Basic Web Application VAPT Using OWASP Juice shop

Project Objective

The objective of this project is to perform a **Vulnerability Assessment and Penetration Test (VAPT)** on OWASP Juice Shop — a deliberately insecure web application. This hands-on activity helps to understand how common web vulnerabilities work, how to exploit them ethically, and how to suggest proper remediations to improve web application security.

Tools Used

Burp Suite Community Edition: Used to intercept, modify, and analyze HTTP requests/responses to identify vulnerabilities in the web application.

VULNERABILITIES

Vulnerability 1: SQL Injection

Objective:

Exploit a SQL injection vulnerability and discover the admin password using Burp Suite to complete the "Login Admin" challenge in OWASP Juice Shop.

Steps to Reproduce

Step 1: Attempted Normal Login

Navigated to the Account > Login page.

Attempted login using:

Email: admin

Password: admin

The login attempt failed.

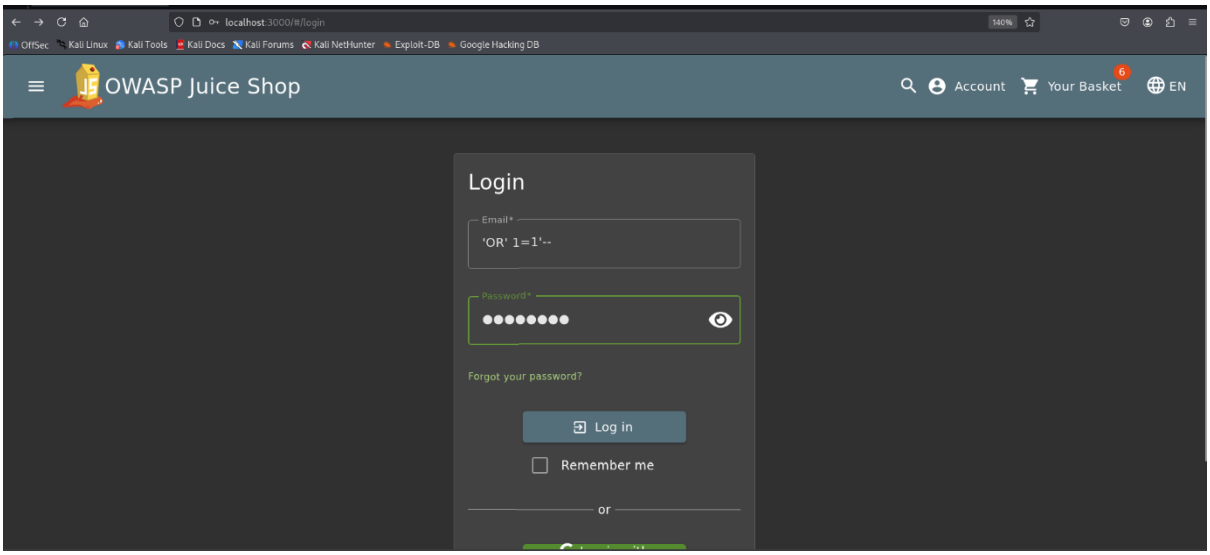
Step 2: SQL Injection to Bypass Authentication

In the login form:

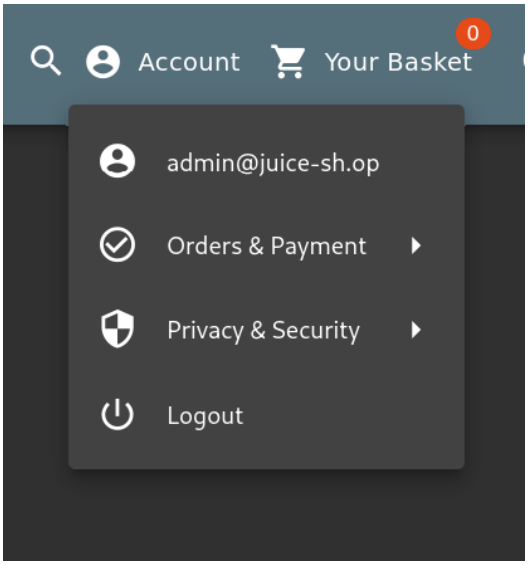
Entered the following SQL injection payload in the email field:

```
'OR' 1=1'--
```

Used a random string in the password field.



The application granted access, logging in as the admin user.



Step 3: Identified the Admin Email

After bypassing authentication, navigated to the account/profile section.

The admin email was displayed as:

admin@juice-sh.op

Step 4: Finding the Admin Password Using Burp Suite

1. Captured Login Request

Opened Burp Suite.

Enabled intercept and submitted a login form using:

Email: admin@juice-sh.op

Password: randomvalue

Captured the HTTP POST request.

2. Sent to Repeater

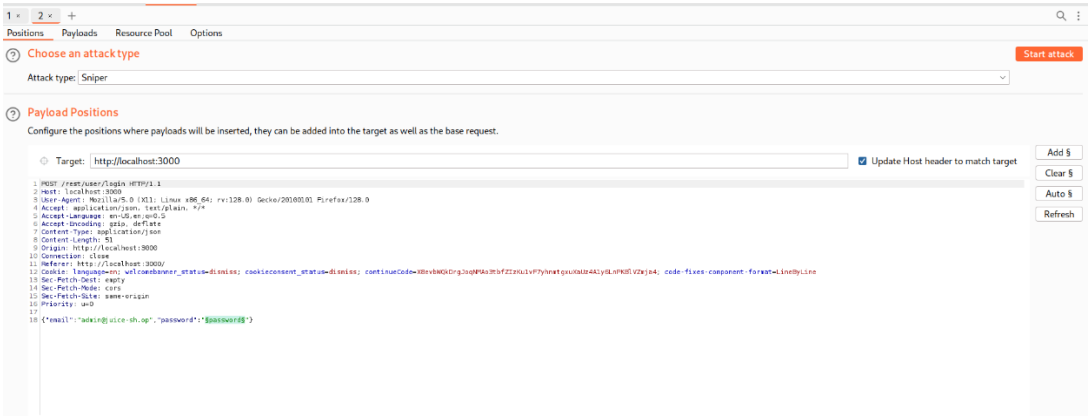
Sent the request to the Repeater tab.

Tested a few random passwords manually, but login attempts were unsuccessful.

3. Sent to Intruder

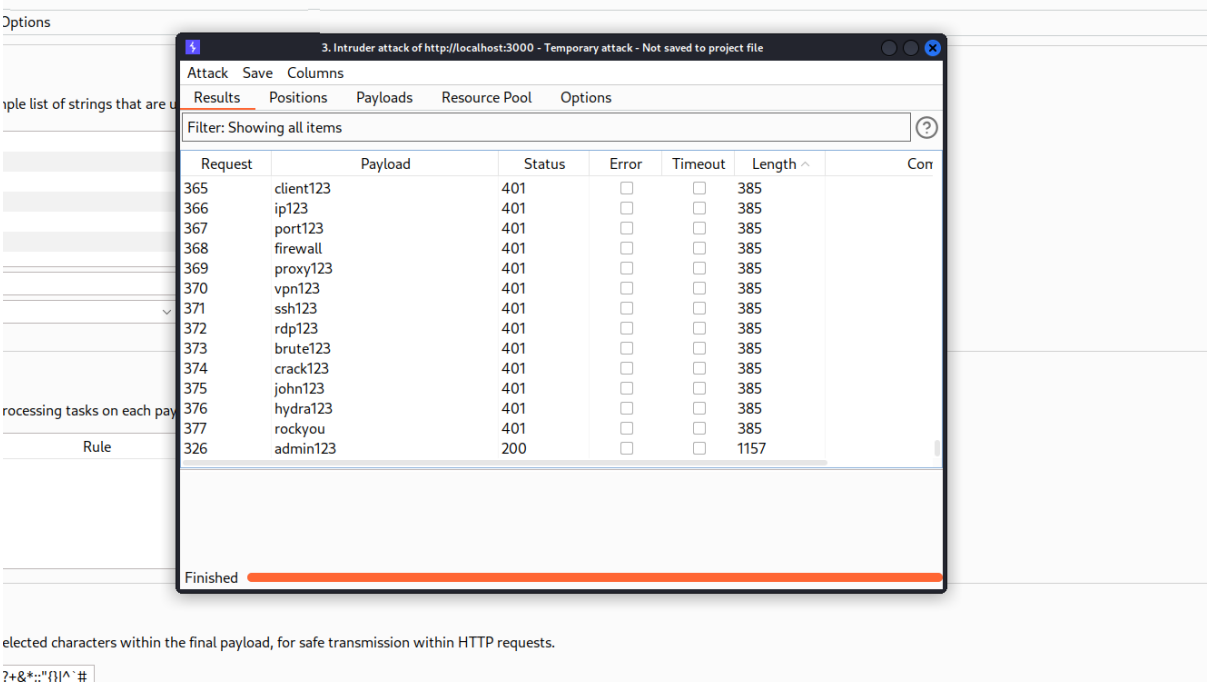
Sent the same request to the Intruder tab.

Configured the password field as the payload position.



Loaded a wordlist of 1000 common passwords.

Launched the attack.



4. Identified the Correct Password

During the attack, observed one request that returned a different response:

HTTP Status Code: 200 OK

Response length was different from others

The matching password was:

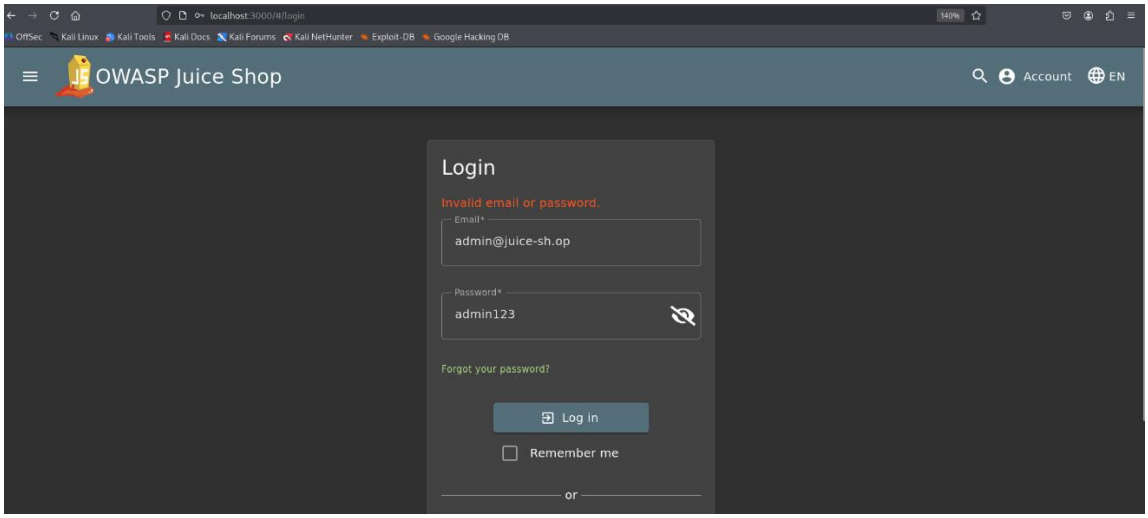
admin123

Step 5: Final Login Using Discovered Credentials

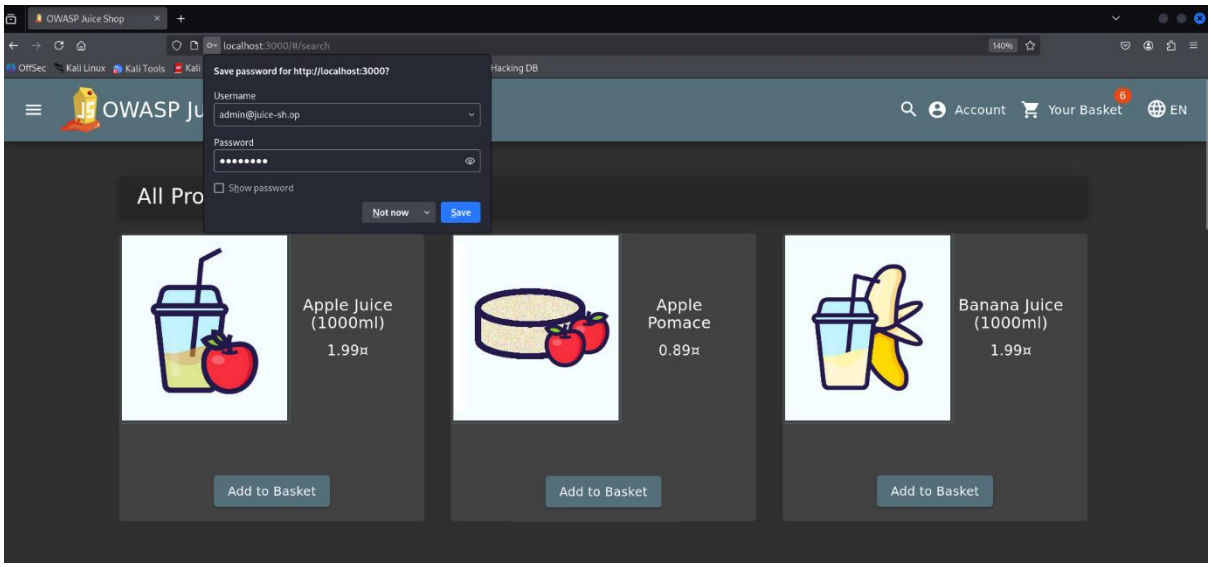
Performed a clean login using:

Email: admin@juice-sh.op

Password: admin123



Login was successful.



This confirmed that the credentials were valid.

Suggested Remediations

- Use prepared statements to prevent SQL injection.
- Validate and sanitize all user inputs.
- Encode output to prevent cross-site scripting (XSS).
- Implement proper access control checks on sensitive resources.
- Use HTTPS and secure cookies to protect sensitive data.

Result

Successfully gained access to the admin account using SQL Injection and password brute force techniques. The challenge "Login Admin" was marked as solved.

Vulnerability 2: Reflected Cross-Site Scripting (XSS)

Objective:

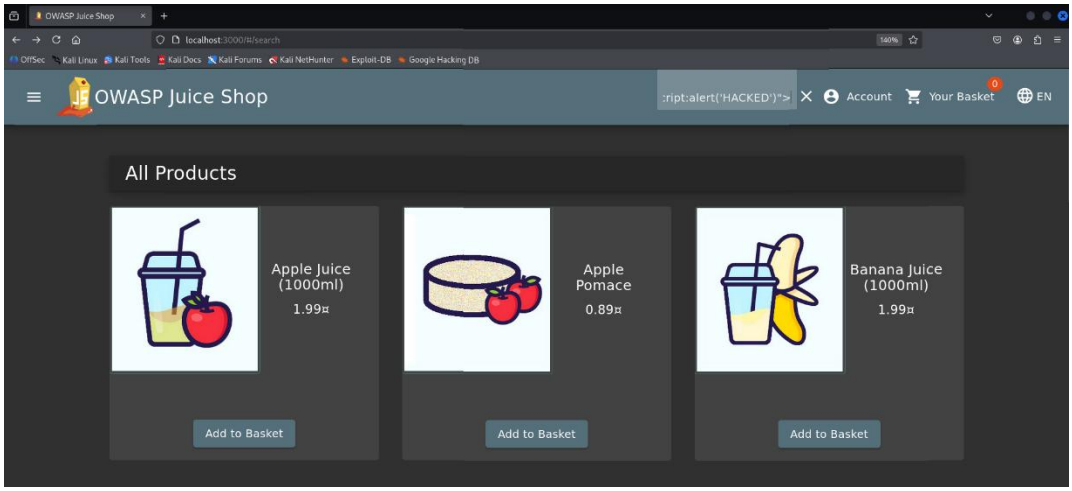
To test if the search functionality in OWASP Juice Shop is vulnerable to **Reflected Cross-Site Scripting (XSS)** by injecting a malicious JavaScript payload.

Steps to Reproduce:

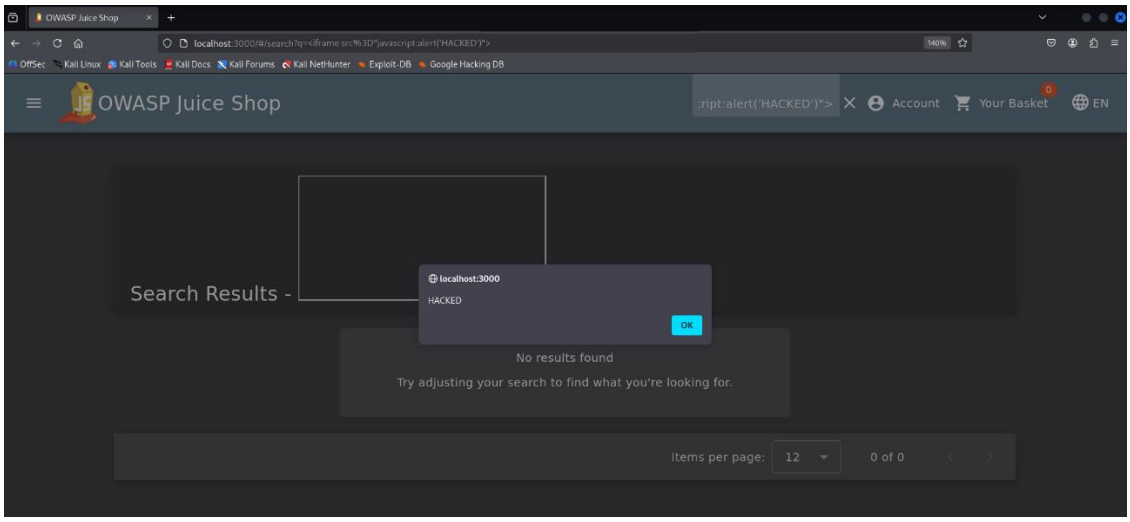
Step 1: Navigated to the search bar on the Juice Shop application.

Step 2: Injected the following payload into the search field:

```
<iframe src="javascript:alert('HACKED')">
```



Upon submitting the payload, a JavaScript alert() box with the message “HACKED” was executed in the browser.



This confirms the presence of a Reflected XSS vulnerability.

Suggested Remediations

- Encode all user input before rendering it in HTML (e.g., use HTML entity encoding).
- Implement Content Security Policy (CSP) headers to restrict script execution.
- Sanitize input on both client and server side to block harmful scripts.
- Avoid reflecting user input directly into the response without validation.
- Use security libraries or frameworks that auto-escape output by default.

Result

The application executed the injected script and displayed a JavaScript alert box with the message “HACKED”, confirming the presence of a **Reflected XSS vulnerability** in the search input.

Vulnerabilty 3:Insecure Direct Object Reference (IDOR)

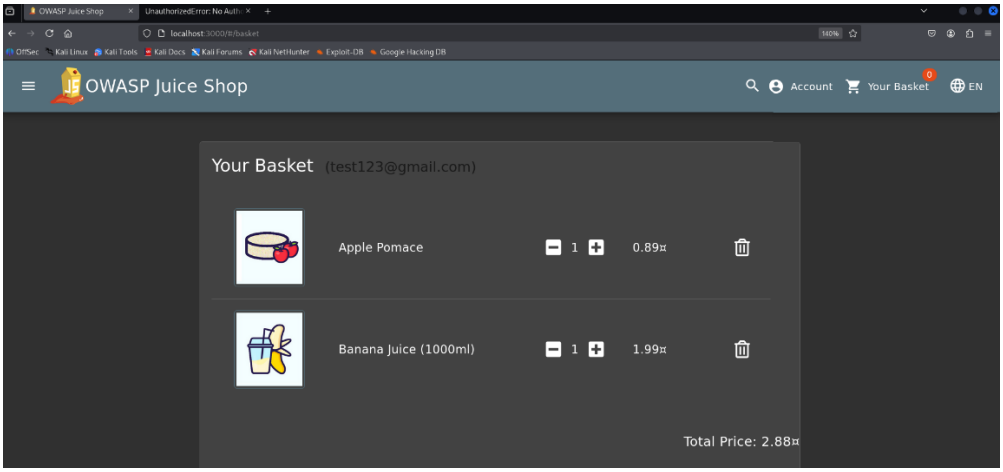
Objective:

To test for Insecure Direct Object Reference (IDOR) by manipulating object IDs in HTTP requests and verify if unauthorized access to other users' data is possible, completing the "View Basket" challenge in OWASP Juice Shop.

Steps to Reproduce:

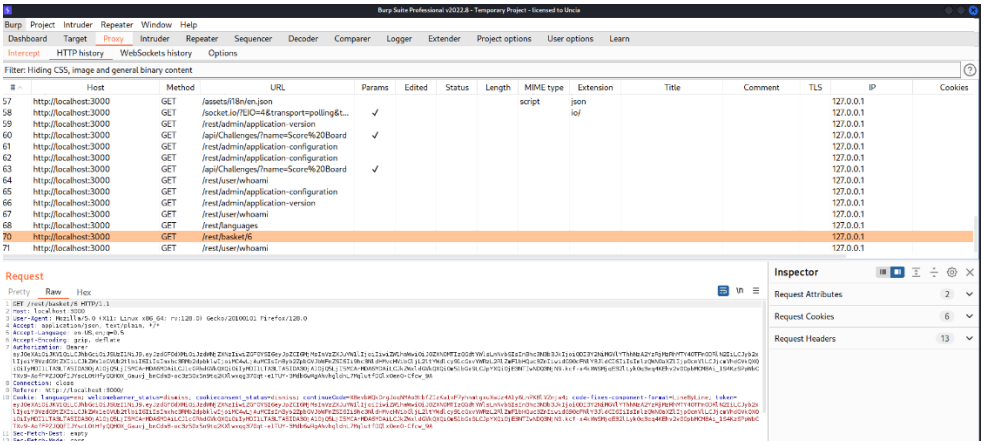
Step 1: Created a normal user account in OWASP Juice Shop.

Logged in and added items to the basket.



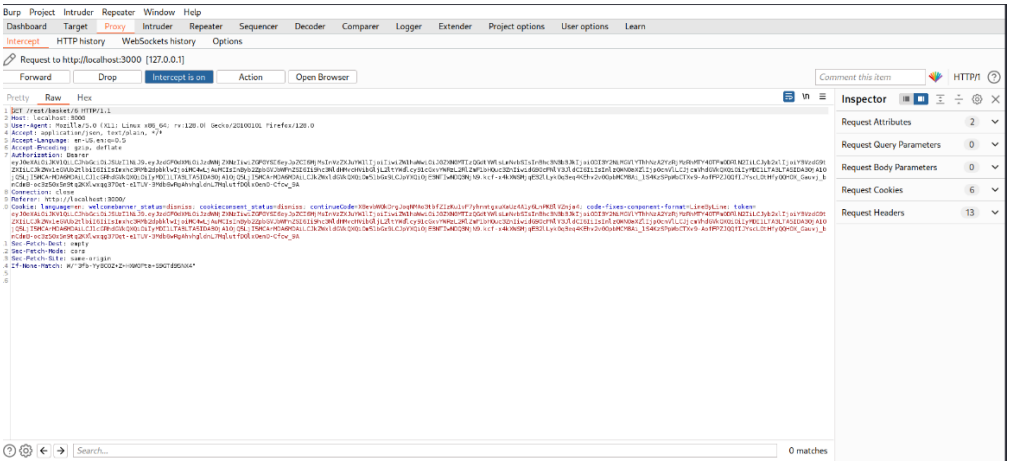
Step 2: Opened Burp Suite and enabled the intercept.

Navigated to Proxy > HTTP History and found the following request:



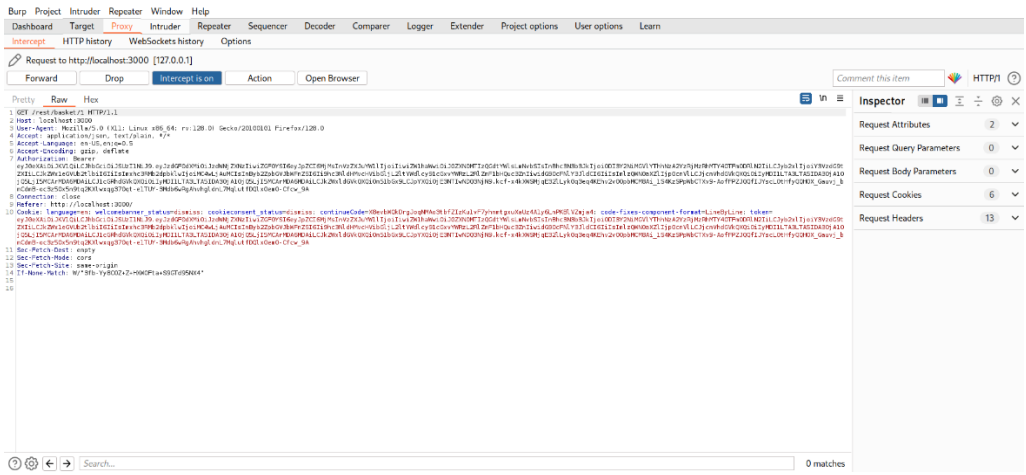
GET /rest/basket/6 HTTP/1.1

(Here, 6 is the basket ID associated with the logged-in user.)



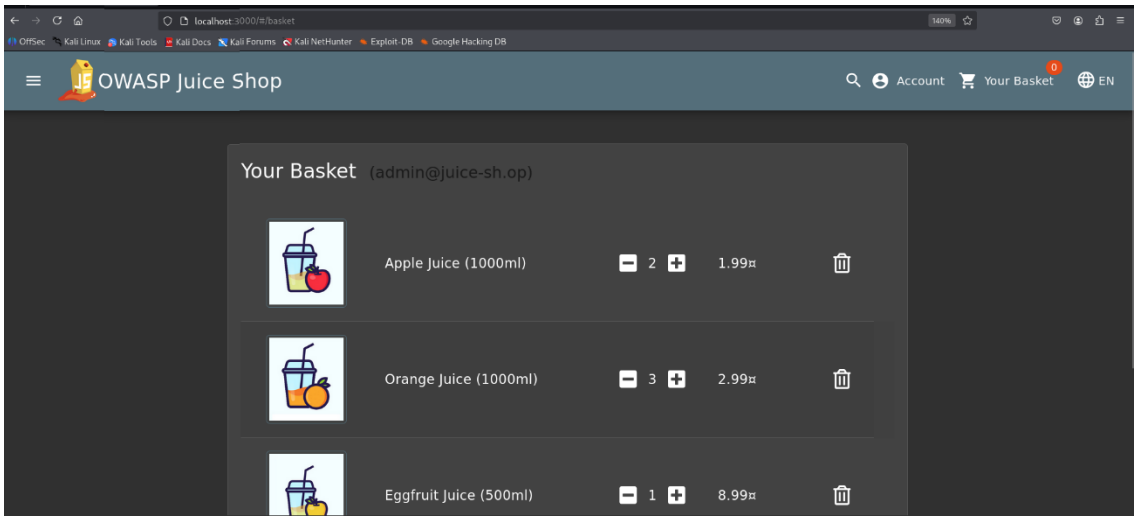
Step 3: Modified the request to:

GET /rest/basket/1 HTTP/1.1



and forwarded it.

Upon forwarding the modified request, the application displayed the admin's basket items inside the normal user's basket view.



Suggested Remediation:

- Implement server-side authorization checks for all resource access.
- Ensure users can only access objects they own (object-level permission checks).
- Use indirect object references (e.g., UUIDs instead of numeric IDs).
- Apply role-based access control (RBAC) to restrict access by user role.
- Log and monitor access to sensitive endpoints
- Include authorization checks in security testing and code reviews.

Result:

This confirms a Broken Access Control vulnerability, specifically an Insecure Direct Object Reference (IDOR), as the application does not validate ownership of the requested basket.

By changing the basket ID in the request, a regular user can view another user's (admin's) basket without authorization.

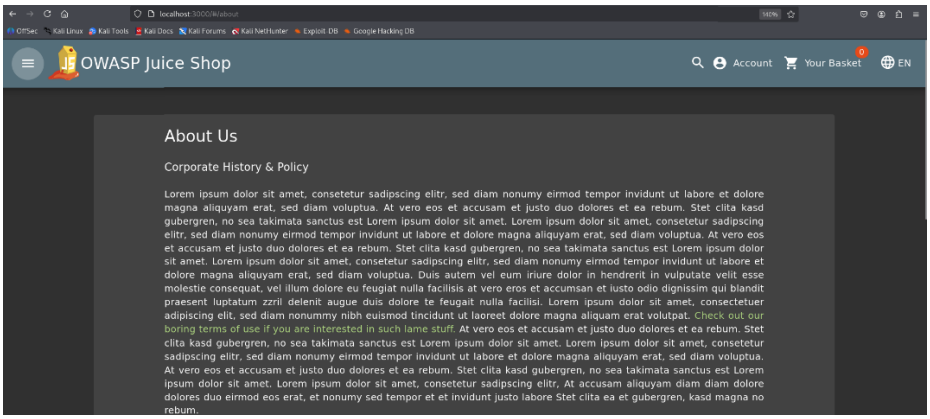
The challenge “View Basket” was successfully solved.

Vulnerability 4: Sensitive Data Exposure

Objective:

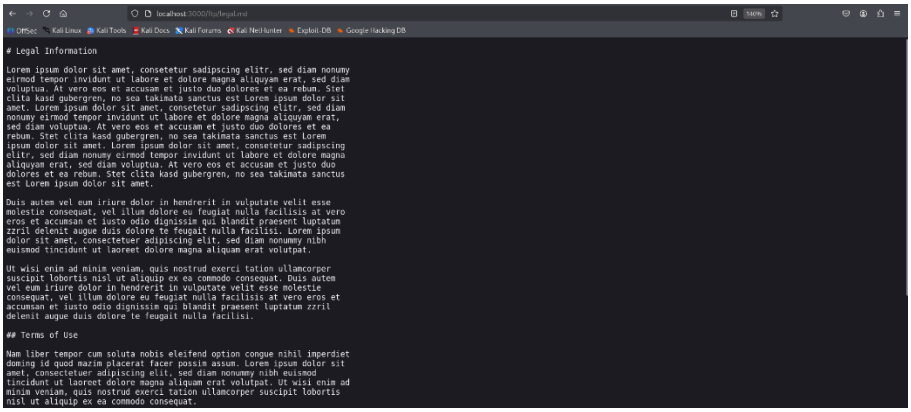
Step 1: click the three lines (hamburger menu) in the top-left corner and navigated to the About Us page.

On the About Us page, I found a green-colored text link, which I clicked.



This opened a URL like

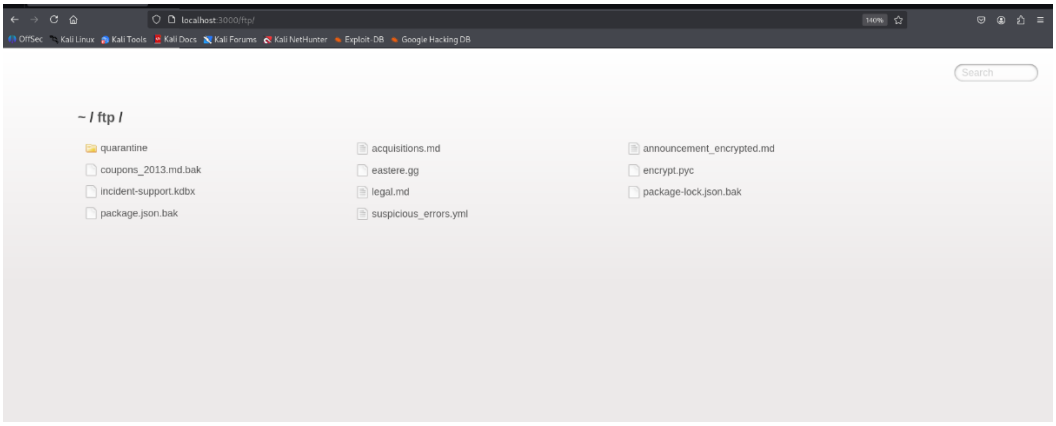
http://localhost:3000/ftp/legal.md



Step 2 :I noticed the URL used an /ftp/ path, which hinted at a file directory.

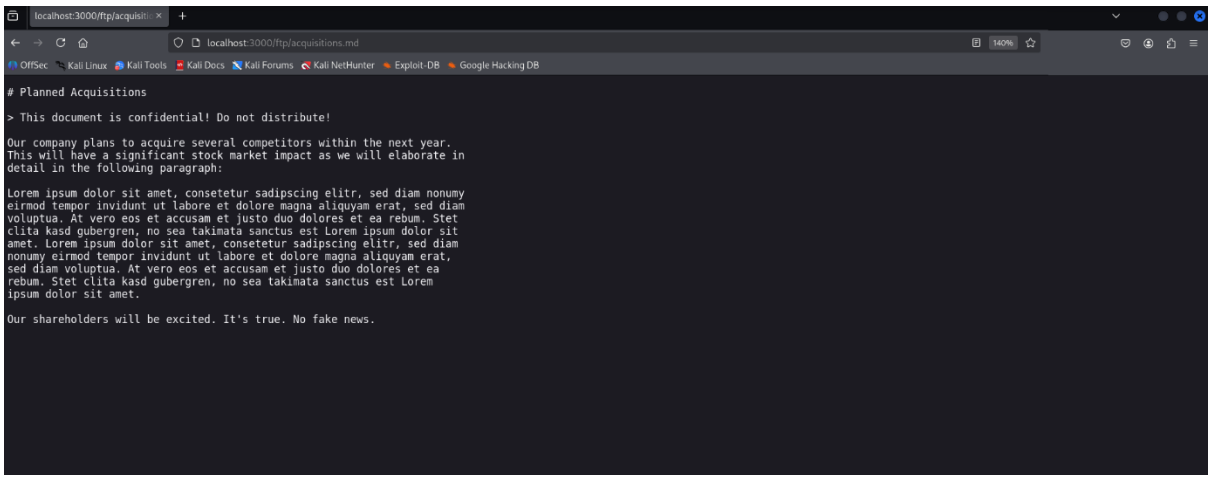
I manually navigated to:

http://localhost:3000/ftp/



and discovered an open directory listing.

Step 3 :Click the file named acquisitions.md and I downloaded the file named acquisitions.md, which contained confidential information.



Suggested Remediation:

- Disable directory listing on the server to prevent browsing of unlinked files.
- Restrict access to sensitive files using proper authentication and authorization checks.
- Store sensitive documents outside the web root or in protected areas.
- Review and audit file paths exposed in the application to ensure they don’t lead to confidential data.
- Implement proper access control policies and least privilege principles for all file access.

Result:

By navigating to an exposed /ftp/ directory from the About Us page, the file acquisitions.md was accessed and downloaded. This file contained confidential business information, confirming a **Sensitive Data Exposure** vulnerability. The "Confidential Document" challenge was successfully solved.

Conclusion

This internship project on OWASP Juice Shop helped me gain hands-on experience in web application vulnerability assessment and penetration testing. I learned how to identify and exploit common web vulnerabilities such as SQL Injection, XSS, IDOR, and Sensitive Data Exposure using Burp Suite. The practical exposure to real-world security flaws enhanced my understanding of web application security and reinforced the importance of secure coding practices. This experience has strengthened my foundation in ethical hacking and cybersecurity.

References

1. https://github.com/juice-shop/juice-shop
2. https://portswigger.net/burp/documentation
3. https://owasp.org/www-project-top-ten/