Report On Security Testing Assignment -1

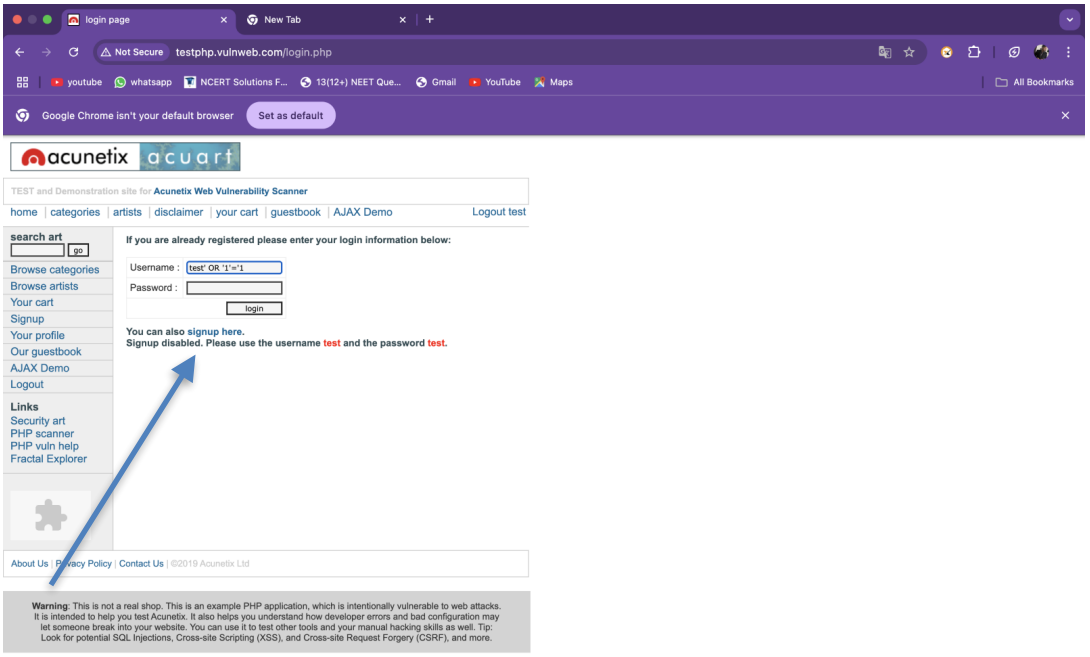
Summary Of Topics:

SQL injection (SQLi):

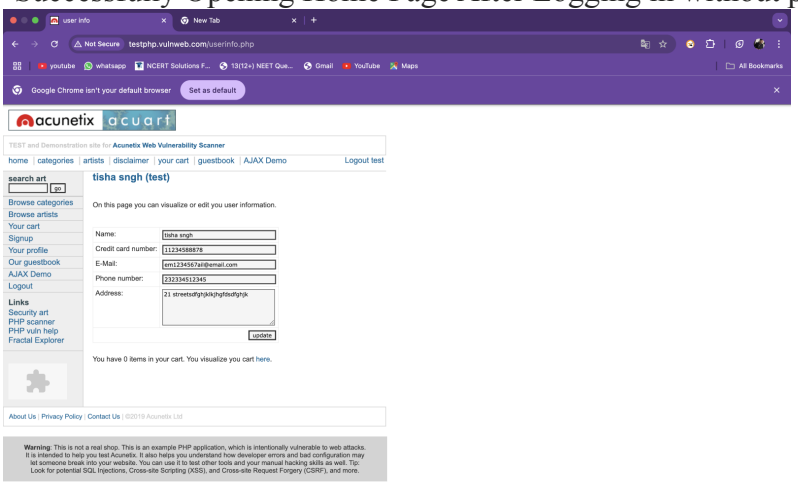
- SQL injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database.
- This can allow an attacker to view data that they are not normally able to retrieve.
- This might include data that belongs to other users, or any other data that the application can access.
- In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior.

Steps To Check the SQL injection Vulnerability:

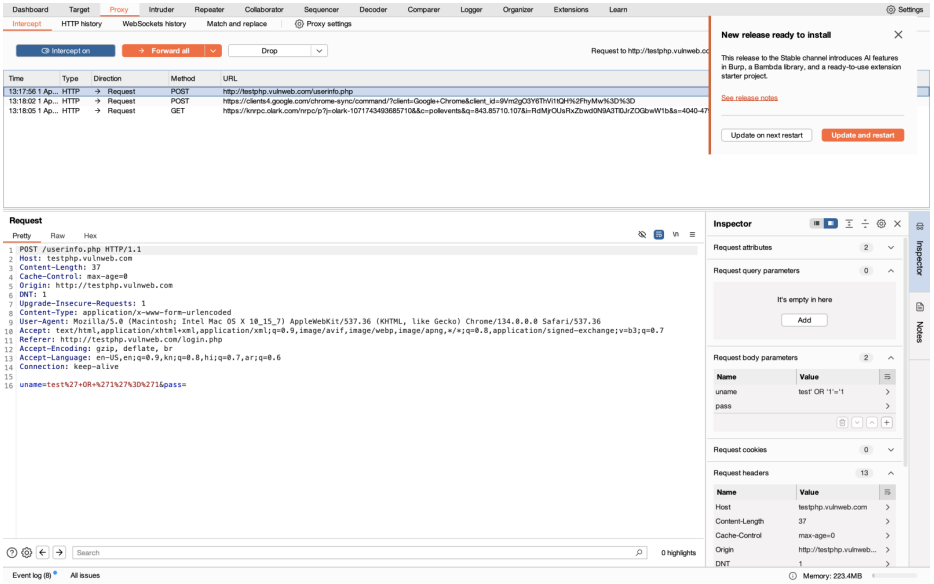
- Open the target site.
- Open Login Page, Where Injecting SQL query as username without password.



- Successfully Opening Home Page After Logging in without password .



- Interceptor Page Where we can see the username and we can also edit it and execute it.



XSS Attack:

Cross-Site Scripting (XSS) is a web security vulnerability that allows attackers to inject malicious scripts into web applications. These scripts are executed in the browser of an unsuspecting user, leading to data theft, session hijacking, or even the takeover of user accounts. XSS is one of the most prevalent vulnerabilities found in web applications today.

Types of XSS:

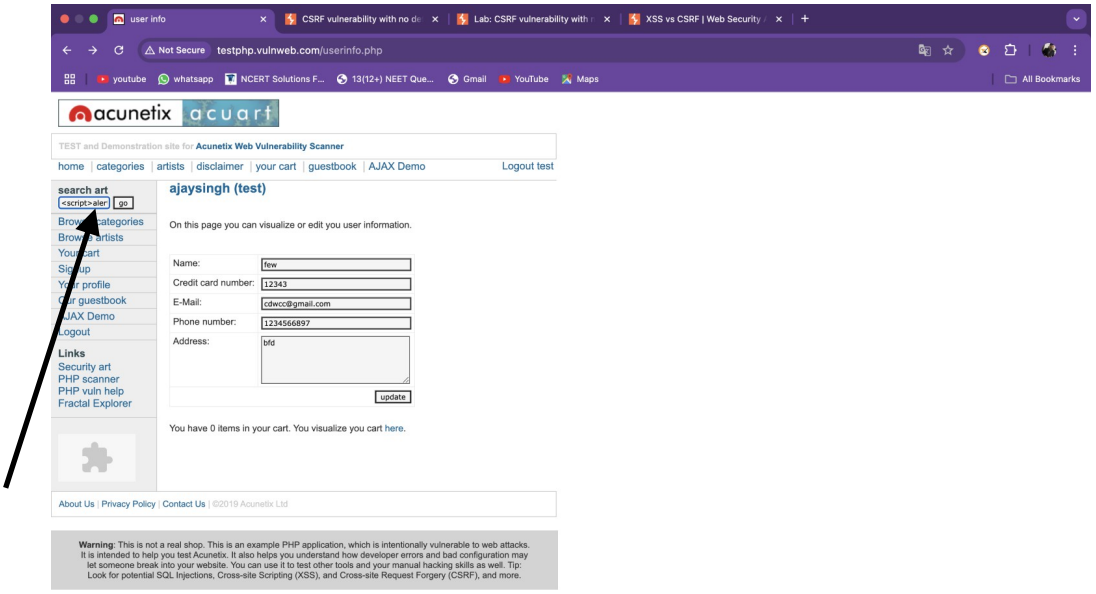
i) Reflected XSS:

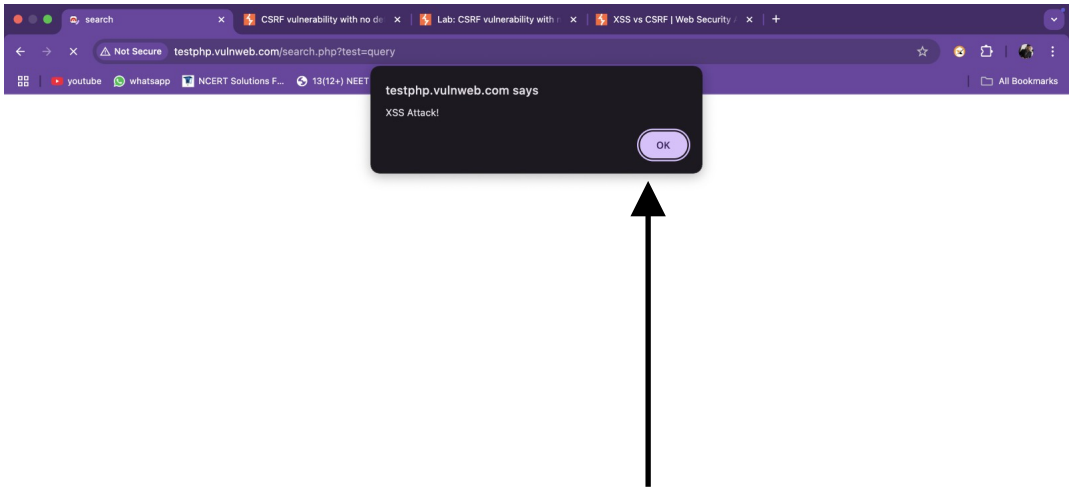
Description: Reflected XSS occurs when user input is immediately reflected on a webpage without proper sanitization. The attack is executed when a victim clicks on a maliciously crafted link containing a malicious script.

Malicious Code: <script>alert('XSS Attack!')</script> **Steps To Check the Reflected XSS Vulnerability:**

- Open the target site.
- Enter the Malicious code into the targeted field, here the Search text field is target site.
- Then hit go button.
- After that automatically a pop message will come saying as XSS Attack, which is the Malicious code which i have injected.

Snapshots:





ii) Stored XSS:

Description: Stored XSS occurs when a malicious script is permanently stored on the server, such as in a database, and is executed whenever a user accesses the affected page.

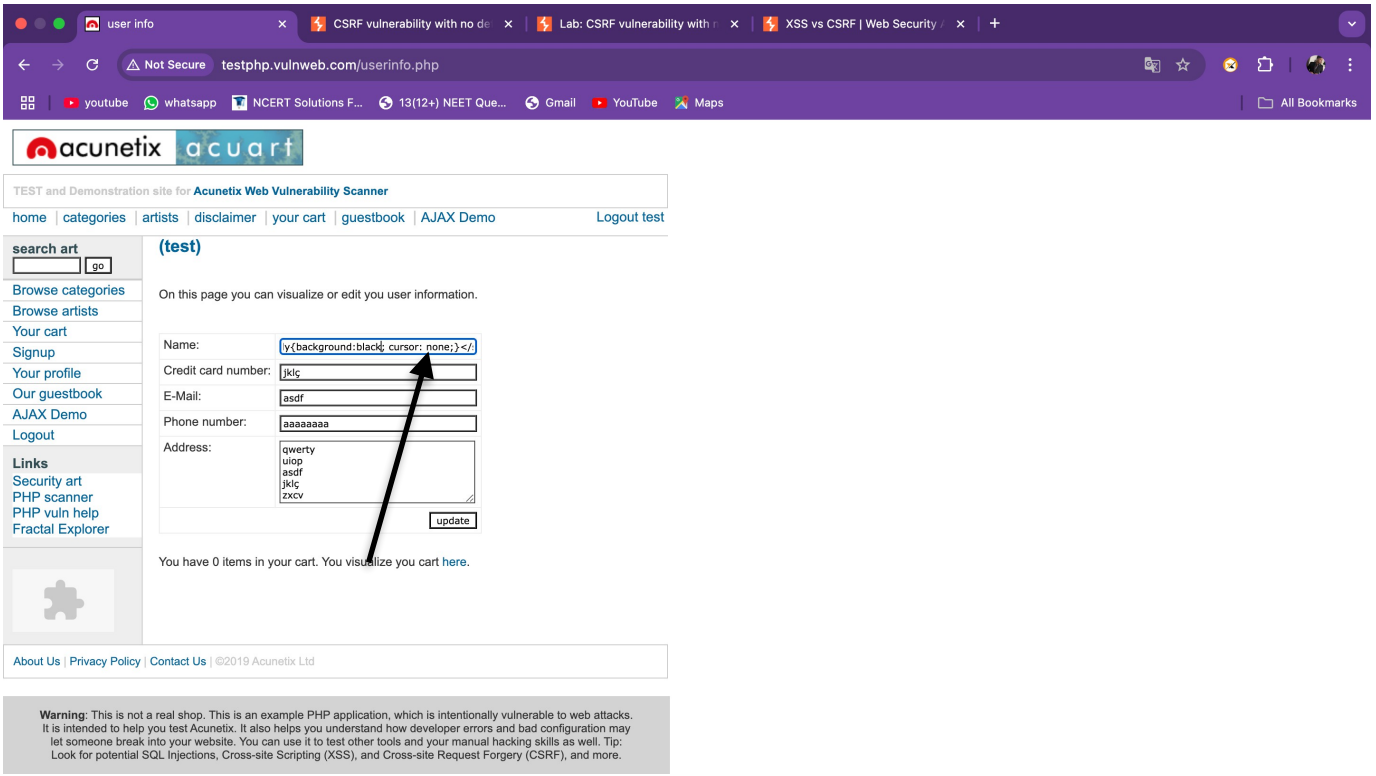
Malicious Code: <style>a,textarea,input,body{background:black; cursor:none;}</style>

Steps To Check the Reflected XSS Vulnerability:

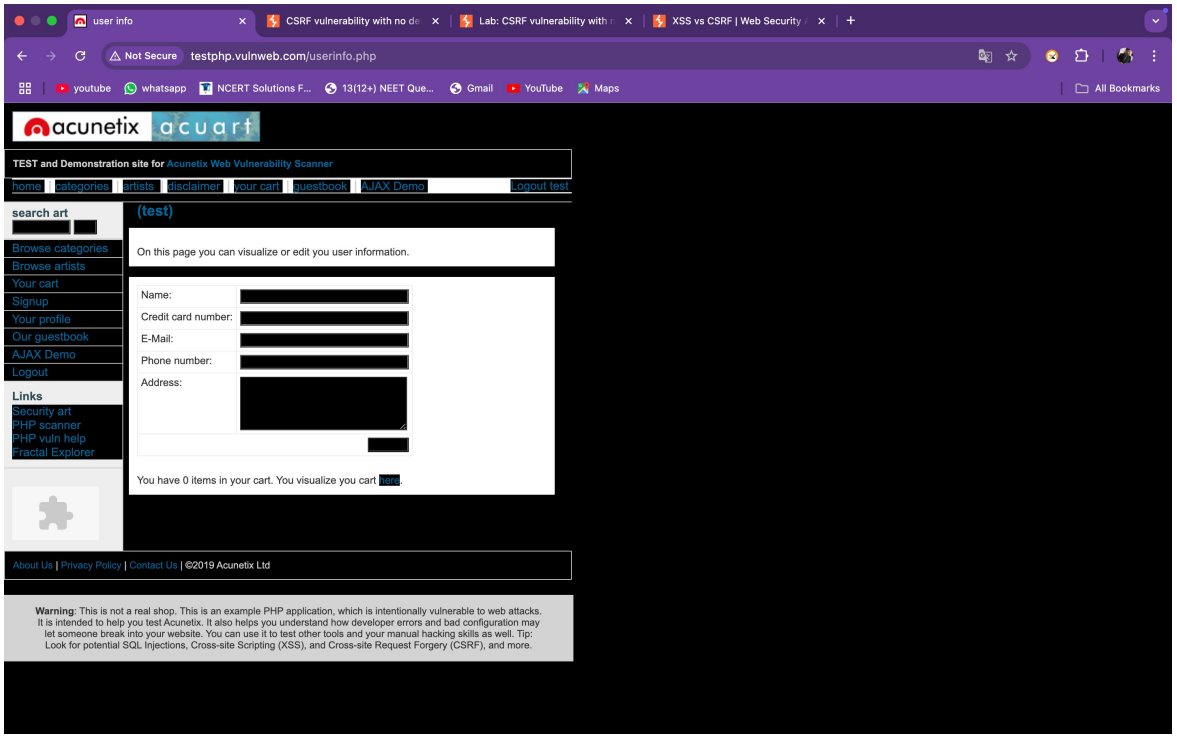
- Open the target site.
- Enter the Malicious code into the targeted field, here the Name text field is target site.
- Then hit update button.
- After that automatically the background colour changes to black, which is caused due to the Malicious code which i have injected.

Snapshots:

Injecting the malicious code to change the colour



Output of the Code



iii) DOM-Based XSS

Description: DOM-Based XSS occurs when JavaScript modifies the Document Object Model (DOM) without proper validation, causing malicious scripts to execute in the browser.

Example: A vulnerable webpage contains the following JavaScript:

```
var search = window.location.hash.substring(1);
document.write("<h1>" + search + "</h1>");
```

If a user visits:

http://victim.com/#<script>alert('XSS')</script>
The script is injected and executed on the page.

Impact of XSS Attacks

XSS attacks can have severe consequences, including:

- **Session Hijacking:** Attackers steal session cookies, leading to unauthorized access to user accounts.
- **Phishing Attacks:** Attackers create fake login forms to steal credentials.
- **Defacement:** Hackers modify the appearance of websites to spread misinformation.
- **Malware Distribution:** Attackers inject malicious code to distribute malware.

Common Remedies against CSRF:

Input Validation & Sanitization:
- Sanitize user input before storing or displaying it.
- Remove or escape characters like `<script>`, `<iframe>`, `'`, `"`, `&`, etc.

Use Content Security Policy (CSP)
- CSP restricts script execution to only trusted sources.
- Add this header in your server configuration:

Use HTTP-Only & Secure Cookies
- Prevent stealing session cookies with JavaScript (`document.cookie`).
- Set cookies with `HttpOnly` and `Secure` flags:

Implement Web Application Firewalls (WAF)
- Use security tools like ModSecurity to block XSS payloads.
- Example WAF rule to block `<script>` injection:

CSRF Attack:

Cross-site request forgery (also known as CSRF) is a web security vulnerability that allows an attacker to induce users to perform actions that they do not intend to perform. It allows an attacker to partly circumvent the same origin policy, which is designed to prevent different websites from interfering with each other.

Impact of CSRF Attacks:

- In a successful CSRF attack, the attacker causes the victim user to carry out an action unintentionally.
- For example, this might be to change the email address on their account, to change their password, or to make a funds transfer.
- Depending on the nature of the action, the attacker might be able to gain full control over the user's account.
- If the compromised user has a privileged role within the application, then the attacker might be able to take full control of all the application's data and functionality.

Common Remedies against CSRF:

- **CSRF tokens** - A CSRF token is a unique, secret, and unpredictable value that is generated by the server-side application and shared with the client. When attempting to perform a sensitive action, such as submitting a form, the client must include the correct CSRF token in the request. This makes it very difficult for an attacker to construct a valid request on behalf of the victim.
- **SameSite cookies** - SameSite is a browser security mechanism that determines when a website's cookies are included in requests originating from other websites. As requests to perform sensitive actions typically require an authenticated session cookie, the appropriate SameSite restrictions may prevent an attacker from triggering these actions cross-site. Since 2021, Chrome enforces Lax SameSite restrictions by default. As this is the proposed standard, we expect other major browsers to adopt this behavior in future.
- **Referer-based validation** - Some applications make use of the HTTP Referer header to attempt to defend against CSRF attacks, normally by verifying that the request originated from the application's own domain. This is generally less effective than CSRF token validation.

Steps To check CSRF Attack:

i) open the target website and login.

ii) Then copy the form code from the victims page.

iii) Host one new website where we insert the malicious code of form, which we have copied from victims page.

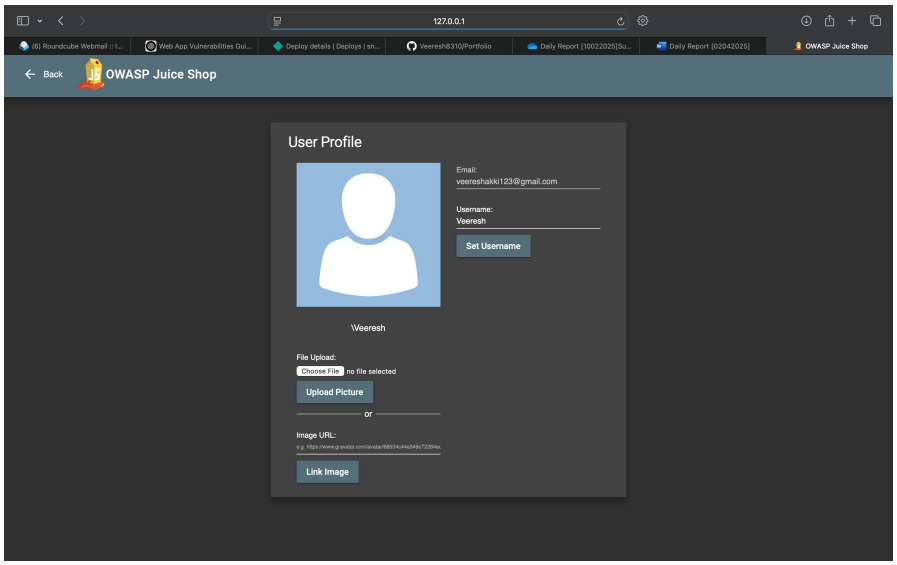
iv) Then send the link of the hosted website to victim and made him to click on it.

v) Once the victim clicks on it, automatically the username get changed.

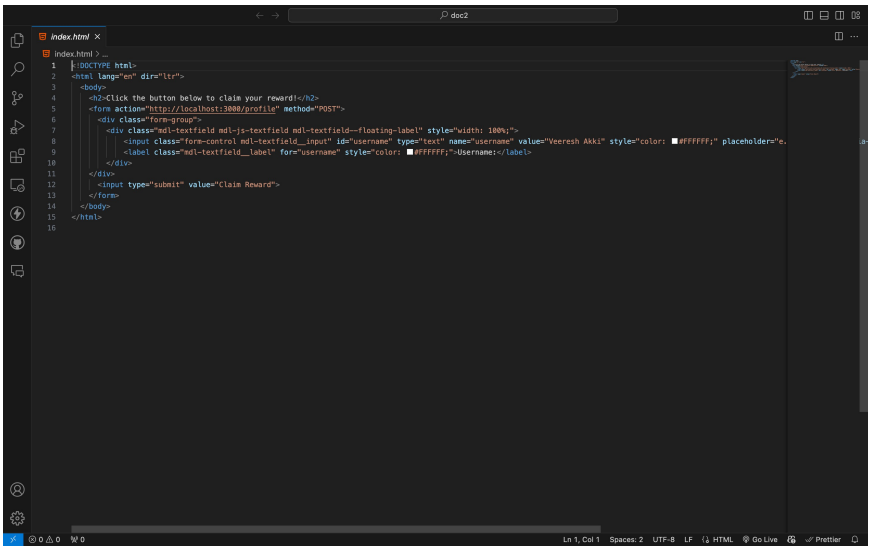
vi) Without victim consent the username got changed.

Snapshots:

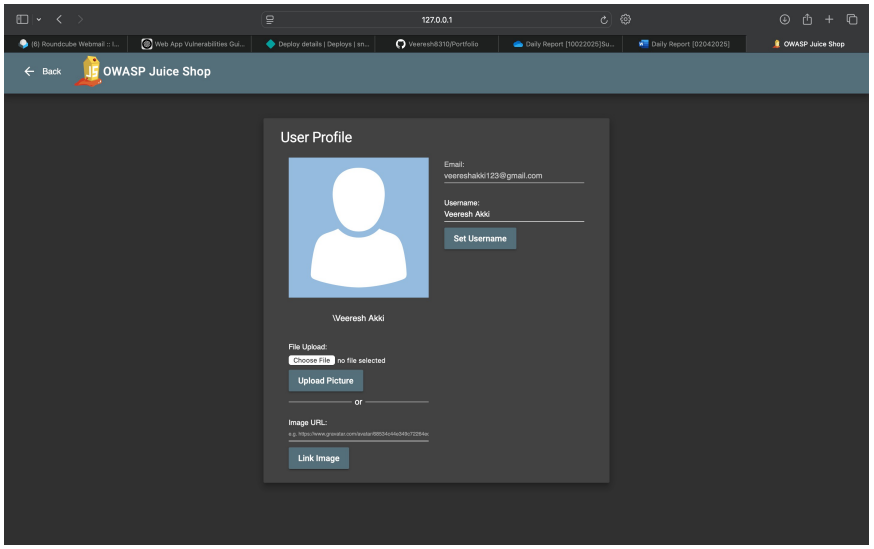
Before Clicking on the link the username of victim’s page.



The malicious code hosting website.



The victim's page username after clicking on the website link.



Broken Access Control:

Broken access control occurs when users can access resources or perform actions beyond their intended permissions due to improper access restrictions.

Prevention:

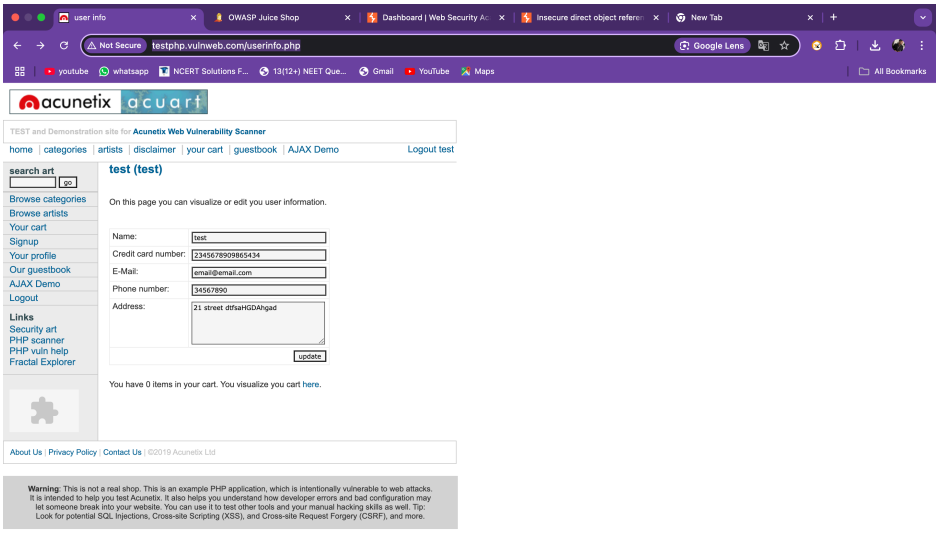
- Implement **Role-Based Access Control (RBAC)**
- Enforce **server-side access controls**
- Use **session management** properly

Example Scenario steps:

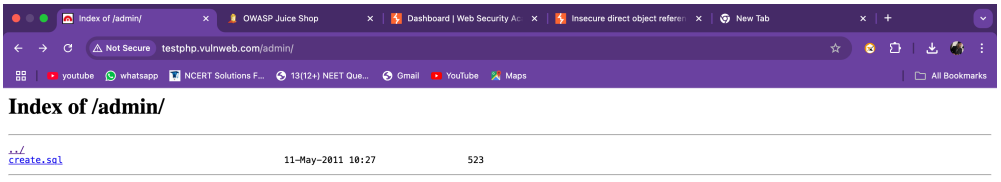
- Login in to a target website as Normal user (http://testphp.vulnweb.com/userinfo.php) .
- In the URL change the path to (http://testphp.vulnweb.com/admin/).
- A normal user is not allowed to access an admin dashboard.
- However, if they manually enter http://testphp.vulnweb.com/admin/ in the URL and gain access, it indicates broken access control.

Snapshots:

This is the normal Userinfo page.



This is the admin Page of website.



Hijack a Session (Session Hijacking)

Definition:
Session hijacking occurs when an attacker steals a user's session token and gains unauthorized access to their account.

Prevention:

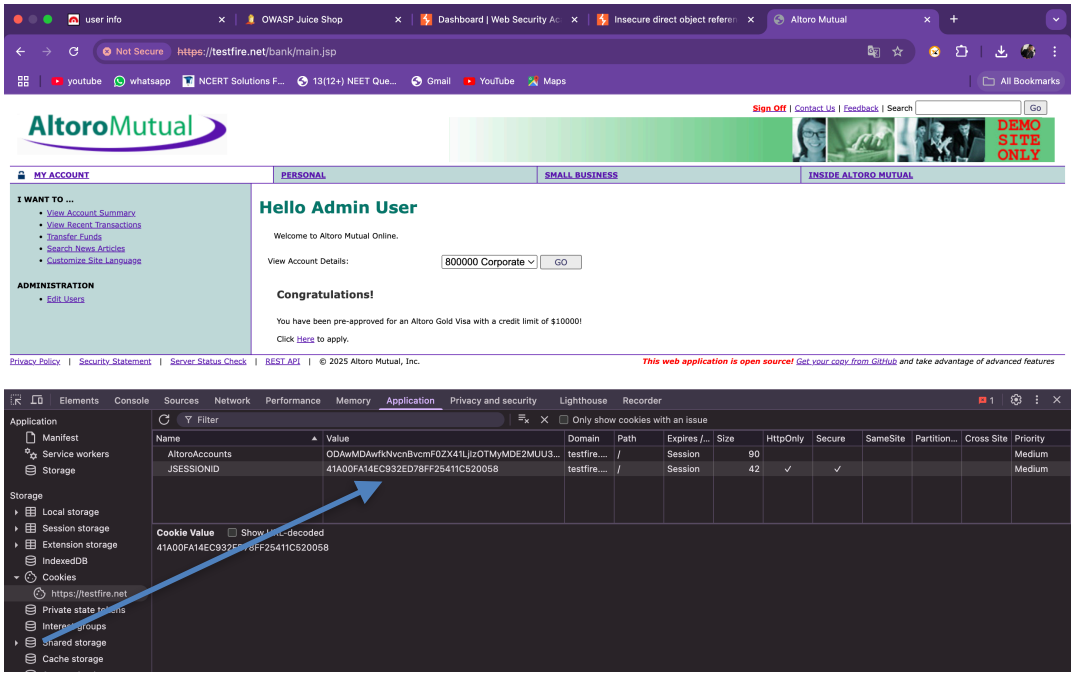
- Use **secure cookies** (`HttpOnly, Secure, SameSite`).
- Implement **session expiration and regeneration**.
- Use **multi-factor authentication (MFA)**.

Example Scenario steps :

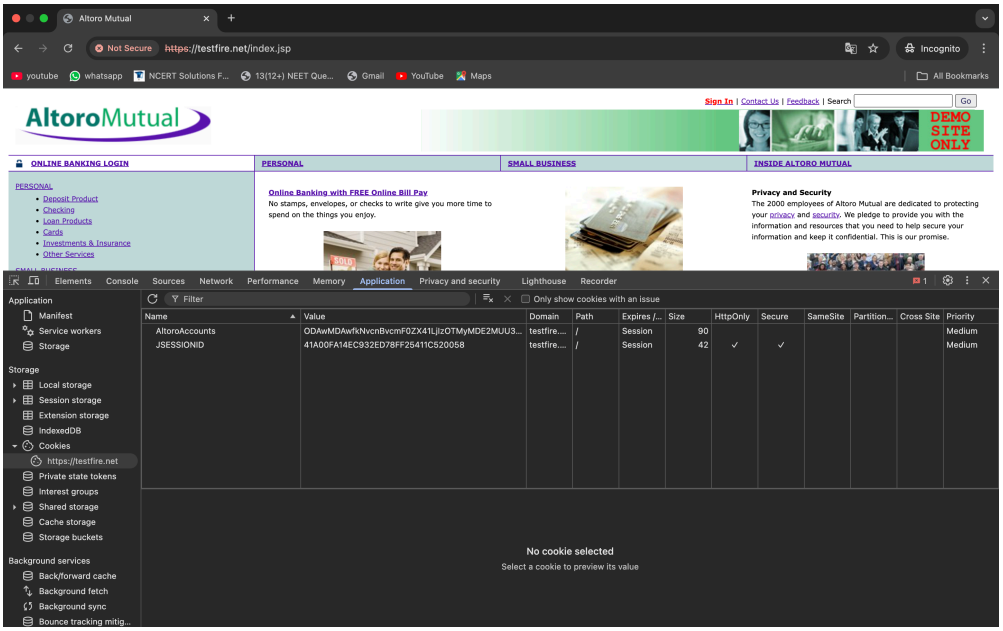
1. A user logs into a target website, with the username and password.
2. An attacker intercepts the request copy the session ID or copied the session id from the Developer tools —> Application.
3. The attacker reuses the session token in incognito window to access the victim account.

Snapshots:

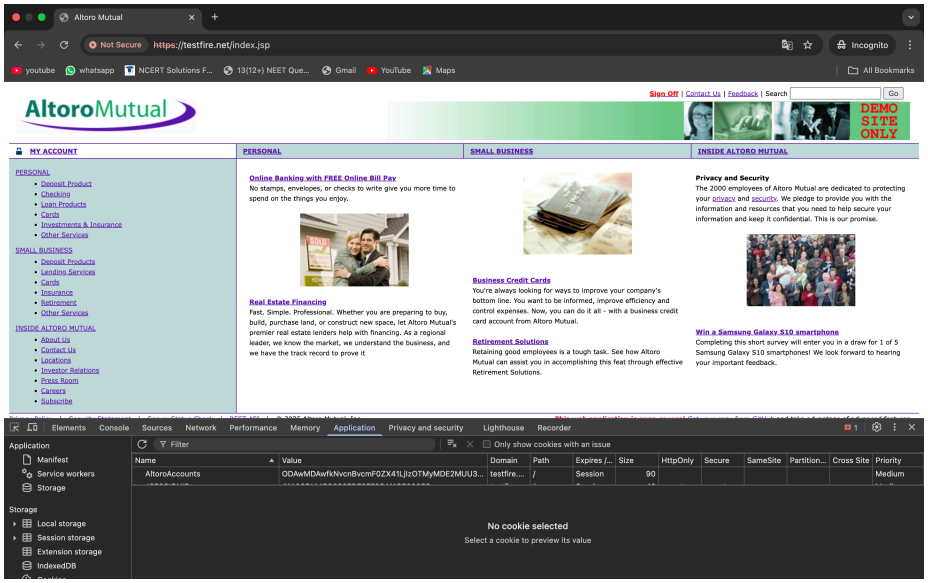
In this Browser victim login in using the username and password.



Here the attacker uses the session id of victims account and trying to access his account



Here we can see that Attacker successfully logged in to victims account using session id.



Insecure direct object references (IDOR)

Insecure direct object references (IDOR) are a type of access control vulnerability that arises when an application uses user-supplied input to access objects directly. The term IDOR was popularized by its appearance in the OWASP 2007 Top Ten. IDOR vulnerabilities are most commonly associated with horizontal privilege escalation, but they can also arise in relation to vertical privilege escalation.

IDOR vulnerability with direct reference to static files:

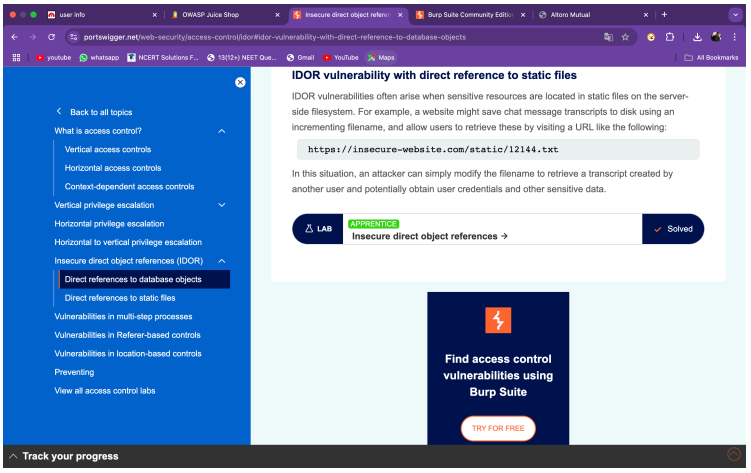
IDOR vulnerabilities often arise when sensitive resources are located in static files on the server-side filesystem. For example, a website might save chat message transcripts to disk using an incrementing filename, and allow users to retrieve these by visiting a URL like the following:

https://insecure-website.com/static/12144.txt

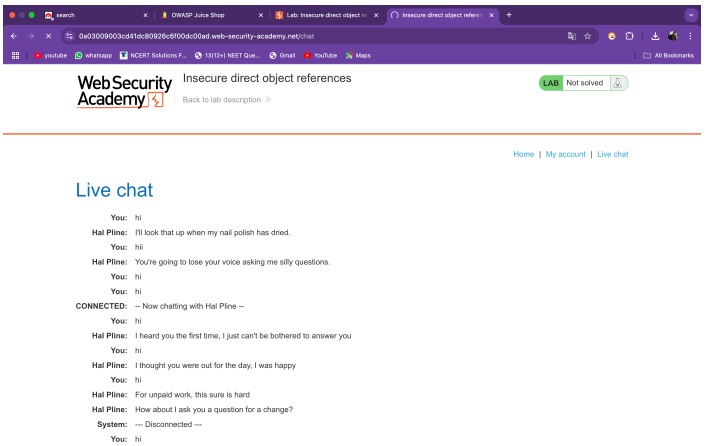
In this situation, an attacker can simply modify the filename to retrieve a transcript created by another user and potentially obtain user credentials and other sensitive data.

Example Scenario steps:

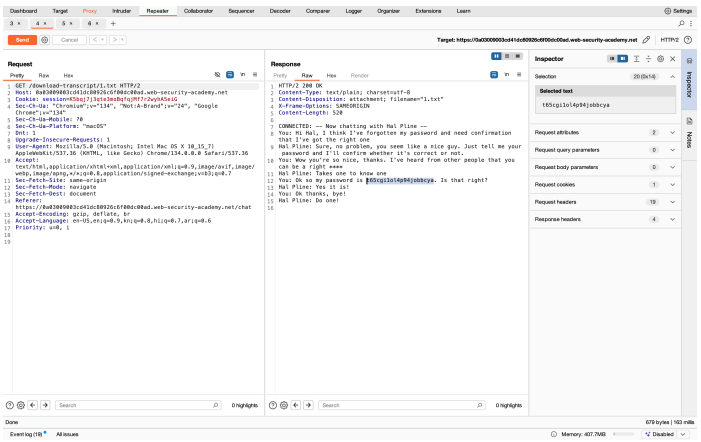
- Login in to a Portswigger learning path and access the lab of IDOR.
- In the Lab We need to access the password for the username **Carlos.**

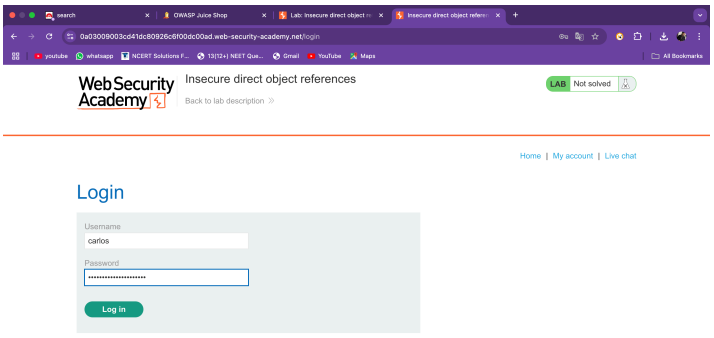


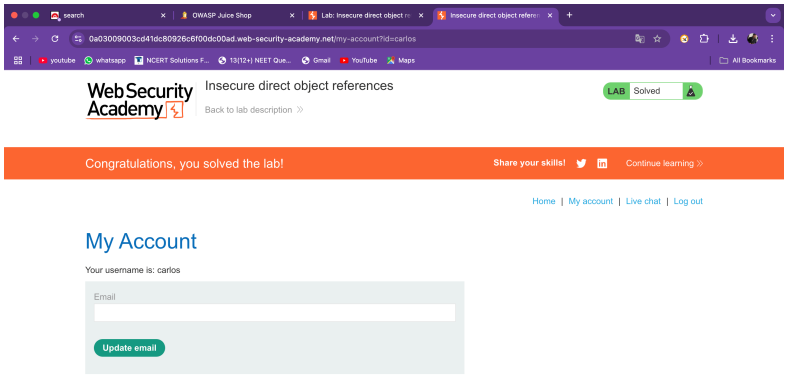
- Using Live chat, we need to send the chat with bot chat assistant.



- Then intercept the request of the sent chat and send it to repeater.
- There change the text file name to 1.txt.
- It automatically gives the password.
- Using password we can easily log in to Carlos account.







Prevention:

- Implement **access control checks on the backend**.
- Use **UUIDs instead of incremental IDs**.
- Ensure **authorization verification** before serving data