

Penetration Testing Report

Full Name:G M Sohanur Rahman

Program: HCPT

Date: 10 March, 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	Server-Side Request Forgery, Cross-Site Request Forgery
------------------	---

3. Summary

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

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

High	Medium	Low
4	5	6

- High

-

Number of Sub-labs with hard difficulty level
- Medium

-

Number of Sub-labs with Medium difficulty level
- Low

-

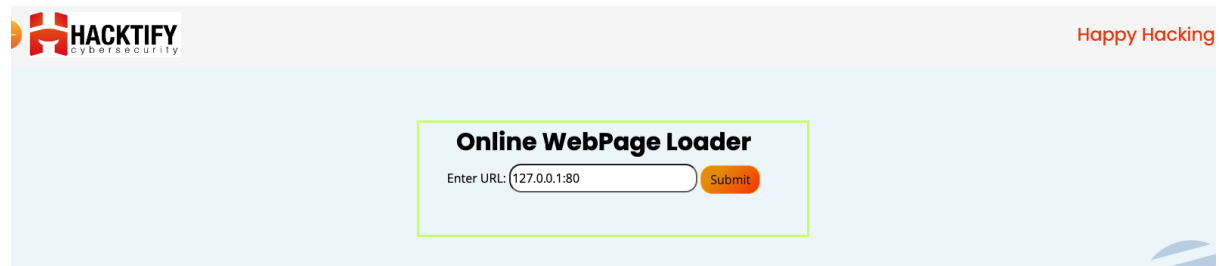
Number of Sub-labs with Easy difficulty level

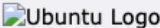
1. Server-Side Request Forgery

1.1. Get The 127.0.0.1

Reference	Risk Rating
Get The 127.0.0.1	Low
Tools Used	
Browser	
Vulnerability Description	
In the payload, "127.0.0.1:80" is an example of such an attack. The payload is instructing the vulnerable server to make a request to localhost (the server itself) on port 80. This is an example of an intraservice interaction, where the attacker can interact with services on the same server.	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/ssrf_lab/localhost/index.html	
Consequences of not Fixing the Issue	
<p>The consequences of not fixing a Server Side Request Forgery (SSRF) vulnerability can be significant and potentially catastrophic. Here are some potential problems:</p> <p>Internal Network Exposure: SSRF vulnerabilities can expose internal infrastructures that are generally protected by firewalls. An attacker can carry out actions as if they originated from the vulnerable server itself.</p> <p>Arbitrary Read/Write Operations: Depending on the nature of the vulnerability and the server's permissions, an attacker could potentially read or write files to the server.</p> <p>Information Disclosure: If the server's response from the exploited request is processed and rendered by the vulnerable application, an attacker could gain access to sensitive information.</p> <p>Remote Code Execution: In severe cases, SSRF can even lead to Remote Code Execution (RCE), where the attacker can run arbitrary commands on the server.</p> <p>Service Disruption: Attackers can use SSRF to make arbitrary HTTP requests, potentially leading to denial-of-service attacks on internal services.</p> <p>Escalation: Since SSRF can bypass firewalls, this could lead to further vulnerabilities being discovered and exploited by attackers.</p>	
Suggested Countermeasures	
Mitigating SSRF vulnerabilities mainly involves input validation and restricting what the server can send requests to. Safe inputs should be identified and white-listed, and anything that does not match these safe inputs should be rejected. You can use Allowlisting of IPs, meaning only permitted IP addresses should be addressed from the server-side.	
References	
https://owasp.org/www-community/attacks/Server_Side_Request_Forgery	

Proof of Concept



Ubuntu Logo

Apache2 Ubuntu Default Page

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in `/usr/share/doc/apache2/README.Debian.gz`**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

```
/etc/apache2/
|-- apache2.conf
/   |-- ports.conf
|-- mods-enabled
/   |-- *.load
/   |-- *.conf
|-- conf-enabled
/   |-- *.conf
|-- sites-enabled
/   |-- *.conf
```

1.2. Http(S)? Nevermind!!

Reference	Risk Rating
Http(S)? Nevermind!!	Low
Tools Used	
Browser	
Vulnerability Description	
The payload "http://localhost:80" is used to illustrate a typical Server Side Request Forgery (SSRF) vulnerability. In an SSRF attack, the attacker causes the server to make a network request to an	

arbitrary URL. In this case, the arbitrary URL is "http://localhost:80". "localhost" is a hostname that refers to the current device, and "80" is the default port number for HTTP.

If an application is vulnerable to SSRF, an attacker might use a payload like this to induce the server to make requests to resources on the server's own network (i.e., "localhost"), bypassing restrictions and potentially gaining access to information or functionality that would normally be protected or hidden.

How It Was Discovered

Manual Analysis

Vulnerable URLs

https://labs.hacktify.in/HTML/ssrf_lab/localhost/index.html

Consequences of not Fixing the Issue

Not fixing a Server Side Request Forgery (SSRF) vulnerability, such as the one demonstrated by the "http://localhost:80" payload, can lead to serious consequences. These can include exposure of internal systems and data, unauthorized actions carried out under the server's identity, and potential for an attacker to pivot to more critical systems within the network. In certain scenarios, it may even allow file read/write operations on the server, leading to a significant breach of security. Not addressing this vulnerability could result in extensive damage to your data or system infrastructure and could pose a significant risk to your organization's operations and reputation.

Suggested Countermeasures

Mitigating Server Side Request Forgery (SSRF) essentially involves strict input validation and limiting the ability of your server to make arbitrary requests. URLs should be properly validated and requests should only be made to trusted domains. Safely fetching remote resources is recommended, which is best handled by utilizing secure libraries. Implementing network restrictions such as firewalls, and adopting same-origin policies can help block undesired traffic. Regular patching and software updates should also be carried out to ensure all security measures are updated. Additionally, securing your internal systems and limiting their access to minimize the impact in case of a breach is highly recommended. Especially, server permissions should be as restrictive as possible. These steps, combined with closely monitoring HTTP requests for unusual patterns, can help protect your systems against SSRF attacks.

References

https://owasp.org/www-community/attacks/Server_Side_Request_Forgery

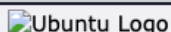
Proof of Concept



Happy Hacking

Online WebPage Loader

Enter URL:



Apache2 Ubuntu Default Page

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in `/usr/share/doc/apache2/README.Debian.gz`**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

```
/etc/apache2/  
|-- apache2.conf  
|   |-- ports.conf  
|-- mods-enabled  
|   |-- *.load  
|   |-- *.conf  
|-- conf-enabled  
|   |-- *.conf  
|-- sites-enabled  
|   |-- *.conf
```

1.3. ":" The Saviour!

Reference	Risk Rating
":" The Saviour!	Low
Tools Used	
Browser	
Vulnerability Description	
<p>The payload "<code>http://[::]:80</code>" is another example of a Server Side Request Forgery (SSRF) attack, where the target is specified using IPv6 syntax. The ":" in an IPv6 address represents a series of consecutive zeros, thus "<code>http://[::]:80</code>" is equivalent to targeting "<code>http://0.0.0.0:80</code>" or "<code>http://localhost:80</code>" in IPv4 notation.</p> <p>This payload, when used in an SSRF vulnerable context, tries to trick the server to send a request to itself over port 80, the default HTTP port. What actions are performed with this request will depend on what functionality is exposed at that address on the server. This could range from harmless operations to actions that have serious security implications like accessing unexposed internal services, exploiting unpatched internal vulnerabilities, or exfiltrating data.</p>	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	

https://labs.hacktify.in/HTML/ssrf_lab/localhost/index.html

Consequences of not Fixing the Issue

The consequences of not resolving a Server Side Request Forgery (SSRF) vulnerability can be serious. If an attacker can trick your server into making requests on their behalf, they could potentially interact with internal services that are normally protected from external access. This could lead to unauthorized actions being performed and sensitive information being exposed. For example, database credentials or other internal sensitive resources could be accessed. Moreover, if your server has a service running with unmatched vulnerabilities, this service can be exploited through this SSRF. Leaving this vulnerability unattended could result in a full system compromise, significant damage to your infrastructure, loss of sensitive data, and a potential breach of legal data protection requirements. This could also harm your organization's reputation because of a privacy violation.

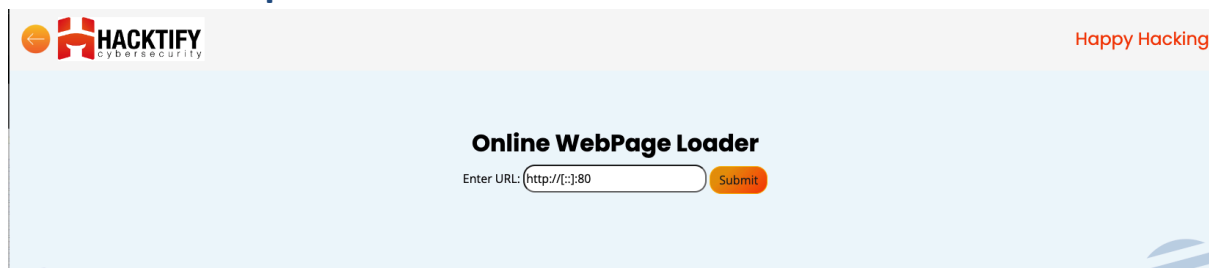
Suggested Countermeasures

To mitigate Server Side Request Forgery (SSRF) vulnerabilities like the one indicated by "http://[::]:80", you could strengthen input validation routines and adopt safe handling methods for URLs. Inputs should be properly validated and limited to trusted domains. Incorporate a more stringent policy of blocking connections to localhost or private networks from web applications. Regular patching and updating of server software should be implemented to fix any known bugs or vulnerabilities. If possible, limit the ability of your server to initiate outbound connections. Use firewall rules to segment your network and strictly limit outbound traffic from your servers. Consider using a Web Application Firewall (WAF) to help detect and block SSRF attacks. Lastly, improve the monitoring of your server and alert on unusual outgoing traffic patterns that may indicate an SSRF attack.

References

https://owasp.org/www-community/attacks/Server_Side_Request_Forgery

Proof of Concept



1.4. Messed Up Domain!

Reference	Risk Rating
Messed Up Domain!	Medium
Tools Used	
Browser	
Vulnerability Description	
In the payload URL, an attacker is using a localhost bypass technique to potentially exploit SSRF vulnerability in an application running on the server. "http://customer1.app.localhost.my.company.127.0.0.1.nip.io" will resolve to the local IP "127.0.0.1". If the targeted server-side application is vulnerable to SSRF, this payload might trick the application into	

making a request back to itself or access other services running on the localhost that should not be exposed, leading to unauthorized actions.

How It Was Discovered

Manual Analysis

Vulnerable URLs

https://labs.hacktify.in/HTML/ssrf_lab/lab_4/lab_4.php

Consequences of not Fixing the Issue

If an SSRF vulnerability is not fixed, it can allow an attacker to abuse the functionality of a server, leading to a range of harmful actions. These could include executing requests to internal resources, triggering actions on behalf of the server, gaining access to internal databases, and even remote code execution.

Suggested Countermeasures

Here are few countermeasures to protect against SSRF attacks:

Input Validation: Validate, sanitize and limit the input received from the user. Use a maintain list of safe or allowed input and reject the requests not meeting the criteria.

Block Private IPs: Implement mechanisms to block requests to loopback and private IP addresses to prevent the application from interacting with internal services.

Allowlist URLs: If the application needs to fetch remote resources, maintain a allowlist of trusted domains or IP addresses and only permit http/s traffic to those.

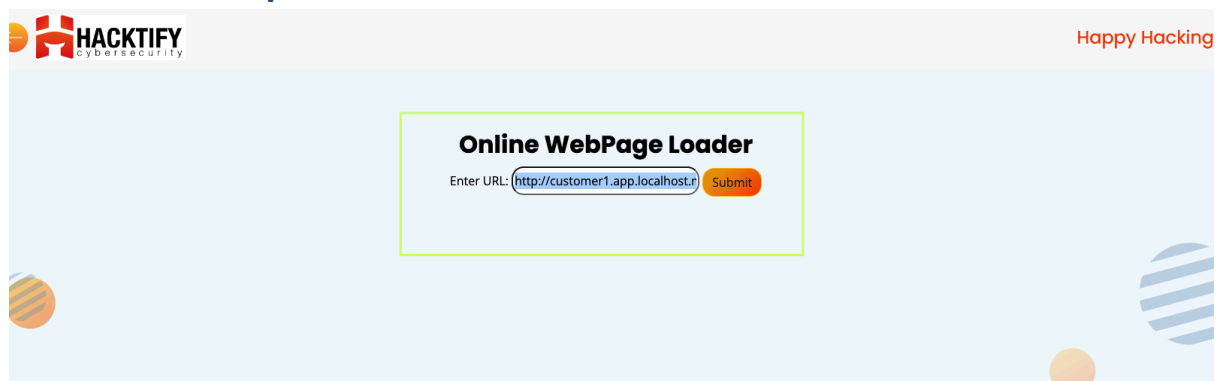
Use a Server-side Proxy: Instead of directly sending the user-provided Url to the vulnerable function, use a proxy service on server side to route the request.

Update and Patch Systems: Keep the systems and server-side scripts updated and apply security patches regularly.

References

https://owasp.org/www-community/attacks/Server_Side_Request_Forgery

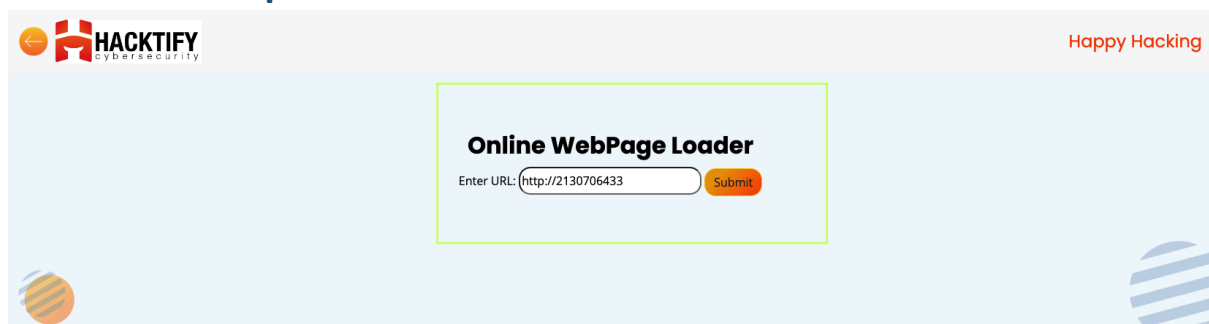
Proof of Concept



1.5. Decimal IP

Reference	Risk Rating
Decimal IP	Medium
Tools Used	
Browser	
Vulnerability Description	
<p>IP addresses are typically represented in human-friendly four-part decimal format (e.g., 192.168.1.1), but they can also be represented as a single decimal number. For example, the IP address 192.168.1.1 can be converted into a decimal format like 3232235777.</p> <p>If a system only blocks known suspicious IP addresses in the standard decimal format, an attacker could use the decimal representation to bypass this restriction. For example, if the "localhost" IP address 127.0.0.1 (which converts to 2130706433 in decimal format) is blocked, attackers may use the decimal equivalent to reach these internal resources, thereby exploiting an SSRF vulnerability.</p>	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/ssrf_lab/lab_5/lab_5.php	
Consequences of not Fixing the Issue	
<p>Internal systems Exposure: An attacker could use the vulnerability to gain access to internal systems that are generally inaccessible from the internet. This might disclose sensitive internal information or enable further network exploitation.</p> <p>Data Leakage: The vulnerability could be exploited to access and leak sensitive data, which could lead to data breaches, violation of privacy laws, and loss of customer trust.</p> <p>Denial of Service: By forcing the server to make unnecessary or resource-intensive requests, an attacker could potentially cause denial of service.</p>	
Suggested Countermeasures	
To prevent this, security measures must account for various IP representations, including decimal format. Potential evasion techniques like IP transformations should be considered in the defense mechanisms. Also, validate or block all non-routable or reserved IP address spaces commonly used for private or internal networks.	
References	
https://owasp.org/www-community/attacks/Server_Side_Request_Forgery	

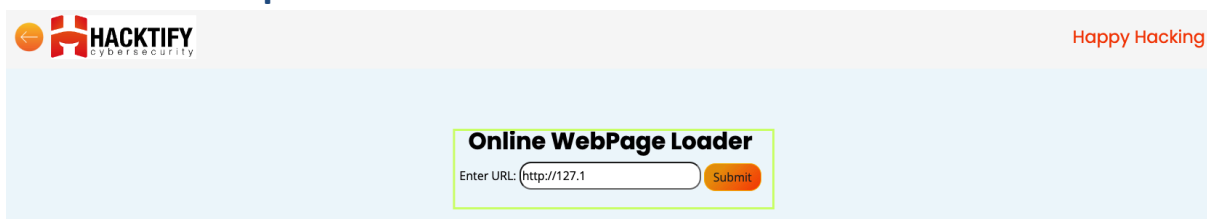
Proof of Concept



1.6. Short-Hand IP Address

Reference	Risk Rating
Short-Hand IP Address	Medium
Tools Used	
Browser	
Vulnerability Description	
Short-hand IP address notation is one of the methods used by attackers to exploit SSRF (Server Side Request Forgery) vulnerabilities and bypass weak protection mechanisms. Typically, we are most familiar with IP addresses in their standard form, like 192.0.2.0. However, this same IP can also be represented in a short-hand format by omitting the trailing zeros. So, 192.0.2.0 becomes simply "192.2".	
How It Was Discovered	
Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/ssrf_lab/lab_6/index.php	
Consequences of not Fixing the Issue	
If the system employs input validation to block known internal IP addresses (like 127.0.0.1 for localhost) but does not account for short-hand notation, an attacker can use the short-hand format to bypass the validation and interact with internal services accessible by the server.	
Suggested Countermeasures	
To mitigate Short-hand IP address SSRF vulnerability, it's crucial to use robust input validation that considers all possible representations of IP addresses, including short-hand or alternative formats. Furthermore, block all non-routable IP address spaces typically used for internal networks to minimize the potential exposure of internal system resources to such SSRF attacks.	
References	
https://owasp.org/www-community/attacks/Server_Side_Request_Forgery	

Proof of Concept

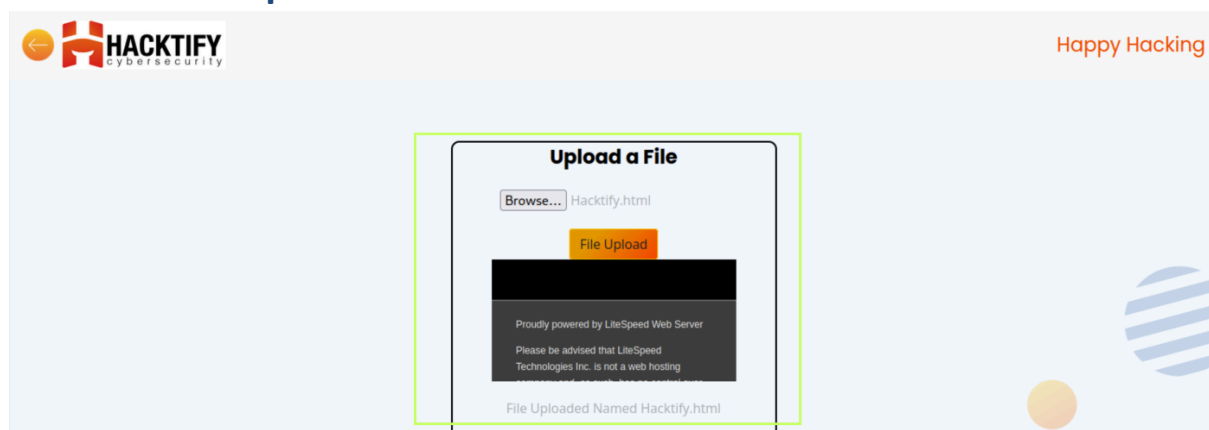


1.7. Short-Hand IP Address

Reference	Risk Rating
File Upload To SSRF!	Low
Tools Used	

Browser
Vulnerability Description
"File Upload SSRF" is a specific form of SSRF where the vulnerability happens during the process of file uploading. An attacker can upload a malicious file which contains scripts to force the server to make requests to any URL the attacker wants. If the web server has inadequate security settings, this could lead to stealing sensitive data, performing actions without permissions, or even attacking other servers within the network.
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
<p>Failing to fix a File Upload SSRF (Server-Side Request Forgery) vulnerability can have several severe consequences, including:</p> <p>Data Exposure: Sophisticated attackers can exploit SSRF vulnerabilities to access sensitive information, which could include user data, internal network details, API keys, or server configurations.</p> <p>Internal Network Attack: SSRF vulnerabilities can provide an opportunity for attackers to target internal systems that are not exposed to the internet.</p> <p>Denial of Service: Attackers could cause a Denial of Service (DoS) by diverting the server to a target that overwhelms it, forcing the legit users to face service outages.</p>
Suggested Countermeasures
<p>Here are some suggested countermeasures to prevent File Upload SSRF (Server-Side Request Forgery) vulnerabilities:</p> <p>Whitelist Input Validation: The system should white-list input validation, that is, strictly define what input is allowed. This can dramatically reduce the chance of SSRF.</p> <p>Firewalling Internal Services: Block requests originating from the server itself or unexpected outbound traffic to keep your internal systems safe.</p> <p>Rate Limiting: Impose limits on how many requests a user/IP can make in a certain timeframe.</p> <p>Disable Unnecessary URL Schemas: Only allow the schemas that are required for your application to function correctly.</p>
References
https://sm4rty.medium.com/hunting-for-bugs-in-file-upload-feature-c3b364fb01ba

Proof of Concept



2. Cross-Site Request Forgery

2.1. Eassyy CSRF

Reference	Risk Rating
Eassyy CSRF	Low
Tools Used	
Burp	
Vulnerability Description	
<p>CSRF (Cross-Site Request Forgery) is a type of security vulnerability in web applications that tricks the victim into performing actions they did not intend to. This happens when a malicious website, email, or program causes a user's web browser to perform an unwanted action on a trusted site on which the user is authenticated.</p> <p>For example, if a user is logged into a banking website and simultaneously visits a malicious website, the malicious site could potentially make a request to the banking site to transfer money without the user's consent.</p>	
How It Was Discovered	
Automated Tools / Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/csrf_lab/lab_1/index.php	
Consequences of not Fixing the Issue	
Some potential hazards of CSRF vulnerability:	

Unauthorized actions: The attacker can make the user perform actions they didn't intend to, such as changing their email address or password.

Data loss: The attacker can delete data on behalf of the user.

Account theft: If the attacker makes the user perform actions on their own account, they can effectively steal the user's account.

Suggested Countermeasures

To prevent CSRF vulnerabilities, it's recommended to:

Use anti-forgery tokens, also known as CSRF tokens. This involves server-side generation of a random, unique value associated with the user's current session which must be included with every state-changing request.

Use the SameSite cookie attribute, which allows you to declare if your cookie should be restricted to a first-party or same-site context, effectively preventing cross-origin requests.

Incorporate re-authentication, CAPTCHAs, or multi-factor authentication for sensitive actions.

Regularly update and patch all systems, libraries, and frameworks used by your web application to minimize the risk of known CSRF vulnerabilities.

References

https://owasp.org/www-project-cheatsheet-series/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

Proof of Concept

```
CSRF HTML:
1 <html>
2 <!-- CSRF PoC - generated by Burp Suite Professional -->
3 <body>
4   <form action="https://labs.hacktify.in/HTML/csrf_lab/lab_1/passwordChange.php" method="POST">
5     <input type="hidden" name="newPassword" value="hacker" />
6     <input type="hidden" name="newPassword2" value="hacker" />
7     <input type="submit" value="Submit request" />
8   </form>
9   <script>
10    history.pushState('', '', '/');
11    document.forms[0].submit();
12  </script>
13 </body>
14 </html>
15
```

2.2. Always Validate Tokens

Reference	Risk Rating
Always Validate Tokens	Medium
Tools Used	
Burp	
Vulnerability Description	
<p>A CSRF Validate Token Vulnerability arises when an application defines its CSRF token improperly or does not check the CSRF token properly. CSRF (Cross-Site Request Forgery) token is a measure to prevent CSRF attacks, but if mishandled, can lead to the very attacks it is supposed to prevent.</p> <p>When the server receives a potentially unsafe HTTP request such as POST, PATCH, DELETE, etc., it should check whether this request includes a valid CSRF token. This token should be securely randomly generated and associated with the user's session. If the token is not included in the request or does not match the value on the server-side, the server should reject the request.</p>	
How It Was Discovered	
Automated Tools / Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/csrf_lab/lab_2/index.php	
Consequences of not Fixing the Issue	
<p>If an application has a CSRF Validate Token Vulnerability, it may allow such requests without a valid CSRF token. This could happen due to logic errors in the code that checks for the CSRF token or incorrect session handling, which mismatches user sessions with CSRF tokens.</p> <p>As a result, an attacker could perform actions on the web application on behalf of the targeted user without their knowledge or consent. Examples of these actions include changing the user's email or password, performing financial transactions, or any other action that the user is authorized to perform on their own account. Therefore, properly validating CSRF tokens is crucial in protecting web applications from such security threats.</p>	
Suggested Countermeasures	
<p>To properly manage CSRF (Cross-Site Request Forgery) Validate Token Vulnerabilities, here are some recommended countermeasures:</p> <p>Implement Anti-CSRF Tokens: Ensure that every sensitive action or request must be accompanied by a unique CSRF token that is securely randomly generated and associated with the user's session.</p> <p>Check Every Request: Every state-changing request (POST, DELETE, UPDATE, etc.) should be checked for a valid CSRF token. If the token is missing or does not match the expected value, reject the request.</p> <p>Tie Tokens to User Session: The CSRF token should be tied to the individual user's session. This will make it extremely difficult for an attacker to guess the CSRF token for a particular user session.</p>	
References	
https://owasp.org/www-project-cheatsheet-series/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html	

Proof of Concept

```
9 Content-Length: 78
10 Origin: https://labs.hacktify.in
11 Referer: https://labs.hacktify.in/HTML/csrf_lab/lab_2/passwordChange.php
12 Upgrade-Insecure-Requests: 1
13 Sec-Fetch-Dest: document
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-User: ?1
17 Te: trailers
18
19 newPassword=Xkernel&newPassword2=Xkernel&csrf=5f9653d9eb16376ec65a14f6dfb43651
```

2.3. I Hate When Someone Uses My Tokens!

Reference	Risk Rating
I Hate When Someone Uses My Tokens!	Medium
Tools Used	
Burp	
Vulnerability Description	
<p>Cross-Site Request Forgery (CSRF) token vulnerability is a specific kind of security flaw where a website or web application does not properly implement or handle CSRF tokens, a critical security measure to prevent CSRF attacks. CSRF tokens are unique, random values that are assigned and validated for each session and user interaction, and they help to ensure that actions and requests made on a site are genuinely from the authenticated user, not from a malicious attacker. If these tokens are missing, not properly associated with the user session, or not adequately randomized, it can lead to CSRF token vulnerability. This allows an attacker to forge malicious requests mimicking the authenticated user's actions, potentially leading to unauthorized data access, data manipulation, account theft, or other unwanted actions.</p>	
How It Was Discovered	
Automated Tools / Manual Analysis	
Vulnerable URLs	
https://labs.hacktify.in/HTML/csrf_lab/lab_4/index.php	
Consequences of not Fixing the Issue	
<p>Not fixing the Cross-Site Request Forgery (CSRF) token vulnerability can have severe consequences. Here are a few:</p> <p>Unauthorized Actions: Attackers could trick users into performing unwanted actions on their behalf. They can create requests that could potentially change user email, password, perform financial transactions etc.</p> <p>Data Breach: Weak CSRF token mechanism could lead to data breaches since attackers might gain access to sensitive information.</p>	

Legal Implications: If a data breach occurs because of a CSRF vulnerability, the organization could face legal actions, especially if it's dealing with sensitive user data.

Reputational Damage: Repeated attacks could lead to loss of trust among users and customers, leading to reputational damage and potential loss of revenue.

Suggested Countermeasures

Use Anti-CSRF Tokens: The most commonly used mitigation technique is the use of anti-CSRF tokens. These tokens can be tied to a user's session and included with each HTTP request.

Same Site Cookies: These can be used to assert that a certain cookie should only be sent via requests from the same site. This is supported in new browsers and will automatically withhold cookies in cross-site POST requests.

Same Origin Policy Checks: Employ Same-Origin Policy (SOP) checks in XMLHttpRequest or fetch APIs to restrict which scripts running on your website can access the response from a request your websites sends.

References

<https://portswigger.net/support/using-burp-suites-session-handling-rules-with-anti-csrf-tokens>

Proof of Concept

```
9 Content-Length: 78
10 Origin: https://labs.hacktify.in
11 Referer: https://labs.hacktify.in/HTML/csrf_lab/lab_2/passwordChange.php
12 Upgrade-Insecure-Requests: 1
13 Sec-Fetch-Dest: document
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-User: ?1
17 Te: trailers
18
19 newPassword=xkernel&newPassword2=xkernel&csrf=5f9653d9eb16376ec65a14f6dfb43651
```

2.5. XSS The Saviour

Reference	Risk Rating
XSS The Saviour	Hard
Tools Used	
Browser	
Vulnerability Description	
The payload <code><script>alert(document.cookie)</script></code> is used in an XSS attack. This simple JavaScript code makes an alert box displaying the website's cookies in the victim's browser.	

Here's how it works:

An attacker injects this malicious script into your website.

When a user visits the compromised webpage, the script runs in their browser.

The script accesses document.cookie - JavaScript's way of accessing all of the cookies associated with the domain.

Then, alert(document.cookie) pops up an alert box displaying all the cookies.

How It Was Discovered

Manual Analysis

Vulnerable URLs

https://labs.hacktify.in/HTML/csrf_lab/lab_7/lab_7.php?name=%3Cscript%3Ealert%28document.cookie%29%3C%2Fscript%3E&show=Save

Consequences of not Fixing the Issue

If the website doesn't properly sanitize user inputs, an attacker could inject such scripts. If they are using session-cookies for login, an attacker could steal these cookies and impersonate the victim's session.

Suggested Countermeasures

Preventing XSS and CSRF requires different approaches. For XSS, it's crucial to sanitize user inputs and for CSRF, it's important to verify the origin of requests, often achieved by using anti-CSRF tokens.

References

<https://owasp.org/www-community/attacks/csrf>

Proof of Concept

