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:
    - SQL Injection
    - Cross-Site Scripting (XSS)
    - Broken Authentication
    - Broken Access Control
    - 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**
    - Manual browsing and feature discovery
    - Identification of login, search, product, and admin-related functionality
2. **Vulnerability Discovery**
    - Manual input fuzzing and parameter tampering
    - Interception and modification of HTTP requests
3. **Exploitation & Proof of Concept**
    - Controlled exploitation of confirmed vulnerabilities
    - Collection of screenshots and HTTP traces
4. **Analysis & Reporting**
    - Mapping to OWASP Top 10
    - Risk rating (Low/Medium/High)
    - 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.

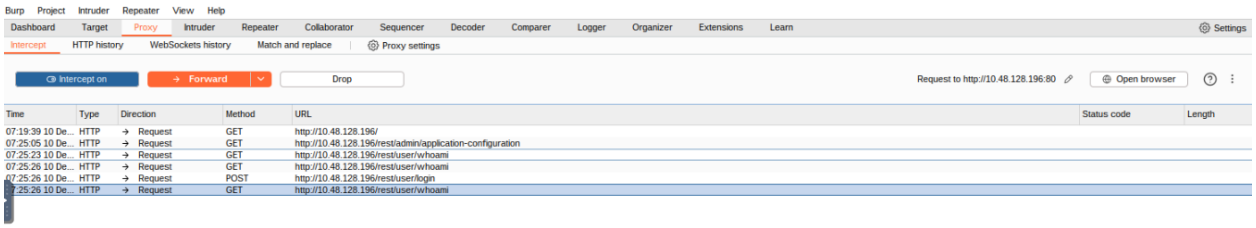
Affected Functionality / Endpoint

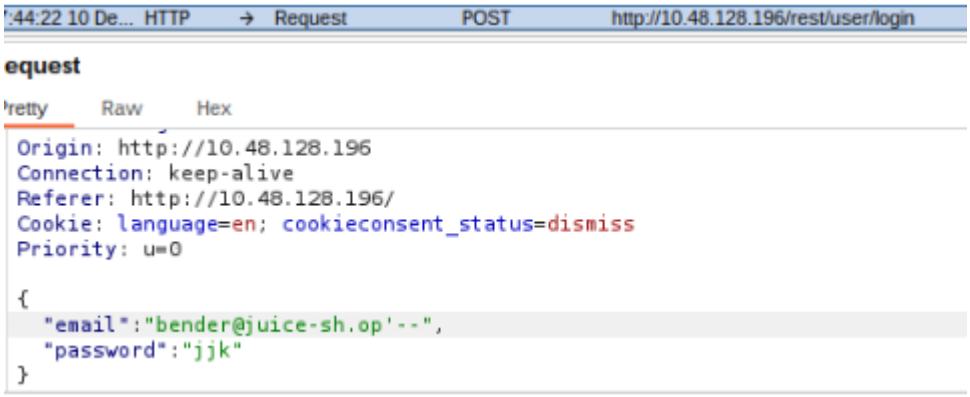
- Login page: `/rest/user/login` (or whatever URL you saw in Burp)
- (Optional) Any product search or filter parameter that was injectable

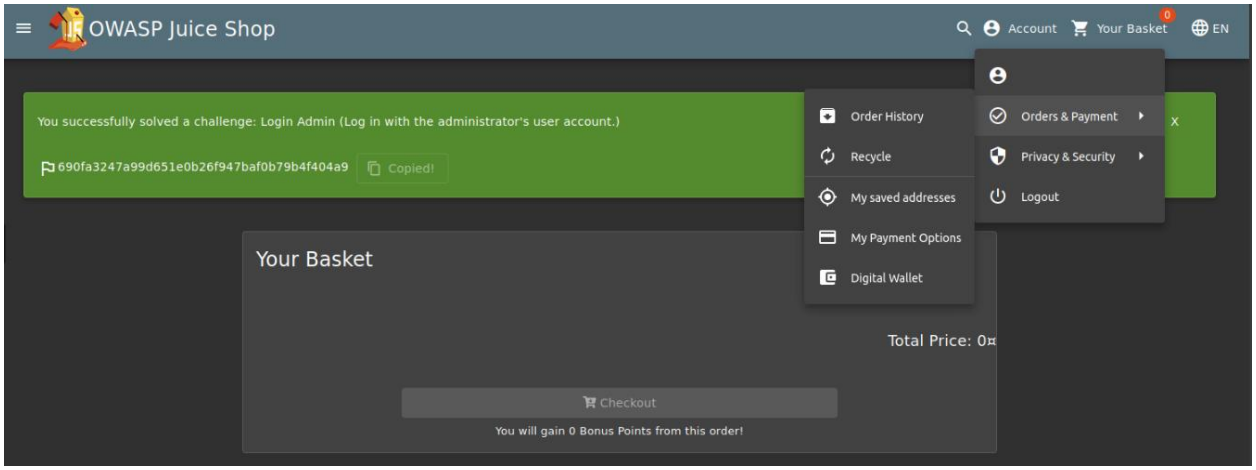
Steps to Reproduce

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.:
    - `' OR 1=1--` or `' 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).







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`

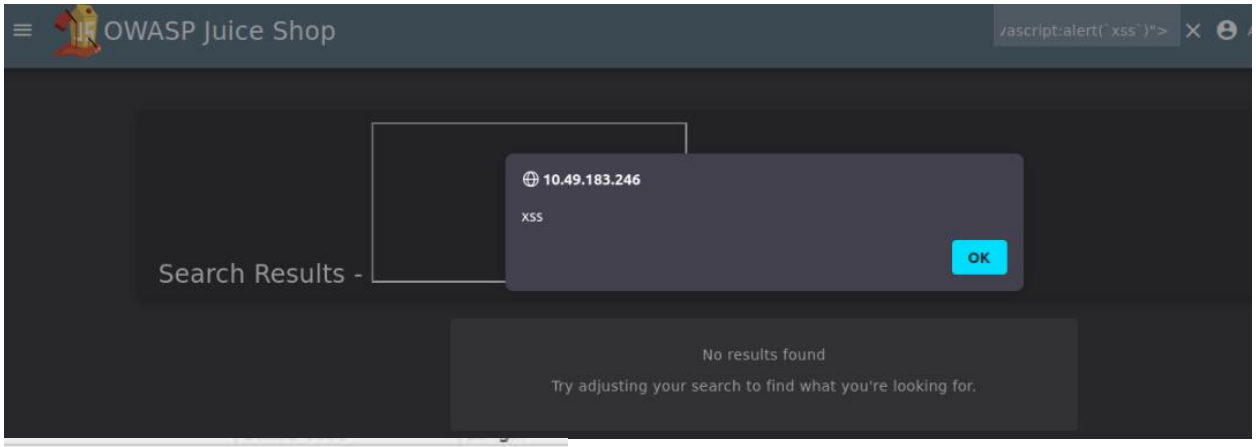
Steps to Reproduce

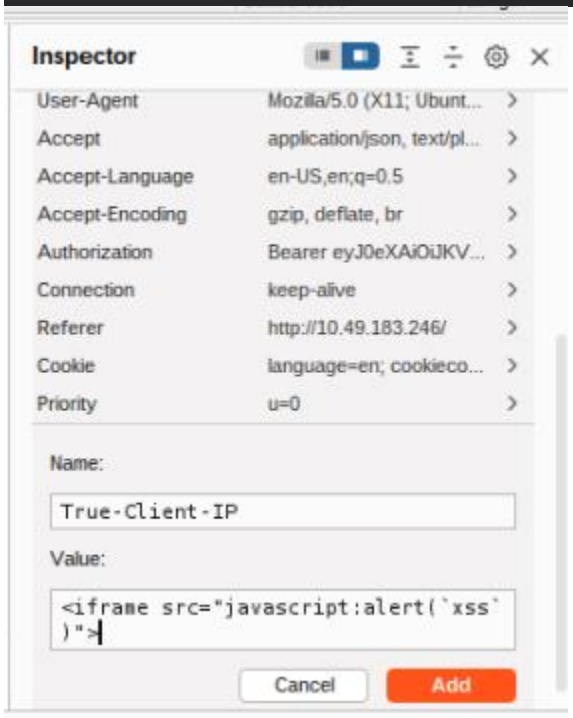
1. Navigate to the input field (e.g., **review**, **comment**, or **feedback** form).
2. Submit a payload such as:

```
<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).





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.

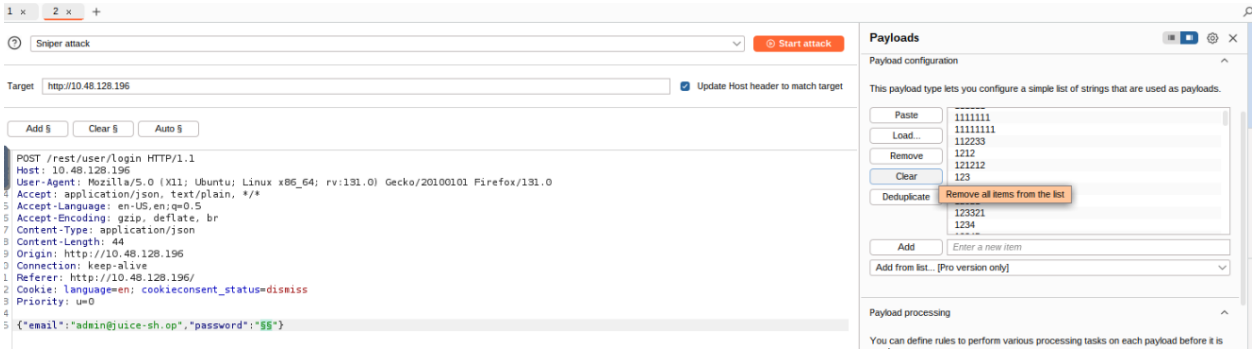
Typical examples in Juice Shop:

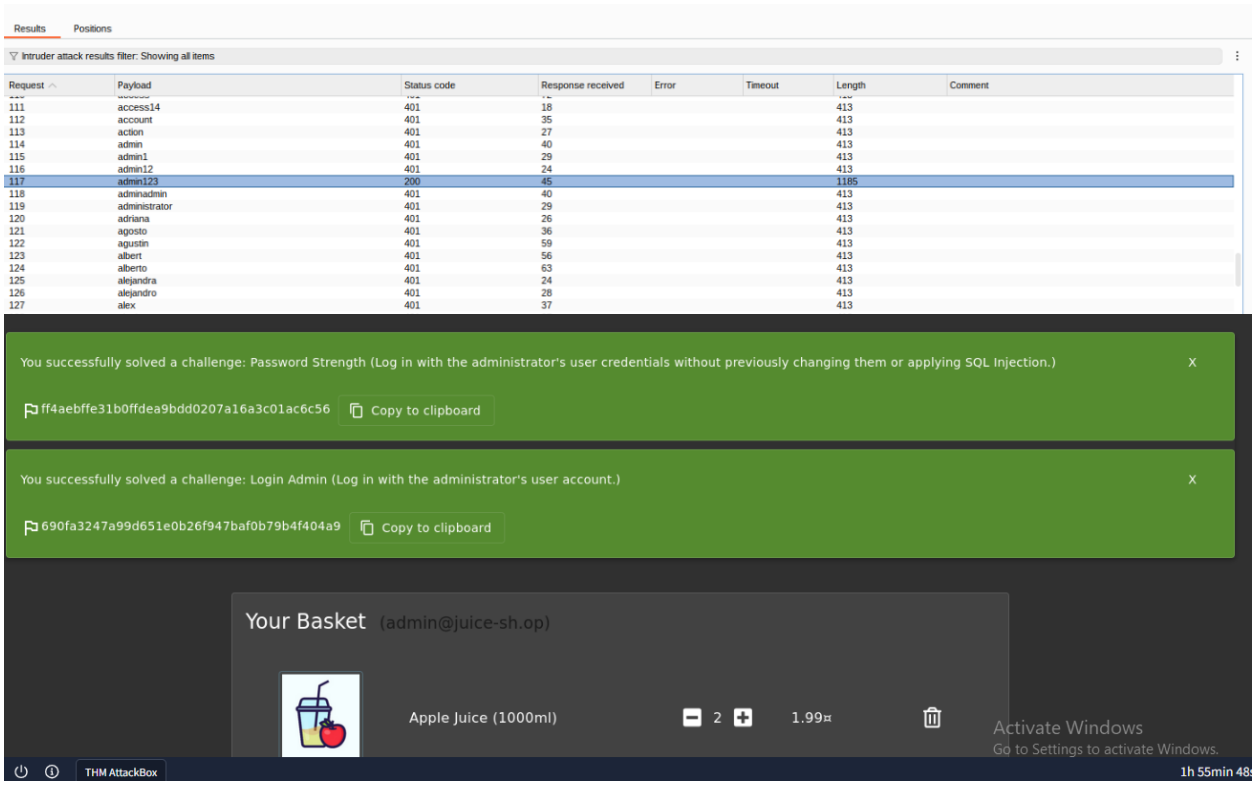
- Default or predictable admin credentials
- Logic bypass using manipulated tokens or email addresses
- Password reset flows that can be abused

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 authetication** showing the login PoC or resulting admin access.





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.

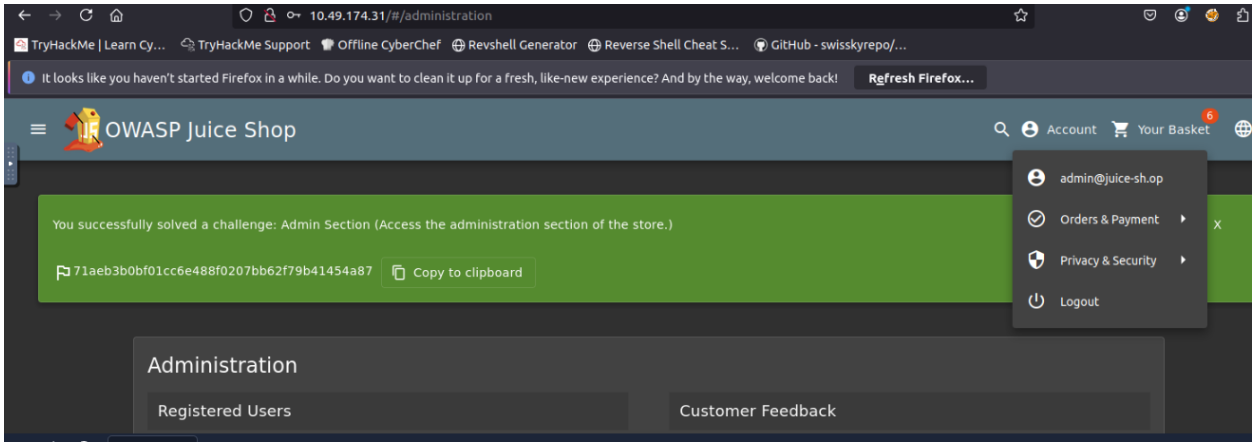
Typical Juice Shop examples:

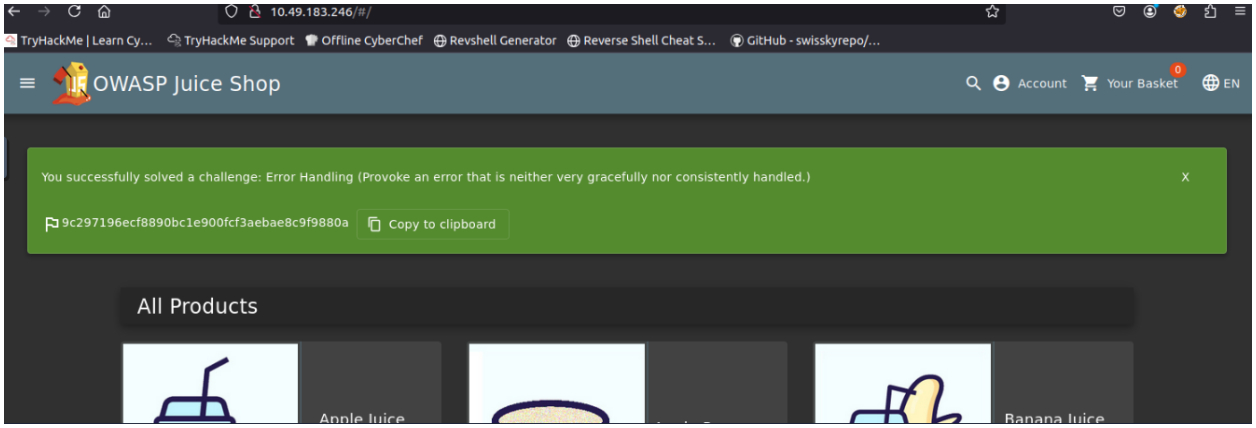
- Accessing `/#/administration` or admin-only APIs without proper checks
- Viewing or editing other users' details via predictable IDs

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).





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:

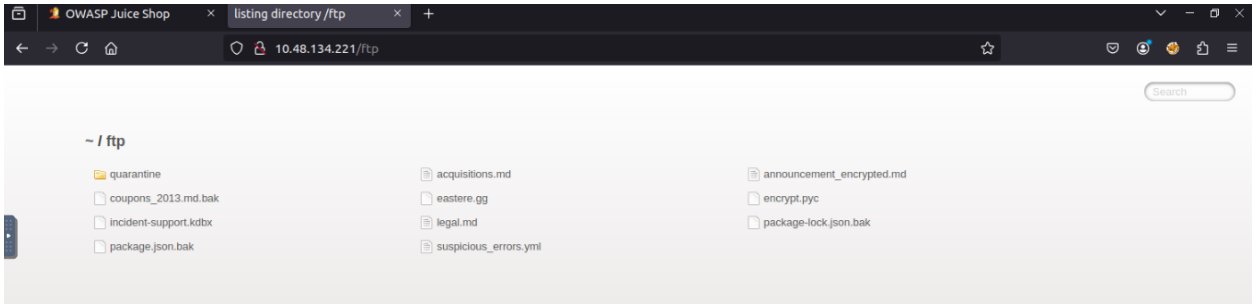
- Tokens or API keys visible in responses or browser storage
- Personal data returned in overly verbose APIs
- Debug messages showing stack traces or database info

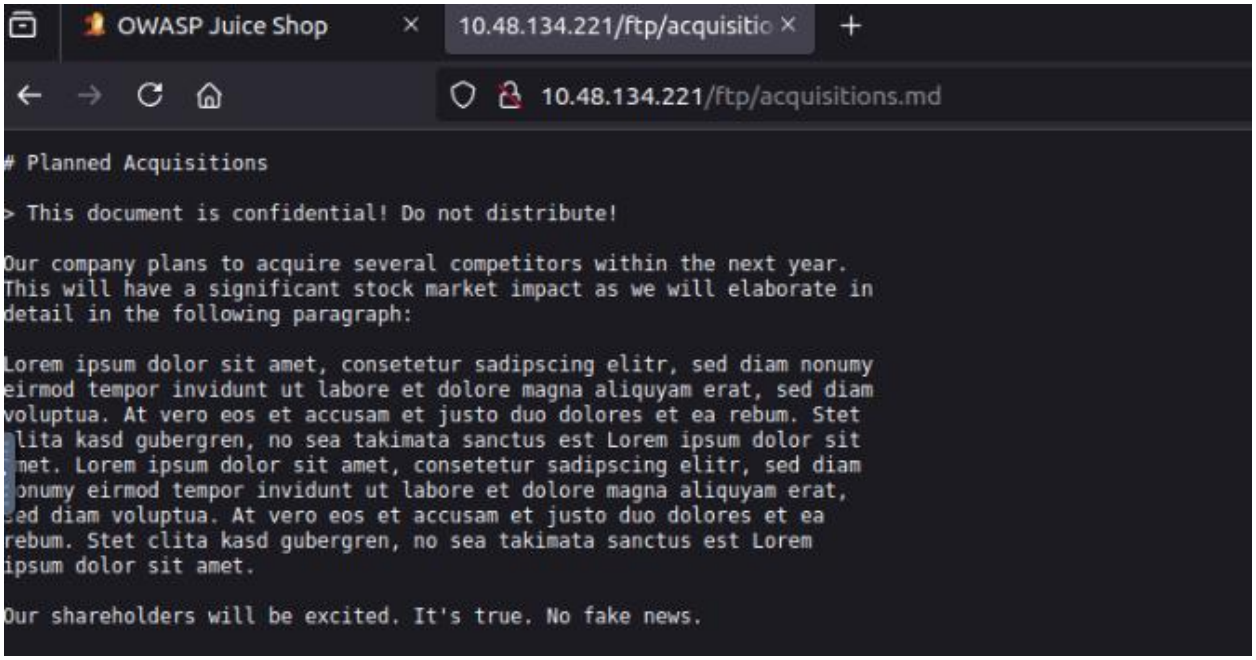
Steps to Reproduce

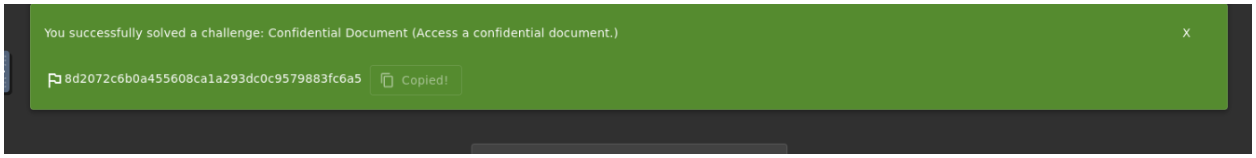
1. Interact with the application and inspect responses in Burp or DevTools.

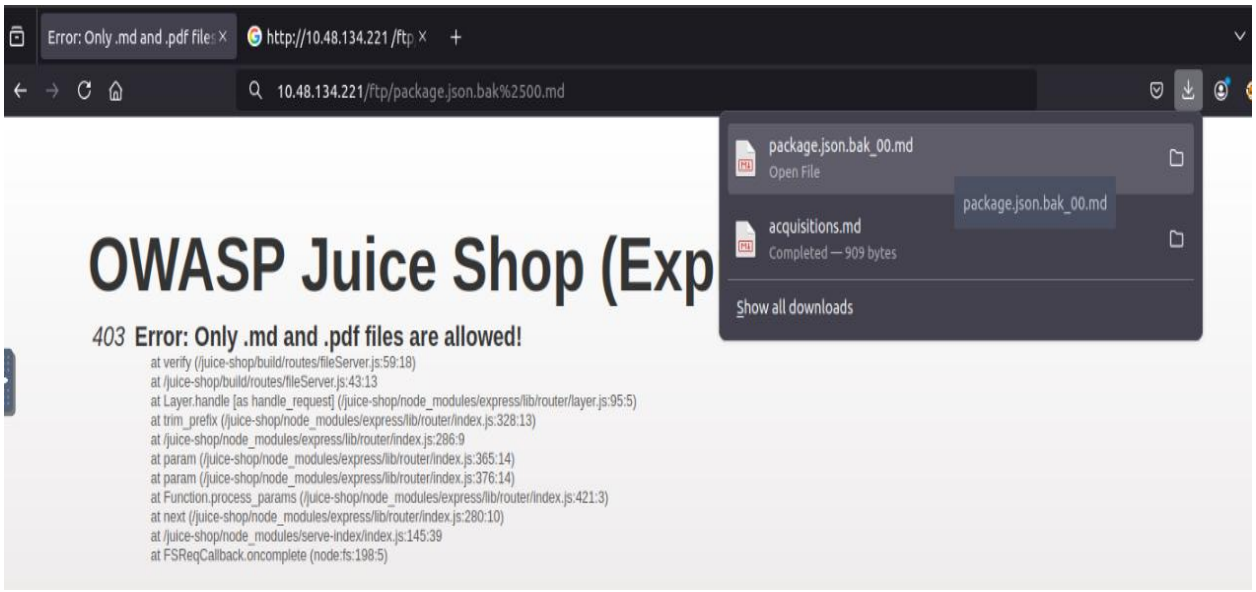
2. Identify any responses that contain sensitive fields which are not necessary for the user.
3. Confirm that data is in cleartext or weakly protected.

Screenshot Here:









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	□ Yes	Direct access to admin/sensitive functionality
A02 – Cryptographic Failures / Sensitive Data Exposure	□ Yes	Sensitive data visible or weakly protected
A03 – Injection (SQLi)	□ Yes	Login bypass via SQL Injection
A04 – Insecure Design	□□ Partially	Some challenges show poor design choices
A05 – Security Misconfiguration / XSS	□ Yes	Reflected / stored XSS cases
A06 – Vulnerable & Outdated Components	□□ Not evaluated	Out of scope for this small project

OWASP Top 10 Category	Observed in Juice Shop Lab?	Notes
Others (A07–A10)	□□ 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**