

Title:

Cyber Security Task 1

Web Application Security Assessment – OWASP Juice Shop – Code – CS_01

Author: Rahul Choudhary

Role: Cybersecurity Learner / SOC Analyst Aspirant

Email: Choudharyrahul8844@gmail.com

Date: December 2025

Lab Source: OWASP Juice Shop Room – TryHackMe

This report documents a hands-on web application security assessment of the OWASP Juice Shop application, performed using the TryHackMe lab environment.

1. Introduction

The **OWASP Juice Shop** is a deliberately insecure web application maintained by OWASP. It contains vulnerabilities from the OWASP Top 10 and other real-world flaws, and is widely used for training, CTFs, and tool testing.[OWASP+1](#)

This project simulates a **junior penetration testing / security internship task**:

- Target: OWASP Juice Shop
 - Objective: Identify and document **5 real vulnerabilities**
 - Focus: OWASP Top 10 categories:
 - Injection (SQL Injection)
 - Broken Authentication
 - Sensitive Data Exposure
 - Broken Access Control
 - Cross-Site Scripting (XSS)
-

2. Scope & Objectives

In-Scope:

- Frontend web UI of OWASP Juice Shop (TryHackMe room instance)
- HTTP/S requests and responses
- Authenticated and unauthenticated areas provided in the lab

Objectives:

1. Detect and exploit at least **five** web application vulnerabilities:
 - o SQL Injection
 - o Cross-Site Scripting (XSS)
 - o Broken Authentication
 - o Broken Access Control
 - o Sensitive Data Exposure
 2. Map each finding to the **OWASP Top 10**.
 3. Rate the **risk** and propose **remediation steps**.
 4. Produce a **professional report** suitable for internship applications.
-

3. Methodology & Tools

Testing was conducted using a **black-box approach** following OWASP-style testing methodology:

1. **Information Gathering**
 - o Manual browsing and feature discovery
 - o Identification of login, search, product, and admin-related functionality
2. **Vulnerability Discovery**
 - o Manual input fuzzing and parameter tampering
 - o Interception and modification of HTTP requests
3. **Exploitation & Proof of Concept**
 - o Controlled exploitation of confirmed vulnerabilities
 - o Collection of screenshots and HTTP traces
4. **Analysis & Reporting**
 - o Mapping to OWASP Top 10
 - o Risk rating (Low/Medium/High)
 - o Suggested mitigations

Tools Used:

- Web Browser (Firefox/Chrome DevTools)
 - **Burp Suite Community Edition** – intercepting and modifying HTTP requests
 - TryHackMe Platform – hosted OWASP Juice Shop lab environment [TryHackMe](#)
-

4. Summary of Findings

#	Vulnerability	OWASP Category	Risk	Status
---	---------------	----------------	------	--------

#	Vulnerability	OWASP Category	Risk	Status
1	SQL Injection in Login / Data Retrieval	A03: Injection	High	Confirmed PoC
2	Cross-Site Scripting (XSS)	A05: Security Misconfiguration / XSS	Medium/High	Confirmed PoC
3	Broken Authentication	A07: Identification & Auth Failures	High	Confirmed PoC
4	Broken Access Control	A01: Broken Access Control	High	Confirmed PoC
5	Sensitive Data Exposure	A02: Cryptographic Failures / Data Leak	Medium/High	Confirmed PoC

5. Detailed Findings

5.1 SQL Injection in Login / Product Functionality

- **Vulnerability ID:** JSHOP-01
- **Category:** SQL Injection (OWASP A03: Injection)
- **Risk Rating:** High

Description

The application fails to properly sanitize user input in SQL queries. By injecting crafted characters into the login or search fields, it is possible to bypass authentication or retrieve unintended data from the database.

Affected Functionality / Endpoint

- Login page: `/rest/user/login` (or whatever URL you saw in Burp)
- (Optional) Any product search or filter parameter that was injectable

Steps to Reproduce

1. Navigate to the **Login** page.
2. Intercept the request using **Burp**.
3. In the *email/username* field, enter a classic test payload, e.g.:
 - o `' OR 1=1-- OR 1=1#`
4. Provide any value for the password and submit the form.
5. Observe that the login succeeds without valid credentials, indicating SQL injection.

Screenshot Here: from **Sql Injection** (request/response + application screenshot).

Screenshot of Burp Suite showing network traffic and a request payload.

Network Tab:

Time	Type	Direction	Method	URL	Status code	Length
07:19:39 10 Dec... HTTP	Request	→ Request	GET	http://10.48.128.196/		
07:25:05 10 Dec... HTTP	Request	→ Request	GET	http://10.48.128.196/rest/admin/application-configuration		
07:25:23 10 Dec... HTTP	Request	→ Request	GET	http://10.48.128.196/rest/user/whoami		
07:25:40 10 Dec... HTTP	Request	→ Request	GET	http://10.48.128.196/rest/user/whoami		
07:25:49 10 Dec... HTTP	Request	→ Request	POST	http://10.48.128.196/rest/user/login		
07:26:10 10 Dec... HTTP	Request	→ Request	GET	http://10.48.128.196/rest/user/whoami		

Request Tab:

```

1 GET /rest/user/whoami HTTP/1.1
2 Host: 10.48.128.196
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0
4 Accept: application/json, text/plain, */*
5 Accept-Language: en-US;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Referer: http://10.48.128.196/

```

Inspector Tab:

- Request attributes: 2
- Request query parameters: 0
- Request body parameters: 0
- Request cookies: 2
- Request headers: 9

Request Details:

POST http://10.48.128.196/rest/user/login

```

Origin: http://10.48.128.196
Connection: keep-alive
Referer: http://10.48.128.196/
Cookie: language=en; cookieconsent_status=dissmiss
Priority: u=0

{
  "email": "bender@juice-sh.op'--",
  "password": "jjk"
}

```

OWASP Juice Shop Application:

You successfully solved a challenge: Login Admin (Log in with the administrator's user account.)

690fa3247a99d651e0b26f947baf0b79b4f404a9 Copied!

Your Basket

Total Price: 0€

Checkout

You will gain 0 Bonus Points from this order!

Account Your Basket 0 EN

Order History

Orders & Payment

Privacy & Security

Logout

Impact

- Attackers can bypass authentication.
- Potential access to other users' data.
- With more advanced payloads, they may read or modify database contents.

Likelihood

- Easy to exploit with simple payloads and a proxy tool.

Risk: High

Recommendations

- Use **parameterized queries / prepared statements** rather than string concatenation.
 - Implement server-side input validation and output encoding.
 - Enforce **least privilege** on database accounts.
 - Add **centralized error handling** to avoid leaking SQL error details.
-

5.2 Cross-Site Scripting (XSS)

- **Vulnerability ID:** JSHOP-02
- **Category:** Cross-Site Scripting (OWASP A03/A05 depending on Top 10 version)
- **Risk Rating:** Medium / High

Description

User-controlled input is reflected or stored in the page without proper output encoding. This allows attackers to inject malicious JavaScript that executes in other users' browsers.

Affected Functionality / Endpoint

- Example: Product review / feedback field
- Example URL: /#/contact or /rest/feedback

Steps to Reproduce

1. Navigate to the input field (e.g., **review**, **comment**, or **feedback** form).
2. Submit a payload such as:

```
<script>alert('XSS')</script>
```

3. Browse to the page where that input is displayed.
4. Observe that the JavaScript executes (e.g., `alert()` pops up), confirming XSS.

Screenshot Here: from XXs (showing the alert, payload, or Burp request).

The screenshot shows a browser window for the OWASP Juice Shop application. At the top, there is an alert dialog box with the text "10.49.183.246" and "XSS" with an "OK" button. Below the alert, the main page displays a search results section with the message "No results found" and "Try adjusting your search to find what you're looking for." On the left side of the screen, the browser's developer tools are open, specifically the "Inspector" tab. The "Network" tab is selected, showing a list of request headers:

User-Agent	Mozilla/5.0 (X11; Ubuntu; ...)
Accept	application/json, text/pl...
Accept-Language	en-US,en;q=0.5
Accept-Encoding	gzip, deflate, br
Authorization	Bearer eyJ0eXAiOiJKV...
Connection	keep-alive
Referer	http://10.49.183.246/
Cookie	language=en; cookieco...
Priority	u=0

Below the headers, there are input fields for "Name:" containing "True-Client-IP" and "Value:" containing "<iframe src="javascript:alert('xss')">". At the bottom of the inspector panel are "Cancel" and "Add" buttons.

Impact

- Session hijacking (if cookies accessible by JavaScript).
- Credential theft via fake login prompts.
- Defacement or malicious redirects.
- Pivot to more advanced attacks (e.g., CSRF, key logging).

Recommendations

- Apply **output encoding** based on context (HTML, attributes, JS, etc.).
- Use frameworks / templating engines that auto-escape by default.
- Implement **Content Security Policy (CSP)** to limit script execution.
- Sanitize user input where rich text is not required.

5.3 Broken Authentication

- **Vulnerability ID:** JSHOP-03
- **Category:** Broken Authentication / Identification & Auth Failures
- **Risk Rating:** High

Description

The application implements flawed authentication logic that allows users to log in as other users or as an administrator without valid credentials.

Typical examples in Juice Shop:

- Default or predictable admin credentials
- Logic bypass using manipulated tokens or email addresses
- Password reset flows that can be abused

Steps to Reproduce

1. Navigate to the **Login** page.
2. Provide a username/email (e.g., Get By using Brup Suite by Running Script For Brup).
3. Use a weak or guessed password, or exploit logic from previous SQLi steps.
4. Observe successful login as a higher-privileged user.

□ **Screenshot Here:** from **broken authentication** showing the login PoC or resulting admin access.

The screenshot shows the Brup Suite interface. On the left, there's a terminal window displaying a POST request to the '/rest/user/login' endpoint. The request includes various headers (User-Agent, Accept, Accept-Language, Accept-Encoding, Content-Type) and a JSON payload with 'email' set to 'admin@juice-shop.com' and 'password' set to '55'. On the right, there's a 'Payloads' configuration window. It shows a list of payloads: 1111111, 1222333, 1212, 121212, and 123. A button labeled 'Remove all items from the list' is highlighted with a red box. Below the payload list, there's a section for 'Payload processing' with a note: 'You can define rules to perform various processing tasks on each payload before it is sent.'

```
POST /rest/user/login HTTP/1.1
Host: 10.48.128.196
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
Cache-Control: no-cache
Origin: http://10.48.128.196
Connection: keep-alive
Referer: http://10.48.128.196/
Cookie: language=en; cookieconsent_status=dismiss
Priority: u=0
(*"email": "admin@juice-shop.com", "password": "55")
```

Results Positions

▼ Intruder attack results filter: Showing all items

Request ^	Payload	Status code	Response received	Error	Timeout	Length	Comment
...	...	401	18		413	...	
111	access14	401	35		413		
112	account	401	27		413		
113	action	401	40		413		
114	admin	401	29		413		
115	admin1	401	24		413		
116	admin12	401	37		413		
117	admin13	200	45		1165		
118	adminadmin	401	40		413		
119	administrator	401	29		413		
120	adriana	401	26		413		
121	agosto	401	36		413		
122	agustin	401	59		413		
123	albert	401	56		413		
124	alberto	401	63		413		
125	aleandra	401	24		413		
126	alejandro	401	28		413		
127	alex	401	37		413		

You successfully solved a challenge: Password Strength (Log in with the administrator's user credentials without previously changing them or applying SQL Injection.) X

You successfully solved a challenge: Login Admin (Log in with the administrator's user account.) X

Impact

- Full account takeover of other users.
- Possible admin panel access.
- Ability to manipulate data, view orders, user details, etc.

Recommendations

- Enforce **strong password policy** and lockout / rate-limiting.
- Remove default or hardcoded credentials.
- Ensure authentication logic is **not bypassable** via crafted parameters.
- Use secure session management (HttpOnly, Secure cookies, short session lifetime).

5.4 Broken Access Control

- **Vulnerability ID:** JSHOP-04
- **Category:** Broken Access Control (OWASP A01)
- **Risk Rating:** High

Description

The application fails to enforce proper authorization checks on sensitive resources. Users can

access restricted data or admin functionality by directly browsing to hidden endpoints or modifying parameters.

Typical Juice Shop examples:

- Accessing `/#/administration` or admin-only APIs without proper checks
- Viewing or editing other users' details via predictable IDs

Steps to Reproduce

1. Browse the application and identify sensitive functionality (e.g., admin, user list, logs).
2. Manually change URLs or IDs in the address bar or intercepted requests.
3. Observe access to resources that should be restricted to admins or specific users.

□ **Screenshot Here:** from **Broken Access Control**(e.g., admin page, unauthorized data access).

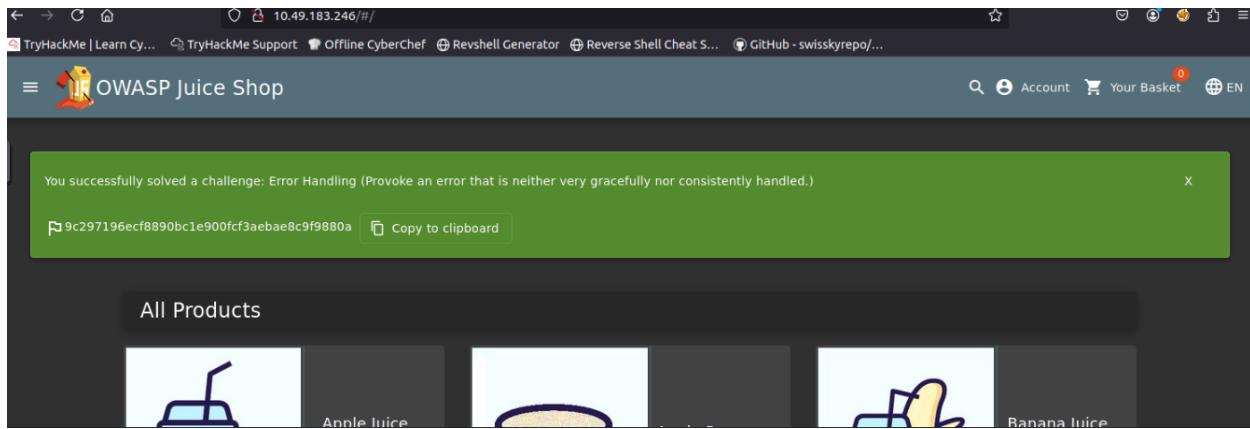
The screenshot shows a Firefox browser window with the URL `10.49.174.31/#/administration`. The page displays a success message: "You successfully solved a challenge: Admin Section (Access the administration section of the store.)" with a copy-to-clipboard button. A sidebar menu is open, showing options like "admin@juice-sh.op", "Orders & Payment", "Privacy & Security", and "Logout". Below the main content, a Network tab in the developer tools shows a table of network requests, and a Request tab shows the raw HTTP traffic for a "GET /rest/basket/1" request.

Time	Type	Direction	Method	URL
13:18:06 10 De...	HTTP	→ Request	GET	http://10.49.174.31/rest/products/search?q=
13:18:25 10 De...	HTTP	→ Request	GET	http://10.49.174.31/socket.io/?EIO=4&transport=polling&t=P18QRIV
13:18:28 10 De...	HTTP	→ Request	GET	http://10.49.174.31/rest/basket/1

Request

Pretty Raw Hex

```
GET /rest/basket/1 HTTP/1.1
Host: 10.49.174.31
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0
Accept: application/json, text/plain, /*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwibmF0YSI6eyJpZCI6MSwidXNlc3ShbWUiOiiIiLCJlbWFpbCI6ImFkbWluQGplaNWNlLXNoLm9wIiwiGfzc3dvcmQ1OiiIwMTkymDIZYTdiYnq3MzI1MDUxNmYnjlkZjEYjUwMCIsInJvbGLiOiJhZGlpbiIsImRlbHVAZVRva2VuIjoiIiwiZFzdExvZ2luSXAiOiiIiLCJwcw9maWxlSW1hZ2UiOiJhc3NldHMvchVibGLjL2ltYnldcy9lcGx
```



Impact

- Unauthorized access to other users' accounts or data.
- Exposure of internal configuration, logs, or user management panels.
- Step towards full compromise of the application.

Recommendations

- Enforce **server-side authorization** on every request.
- Use **role-based access control (RBAC)**.
- Avoid relying on hidden links or client-side checks.
- Implement **access control tests** as part of CI/CD.

5.5 Sensitive Data Exposure

- **Vulnerability ID:** JSHOP-05
- **Category:** Sensitive Data Exposure / Cryptographic Failures (OWASP A02)
- **Risk Rating:** Medium / High

Description

Sensitive information (e.g., user details, tokens, or technical data) is exposed in plaintext or in weakly protected form, either in responses, error messages, or logs.

Examples:

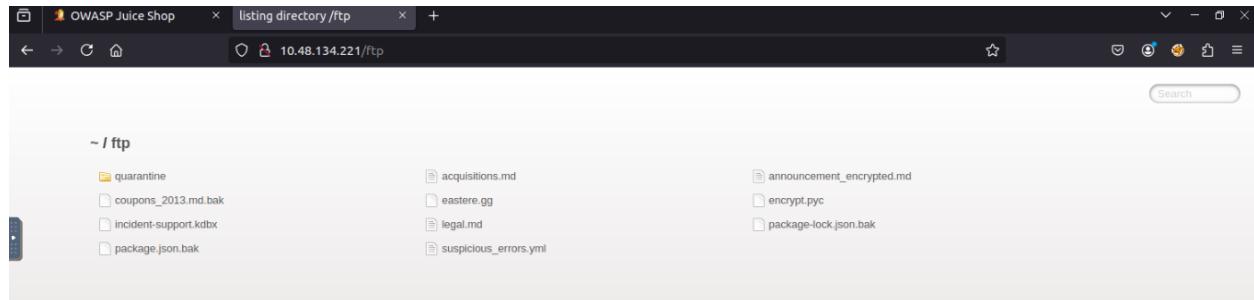
- Tokens or API keys visible in responses or browser storage
- Personal data returned in overly verbose APIs
- Debug messages showing stack traces or database info

Steps to Reproduce

1. Interact with the application and inspect responses in Burp or DevTools.

2. Identify any responses that contain sensitive fields which are not necessary for the user.
3. Confirm that data is in cleartext or weakly protected.

Screenshot Here:



The screenshot shows a browser window titled "OWASP Juice Shop" with the URL "10.48.134.221/ftp/acquisitions.md". The address bar also shows "10.48.134.221/ftp/acquisitions.md". The page displays the content of the "acquisitions.md" file:

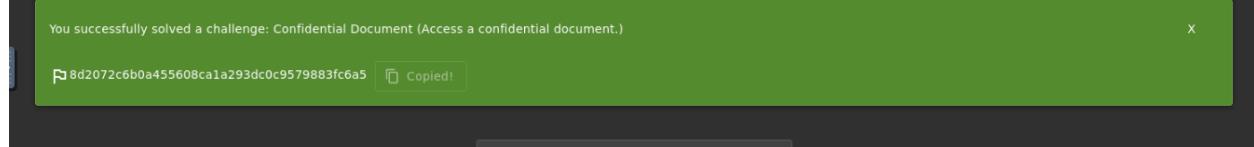
```
# Planned Acquisitions

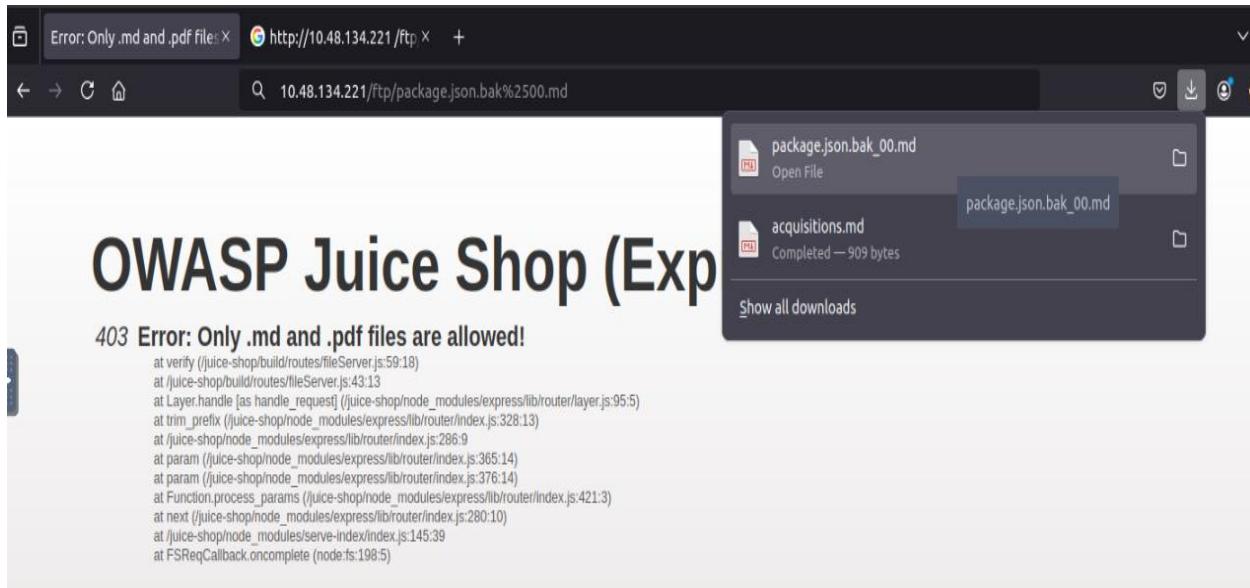
> This document is confidential! Do not distribute!

Our company plans to acquire several competitors within the next year.
This will have a significant stock market impact as we will elaborate in
detail in the following paragraph:

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy
eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam
voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet
clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit
amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam
nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat,
sed diam voluptua. At vero eos et accusam et justo duo dolores et ea
rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem
ipsum dolor sit amet.

Our shareholders will be excited. It's true. No fake news.
```





Impact

- Leakage of personal data or internal technical information.
- Facilitation of further attacks (credential stuffing, privilege escalation).

Recommendations

- Return **only necessary fields** to the client.
- Use **strong encryption** for sensitive data at rest and in transit (HTTPS).
- Disable verbose error messages in production.
- Review logs for sensitive data and redact where needed.

6. OWASP Top 10 Mapping Checklist

OWASP Top 10 Category	Observed in Juice Shop Lab?	Notes
A01 – Broken Access Control	<input type="checkbox"/> Yes	Direct access to admin/sensitive functionality
A02 – Cryptographic Failures / Sensitive Data Exposure	<input type="checkbox"/> Yes	Sensitive data visible or weakly protected
A03 – Injection (SQLi)	<input type="checkbox"/> Yes	Login bypass via SQL Injection
A04 – Insecure Design	<input type="checkbox"/> <input checked="" type="checkbox"/> Partially	Some challenges show poor design choices
A05 – Security Misconfiguration / XSS	<input type="checkbox"/> Yes	Reflected / stored XSS cases
A06 – Vulnerable & Outdated Components	<input type="checkbox"/> <input checked="" type="checkbox"/> Not evaluated	Out of scope for this small project

OWASP Top 10 Category	Observed in Juice Shop Lab?	Notes
Others (A07–A10)	<input type="checkbox"/> <input type="checkbox"/> Not fully tested	Future work

7. Conclusion & Next Steps

This assessment demonstrated **practical exploitation** of five core web application vulnerabilities in OWASP Juice Shop via the TryHackMe lab:

- **Injection attacks** that bypass authentication
- **XSS** that executes arbitrary JavaScript in the browser
- **Broken authentication & access control** leading to privilege misuse
- **Sensitive data exposure** that leaks internal information

Through this project, I strengthened skills in:

- Web application testing methodology
- Using intercepting proxies (Burp Suite)
- Mapping findings to **OWASP Top 10**
- Writing structured **security reports**