

Penetration Testing Report

Full Name: AVIREDDY RUSHIKESH

Program: HCPT

Date:02/03/2025

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 Name – Cross-Site Request Forgery , Lab 2 Name – Cross Origin Resource Sharing |
|-------------------------|---|

3. Summary

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

Total number of Sub-labs: 13

| High | Medium | Low |
|-------------|---------------|------------|
| 5 | 3 | 5 |

High - **5**

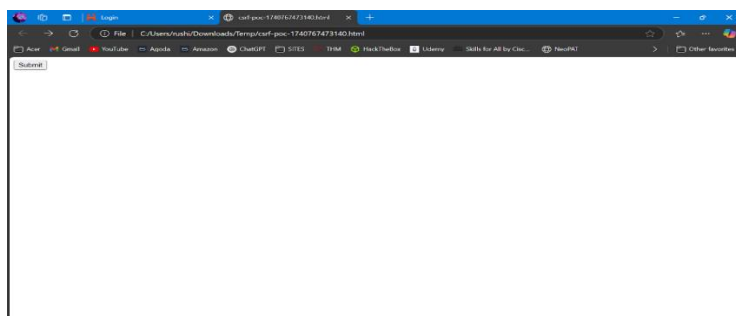
Medium - **3**

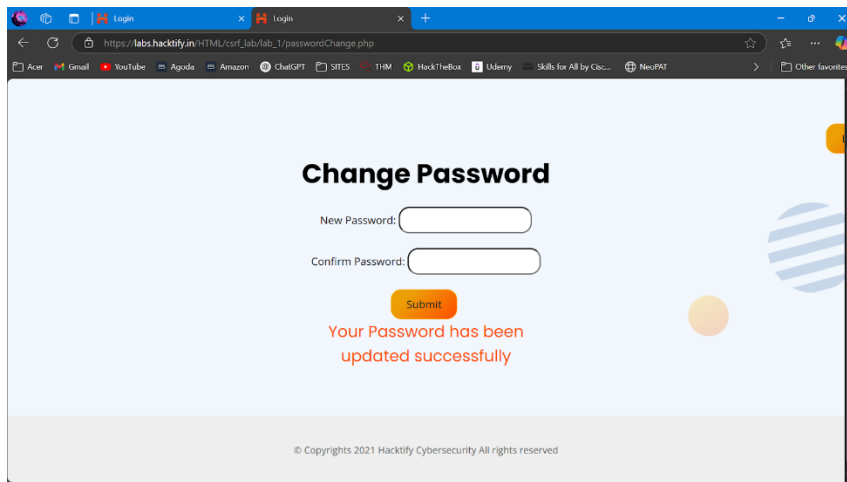
1. Cross – Site Request Forgery

1.1. Eassyy CSRF

| Reference | Risk Rating |
|---|-------------|
| Eassyy CSRF | Low |
| Tools Used | |
| Hacktify CSRF Poc Generator | |
| Vulnerability Description | |
| <p>Cross-Site Request Forgery (CSRF) is a vulnerability that allows an attacker to trick a logged-in user into unknowingly executing unauthorized actions on a web application. This occurs when the application fails to verify the origin of requests, allowing an attacker to embed malicious requests in a link, form, or script that executes with the victim's session credentials. If exploited, CSRF can lead to account takeover, data modification, fund transfers, or privilege escalation without the victim's consent. Proper CSRF protection tokens and same-site cookie policies are essential to prevent this attack.</p> | |
| How It Was Discovered | |
| Automated Tools – By CSRF Poc Generator | |
| Vulnerable URLs | |
| https://labs.hacktify.in/HTML/csrf_lab/lab_1/index.php | |
| Consequences of not Fixing the Issue | |
| <p>If a CSRF vulnerability is not fixed, attackers can force users to perform unintended actions like changing account details, transferring funds, or deleting data without their consent. This can lead to account hijacking, financial fraud, and data loss, causing reputational damage and regulatory non-compliance. In critical cases, attackers may gain full control over user accounts or administrative functions, leading to mass exploitation and security breaches.</p> | |
| Suggested Countermeasures | |
| <p>To prevent CSRF attacks, implement CSRF tokens in all state-changing requests to verify legitimate user actions. Use SameSite cookies to restrict cookie transmission across sites and enforce user re-authentication for sensitive actions. Implement CORS policies to limit cross-origin requests and use HTTP headers like Referer and Origin to validate request sources. Additionally, educate users to avoid clicking on untrusted links while logged in and conduct regular security audits to detect vulnerabilities.</p> | |
| References | |
| https://owasp.org/www-community/attacks/csrf | |

Proof of Concept





1.2. Always Validate Tokens!

| Reference | Risk Rating |
|---|-------------|
| Always Validate Tokens | Medium |
| Tools Used | |
| Hacktify CSRF Poc Generator | |
| Vulnerability Description | |
| <p>Cross-Site Request Forgery (CSRF) occurs when an attacker tricks a logged-in user into unknowingly making a malicious request that changes a security token, such as an API key, authentication token, or session token, without their consent. This happens due to the application failing to validate request origins and relying solely on session-based authentication. If exploited, an attacker can change security tokens linked to the victim's account, leading to unauthorized access, privilege escalation, or account takeover. Implementing CSRF tokens, SameSite cookie attributes, and user re-authentication can prevent such attacks.</p> | |
| How It Was Discovered | |
| Automated Tools – By CSRF Poc Generator | |
| Vulnerable URLs | |
| https://labs.hacktify.in/HTML/csrf_lab/lab_2/login.php | |
| Consequences of not Fixing the Issue | |
| <p>If a CSRF vulnerability is not fixed, attackers can force users to perform unintended actions like changing account details, transferring funds, or deleting data without their consent. This can lead to account hijacking, financial fraud, and data loss, causing reputational damage and regulatory non-compliance. In critical cases, attackers may gain full control over user accounts or administrative functions, leading to mass exploitation and security breaches.</p> | |
| Suggested Countermeasures | |
| <p>To prevent CSRF attacks, implement CSRF tokens in all state-changing requests to verify legitimate user actions. Use SameSite cookies to restrict cookie transmission across sites and enforce user re-authentication for sensitive actions. Implement CORS policies to limit cross-origin requests and use HTTP headers like Referer and Origin to validate request sources. Additionally, educate users to avoid clicking on untrusted links while logged in and conduct regular security audits to detect vulnerabilities.</p> | |
| References | |
| https://owasp.org/www-community/attacks/csrf | |

Proof of Concept

The screenshot shows the 'Hacktify CSRF PoC Generator' web application. It has a dark-themed browser window with multiple tabs. The main content area is titled 'CSRF PoC Generator' and is divided into two panels. The left panel, 'REQUEST', displays a detailed HTTP request for a password change endpoint, including headers like 'User-Agent', 'Origin', 'Content-Type', 'Accept', 'Referer', and 'Accept-Language', as well as a CSRF token in the body. The right panel, 'CSRF PoC FORM', shows the generated HTML form with a POST method, the same endpoint URL, and hidden inputs for 'newPassword' and 'csrf'. Below the panels are buttons for 'Generate PoC Form', 'Copy it', and 'Save as HTML'. At the bottom, there are radio buttons for 'HTTP' and 'HTTPS'.

CSRF PoC Generator

REQUEST

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
Origin: https://labs.hacktify.in
Content-Type: application/x-www-form-urlencoded
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Referer: https://labs.hacktify.in/HTML/csrflablab_2/passwordChange.php
Accept-Encoding: gzip, deflate, br
Accept-Language: en,en-US;q=0.9,en-US;q=0.8,en;q=0.7,he;q=0.6,hi;q=0.5,da;q=0.4
Priority: u=0, l
newPassword=12345&newPassword2=12345&csrf=7397061862c2b57bd78a9bad5a23ee3

CSRF PoC FORM

```
<html>
  <body>
    <form method="POST"
      action="https://labs.hacktify.in/HTML/csrflablab_2/passwordChange.php">
      <input type="hidden" name="newPassword" value="2328"/>
      <input type="hidden" name="newPassword2" value="2328"/>
      <input type="hidden" name="csrf" value="abc123"/>
      <input type="submit" value="Submit"/>
    </form>
  </body>
</html>
```

Generate PoC Form Copy it Save as HTML

HTTP HTTPS

The screenshot shows a web browser window displaying a 'Change Password' form. The form has two input fields for 'New Password' and 'Confirm Password', followed by an orange 'Submit' button. Below the button, a red message states 'Your Password has been updated successfully'. The footer of the page reads '© Copyrights 2021 Hacktify Cybersecurity All rights reserved'. The browser's address bar shows the URL 'https://labs.hacktify.in/HTML/csrflablab_2/passwordChange.php'.

Change Password

New Password:

Confirm Password:

Submit

Your Password has been updated successfully

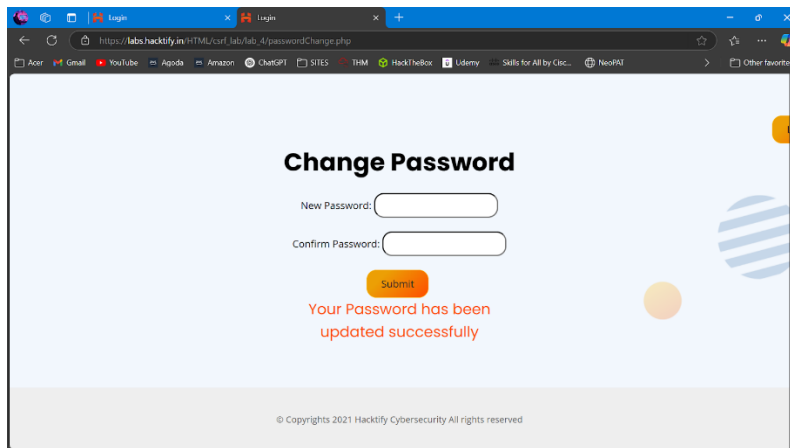
© Copyrights 2021 Hacktify Cybersecurity All rights reserved

1.3. I hate when someone uses my Tokens!

| Reference | Risk Rating |
|---|-------------|
| I hate when someone uses my Tokens! | Low |
| Tools Used | |
| Hacktify CSRF Poc Generator | |
| Vulnerability Description | |
| Cross-Site Request Forgery (CSRF) occurs when an attacker tricks a logged-in user into unknowingly making a malicious request that changes a security token, such as an API key , authentication token , or session token , without their consent. This happens due to the application failing to validate request origins and relying solely on session-based authentication . If exploited, an attacker can change security tokens linked to the victim's account, leading to unauthorized access , privilege escalation , or account takeover . Implementing CSRF tokens , SameSite cookie attributes , and user re-authentication can prevent such attacks. | |
| How It Was Discovered | |
| Automated Tools – By CSRF Poc Generator | |
| Vulnerable URLs | |
| https://labs.hacktify.in/HTML/csrf_lab/lab_4/login.php | |
| Consequences of not Fixing the Issue | |
| If a CSRF vulnerability is not fixed, attackers can force users to perform unintended actions like changing account details , transferring funds , or deleting data without their consent. This can lead to account hijacking , financial fraud , and data loss , causing reputational damage and regulatory non-compliance . In critical cases, attackers may gain full control over user accounts or administrative functions , leading to mass exploitation and security breaches . | |
| Suggested Countermeasures | |
| To prevent CSRF attacks , implement CSRF tokens in all state-changing requests to verify legitimate user actions. Use SameSite cookies to restrict cookie transmission across sites and enforce user re-authentication for sensitive actions. Implement CORS policies to limit cross-origin requests and use HTTP headers like Referrer and Origin to validate request sources. Additionally, educate users to avoid clicking on untrusted links while logged in and conduct regular security audits to detect vulnerabilities. | |
| References | |
| https://owasp.org/www-community/attacks/csrf | |

Proof of Concept

The screenshot shows the Hacktify Labs interface with the CSRF PoC Generator tool. The tool displays a 'REQUEST' tab with a detailed HTTP request for a password change endpoint. The request includes headers like User-Agent, Origin, and Accept, and a body with form data including 'newPassword' and 'csrf' tokens. The 'CSRF PoC FORM' tab shows the corresponding HTML form structure with hidden input fields for the password and CSRF token. The interface also includes a 'Generate PoC Form' button and a 'Copy It' button.



1.4. GET Me or POST ME!

| Reference | Risk Rating |
|---|-------------|
| GET Me or POST ME! | Low |
| Tools Used | |
| Hacktify CSRF Poc Generator | |
| Vulnerability Description | |
| Cross-Site Request Forgery (CSRF) exploits the lack of proper request validation, allowing an attacker to force a victim's browser to perform unauthorized actions. If a web application accepts sensitive actions via both GET and POST requests, an attacker can convert a POST request to GET and embed it in an external link or image, causing unintended changes when the victim visits a malicious page. This can lead to account modifications, privilege escalation, or data exposure. Implementing strict request methods, CSRF tokens, and SameSite cookie policies can prevent such attacks. | |
| How It Was Discovered | |
| Automated Tools – By CSRF Poc Generator | |
| Vulnerable URLs | |
| https://labs.hacktify.in/HTML/csrflab_6/login.php | |
| Consequences of not Fixing the Issue | |
| If this CSRF vulnerability (changing POST to GET) is not fixed, attackers can exploit it to perform unauthorized actions on behalf of authenticated users without their knowledge. This can lead to account takeovers, unauthorized data modifications, financial fraud, or privilege escalations . Since GET requests can be triggered automatically (e.g., by loading an image or clicking a malicious link), attackers can easily exploit users just by making them visit a compromised webpage. Failure to enforce proper request validation can result in data breaches, reputational damage, and compliance violations for the affected organization. | |
| Suggested Countermeasures | |
| To prevent this CSRF vulnerability (changing POST to GET) , enforce strict request methods by ensuring sensitive actions only accept POST requests . Implement CSRF tokens for all state-changing operations and validate them on the server side. Use SameSite cookie attributes (Lax or Strict) to prevent cross-origin requests from automatically sending session cookies. Additionally, require re-authentication or multi-factor authentication (MFA) for critical actions like password or email changes. Regular security audits and penetration testing can help identify and mitigate such vulnerabilities before they are exploited. | |
| References | |
| https://owasp.org/www-community/attacks/csrf | |

Proof of Concept

hacktify.in/csrf/

CSRF PoC Generator

REQUEST

POST /HTML/csrf_lab/lab_6/passwordChange.php HTTP/2
Host: labs.hacktify.in
Cookie: PHPSESSID=378aaef131881a4eea2359752561efdc
Content-Length: 76
Cache-Control: max-age=0
Sec-Ch-Ua: "Not(A.Brand)" ;v="99", "Google Chrome" ;v="133", "Chromium" ;v="133"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
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
Origin: https://labs.hacktify.in
Content-Type: application/x-www-form-urlencoded
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

Generate PoC Form

CSRF PoC FORM

```
<html>
<body>
<form method="GET"
action="https://labs.hacktify.in/HTML/csrf_lab/lab_6/passwordChange.php">
<input type="hidden" name="newPassword" value="hacked"/>
<input type="hidden" name="newPassword2" value="hacked"/>
<input type="hidden" name="csrf"
value="9030abfb7a0141bb657fa6d587a5d785"/>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

Copy It Save as HTML

Login

https://labs.hacktify.in/HTML/csrf_lab/lab_6/passwordChange.php?newPassword=hacked&newPassword2=hacked&csrf=9030abfb7a0141bb657fa6d58...

HACKTIFY
cybersecurity

Happy Hacking

Change Password

New Password:

Confirm Password:

Submit

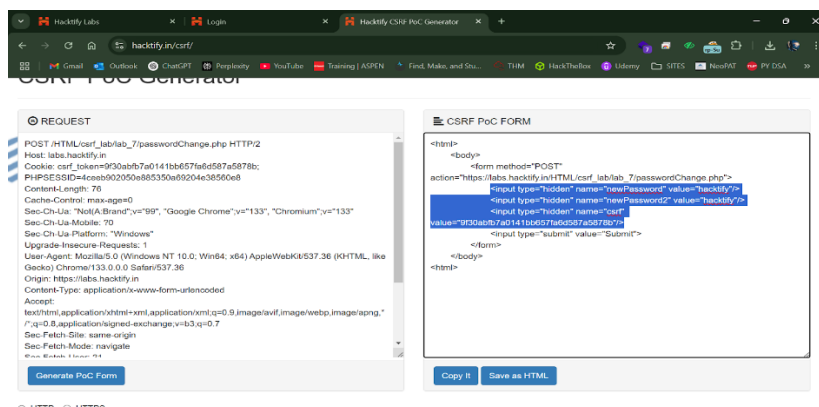
Your Password has been updated successfully

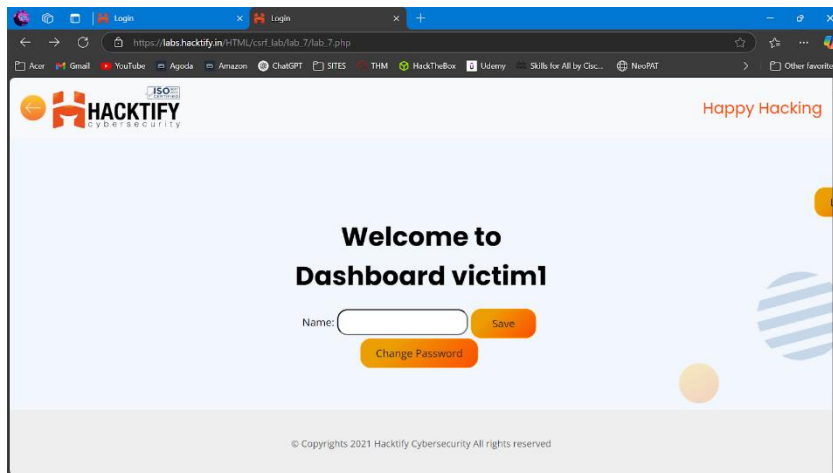
© Copyrights 2021 Hacktify Cybersecurity All rights reserved

1.5. XSS the Saviour!

| Reference | Risk Rating |
|--|-------------|
| XSS the Saviour! | High |
| Tools Used | |
| Burpsuite , Hacktify CSRF Poc Generator | |
| Vulnerability Description | |
| <p>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.</p> <p>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.</p> | |
| How It Was Discovered | |
| Automated Tools – By CSRF Poc Generator | |
| Vulnerable URLs | |
| https://labs.hacktify.in/HTML/csrf_lab/lab_7/login.php | |
| Consequences of not Fixing the Issue | |
| <p>If a CSRF vulnerability is not fixed, attackers can force users to perform unintended actions like changing account details, transferring funds, or deleting data without their consent. This can lead to account hijacking, financial fraud, and data loss, causing reputational damage and regulatory non-compliance. In critical cases, attackers may gain full control over user accounts or administrative functions, leading to mass exploitation and security breaches.</p> | |
| Suggested Countermeasures | |
| <p>To prevent CSRF attacks, implement CSRF tokens in all state-changing requests to verify legitimate user actions. Use SameSite cookies to restrict cookie transmission across sites and enforce user re-authentication for sensitive actions. Implement CORS policies to limit cross-origin requests and use HTTP headers like Referer and Origin to validate request sources. Additionally, educate users to avoid clicking on untrusted links while logged in and conduct regular security audits to detect vulnerabilities. Additionally, require re-authentication or multi-factor authentication (MFA) for critical actions like password or email changes. Regular security audits and penetration testing can help identify and mitigate such vulnerabilities before they are exploited.</p> | |
| References | |
| https://portswigger.net/web-security/csrf | |

Proof of Concept

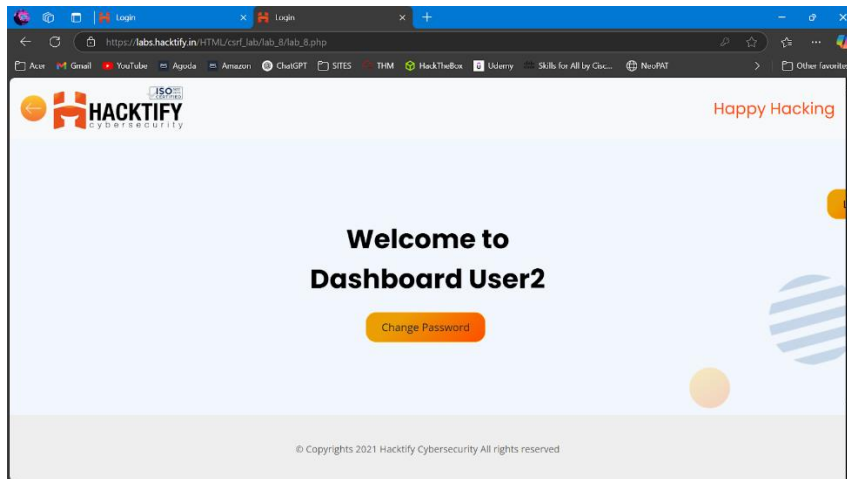
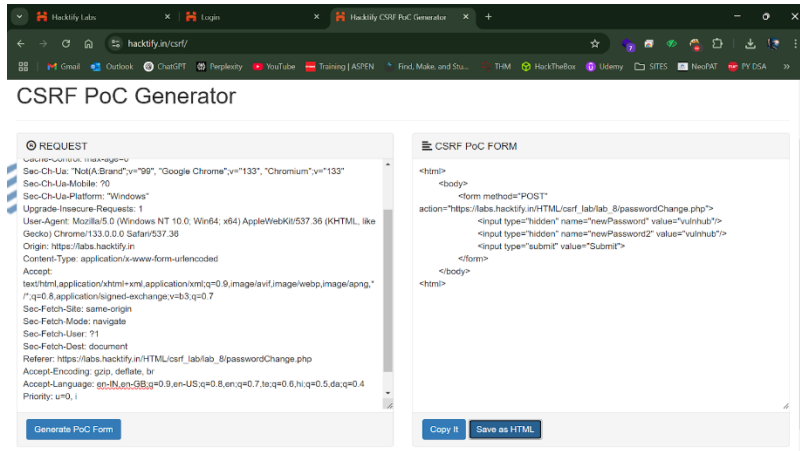




1.6. rm -rf token

| Reference | Risk Rating |
|--|-------------|
| rm -rf token | High |
| Tools Used | |
| Burpsuite , Hacktify CSRF Poc Generator | |
| Vulnerability Description | |
| <p>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.</p> <p>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.</p> | |
| How It Was Discovered | |
| Automated Tools – By CSRF Poc Generator | |
| Vulnerable URLs | |
| https://labs.hacktify.in/HTML/csrf_lab/lab_8/login.php | |
| Consequences of not Fixing the Issue | |
| <p>If a CSRF vulnerability is not fixed, attackers can force users to perform unintended actions like changing account details, transferring funds, or deleting data without their consent. This can lead to account hijacking, financial fraud, and data loss, causing reputational damage and regulatory non-compliance. In critical cases, attackers may gain full control over user accounts or administrative functions, leading to mass exploitation and security breaches.</p> | |
| Suggested Countermeasures | |
| <p>To prevent CSRF attacks, implement CSRF tokens in all state-changing requests to verify legitimate user actions. Use SameSite cookies to restrict cookie transmission across sites and enforce user re-authentication for sensitive actions. Implement CORS policies to limit cross-origin requests and use HTTP headers like Referer and Origin to validate request sources. Additionally, educate users to avoid clicking on untrusted links while logged in and conduct regular security audits to detect vulnerabilities. Additionally, require re-authentication or multi-factor authentication (MFA) for critical actions like password or email changes. Regular security audits and penetration testing can help identify and mitigate such vulnerabilities before they are exploited.</p> | |
| References | |
| https://www.invicti.com/learn/cross-site-request-forgery-csrf/ | |

Proof of Concept

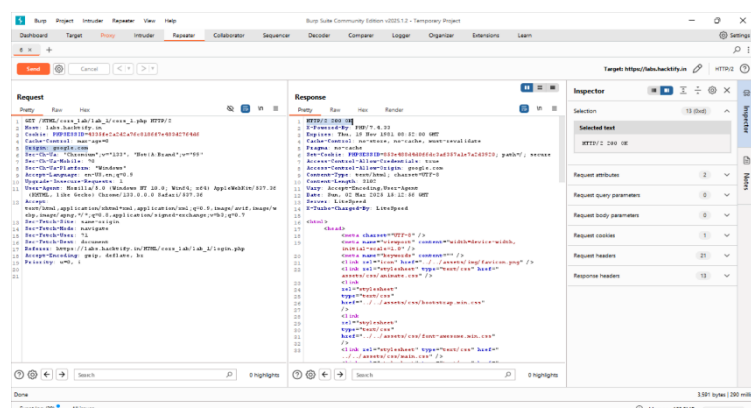


2. Cross-Origin Resource Sharing

2.1. CORS with arbitrary Origin

| Reference | Risk Rating |
|---|-------------|
| CORS with arbitrary Origin | Low |
| Tools Used | |
| Burpsuite | |
| Vulnerability Description | |
| <p>A Cross-Origin Resource Sharing (CORS) vulnerability occurs when a web application improperly configures its CORS policy, allowing unauthorized origins to access sensitive resources. If an application allows requests from any origin (Access-Control-Allow-Origin: *) or whitelists untrusted domains, attackers can exploit this to steal user data, hijack sessions, or perform API abuse via malicious websites. This is especially dangerous if authentication credentials (cookies, tokens) are exposed to untrusted origins, leading to data breaches and unauthorized actions. Properly restricting allowed origins and methods is crucial for preventing CORS-based attacks.</p> | |
| How It Was Discovered | |
| Manual Analysis – Intercepting and modifying the request | |
| Vulnerable URLs | |
| https://labs.hacktify.in/HTML/cors_lab/lab_1/login.php | |
| Consequences of not Fixing the Issue | |
| <p>If a CORS vulnerability is not fixed, attackers can exploit it to perform unauthorized cross-origin requests, leading to data theft, session hijacking, and API abuse. Malicious websites can trick a victim's browser into making requests to a vulnerable application, potentially exposing sensitive user data, authentication tokens, or financial information. If credentials are included in CORS requests, attackers could bypass authentication mechanisms and gain unauthorized access. This can result in data breaches, compliance violations (e.g., GDPR), and reputational damage for the affected organization.</p> | |
| Suggested Countermeasures | |
| <p>To prevent CORS vulnerabilities, configure the CORS policy securely by restricting Access-Control-Allow-Origin to trusted domains instead of using *. Avoid allowing credentials (Access-Control-Allow-Credentials: true) unless absolutely necessary, and never expose sensitive endpoints to untrusted origins. Implement proper authentication and authorization checks on the server-side rather than relying on CORS for security. Regularly audit CORS configurations and use security headers like Content-Security-Policy (CSP) to reduce the risk of cross-origin attacks.</p> | |
| References | |
| https://portswigger.net/web-security/cors | |

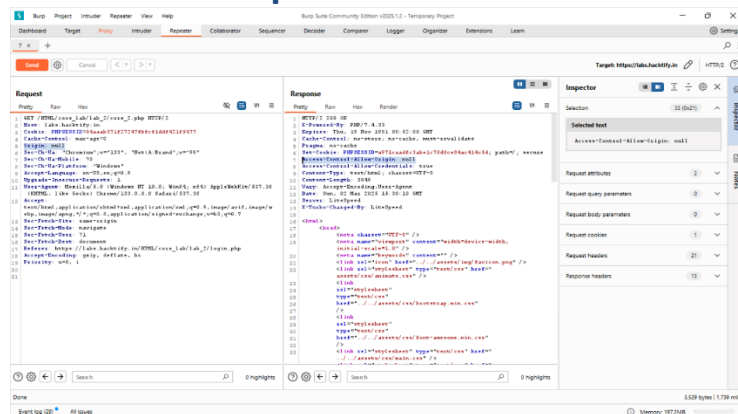
Proof of Concept



2.2. CORS with Null Origin

| Reference | Risk Rating |
|---|-------------|
| CORS with Null Origin | Low |
| Tools Used | |
| Burpsuite | |
| Vulnerability Description | |
| <p>A null origin CORS vulnerability occurs when a web application allows requests from the null origin (Access-Control-Allow-Origin: null), which is often misused in sandboxed iframes, file:// URLs, and certain cross-origin requests. Attackers can exploit this by embedding the vulnerable site in a malicious iframe or using a local HTML file to bypass origin restrictions, leading to data theft, unauthorized API access, or session hijacking. To mitigate this, developers should restrict allowed origins to trusted domains and avoid using null unless absolutely necessary in controlled environments.</p> | |
| How It Was Discovered | |
| Manual Analysis – Intercetpting and modifying the request | |
| Vulnerable URLs | |
| https://labs.hacktify.in/HTML/cors_lab/lab_2/login.php | |
| Consequences of not Fixing the Issue | |
| <p>If a null origin CORS vulnerability is not fixed, attackers can exploit it to bypass origin restrictions and gain unauthorized access to sensitive data or APIs. Malicious websites or local HTML files can send requests on behalf of users, potentially leading to data theft, session hijacking, or API abuse. This is especially dangerous for applications handling personal, financial, or authentication-related data, as it can result in data breaches, account compromises, and regulatory non-compliance, ultimately damaging the organization's reputation and security.</p> | |
| Suggested Countermeasures | |
| <p>To prevent null origin CORS vulnerabilities, avoid using Access-Control-Allow-Origin: null unless absolutely necessary in controlled environments. Instead, explicitly define trusted origins and restrict cross-origin access to only those domains. Implement strict authentication and authorization checks on the server-side rather than relying on CORS for security. Additionally, disable null origin access for sensitive endpoints, enforce SameSite cookies, and use Content Security Policy (CSP) to restrict resource loading from untrusted sources. Regular security audits can help identify and mitigate such misconfigurations.</p> | |
| References | |
| https://www.tenable.com/blog/understanding-cross-origin-resource-sharing-vulnerabilities | |

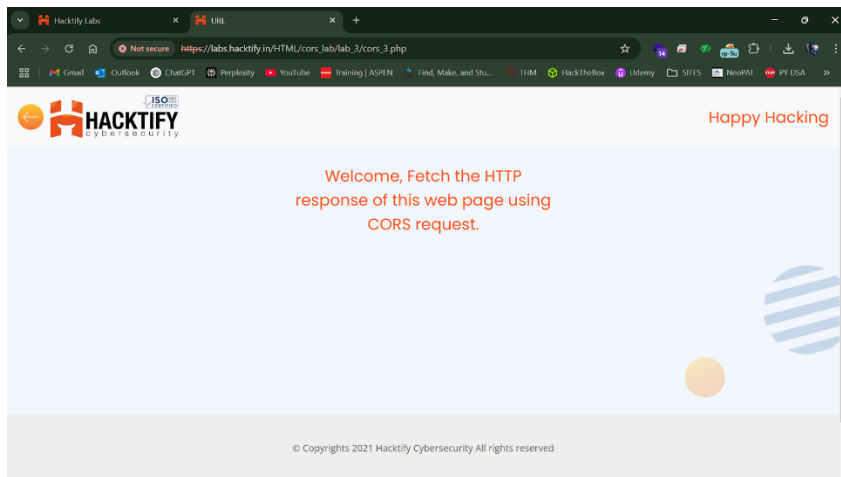
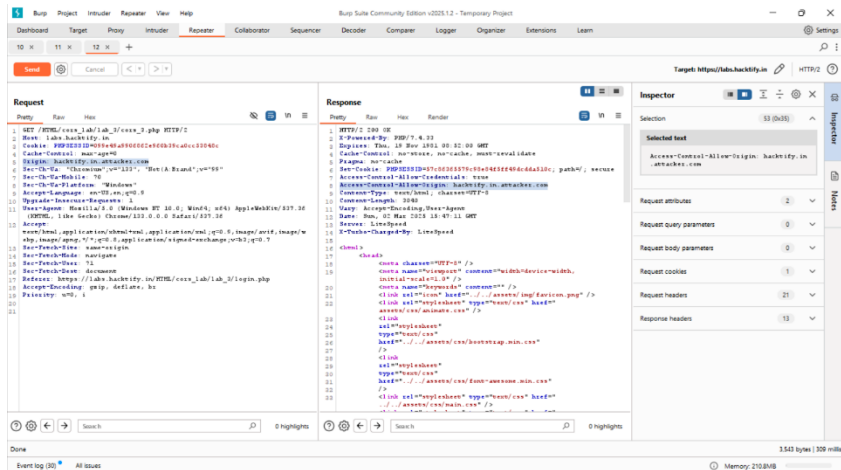
Proof of Concept



2.3. CORS with prefix match

| Reference | Risk Rating |
|--|-------------|
| CORS with prefix match | Medium |
| Tools Used | |
| Burpsuite | |
| Vulnerability Description | |
| <p>A prefix match CORS vulnerability occurs when a server improperly validates the Origin header by using partial string matching instead of an exact match. For example, if a server allows requests from "https://trusted.com" but incorrectly matches any domain starting with "https://trusted.com", an attacker can exploit this by hosting a malicious site at "https://trusted.com.attacker.com". This can lead to unauthorized cross-origin access, data theft, and API abuse, allowing attackers to steal user credentials, tokens, or sensitive data through manipulated requests.</p> | |
| How It Was Discovered | |
| Manual Analysis – Intercepting and modifying the request | |
| Vulnerable URLs | |
| https://labs.hacktify.in/HTML/cors_lab/lab_3/login.php | |
| Consequences of not Fixing the Issue | |
| <p>If a prefix match CORS vulnerability is not fixed, attackers can exploit it by hosting malicious sites with similar domain prefixes (e.g., https://trusted.com.attacker.com) to bypass origin restrictions. This can lead to unauthorized API access, data leaks, session hijacking, and account takeovers, as users' sensitive information may be exposed to untrusted origins. Such vulnerabilities can be particularly dangerous for applications handling financial, authentication, or personal data, potentially resulting in data breaches, compliance violations, and reputational damage.</p> | |
| Suggested Countermeasures | |
| <p>To prevent prefix match CORS vulnerabilities, always use exact string matching when validating the Origin header instead of partial or substring-based checks. Define a strict allowlist of trusted origins and avoid using wildcard (*) or dynamically reflecting origins without proper validation. Additionally, ensure sensitive endpoints do not expose credentials via CORS (Access-Control-Allow-Credentials: true) unless absolutely necessary. Regular security audits and testing can help detect and mitigate misconfigurations before they are exploited.</p> | |
| References | |
| https://www.tenable.com/blog/understanding-cross-origin-resource-sharing-vulnerabilities | |

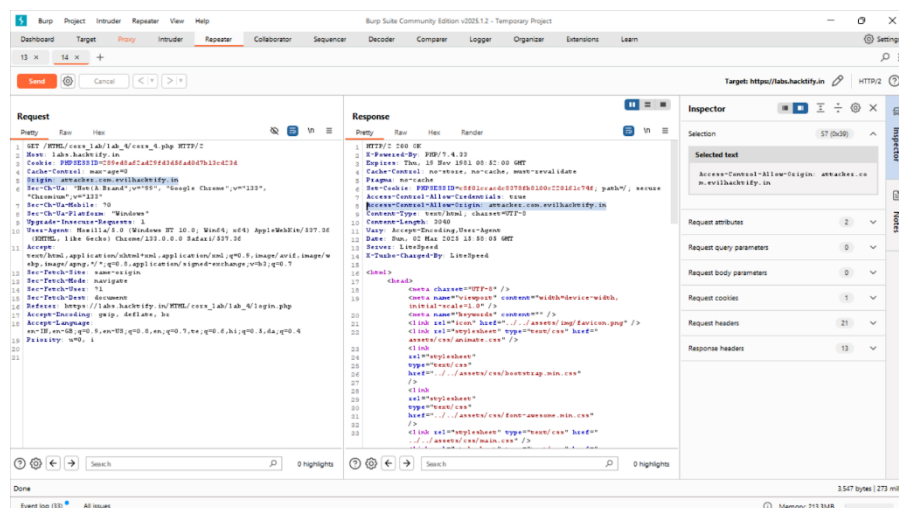
Proof of Concept

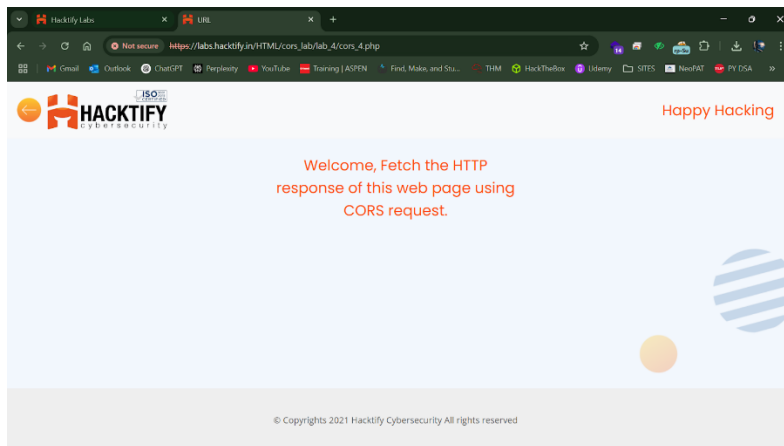


2.4. CORS with suffix match

| Reference | Risk Rating |
|---|-------------|
| CORS with suffix match | Medium |
| Tools Used | |
| Burpsuite | |
| Vulnerability Description | |
| A suffix match CORS vulnerability occurs when a server incorrectly validates the Origin header by allowing any domain that ends with a trusted string (e.g., allowing example.com but also mistakenly permitting attackerexample.com). Attackers can exploit this by hosting malicious domains that trick the server into granting cross-origin access , leading to data theft, unauthorized API requests, and session hijacking . | |
| How It Was Discovered | |
| Manual Analysis – Intercepting and modifying the request | |
| Vulnerable URLs | |
| https://labs.hacktify.in/HTML/cors_lab/lab_4/login.php | |
| Consequences of not Fixing the Issue | |
| If a suffix match CORS vulnerability is not fixed, attackers can create deceptive domains (e.g., malicious-example.com) to bypass security restrictions and gain unauthorized access to sensitive data or APIs . This can lead to user impersonation, data leaks, and financial fraud , especially in applications dealing with personal, authentication, or financial information , ultimately causing data breaches, compliance violations, and reputational damage . | |
| Suggested Countermeasures | |
| To prevent suffix match CORS vulnerabilities , always perform exact string matching for trusted origins instead of relying on partial or wildcard matching . Use a whitelist of fully qualified, explicitly trusted domains , and never allow dynamic or user-controlled origins unless validated. Additionally, implement strict authentication and authorization checks on the server-side, enforce SameSite cookies , and use security headers like Content-Security-Policy (CSP) to mitigate risks. Regular audits should be conducted to ensure proper CORS configurations. | |
| References | |
| https://www.tenable.com/blog/understanding-cross-origin-resource-sharing-vulnerabilities | |

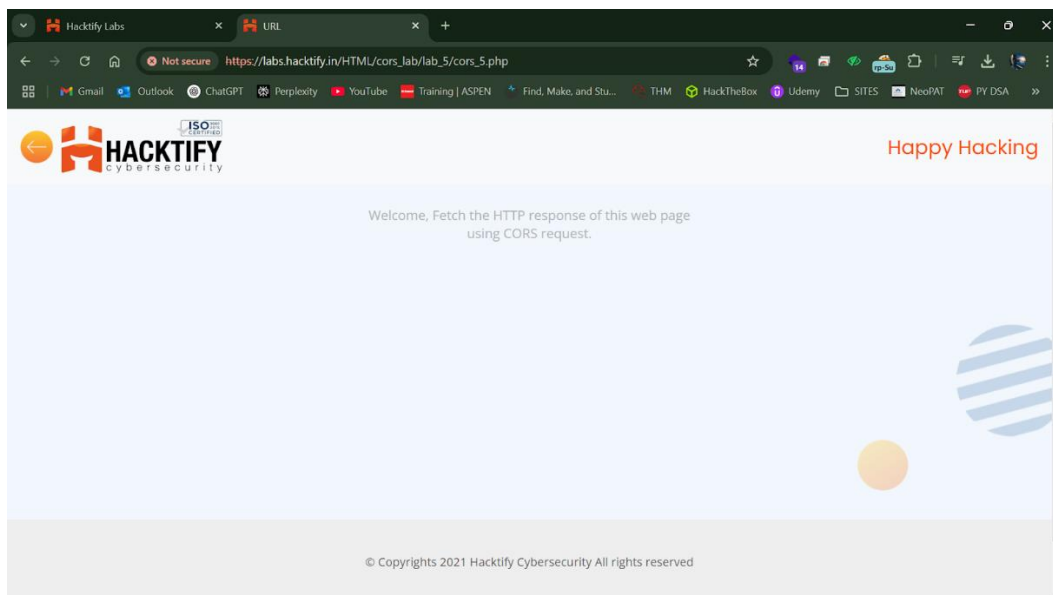
Proof of Concept





2.5. CORS with Escape dot

| Reference | Risk Rating |
|--|-------------|
| CORS with Escape dot | High |
| Tools Used | |
| Burpsuite | |
| Vulnerability Description | |
| A CORS with escape dot vulnerability occurs when a server misinterprets domain validation due to improper handling of escaped characters (e.g., \. instead of .). Attackers can exploit this by crafting malicious subdomains (trusted\.malicious.com instead of trusted.malicious.com) that bypass CORS restrictions. This can lead to unauthorized API access, data exfiltration, and session hijacking , compromising sensitive user information. | |
| How It Was Discovered | |
| Manual Analysis – Intercepting and modifying the request | |
| Vulnerable URLs | |
| https://labs.hacktify.in/HTML/cors_lab/lab_5/login.php | |
| Consequences of not Fixing the Issue | |
| If not fixed, attackers can exploit CORS misconfigurations to steal user data, hijack sessions, and perform unauthorized actions on behalf of users. This can affect authentication mechanisms, financial transactions, and private communications , leading to data breaches, compliance failures, and reputational damage . | |
| Suggested Countermeasures | |
| To mitigate CORS with escape dot vulnerabilities , always use exact string matching for trusted origins and avoid regex-based or partial matching . Implement strict CORS policies , validate and sanitize input properly, and enforce server-side authentication and authorization rather than relying on CORS alone. Regular security audits should be conducted to detect and fix misconfigurations. | |
| References | |
| https://www.tenable.com/blog/understanding-cross-origin-resource-sharing-vulnerabilities | |

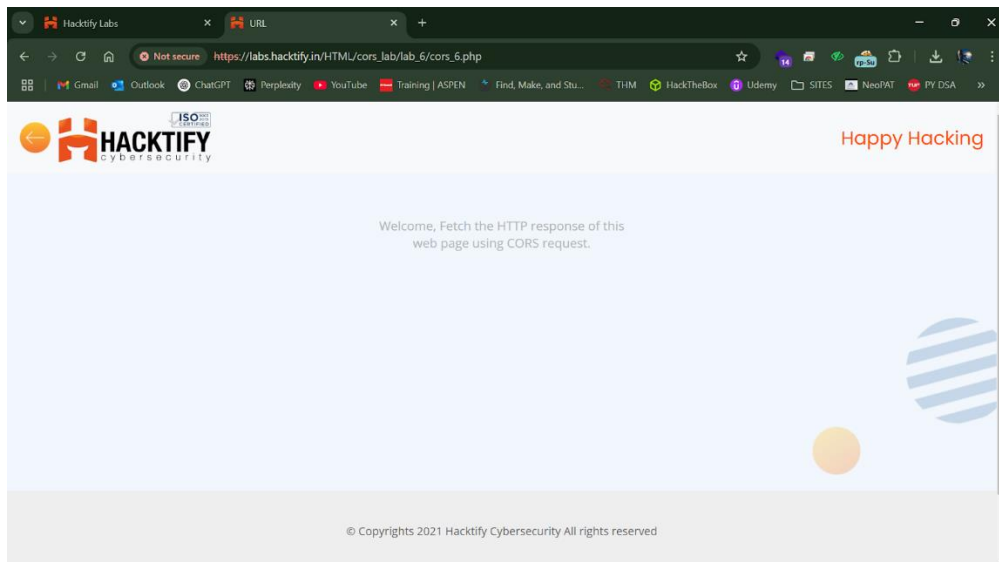
[illegible]

2.6. CORS with Substring match

| Reference | Risk Rating |
|--|-------------|
| CORS with Substring match | High |
| Tools Used | |
| Burpsuite | |
| Vulnerability Description | |
| A CORS with substring match vulnerability occurs when a server improperly validates the Origin header by allowing any domain that contains a trusted string (e.g., permitting trusted.com but also allowing eviltrusted.com). Attackers can exploit this misconfiguration by registering malicious domains containing the trusted string , gaining unauthorized cross-origin access to sensitive APIs and user data. | |
| How It Was Discovered | |
| Manual Analysis – Intercepting and modifying the request | |
| Vulnerable URLs | |
| https://labs.hacktify.in/HTML/cors_lab/lab_6/login.php | |
| Consequences of not Fixing the Issue | |
| If not fixed, attackers can exploit this weakness to steal sensitive data, perform unauthorized actions, hijack user sessions, or abuse APIs by hosting malicious domains containing the trusted substring. This can lead to data breaches, account takeovers, financial fraud, and compliance violations , ultimately damaging user trust and an organization's reputation. | |
| Suggested Countermeasures | |
| To prevent substring match vulnerabilities , always use exact string matching for CORS origin validation rather than partial or regex-based matching. Maintain a strict whitelist of fully qualified trusted domains , avoid dynamically reflecting user-provided origins, and implement server-side authentication and authorization instead of relying solely on CORS. Conduct regular security audits to detect and fix misconfigurations. | |
| References | |
| https://www.tenable.com/blog/understanding-cross-origin-resource-sharing-vulnerabilities | |

Proof of Concept

The screenshot displays the Burp Suite interface with a request and response captured. The request is a GET to `/HTML/cors_lab/6/cors_6.php` with an `Origin` header set to `https://labs.hacktify.in`. The response shows CORS headers, including `Access-Control-Allow-Origin: https://labs.hacktify.in`, indicating that the server is allowing the request from the specified origin.



2.7. CORS with Arbitrary Subdomain

| Reference | Risk Rating |
|---|-------------|
| CORS with Arbitrary Subdomain | High |
| Tools Used | |
| Burpsuite | |
| Vulnerability Description | |
| A CORS with arbitrary subdomain vulnerability occurs when a server overly trusts all subdomains using a wildcard (*.example.com) or improper validation, allowing any subdomain (e.g., attacker.example.com) to make cross-origin requests. If an attacker gains control of a subdomain through subdomain takeover or misconfiguration , they can bypass CORS restrictions, access sensitive data, and exploit API endpoints . | |
| How It Was Discovered | |
| Manual Analysis – Intercepting and modifying the request | |
| Vulnerable URLs | |
| https://labs.hacktify.in/HTML/cors_lab/lab_7/login.php | |
| Consequences of not Fixing the Issue | |
| If not fixed, attackers can create malicious subdomains to perform data theft, session hijacking, and unauthorized API access , leading to account takeovers, financial fraud, and exposure of sensitive information . This can result in data breaches, legal consequences, and reputational damage , especially for applications handling authentication, financial transactions, or personal data . | |
| Suggested Countermeasures | |
| To mitigate CORS with arbitrary subdomain vulnerabilities , avoid using wildcards (* or *.example.com) and enforce exact matching of trusted origins . Secure subdomains against takeover attacks , use strict authentication and authorization checks on the backend, and implement security headers like Content Security Policy (CSP) to prevent exploitation. Regularly audit DNS records and CORS policies to detect and fix misconfigurations. | |
| References | |
| https://www.tenable.com/blog/understanding-cross-origin-resource-sharing-vulnerabilities | |

Proof of Concept

The screenshot displays the Burp Suite interface with the 'Repeater' tab selected. The target is set to `https://labs.hacktify.in`. The request is a GET to `https://labs.hacktify.in/HTML/cors_lab/lab_7/cors_7.php`. The response is an HTTP/2 200 OK from `labs.hacktify.in`. The response headers include `Access-Control-Allow-Origin: https://evil1.hacktify.in`, indicating a successful CORS configuration. The response body contains HTML with a `<script>` tag that sets `document.cookie` to `test=test`.

```
1 GET /HTML/cors_lab/lab_7/cors_7.php HTTP/2
2 Host: labs.hacktify.in
3 Cookie: PHPSESSID=166125a0c9b44272a13d6a0b0c1e
4 Cache-Control: max-age=0
5 Origin: https://evil1.hacktify.in
6 Sec-CH-UA: "Brave Brand",v="99", "Google Chrome",v="113",
7 "Chromium",v="113"
8 Sec-CH-UA-Mobile: ?0
9 Sec-CH-UA-Platform: "Windows"
10 Upgrade-Insecure-Requests: 1
11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
12 (KHTML, like Gecko) Chrome/113.0.0.0 Safari/537.36
13 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
14 Accept-Encoding: gzip, deflate, br
15 Accept-Language: en-US,en;q=0.9,en-US;q=0.8,en;q=0.7,en;q=0.6,hi;q=0.5,da;q=0.4
16 Priority: u=0, i
17
18
19
20
21
```

```
1 HTTP/2 200 OK
2 X-Powered-By: PHP/7.4.33
3 Date: Thu, 18 Nov 2021 08:22:00 GMT
4 Cache-Control: no-cache, no-store, max-age=0, must-revalidate
5 Pragma: no-cache
6 Sec-Content-Security-Policy: default-src 'self'; script-src 'self' https://evil1.hacktify.in
7 Access-Control-Allow-Origin: https://evil1.hacktify.in
8 Access-Control-Allow-Credentials: true
9 Content-Type: text/html; charset=UTF-8
10 Content-Length: 2021
11 Vary: Accept-Encoding,User-Agent
12 Date: Thu, 18 Nov 2021 08:22:00 GMT
13 Server: LiteSpeed
14 X-Push-Changed-By: LiteSpeed
15
16 <html>
17
18 <meta charset="UTF-8" />
19 <meta name="viewport" content="width=device-width,
20 initial-scale=1.0" />
21 <meta name="keywords" content="" />
22 <link rel="icon" href="https://evil1.hacktify.in/assets/img/favicon.png" />
23 <link rel="stylesheet" type="text/css" href="https://evil1.hacktify.in/assets/css/animate.css" />
24 <link rel="stylesheet" type="text/css" href="https://evil1.hacktify.in/assets/css/bootstrap.min.css" />
25 <link rel="stylesheet" type="text/css" href="https://evil1.hacktify.in/assets/css/font-awesome.min.css" />
26 <link rel="stylesheet" type="text/css" href="https://evil1.hacktify.in/assets/css/main.css" />
27
28 <script>
29 document.cookie = "test=test";
30
31 </script>
32
```

The screenshot shows a web browser window with the URL `https://labs.hacktify.in/HTML/cors_lab/lab_7/cors_7.php`. The page features the Hacktify Labs logo and a message: "Welcome, Fetch the HTTP response of this web page using CORS request." The footer includes the copyright notice: "© Copyrights 2021 Hacktify Cybersecurity All rights reserved."