Experiment 9: Full Web Application Penetration Test

Scenario:

You must perform a comprehensive test against the OWASP Juice Shop. The organization wants a detailed understanding of all web vulnerabilities before deployment.

Tasks:

- Use OWASP ZAP to spider and scan the application.
- Identify various vulnerabilities (XSS, SQLi, broken authentication, insecure direct object references) and exploit them.
- Summarize the findings and recommend remediations.

Deliverable:

A full web application penetration test report, including identified vulnerabilities, exploitation proofs, and remediation steps.

INTRODUCTION TO OWASP ZAP

The OWASP Zed Attack Proxy (ZAP) is a free, open-source web application security scanner maintained by the Open Web Application Security Project (OWASP). ZAP is designed to help security professionals and developers identify security vulnerabilities in web applications. It supports both automated scanning and manual testing, making it suitable for both beginners and experienced testers. Key features include:

- Spidering
- Passive and active scanning
- Request interception
- Vulnerability detection (such as XSS, SQLi, and more)

Install OWASP ZAP:

sudo apt install zaproxy

Run ZAP:

zaproxy

Step-by-Step Instructions

Step 1: Start Juice Shop

sudo docker run -d -p 3000:3000 bkimminich/juice-shop

Juice Shop will run at: http://localhost:3000

Step 2: Open OWASP ZAP

- Launch ZAP from Kali's menu:

  Applications > Web Application Analysis > Zaproxy

Step 3: Use ZAP for Manual Scanning

- In ZAP, click "Manual Explore" from the top-left menu.
- In the popup, enter the target URL: http://localhost:3000
- Click "Launch Browser" — this opens ZAP's built-in browser, automatically configured to work with ZAP's proxy.
- You are now ready to manually explore and test the Juice Shop while all traffic is captured by ZAP.

Step 4: Spider the Site

- In ZAP, right-click on the site tree http://localhost:3000
- Choose: Attack > Spider Site
- Let it crawl the entire website.

Note: This initiates a manual exploration-based spidering process rather than a fully automated scan. You are validating pages by manually browsing and observing real-time HTTP requests.

Step 5: Manual Validation

To further confirm vulnerabilities, try these manual tests:

- **SQL Injection Test:**

  For the login page, input the following credentials:

  Email: ' OR 1=1--

  Password: anything

  This will confirm SQL Injection if you bypass the login screen without valid credentials.

- **XSS Test:**

  In Juice Shop's search bar, input the following payload:

  <iframe src="javascript:alert('XSS 1')">

  This will confirm XSS if the payload triggers an alert box.

- **Broken Authentication:**

  Test default credentials for administrative login:

  Username: admin@juice-sh.op

  Password: admin123

  If successful, this confirms broken authentication.

Web Application Penetration Test Report

Vulnerability Focus: SQL Injection (SQLi)
Target Application: OWASP Juice Shop
Test Date: [Insert Date]
Tester: [Insert Name]

Vulnerability Summary

- **Title:** SQL Injection in Login Function
- **Severity:** Critical
    - This vulnerability is considered critical because a successful SQL Injection attack allows attackers to bypass authentication, gaining unauthorized access to the system and potentially exposing sensitive data.
- **CWE Reference:** CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
- **Affected URL:** http://127.0.0.1:3000/#/login
- **HTTP Method:** POST

Description

The login functionality of OWASP Juice Shop is vulnerable to SQL Injection due to improper input sanitization. By injecting malicious SQL code into the username field, an attacker can bypass authentication and gain unauthorized access to the system. This vulnerability can be exploited by sending specially crafted input to the login endpoint.

Steps to Reproduce

Request:

POST /rest/user/login HTTP/1.1

Host: <target-ip>:3000

Content-Type: application/json

{

"email": "' OR 1=1--",

"password": "anything"

}

Explanation:

- The payload ' OR 1=1-- manipulates the SQL query by always returning true, bypassing the authentication check.

Response:

HTTP/1.1 200 OK

{

"authentication": {

"token": "eyJhbGciOiJIUzI1NiIs..."

}

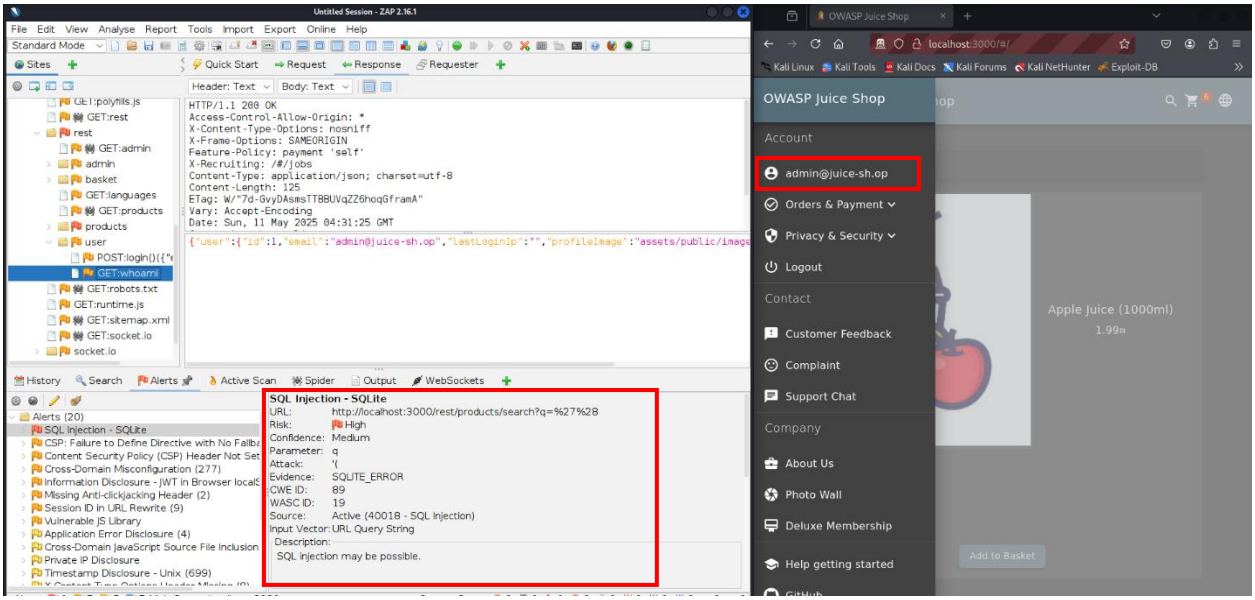
}

Explanation:

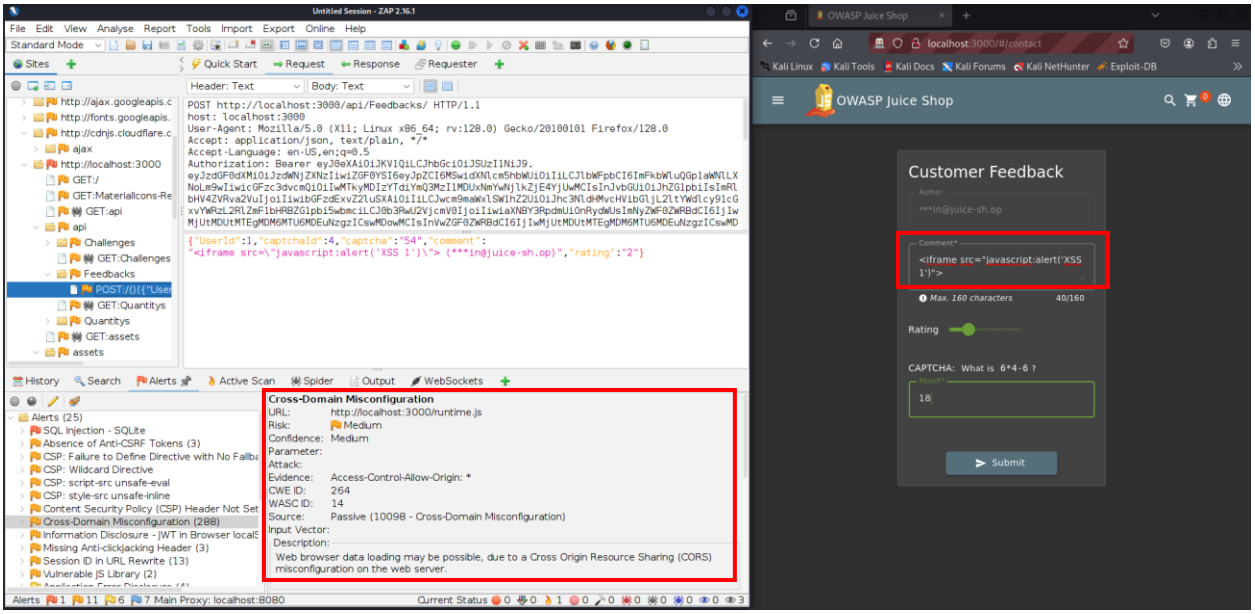
- The response indicates a successful login without valid credentials, confirming the presence of SQL Injection.

Evidence

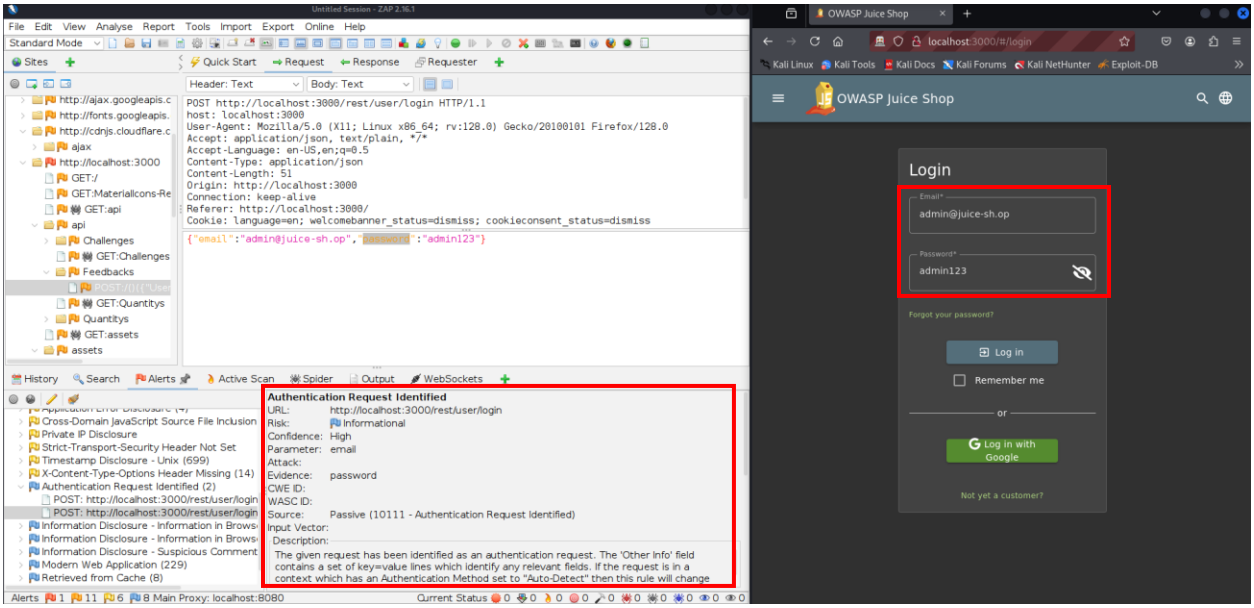
SQL Injection Test:



XSS Test:



Broken Authentication:



Risk & Impact

- **Risk Level:** Critical
- **Impact:**
    - Full account takeover, including administrative access
    - Unauthorized data exposure (users, orders, tokens)
    - Possibility of remote code execution depending on DB backend
    - Reputational and compliance risk

Recommendations

- Use parameterized queries or ORM frameworks (e.g., Sequelize for Node.js)
- Implement server-side input validation and sanitization
- Avoid building SQL queries via string concatenation
- Use Web Application Firewalls (WAFs) for detection and blocking
- Conduct periodic code reviews and security testing

Conclusion

In this penetration test on the OWASP Juice Shop, we found serious security problems like SQL Injection, Cross-Site Scripting (XSS), and weak login protection (broken authentication). These issues can allow attackers to log in without a real account, steal user data, or take full control of the system. To fix these problems, developers should write code that checks and cleans user input, use safe methods for handling login and database queries, and regularly test their applications for security issues. Fixing these problems quickly will help keep the app safe from hackers.

****