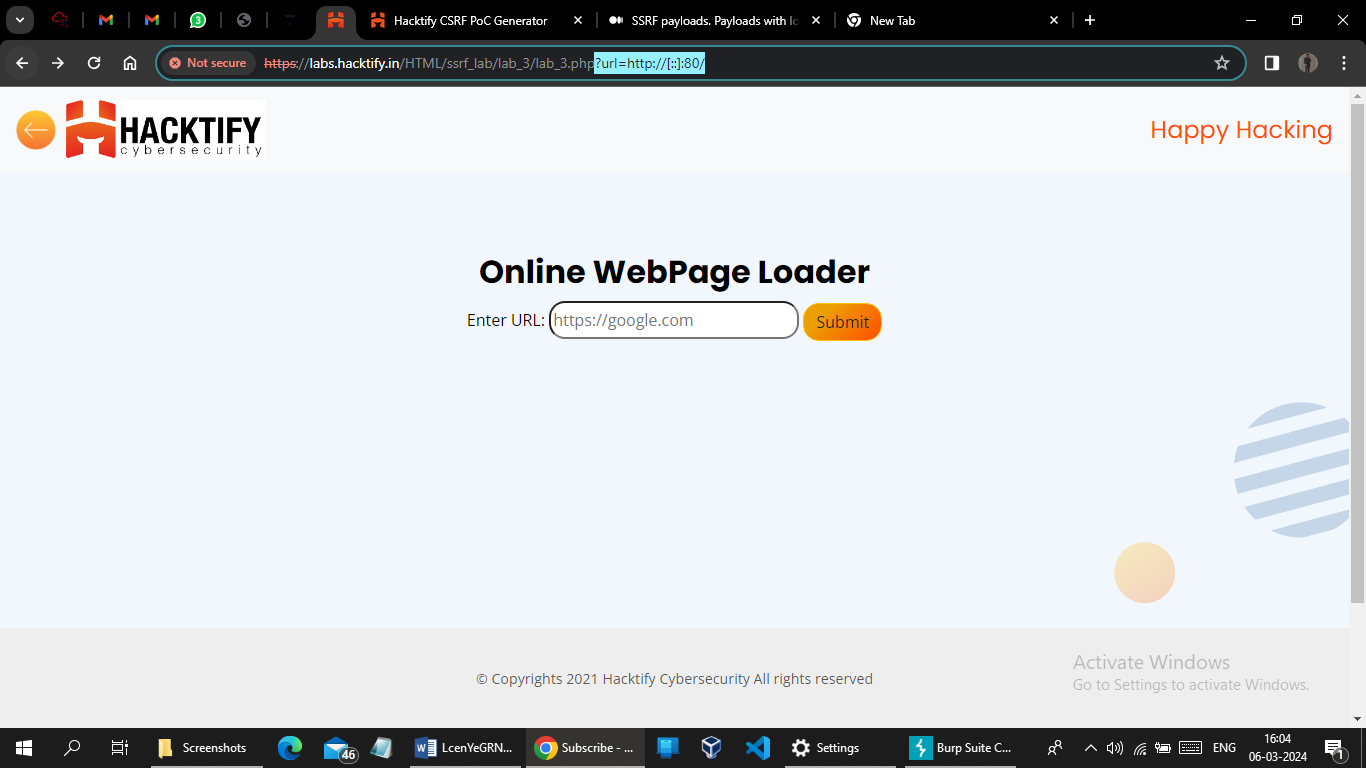


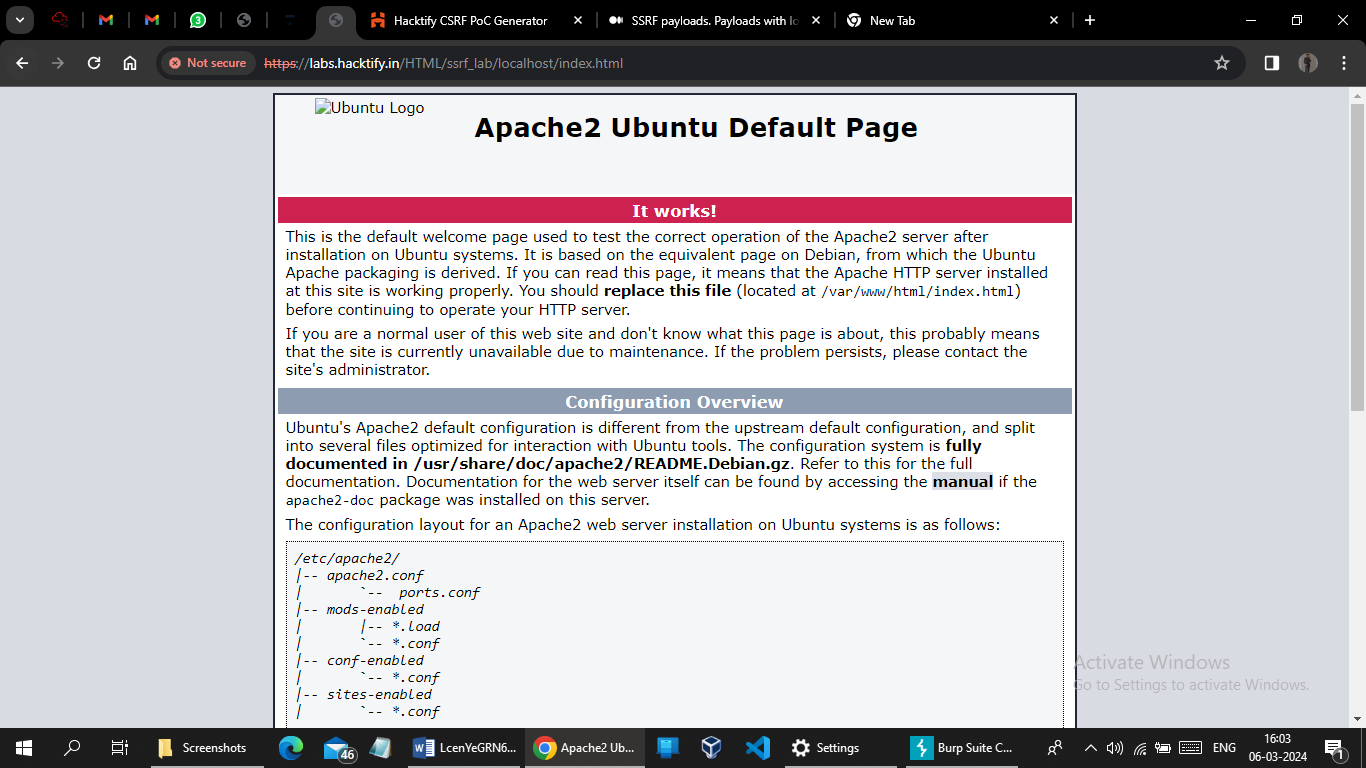
# 2.3. {":" The Saviour!}

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| {":" The Saviour!} | **Low** |
| **Tools Used** | |
| SSRF PAYLOAD | |
| **Vulnerability Description** | |
| Server-side request forgery (also known as SSRF) is a web security vulnerability that allows an attacker to induce the server-side application to make HTTP requests to an arbitrary domain of the attacker's choosing. In a typical SSRF attack, the attacker might cause the server to make a connection to internal-only services within the organization's infrastructure. In other cases, they may be able to force the server to connect to arbitrary external systems, potentially leaking sensitive data such as authorization credentials.  SSRF vulnerabilities let an attacker send crafted requests from the back-end server of a vulnerable application. Criminals usually use SSRF attacks to target internal systems that are behind firewalls and are not accessible from the external network.  SSRF vulnerabilities occur when an attacker has full or partial control of the request sent by the web application. A common example is when an attacker can control the third-party service URL to which the web application makes a request. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/ssrf\_lab/lab\_3/lab\_3.php?url=http://[::]:80/ | |
| **Consequences of not Fixing the Issue** | |
| * Data theft * Server compromised * Server disruption * Unauthorized access * Attacking internal infrastructure | |
| **Suggested Countermeasures** | |
| * Keep software up-to-date * Whitelists and DNS resolution * Response handling * Disable unused URL schemas * Authentication on internal services | |
| **References** | |
| <https://portswigger.net/web-security/ssrf>  <https://owasp.org/www-community/attacks/Server_Side_Request_Forgery>  <https://www.imperva.com/learn/application-security/server-side-request-forgery-ssrf/>  <https://www.geeksforgeeks.org/server-side-request-forgery-ssrf-in-depth/> | |

# Proof of Concept

Payload : url=http://[::]:80/



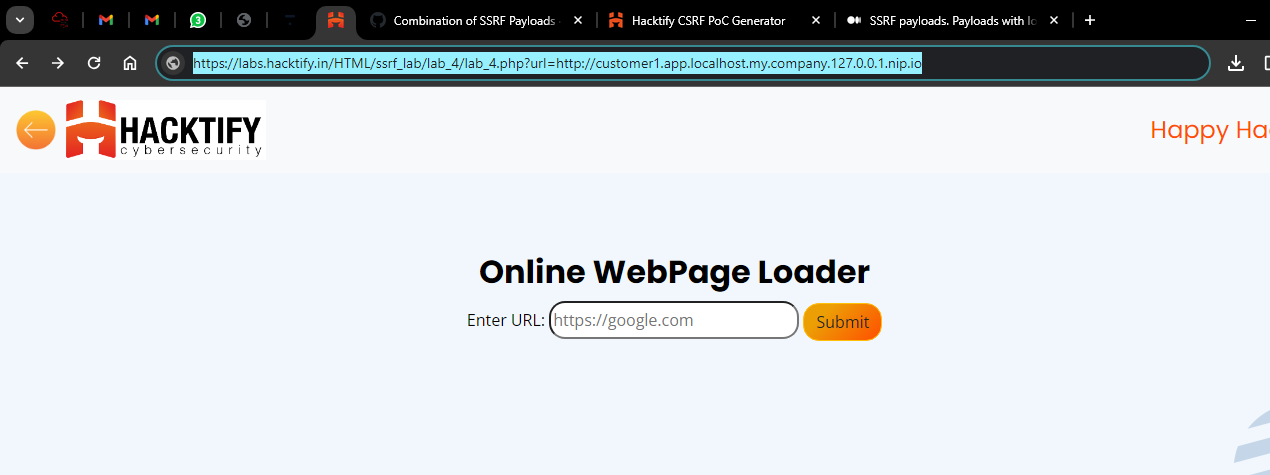


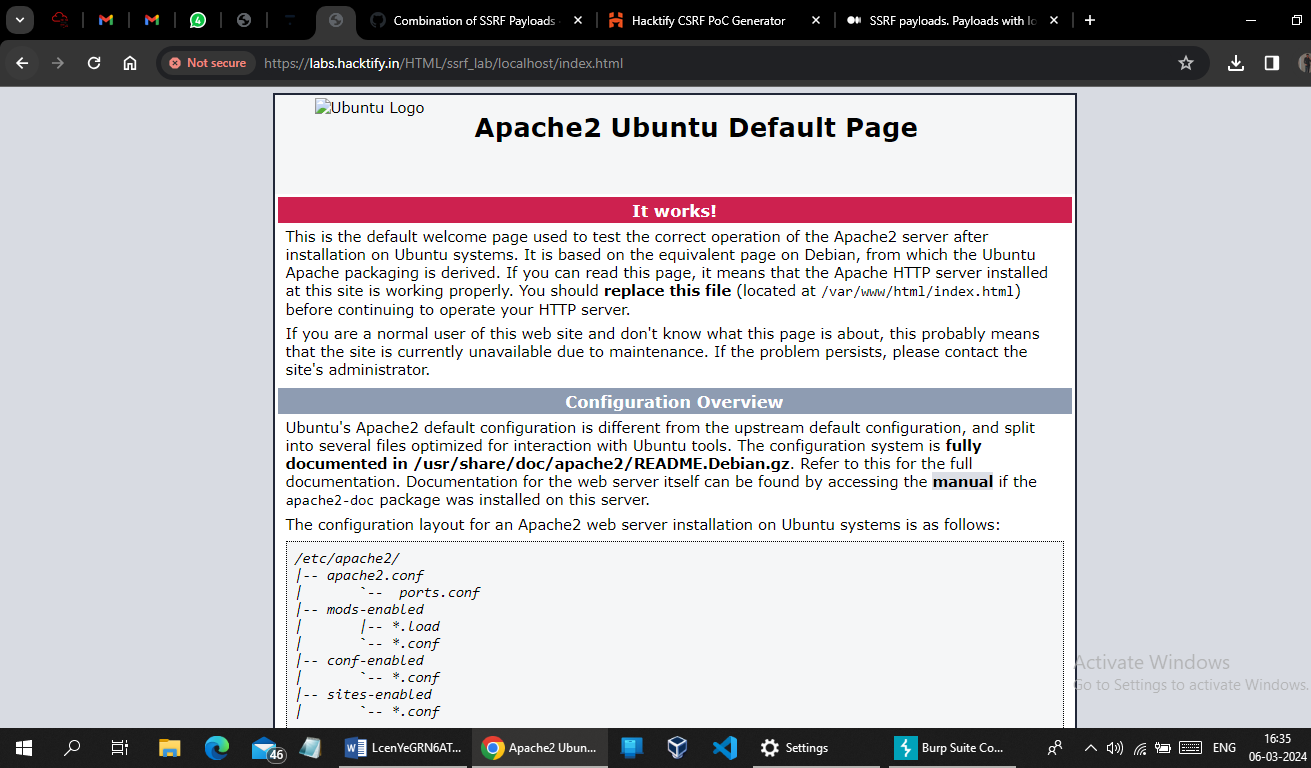
# 2.4. {Messed Up Domain!}

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| { Messed Up Domain!} | **Medium** |
| **Tools Used** | |
| SSRF payload | |
| **Vulnerability Description** | |
| Server-side request forgery (also known as SSRF) is a web security vulnerability that allows an attacker to induce the server-side application to make HTTP requests to an arbitrary domain of the attacker's choosing. In a typical SSRF attack, the attacker might cause the server to make a connection to internal-only services within the organization's infrastructure. In other cases, they may be able to force the server to connect to arbitrary external systems, potentially leaking sensitive data such as authorization credentials.  SSRF vulnerabilities let an attacker send crafted requests from the back-end server of a vulnerable application. Criminals usually use SSRF attacks to target internal systems that are behind firewalls and are not accessible from the external network.  SSRF vulnerabilities occur when an attacker has full or partial control of the request sent by the web application. A common example is when an attacker can control the third-party service URL to which the web application makes a request. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/ssrf\_lab/lab\_4/lab\_4.php?url=http://customer1.app.localhost.my.company.127.0.0.1.nip.io | |
| **Consequences of not Fixing the Issue** | |
| * Data theft * Server compromised * Server disruption * Unauthorized access * Attacking internal infrastructure | |
| **Suggested Countermeasures** | |
| * Keep software up-to-date * Whitelists and DNS resolution * Response handling * Disable unused URL schemas * Authentication on internal services | |
| **References** | |
| <https://portswigger.net/web-security/ssrf>  <https://owasp.org/www-community/attacks/Server_Side_Request_Forgery>  <https://www.imperva.com/learn/application-security/server-side-request-forgery-ssrf/>  <https://www.geeksforgeeks.org/server-side-request-forgery-ssrf-in-depth/> | |

# Proof of Concept

Payload : http://customer1.app.localhost.my.company.127.0.0.1.nip.io



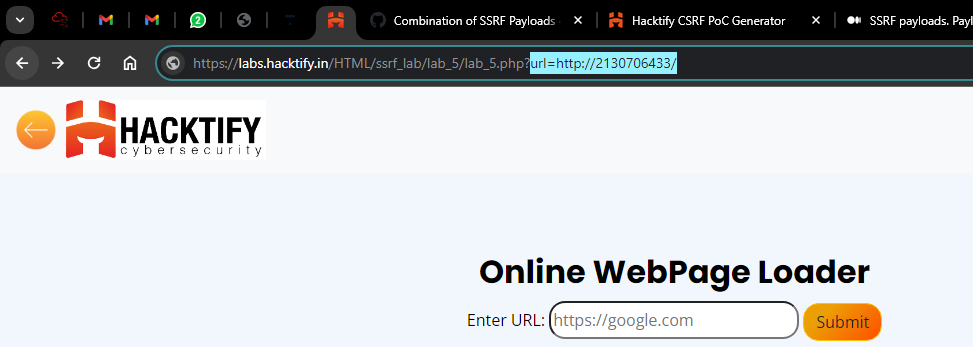


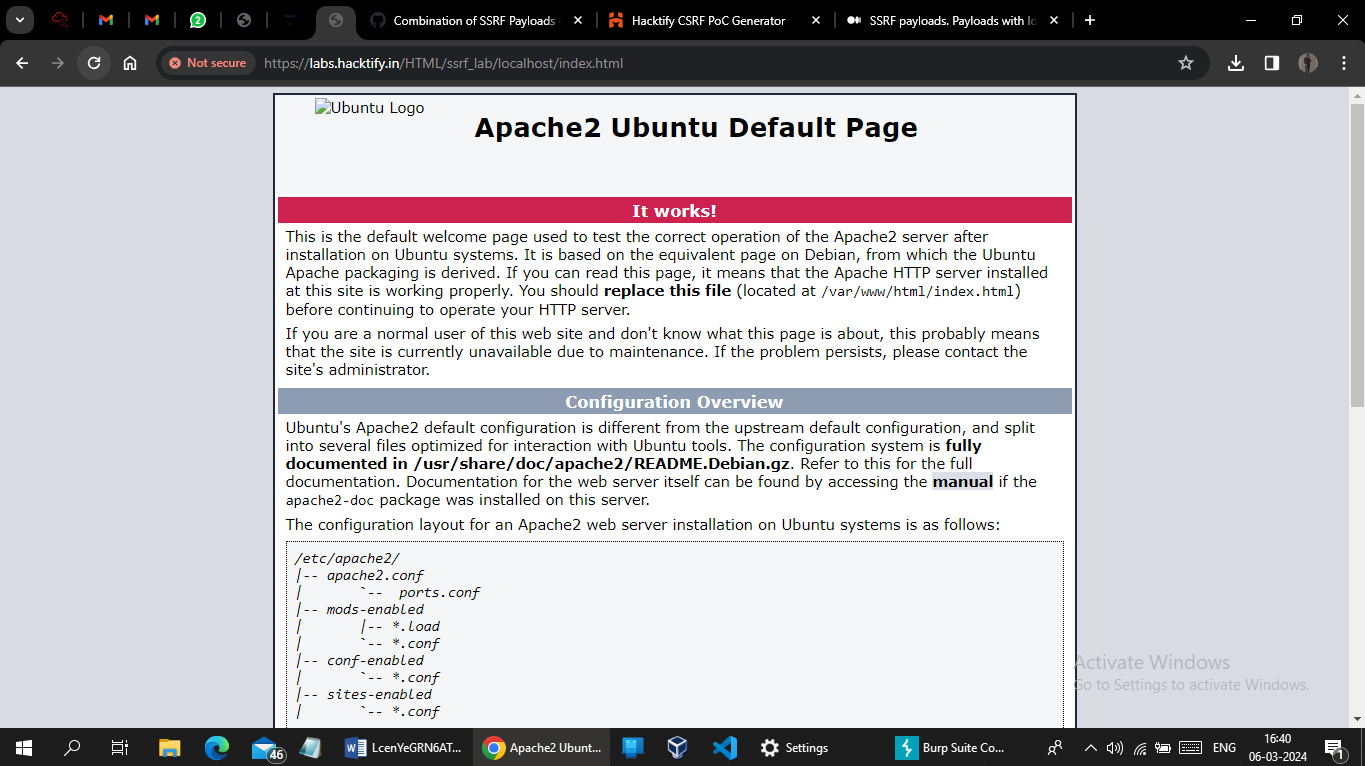
# 2.5. {Decimal IP }

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| {Sub-lab-2 Name} | **Medium** |
| **Tools Used** | |
| SSRF Payload | |
| **Vulnerability Description** | |
| Server-side request forgery (also known as SSRF) is a web security vulnerability that allows an attacker to induce the server-side application to make HTTP requests to an arbitrary domain of the attacker's choosing. In a typical SSRF attack, the attacker might cause the server to make a connection to internal-only services within the organization's infrastructure. In other cases, they may be able to force the server to connect to arbitrary external systems, potentially leaking sensitive data such as authorization credentials.  SSRF vulnerabilities let an attacker send crafted requests from the back-end server of a vulnerable application. Criminals usually use SSRF attacks to target internal systems that are behind firewalls and are not accessible from the external network.  SSRF vulnerabilities occur when an attacker has full or partial control of the request sent by the web application. A common example is when an attacker can control the third-party service URL to which the web application makes a request. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/ssrf\_lab/lab\_5/lab\_5.php?url=http://2130706433/ | |
| **Consequences of not Fixing the Issue** | |
| * Data theft * Server compromised * Server disruption * Unauthorized access * Attacking internal infrastructure | |
| **Suggested Countermeasures** | |
| * Keep software up-to-date * Whitelists and DNS resolution * Response handling * Disable unused URL schemas * Authentication on internal services | |
| **References** | |
| <https://portswigger.net/web-security/ssrf>  <https://owasp.org/www-community/attacks/Server_Side_Request_Forgery>  <https://www.imperva.com/learn/application-security/server-side-request-forgery-ssrf/>  <https://www.geeksforgeeks.org/server-side-request-forgery-ssrf-in-depth/> | |

# Proof of Concept

Payload : url=http://2130706433/





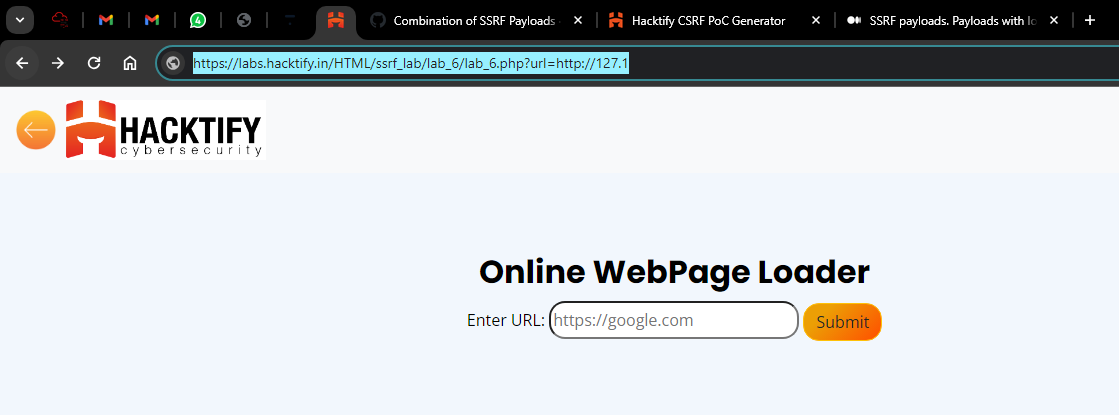
# 2.6. {Short-Hand IP Address}

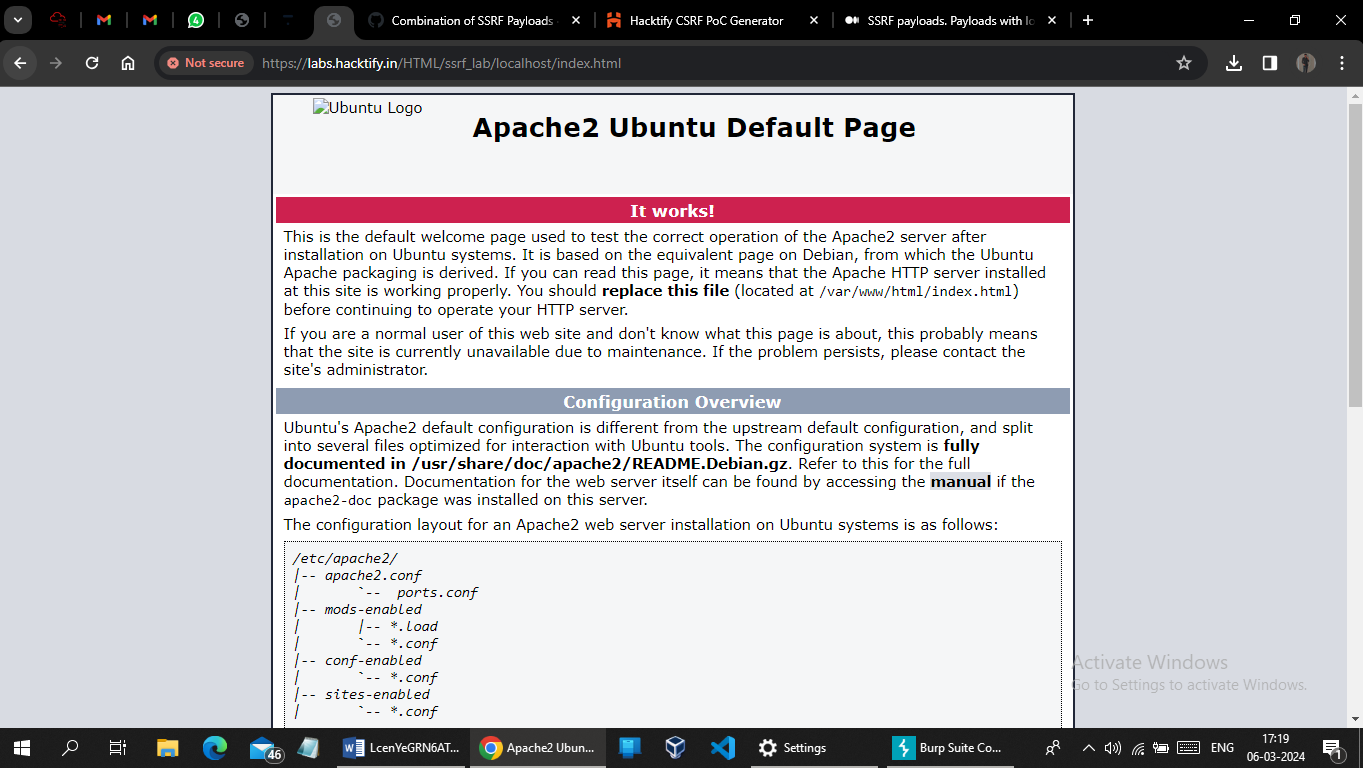
|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| { Short-Hand IP Address } | **Medium** |
| **Tools Used** | |
| SSRF payload | |
| **Vulnerability Description** | |
| Server-side request forgery (also known as SSRF) is a web security vulnerability that allows an attacker to induce the server-side application to make HTTP requests to an arbitrary domain of the attacker's choosing. In a typical SSRF attack, the attacker might cause the server to make a connection to internal-only services within the organization's infrastructure. In other cases, they may be able to force the server to connect to arbitrary external systems, potentially leaking sensitive data such as authorization credentials.  SSRF vulnerabilities let an attacker send crafted requests from the back-end server of a vulnerable application. Criminals usually use SSRF attacks to target internal systems that are behind firewalls and are not accessible from the external network.  SSRF vulnerabilities occur when an attacker has full or partial control of the request sent by the web application. A common example is when an attacker can control the third-party service URL to which the web application makes a request. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/ssrf\_lab/lab\_6/lab\_6.php?url=http://127.1 | |
| **Consequences of not Fixing the Issue** | |
| * Data theft * Server compromised * Server disruption * Unauthorized access * Attacking internal infrastructure | |
| **Suggested Countermeasures** | |
| * Keep software up-to-date * Whitelists and DNS resolution * Response handling * Disable unused URL schemas * Authentication on internal services | |
| **References** | |
| <https://portswigger.net/web-security/ssrf>  <https://owasp.org/www-community/attacks/Server_Side_Request_Forgery>  <https://www.imperva.com/learn/application-security/server-side-request-forgery-ssrf/>  <https://www.geeksforgeeks.org/server-side-request-forgery-ssrf-in-depth/> | |

# Proof of Concept

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

Payload : http://127.1





# 2.7. {File Upload To SSRF! }

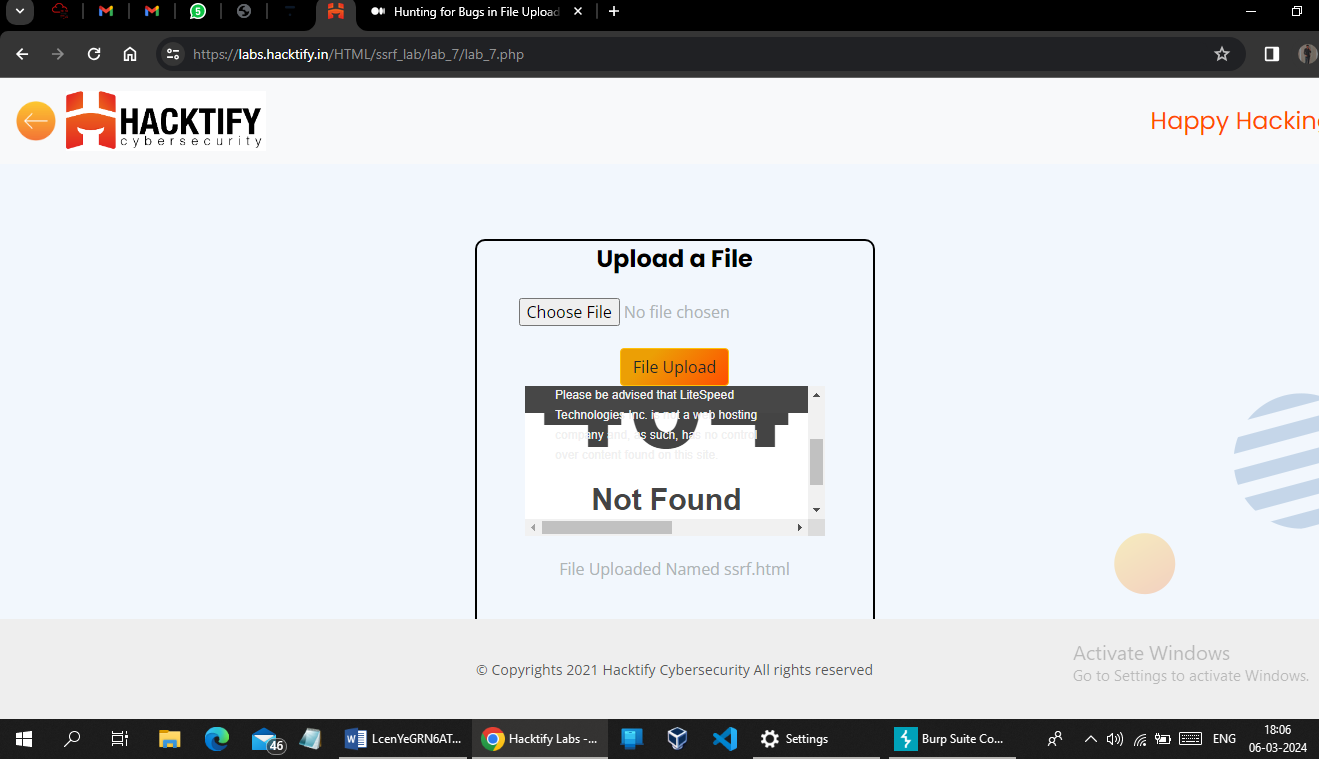
|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| { File Upload To SSRF! } | **Low** |
| **Tools Used** | |
| SSRF payload | |
| **Vulnerability Description** | |
| Server-side request forgery (also known as SSRF) is a web security vulnerability that allows an attacker to induce the server-side application to make HTTP requests to an arbitrary domain of the attacker's choosing. In a typical SSRF attack, the attacker might cause the server to make a connection to internal-only services within the organization's infrastructure. In other cases, they may be able to force the server to connect to arbitrary external systems, potentially leaking sensitive data such as authorization credentials.  SSRF vulnerabilities let an attacker send crafted requests from the back-end server of a vulnerable application. Criminals usually use SSRF attacks to target internal systems that are behind firewalls and are not accessible from the external network.  SSRF vulnerabilities occur when an attacker has full or partial control of the request sent by the web application. A common example is when an attacker can control the third-party service URL to which the web application makes a request. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/ssrf\_lab/lab\_7/lab\_7.php | |
| **Consequences of not Fixing the Issue** | |
| * Data theft * Server compromised * Server disruption * Unauthorized access * Attacking internal infrastructure | |
| **Suggested Countermeasures** | |
| * Keep software up-to-date * Whitelists and DNS resolution * Response handling * Disable unused URL schemas * Authentication on internal services | |
| **References** | |
| <https://portswigger.net/web-security/ssrf>  <https://owasp.org/www-community/attacks/Server_Side_Request_Forgery>  <https://www.imperva.com/learn/application-security/server-side-request-forgery-ssrf/>  <https://www.geeksforgeeks.org/server-side-request-forgery-ssrf-in-depth/> | |

# Proof of Concept

Did not understand what is the actual output of this lab.

I tried to upload one .html file which contain iframe tag and upload it.

So I got this output

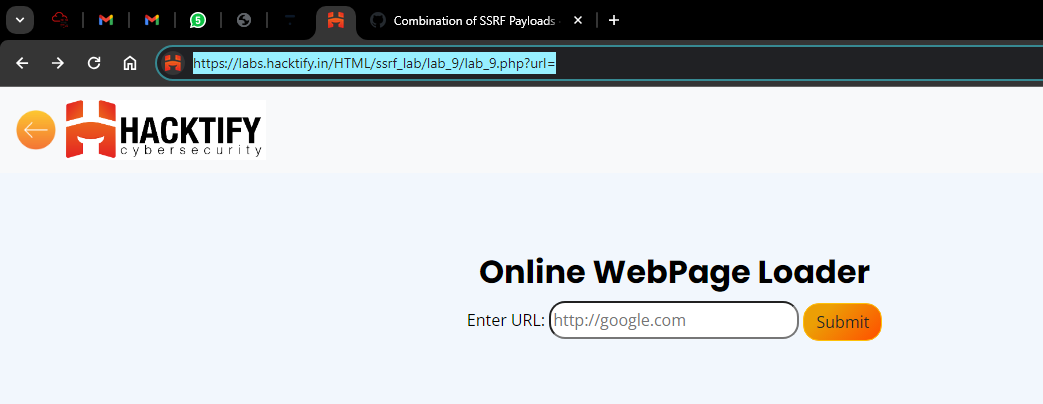


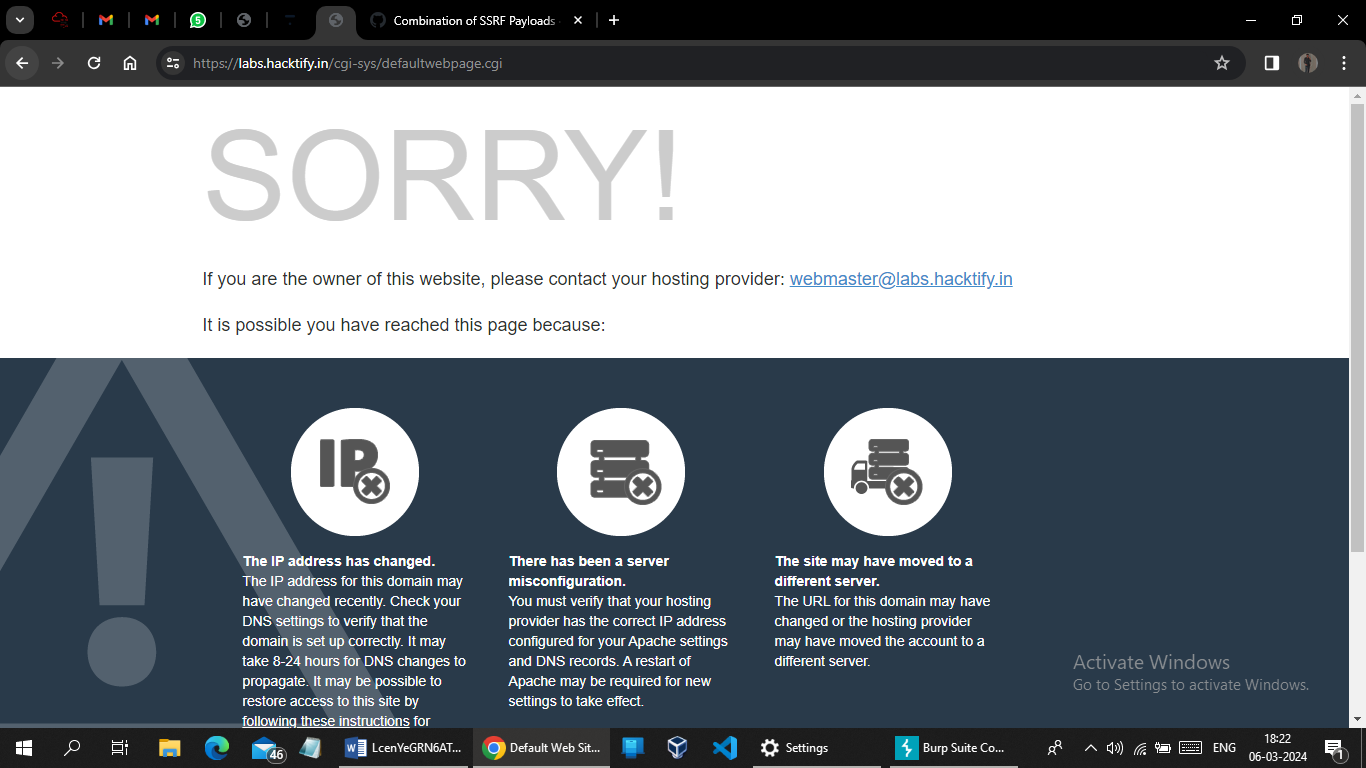
# 2.8. {SSRF With DNS Rebinding}

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| { SSRF With DNS Rebinding } | **High** |
| **Tools Used** | |
| SSRF payload | |
| **Vulnerability Description** | |
| Server-side request forgery (also known as SSRF) is a web security vulnerability that allows an attacker to induce the server-side application to make HTTP requests to an arbitrary domain of the attacker's choosing. In a typical SSRF attack, the attacker might cause the server to make a connection to internal-only services within the organization's infrastructure. In other cases, they may be able to force the server to connect to arbitrary external systems, potentially leaking sensitive data such as authorization credentials.  SSRF vulnerabilities let an attacker send crafted requests from the back-end server of a vulnerable application. Criminals usually use SSRF attacks to target internal systems that are behind firewalls and are not accessible from the external network.  SSRF vulnerabilities occur when an attacker has full or partial control of the request sent by the web application. A common example is when an attacker can control the third-party service URL to which the web application makes a request. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/ssrf\_lab/lab\_9/lab\_9.php | |
| **Consequences of not Fixing the Issue** | |
| * Data theft * Server compromised * Server disruption * Unauthorized access * Attacking internal infrastructure | |
| **Suggested Countermeasures** | |
| * Keep software up-to-date * Whitelists and DNS resolution * Response handling * Disable unused URL schemas * Authentication on internal services | |
| **References** | |
| <https://portswigger.net/web-security/ssrf>  <https://owasp.org/www-community/attacks/Server_Side_Request_Forgery>  <https://www.imperva.com/learn/application-security/server-side-request-forgery-ssrf/>  <https://www.geeksforgeeks.org/server-side-request-forgery-ssrf-in-depth/> | |

# Proof of Concept

# When i simple type “ url= “ in url field and hit enter it is redirect me to cpanel (<https://labs.hacktify.in/cgi-sys/defaultwebpage.cgi>) of the website.





# 2.9. {Look An SSRF On Cloud!}

|  |  |
| --- | --- |
| **Reference** | **Risk Rating** |
| { Look An SSRF On Cloud!} | **High** |
| **Tools Used** | |
| SSRF payload | |
| **Vulnerability Description** | |
| Server-side request forgery (also known as SSRF) is a web security vulnerability that allows an attacker to induce the server-side application to make HTTP requests to an arbitrary domain of the attacker's choosing. In a typical SSRF attack, the attacker might cause the server to make a connection to internal-only services within the organization's infrastructure. In other cases, they may be able to force the server to connect to arbitrary external systems, potentially leaking sensitive data such as authorization credentials.  SSRF vulnerabilities let an attacker send crafted requests from the back-end server of a vulnerable application. Criminals usually use SSRF attacks to target internal systems that are behind firewalls and are not accessible from the external network.  SSRF vulnerabilities occur when an attacker has full or partial control of the request sent by the web application. A common example is when an attacker can control the third-party service URL to which the web application makes a request. | |
| **How It Was Discovered** | |
| Manual Analysis | |
| **Vulnerable URLs** | |
| https://labs.hacktify.in/HTML/ssrf\_lab/lab\_10/lab\_10.php?url=127.0.0.1:80 | |
| **Consequences of not Fixing the Issue** | |
| * Data theft * Server compromised * Server disruption * Unauthorized access * Attacking internal infrastructure | |
| **Suggested Countermeasures** | |
| * Keep software up-to-date * Whitelists and DNS resolution * Response handling * Disable unused URL schemas * Authentication on internal services | |
| **References** | |
| <https://portswigger.net/web-security/ssrf>  <https://owasp.org/www-community/attacks/Server_Side_Request_Forgery>  <https://www.imperva.com/learn/application-security/server-side-request-forgery-ssrf/>  <https://www.geeksforgeeks.org/server-side-request-forgery-ssrf-in-depth/> | |

# Proof of Concept

Payload : url=127.0.0.1:80

