Penetration Testing Report

Full Name: Sahil Naik
Program: HCPT

Date: 27/02/2025

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	**Lab 1 : Cross-Origin Resource Sharing,** **Lab 2 : Cross-Site Request Forgery.**

3. Summary

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

Total number of Sub-labs: {count} Sub-labs

High	Medium	Low
5	4	4

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

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

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

1. Cross-Origin Resource Sharing

1.1. CORS With Arbitrary Origin

Reference	**Risk Rating**
CORS With Arbitrary Origin	**Low**
Tools Used	
BurpSuite	
Vulnerability Description	
CORS (Cross-Origin Resource Sharing) vulnerability arises when a web server misconfigures its CORS policy, allowing untrusted origins to access sensitive data. Attackers can exploit this to steal user information or perform unauthorized actions.	
How It Was Discovered / Steps to Reproduce	
1. Capture the request using Burp Suite and identify an API endpoint with CORS headers. 2. Modify the Origin header to an arbitrary domain and resend the request.	

3. Check if the response includes Access-Control-Allow-Origin with the injected domain.
4. If credentials are used, verify Access-Control-Allow-Credentials: true to confirm exploitation.

Vulnerable URLs

https://labs.hacktify.in/HTML/cors_lab/lab_1/cors_1.php

Consequences of not Fixing the Issue

1. Data Theft: Attackers can steal sensitive user data from APIs.
2. Account Takeover: Exploiting exposed credentials via a poorly configured CORS policy.
3. Privilege Escalation: Unauthorized actions can be performed on behalf of a victim.

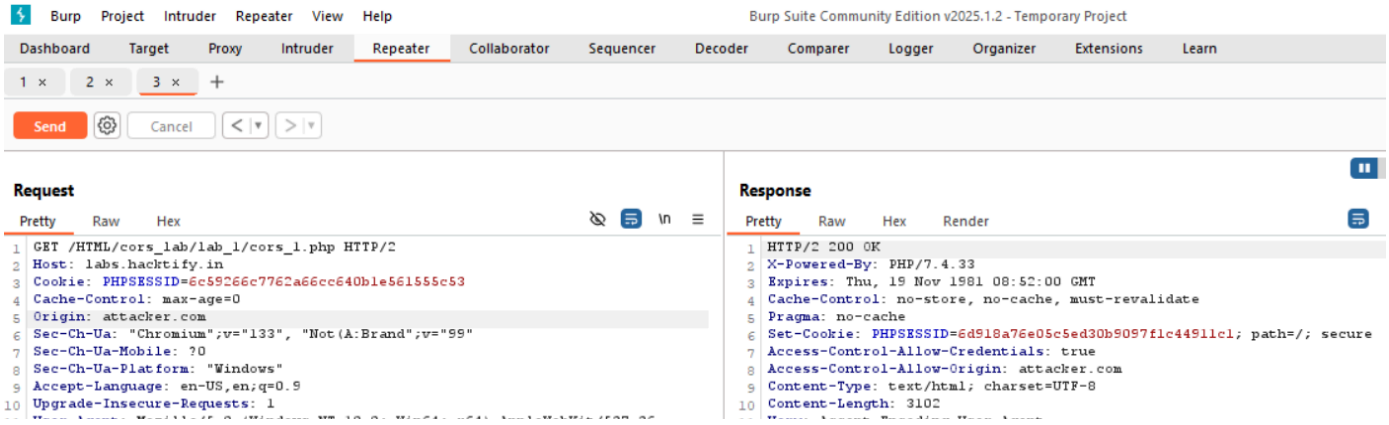
Suggested Countermeasures

1. Implement a strict allowlist for trusted origins instead of using * or dynamically reflecting origins.
2. Restrict allowed HTTP methods and credentials (e.g., Access-Control-Allow-Credentials: false).
3. Use server-side validation to prevent unauthorized access to sensitive API endpoints.

References

https://portswigger.net/web-security/cors

Proof of Concept

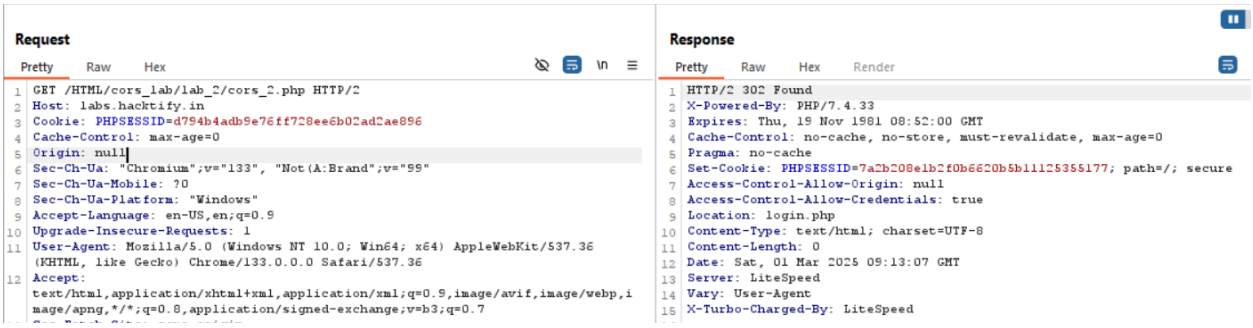


1.2. CORS with Null origin

Reference	**Risk Rating**
CORS with Null origin	**Low**
Tools Used	
BurpSuite	
Vulnerability Description	
This vulnerability occurs when a web server improperly allows Origin: null, enabling attackers to bypass CORS restrictions using sandboxed environments like iframes, data URLs, or local HTML files. This can lead to unauthorized access to sensitive data.	
How It Was Discovered / Steps to Reproduce	
1. Intercept a request in Burp Suite and add Origin: null in the request headers. 2. Send the request and check if the response includes Access-Control-Allow-Origin: null. 3. If Access-Control-Allow-Credentials: true is present, attempt to access sensitive resources. 4. Craft an exploit using a sandboxed iframe or a local HTML file to exfiltrate user data.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/cors_lab/lab_2/cors_2.php	
Consequences of not Fixing the Issue	
1. Data Leakage: Attackers can extract sensitive information from APIs. 2. Exploitation via Local Files: Malicious HTML files loaded locally can abuse null origin to exfiltrate data. 3. Cross-Domain Attacks: If null is accepted, attackers can abuse it to access private resources from restricted origins.	

Suggested Countermeasures
1. Reject requests with Origin: null unless explicitly required for specific use cases. 2. Implement a strict allowlist for trusted origins instead of allowing null. 3. Use server-side authentication and access controls to protect sensitive data.
References
https://portswigger.net/web-security/cors

Proof of Concept



1.3. CORS with prefix match

Reference	Risk Rating
CORS with prefix match	**Medium**
Tools Used	
BurpSuite	
Vulnerability Description	

CORS vulnerability with prefix matching occurs when a server improperly validates origins by using a partial string match instead of an exact match, allowing attackers to craft subdomains (e.g., evil.example.com) that bypass security checks intended for example.com.

How It Was Discovered / Steps to Reproduce

1. Intercept a request in Burp Suite and modify the Origin header to a controlled subdomain (e.g., evil.example.com).
2. Send the request and check if the response includes Access-Control-Allow-Origin: evil.example.com.
3. If Access-Control-Allow-Credentials: true is present, attempt to access sensitive resources.
4. Use an attacker-controlled subdomain to steal user data via a malicious webpage.

Vulnerable URLs

https://labs.hacktify.in/HTML/cors_lab/lab_3/cors_3.php

Consequences of not Fixing the Issue

1. Unauthorized Access: Attackers can create similar-looking subdomains to gain access to sensitive data.
2. Session Hijacking: If Access-Control-Allow-Credentials: true is enabled, an attacker can steal user sessions.
3. API Exploitation: Private API endpoints may be exposed to malicious origins.

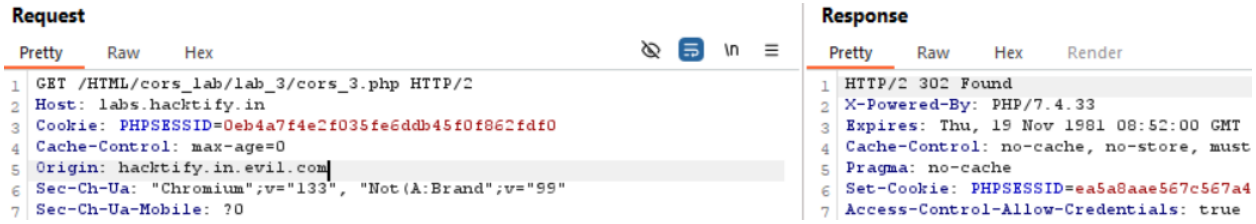
Suggested Countermeasures

1. Use exact origin matching instead of prefix-based or partial string matches.
2. Implement a strict allowlist of trusted origins on the server.
3. Avoid dynamically reflecting the Origin header in Access-Control-Allow-Origin.

References

https://portswigger.net/web-security/cors

Proof of Concept

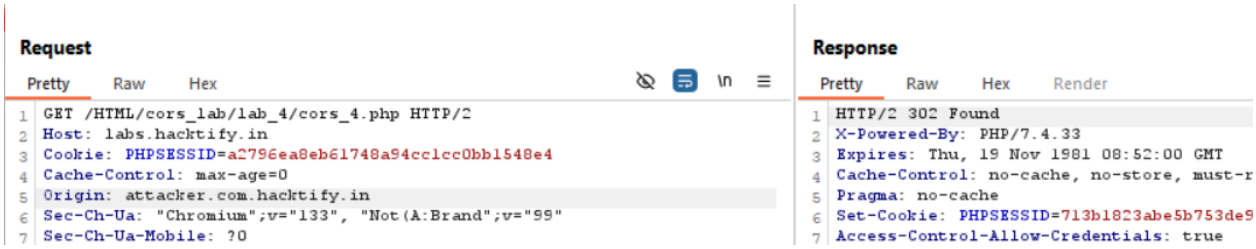


1.4. CORS with suffix match

Reference	**Risk Rating**
CORS with suffix match	**Medium**
Tools Used	
BurpSuite	
Vulnerability Description	
CORS vulnerability with suffix matching occurs when a server incorrectly validates origins by allowing any domain that ends with a trusted string (e.g., .example.com). This enables attackers to register malicious domains like attack-example.com to bypass security controls.	
How It Was Discovered / Steps to Reproduce	
1. Intercept a request in Burp Suite and modify the Origin header to a controlled domain with a matching suffix (e.g., attack-example.com). 2. Send the request and check if the response includes Access-Control-Allow-Origin: attack-example.com. 3. If Access-Control-Allow-Credentials: true is present, attempt to access sensitive resources. 4. Use an attacker-controlled domain to execute a cross-origin request and exfiltrate user data.	
Vulnerable URLs	

https://labs.hacktify.in/HTML/cors_lab/lab_4/cors_4.php
Consequences of not Fixing the Issue
1. Data Exposure: Attackers can host malicious websites on controlled domains that match the trusted suffix. 2. Session Hijacking: If Access-Control-Allow-Credentials: true is enabled, attackers can steal session data. 3. Unauthorized API Access: Sensitive endpoints may be accessible from attacker-controlled domains.
Suggested Countermeasures
1. Use strict, exact origin matching instead of suffix-based validation. 2. Allow only fully qualified domains instead of using wildcards like *.example.com. 3. Implement server-side authentication to restrict sensitive data access.
References
https://portswigger.net/web-security/cors

Proof of Concept



1.5. CORS with Escape dot

Reference	Risk Rating
CORS with Escape dot	**High**

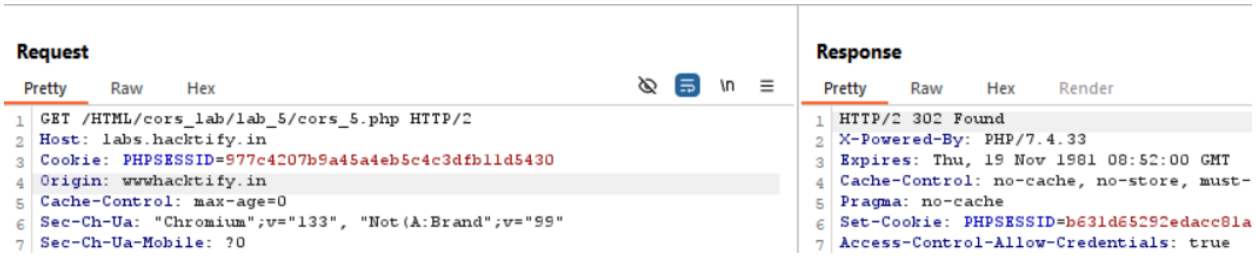
Tools Used
BurpSuite
Vulnerability Description
A CORS vulnerability with escape dot (\.) occurs when a web server incorrectly validates origins by using improper regex matching, allowing attackers to bypass security restrictions. For example, if the server allows trusted.com but uses a flawed regex like trusted\.com, an attacker could use trustedxcom to evade checks.
How It Was Discovered / Steps to Reproduce
1. Intercept a request in Burp Suite and modify the Origin header to a bypass variant (e.g., trustedxcom instead of trusted.com). 2. Send the request and check if the response includes Access-Control-Allow-Origin: trustedxcom. 3. If Access-Control-Allow-Credentials: true is present, attempt to access sensitive resources. 4. Host an attacker-controlled domain that exploits the escape dot misconfiguration to steal user data.
Vulnerable URLs
https://labs.hacktify.in/HTML/cors_lab/lab_5/cors_5.php
Consequences of not Fixing the Issue
1. Bypassing Origin Checks: Attackers can craft domains that match the misconfigured regex and gain access. 2. Session Hijacking: If Access-Control-Allow-Credentials: true is set, attackers can steal authentication tokens. 3. Data Exfiltration: Sensitive API responses may be exposed to attacker-controlled domains.
Suggested Countermeasures

1. Use strict, exact string comparisons instead of flawed regex patterns.
2. Avoid dynamically reflecting the Origin header in Access-Control-Allow-Origin.
3. Implement server-side authentication and authorization to prevent unauthorized access.

References

https://portswigger.net/web-security/cors

Proof of Concept

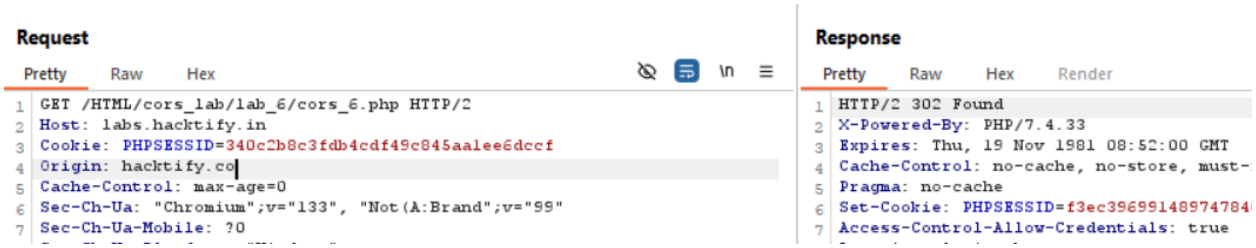


1.6. CORS with Substring match

Reference	**Risk Rating**
CORS with Substring match	**High**
Tools Used	
BurpSuite	
Vulnerability Description	
A CORS vulnerability with substring matching occurs when a web server improperly validates origins by checking if a trusted string appears anywhere in the Origin header. This allows attackers to craft domains like eviltrusted.com to bypass security policies intended for trusted.com.	

How It Was Discovered / Steps to Reproduce
1. Intercept a request in Burp Suite and modify the Origin header to an attacker-controlled domain containing the trusted domain as a substring (e.g., eviltrusted.com). 2. Send the request and check if the response includes Access-Control-Allow-Origin: eviltrusted.com. 3. If Access-Control-Allow-Credentials: true is present, attempt to access sensitive resources. 4. Host an attacker-controlled domain that exploits the substring match misconfiguration to exfiltrate user data.
Vulnerable URLs
https://labs.hacktify.in/HTML/cors_lab/lab_6/cors_6.php
Consequences of not Fixing the Issue
1. Unauthorized Access: Attackers can register similar-looking domains to gain access to sensitive data. 2. Session Hijacking: If Access-Control-Allow-Credentials: true is enabled, attackers can steal session cookies. 3. API Exploitation: Private API endpoints may become accessible to untrusted origins.
Suggested Countermeasures
1. Use exact string matching instead of substring checks for allowed origins. 2. Implement a strict allowlist of fully qualified domains. 3. Avoid dynamically reflecting the Origin header in Access-Control-Allow-Origin.
References
https://portswigger.net/web-security/cors

Proof of Concept



1.7. CORS with Arbitrary Subdomain

Reference	**Risk Rating**
CORS with Arbitrary Subdomain	**High**
Tools Used	
BurpSuite	
Vulnerability Description	
A CORS vulnerability with arbitrary subdomains occurs when a server incorrectly allows all subdomains of a trusted domain using a wildcard (*). Attackers can exploit this by registering a malicious subdomain (e.g., attacker.trusted.com) to bypass security restrictions and access sensitive data.	
How It Was Discovered / Steps to Reproduce	
1. Identify a vulnerable site that allows arbitrary subdomains (e.g., *.trusted.com). 2. Register a subdomain (e.g., attacker.trusted.com) or use an open subdomain takeover. 3. Intercept a request in Burp Suite and modify the Origin header to attacker.trusted.com. 4. Send the request and check if Access-Control-Allow-Origin: attacker.trusted.com is returned. 5. If Access-Control-Allow-Credentials: true is present, use the attacker-controlled subdomain to steal user data via cross-origin requests.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/cors_lab/lab_7/cors_7.php	

Consequences of not Fixing the Issue
1. Unauthorized Access: Attackers can host malicious content on a registered subdomain to steal data. 2. Session Hijacking: If Access-Control-Allow-Credentials: true is enabled, attackers can steal authentication tokens. 3. API Data Leakage: Sensitive API responses may be exposed to attacker-controlled subdomains.
Suggested Countermeasures
1. Avoid using wildcards (* or *.trusted.com) in Access-Control-Allow-Origin. 2. Maintain a strict allowlist of specific trusted subdomains. 3. Implement authentication and authorization checks on sensitive API responses.
References
https://portswigger.net/web-security/cors

Proof of Concept

2. Cross-Site Request Forgery

2.1. Eassyy CSRF

Reference	**Risk Rating**
Eassyy CSRF	**Low**
Tools Used	
BurpSuite and https://hacktify.in/csrf/	
Vulnerability Description	
A Cross-Site Request Forgery (CSRF) vulnerability occurs when a web application fails to verify the origin of incoming requests, allowing attackers to trick authenticated users into performing unintended actions on their accounts.	
How It Was Discovered / Steps to Reproduce	
1. Identify the form request that performs a state-changing action. 2. Captured the request using Burp Suite and noted the absence of a CSRF token. 3. Crafted a malicious HTML form that submits the request automatically when a button is clicked. 4. Host the exploit on an attacker-controlled page and trick a victim into visiting it while authenticated. 5. Verified that the action was executed without the user's consent when I logged into victim acc.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/csrf_lab/lab_1/lab_1.php	
Consequences of not Fixing the Issue	
1. Unauthorized Actions: Attackers can force users to perform unintended actions (e.g., changing passwords).	

2. Account Takeover: If the application lacks proper validation, an attacker can hijack user sessions.
3. Financial Loss: In banking apps, CSRF could allow attackers to initiate unauthorized transactions.

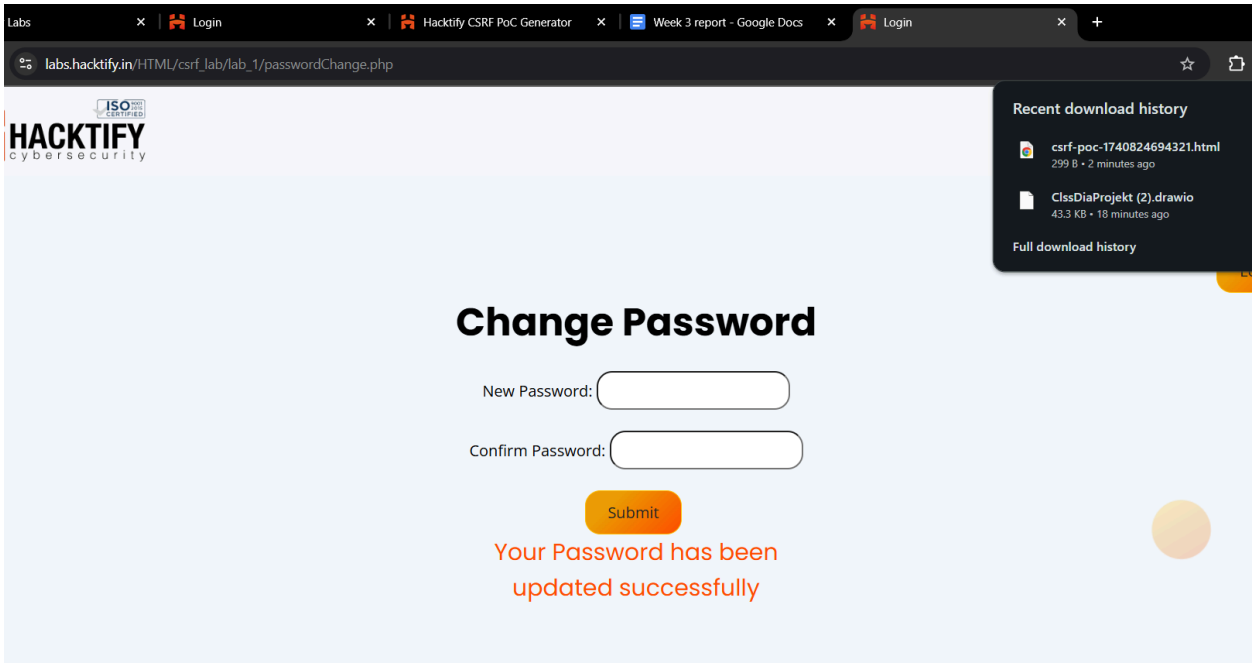
Suggested Countermeasures

1. Implement CSRF tokens (anti-CSRF tokens) in requests that modify user data.
2. Use SameSite attribute for cookies to prevent cross-site request execution.
3. Verify the Origin and Referer headers for sensitive actions.

References

https://portswigger.net/web-security/csrf

Proof of Concept

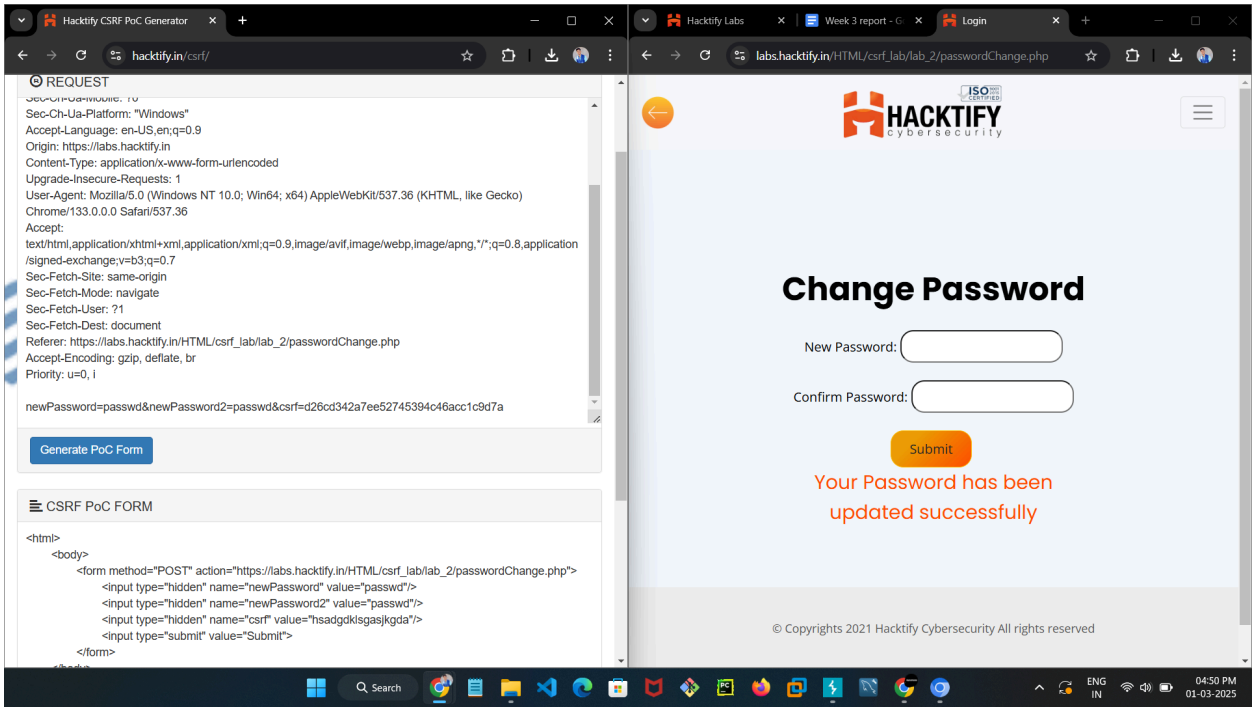


2.2. Always Validate Tokens

Reference	**Risk Rating**
Always Validate Tokens	**Medium**
Tools Used	
BurpSuite and https://hacktify.in/csrf/	
Vulnerability Description	
This CSRF vulnerability occurs when a web application includes a CSRF token in requests but does not properly validate it, allowing attackers to submit arbitrary token values to bypass protection.	
How It Was Discovered / Steps to Reproduce	
1. Captured request using Burp Suite that included a CSRF token. 2. Modifed the CSRF token to a random string and resent the request. 3. The server accepted the request despite the altered CSRF token. 4. Crafted an exploit script that submits the request with a random CSRF token to automate the attack. 5. Tricked the victim into executing the malicious request by making him click a button.	
Vulnerable URLs	
https://labs.hacktify.in/HTML/csrf_lab/lab_2/lab_2.php	
Consequences of not Fixing the Issue	
1. Unauthorized Actions: Attackers can forge requests and modify user data despite the presence of a CSRF token. 2. Session Exploitation: If session cookies are automatically sent, an attacker can perform actions on behalf of the user. 3. Financial & Data Loss: Malicious actions like money transfers or email changes can be executed by an attacker.	

Suggested Countermeasures
1. Ensure CSRF tokens are cryptographically secure and tied to the user session. 2. Reject requests with invalid or missing CSRF tokens. 3. Implement per-request or per-session CSRF tokens to prevent token reuse.
References
https://portswigger.net/web-security/csrf

Proof of Concept



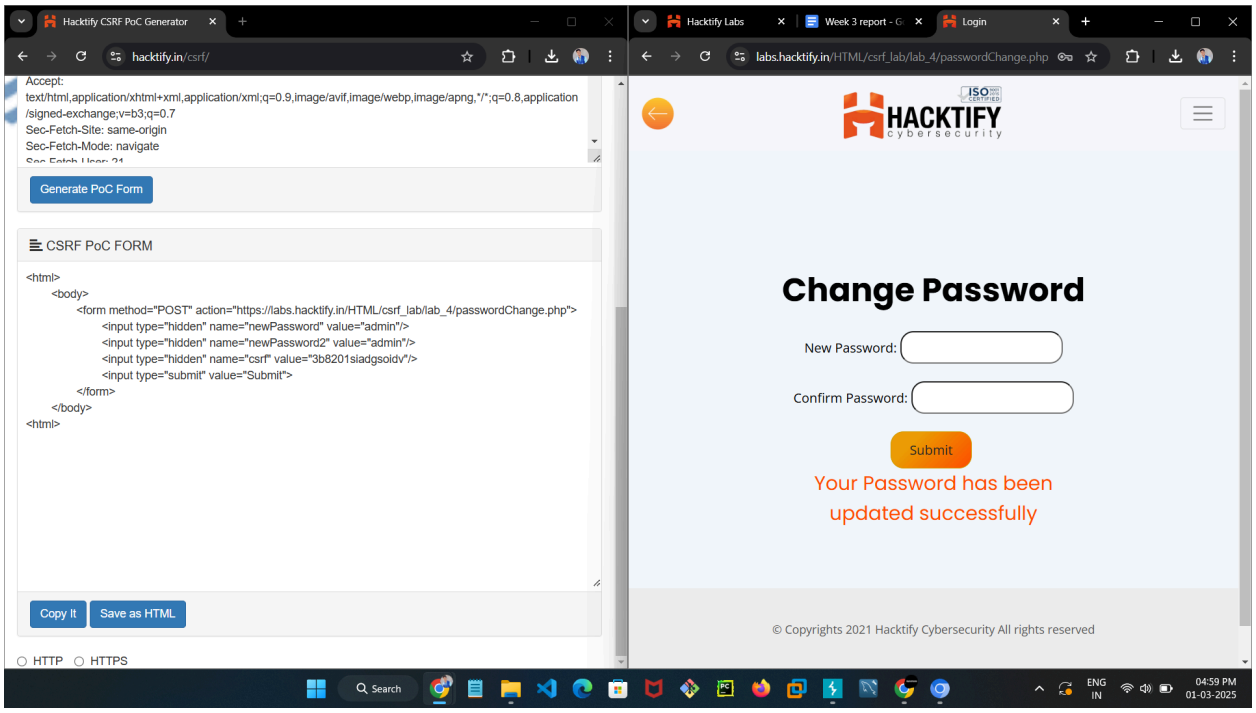
2.3. I hate when someone uses my tokens!

Reference	Risk Rating
I hate when someone uses my tokens!	**Medium**

Tools Used
BurpSuite and https://hacktify.in/csrf/
Vulnerability Description
This CSRF vulnerability occurs when a web application uses weak or predictable tokens, allowing an attacker to guess or obtain another user's token and use it to forge malicious requests on their behalf.
How It Was Discovered / Steps to Reproduce
1. Captured the request containing a CSRF token using Burp Suite. 2. Analyzed multiple requests from different accounts to identify predictable patterns in the CSRF token (e.g., sequential or timestamp-based). 3. Generated a valid CSRF token for another user based on the observed pattern. 4. Used the guessed CSRF token in a forged request and checked if the server accepts it. 5. Craft an exploit that sends requests using the predictable CSRF token to automate the attack.
Vulnerable URLs
https://labs.hacktify.in/HTML/csrf_lab/lab_4/lab_4.php
Consequences of not Fixing the Issue
1. Account Takeover: Attackers can use guessed CSRF tokens to perform unauthorized actions on another user's account. 2. Mass Exploitation: If the CSRF token pattern is predictable, attackers can automate attacks against multiple users. 3. Financial & Data Loss: Sensitive actions like password changes or fund transfers can be performed by an attacker.
Suggested Countermeasures
1. Use cryptographically secure, random, and unique CSRF tokens per user session. 2. Bind CSRF tokens to user sessions to prevent token reuse. 3. Implement per-request CSRF tokens to mitigate token prediction attacks.

References
https://portswigger.net/web-security/csrf

Proof of Concept



2.4. GET Me or POST ME

Reference	Risk Rating
GET Me or POST ME	**Low**
Tools Used	
BurpSuite and https://hacktify.in/csrf/	
Vulnerability Description	

This CSRF vulnerability occurs when a web application does not properly validate the request method, allowing an attacker to change a GET request to POST and execute unintended actions on behalf of an authenticated user.

How It Was Discovered / Steps to Reproduce

1. Captured a legitimate GET request using Burp Suite.
2. Modified the request method from GET to POST.
3. Removed or adjusted parameters if necessary to match the expected POST request format.
4. Sent the modified request and observed that the server processed it successfully.
5. Crafted an exploit using an HTML form or JavaScript that automatically sent a POST request to the vulnerable endpoint.
6. Executed the exploit while authenticated and confirmed that the unintended action was performed.

Vulnerable URLs

https://labs.hacktify.in/HTML/csrf_lab/lab_6/lab_6.php

Consequences of not Fixing the Issue

1. Unauthorized Actions: Attackers can force users to perform unintended actions (e.g., changing account settings).
2. Data Manipulation: Sensitive information like email addresses or passwords can be changed.
3. Financial & Security Risks: Actions such as unauthorized transactions or privilege escalations may occur.

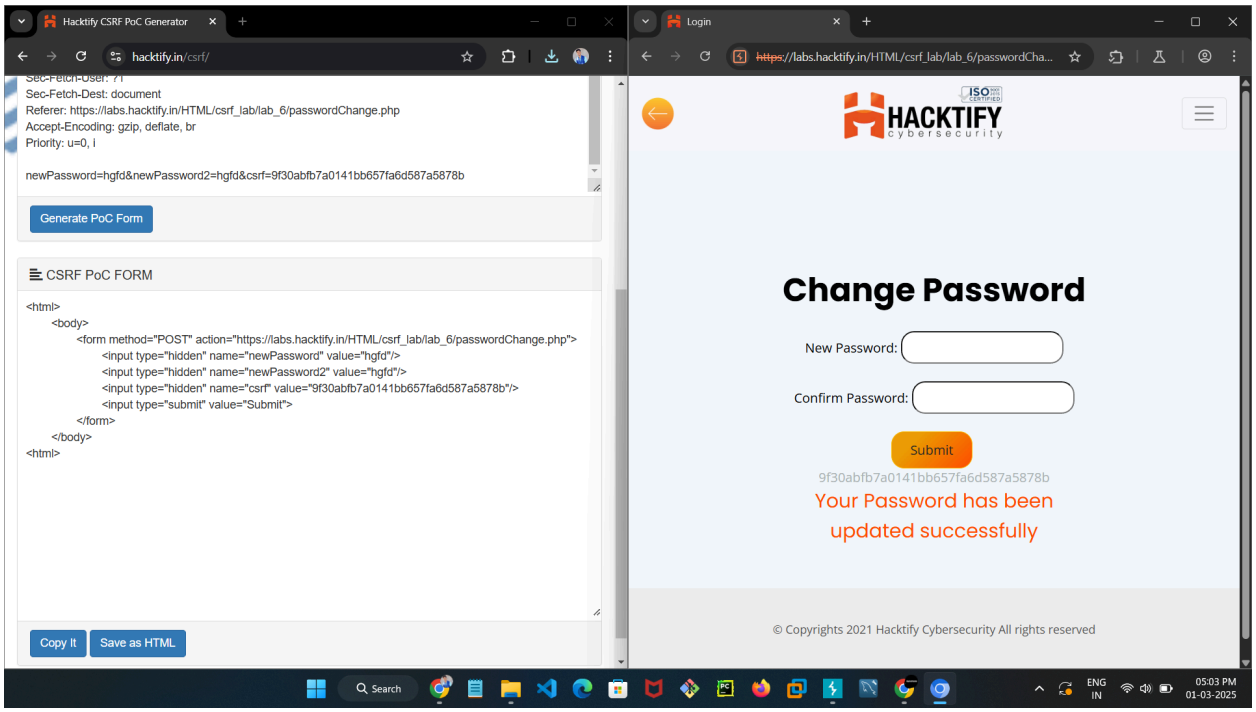
Suggested Countermeasures

1. Enforce strict HTTP method validation to ensure only allowed methods are processed.
2. Require CSRF tokens for all state-changing requests, regardless of the method.
3. Use SameSite=strict cookies to prevent cross-site request execution.

References

https://portswigger.net/web-security/csrf

Proof of Concept

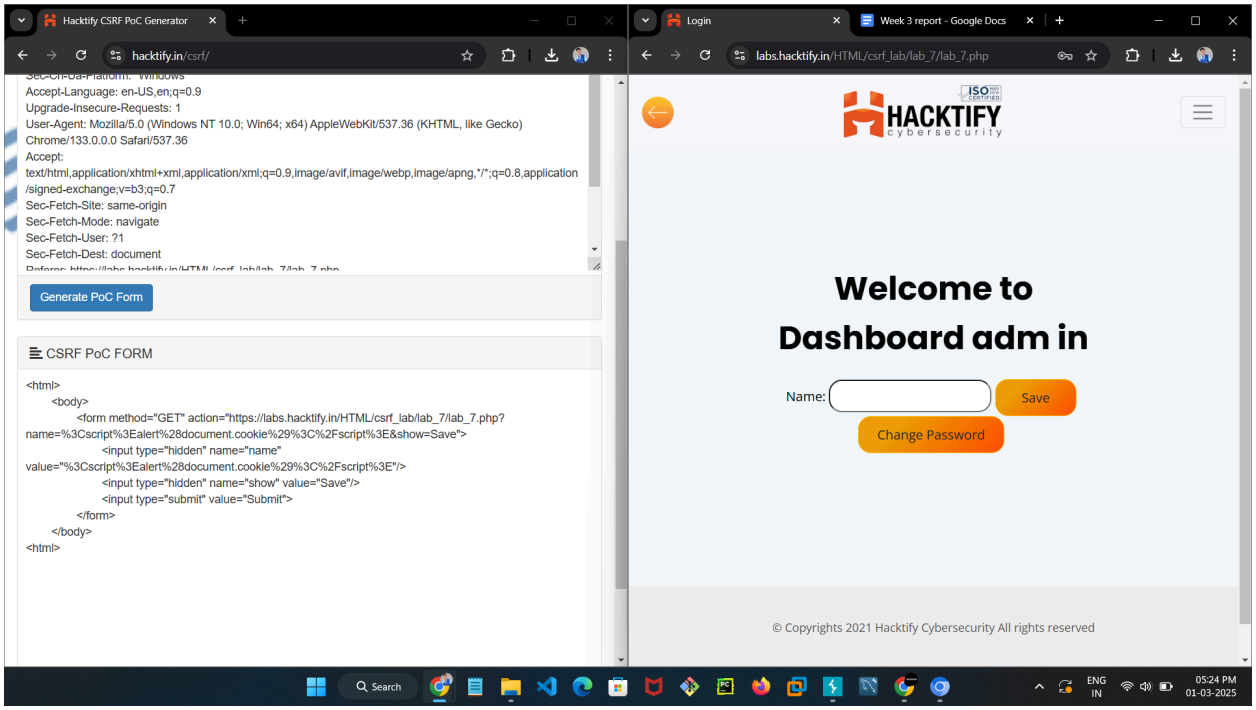


2.5. XSS the saviour

Reference	**Risk Rating**
XSS the saviour	**High**
Tools Used	
BurpSuite and https://hacktify.in/csrf/	
Vulnerability Description	
This vulnerability occurs when an attacker exploits a XSS flaw to steal a user's session cookie and CSRF token, allowing them to forge authenticated requests and perform unauthorized actions.	

How It Was Discovered / Steps to Reproduce
1. Identified an XSS vulnerability that allowed JavaScript execution in the target application. 2. Injected a malicious payload that extracted the user's cookies and CSRF token via document.cookie or by making an AJAX request to a CSRF-protected endpoint. 3. Sent the stolen credentials to an attacker-controlled server. 4. Used the stolen CSRF token and session cookie to craft an authenticated request that performed an unintended action. 5. Sent the forged request and confirmed that the action was successfully executed.
Vulnerable URLs
https://labs.hacktify.in/HTML/csrf_lab/lab_7/lab_7.php
Consequences of not Fixing the Issue
1. Complete Account Takeover: Attackers can steal session cookies and CSRF tokens to impersonate users. 2. Unauthorized Transactions: Sensitive actions (e.g., fund transfers, password changes) can be executed. 3. Data Theft & Mass Exploitation: Attackers can automate the attack and target multiple users.
Suggested Countermeasures
1. Mitigate XSS: Implement strong input sanitization and output encoding to prevent XSS attacks. 2. Use HttpOnly Cookies: Store session tokens in HttpOnly cookies to prevent JavaScript access. 3. Bind CSRF Tokens to Sessions: Ensure CSRF tokens are unique per session and cannot be reused across requests.
References
https://portswigger.net/web-security/csrf

Proof of Concept



2.6. rm -rf token

Reference	Risk Rating
rm -rf token	**High**
Tools Used	
BurpSuite and https://hacktify.in/csrf/	
Vulnerability Description	
This vulnerability occurs when a web application includes a CSRF token in requests but does not properly enforce its validation, allowing an attacker to simply omit the token and still have the request processed successfully.	
How It Was Discovered / Steps to Reproduce	

1. Captured a legitimate request containing a CSRF token using Burp Suite.
2. Removed the CSRF token parameter from the request.
3. Sent the modified request and observed that the server still processed it successfully.
4. Crafted an exploit (e.g., an HTML form or JavaScript code) that sent the same request without a CSRF token.
5. Executed the exploit while authenticated and confirmed that the intended action was performed despite the missing token.

Vulnerable URLs

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

Consequences of not Fixing the Issue

1. Unauthorized Actions: Attackers can submit forged requests without needing a valid CSRF token.
2. Session Exploitation: If session cookies are automatically sent, an attacker can perform actions on behalf of a logged-in user.
3. Data Manipulation & Financial Loss: Sensitive operations like password changes or fund transfers can be executed without user consent.

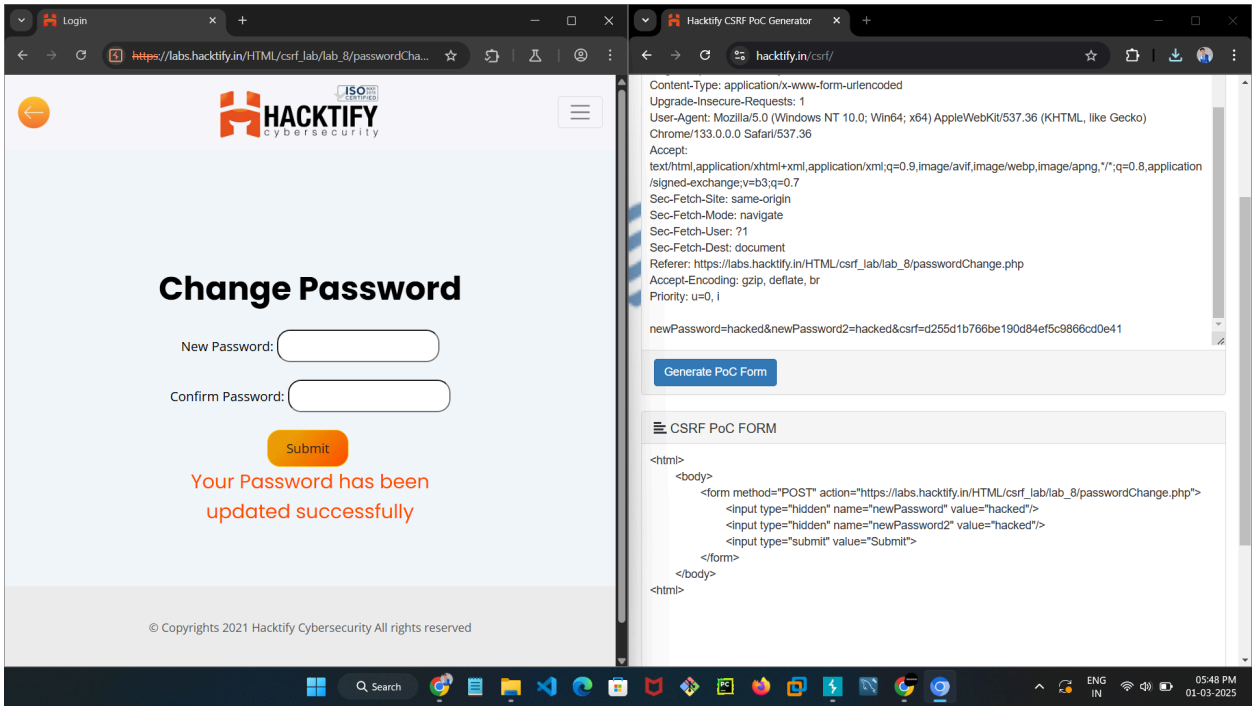
Suggested Countermeasures

1. Enforce CSRF Token Validation: Reject requests with missing or invalid CSRF tokens.
2. Use SameSite Cookies: Set SameSite=strict to prevent cross-site request execution.
3. Verify Origin & Referer Headers: Ensure requests originate from trusted sources before processing them.

References

https://portswigger.net/web-security/csrf

Proof of Concept


Login
https://labs.hacktify.in/HTML/csrf_lab/lab_8/passwordCha...
HACKTIFY
cybersecurity
Change Password
New Password:
Confirm Password:
Submit
Your Password has been updated successfully
© Copyrights 2021 Hacktify Cybersecurity All rights reserved
Hacktify CSRF PoC Generator
hacktify.in/csrf/
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://labs.hacktify.in/HTML/csrf_lab/lab_8/passwordChange.php
Accept-Encoding: gzip, deflate, br
Priority: u=0, i
newPassword=hacked&newPassword2=hacked&csrf=d255d1b766be190d84ef5c9866cd0e41
Generate PoC Form
CSRF PoC FORM
<html>
<body>
<form method="POST" action="https://labs.hacktify.in/HTML/csrf_lab/lab_8/passwordChange.php">
<input type="hidden" name="newPassword" value="hacked"/>
<input type="hidden" name="newPassword2" value="hacked"/>
<input type="submit" value="Submit">
</form>
</body>
<html>
Search
ENG
IN
05:48 PM
01-03-2025

THANK YOU