

Security Assessment Report — Task 1

Target: OWASP Juice Shop (local Docker)

Date: 2025-10-19

Analyst:NIVAS.B.U

Executive Summary

I performed a hands-on assessment of a local OWASP Juice Shop instance running in Docker (<http://localhost:3000>). My goal was to identify common web application vulnerabilities following OWASP Top 10 guidance. I explored the application manually using a proxy and verified issues with Burp Repeater and browser DevTools. The testing uncovered several actionable issues ranging from injection and XSS to authentication and access-control weaknesses. The findings below reflect what I validated and the recommended fixes I propose.

Top findings (high level)

- SQL Injection (High)
 - Sensitive Data Exposure (Medium)
 - Persistent (Stored) Cross-Site Scripting (XSS) (High)
 - Reflected Cross-Site Scripting (XSS) (Medium–High)
 - Broken Authentication (Medium–High)
 - DOM Cross-Site Scripting (DOM XSS) (Medium)
 - Broken Access Control / IDOR (High)
-

Scope & Rules of Engagement

- **Scope:** Local OWASP Juice Shop instance deployed via Docker on my test machine.
 - **Tools I used:** Burp Suite (Community), Firefox DevTools, curl, Docker logs, Nikto (optional), OWASP ZAP (optional).
 - **Constraints:** I performed non-destructive testing only in an isolated lab environment. No data was exfiltrated beyond proof-of-concept evidence.
-

Methodology

I followed a practical testing workflow:

1. Reconnaissance: browsed the app with a proxy enabled, recorded endpoints, and exported HAR/Burp logs (see evidence files).
 2. Passive analysis: reviewed traffic to identify input points and API endpoints.
 3. Manual verification: used Burp Repeater to craft PoCs and confirmed vulnerabilities without causing harm.
 4. Documentation: saved raw requests, screenshots, and concise remediation guidance for each finding.
-

Findings (detailed)

Each finding below includes a concise description, a PoC summary, impact, evidence pointers, and remediation I recommend.

SQL Injection — High

- **Issue:** Injection in authentication or search endpoint allows SQL payloads to manipulate query logic.
- **PoC (summary):** Used Burp Repeater to inject `admin' OR '1'='1` into the login (or search) parameter and observed authentication bypass / indicative database error.
- **Impact:** Attacker could bypass authentication, extract data or manipulate the database.
- **Remediation:** Use parameterized queries / prepared statements and validate input.

The image contains two screenshots of the OWASP Juice Shop application, illustrating SQL injection findings.

Screenshot 1: Login Page

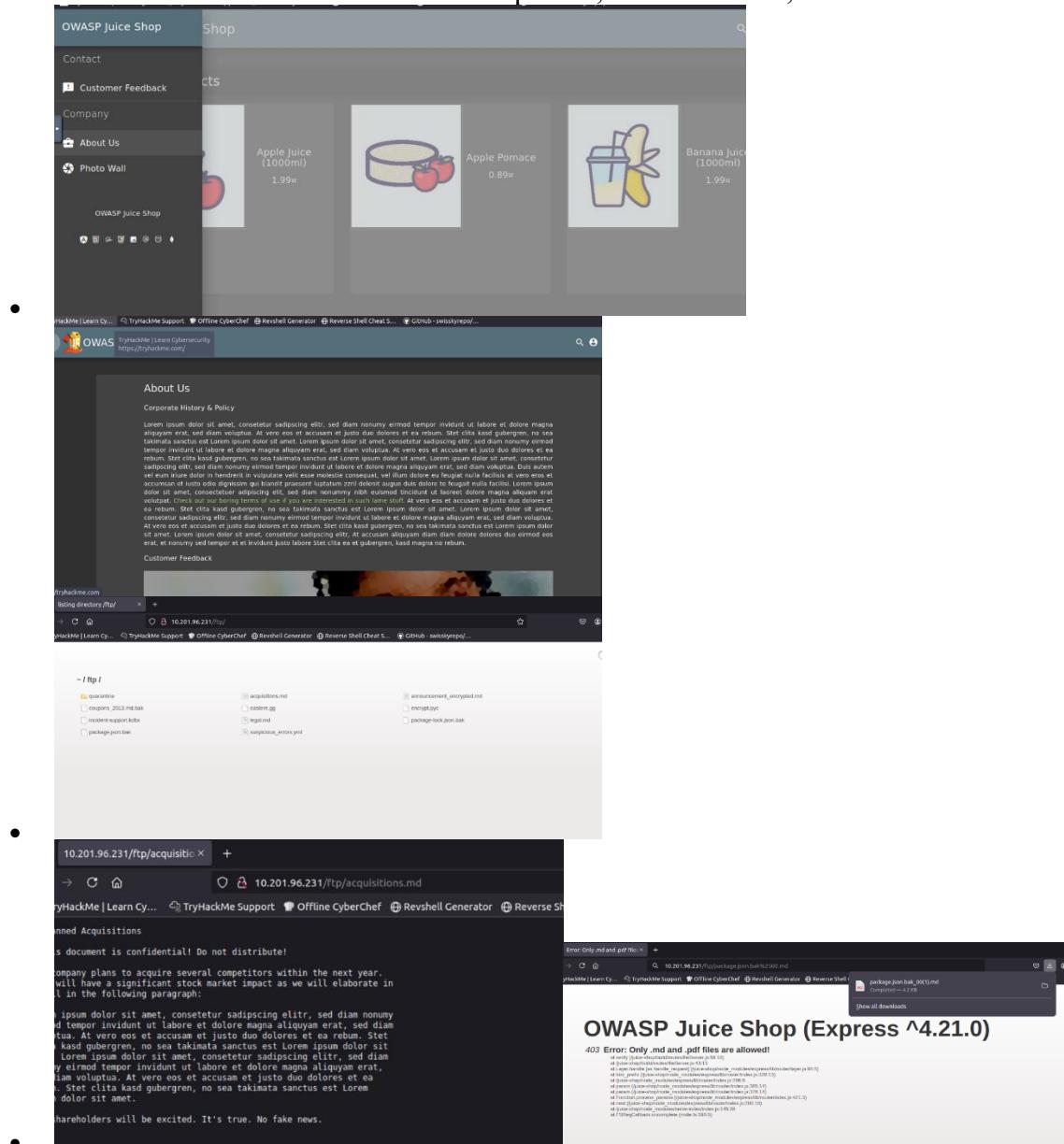
This screenshot shows the login interface of the OWASP Juice Shop. The user has injected the payload `' or '1'='1` into the "Email" field. The password field contains "a". The "Log In" button is visible, and the "Remember me" checkbox is checked. The page displays an error message: "Unknown column '1' in 'where clause'".

Screenshot 2: All Products Page

This screenshot shows the "All Products" page. A dropdown menu is open for the user "admin@juice-shop". The menu items include "Privacy Policy", "Request Data Export", "Request Data Erasure", "Change Password", "2FA Configuration", and "Last Login IP". The "Logout" option is also present. The main content area shows products like "Apple Juice (1000ml)" and "Apple Pomace".

Sensitive Data Exposure — Medium

- **Issue:** API responses and storage expose sensitive fields or tokens; cookies missing flags.
- **PoC (summary):** Observed tokens in JSON responses and cookies without HttpOnly/Secure.
- **Impact:** Token theft or session compromise.
- **Remediation:** Redact secrets from responses, secure cookies, enforce TLS.



Persistent (Stored) XSS — High

- Issue:** Stored XSS in product reviews or feedback that is persisted and later rendered to users.
- PoC (summary):** Submitted ">" in a product review and the alert executed on page load
- Impact:** Persistent client-side code execution; session theft or phishing.
- Remediation:** Context-aware output encoding, whitelist allowed HTML, implement CSP.

The screenshot shows a browser window for the OWASP Juice Shop application. The URL is `http://10.201.96.231/#/privacy-security/last-login-ip`. The page displays the message "Last Login IP" with the IP address "10.201.45.202". Below the page, a NetworkMiner tool is open, showing three captured requests:

- Request 1: GET /rest/saveLoginIp HTTP/1.1 (IP: 10.201.96.231)
- Request 2: GET /api/quantity HTTP/1.1 (IP: 10.201.96.231)
- Request 3: GET /rest/products/search?q= (IP: 10.201.96.231)

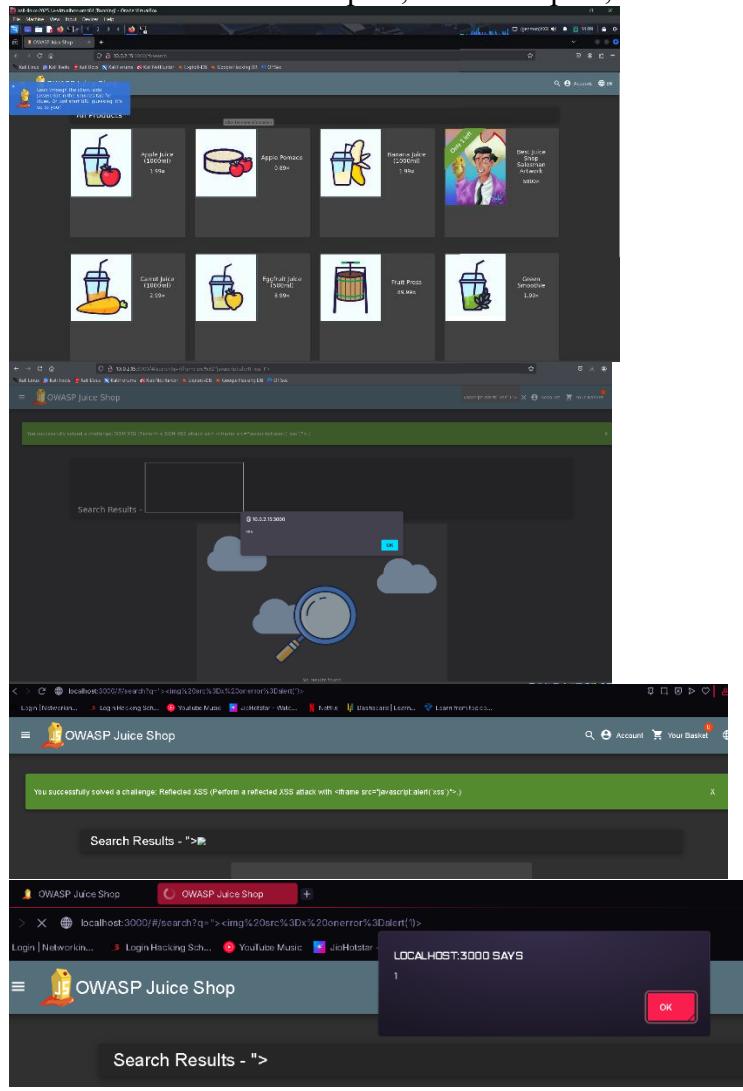
The NetworkMiner interface includes a "Request" tab showing the raw request data, and an "Inspector" tab showing the request headers and the payload being modified. The payload is set to:

```
Name: True-Client_IP  
Value:  
<iframe src="javascript:alert('xss')">
```

On the right, a browser window shows the search results page with the injected XSS payload displayed as "xss".

Reflected XSS — Medium–High

- **Issue:** Input reflected in server responses (search or error pages) leads to script execution when crafted links are used.
- **PoC (summary):** Reflected payload via search parameter executed in browser render.
- **Impact:** Script execution in victim's browser when they follow a crafted link.
- **Remediation:** Encode outputs, validate inputs, use CSP.



Broken Authentication — Medium–High

- **Issue:** Weak session management and authentication flows (e.g., session reuse, predictable reset tokens, missing account lockout).
- **PoC (summary):** Observed session cookie behaviors and token characteristics indicating potential reuse or weak reset flow
- **Impact:** Account takeover and session hijacking.
- **Remediation:** Rotate sessions on login, use `HttpOnly`/`Secure` flags, enforce strong password policies and rate limits.

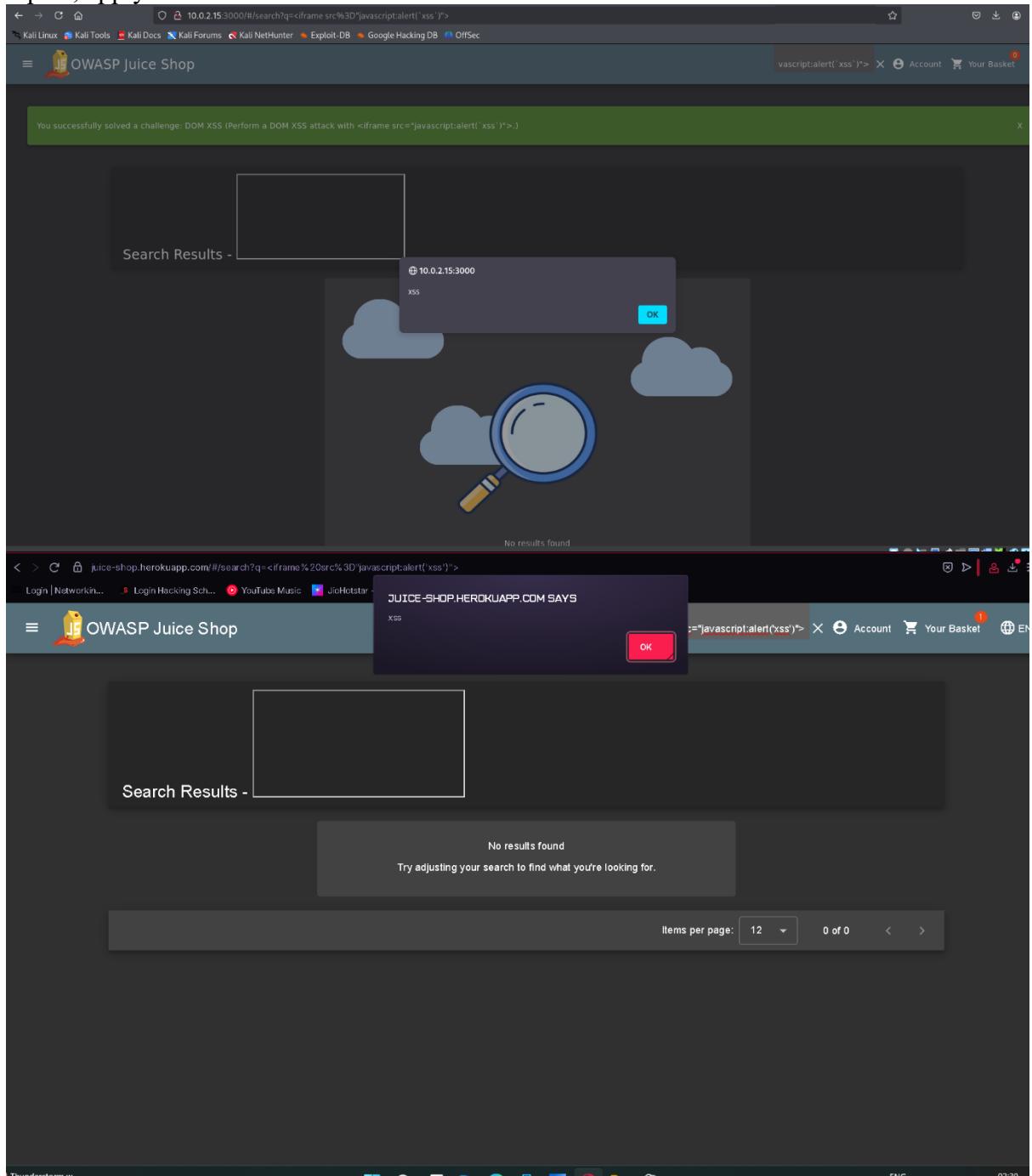
The screenshot shows the OWASP Juice Shop application. On the left, a browser window displays the 'Login' page with the URL `http://10.201.121.177/login`. The user has entered 'admin@juice-shop' in the 'username' field and 'password123' in the 'password' field, then clicked 'Log in'. On the right, another browser window shows the same 'Login' page, but the session has been hijacked, displaying the same credentials. This indicates a session reuse vulnerability.

Below the browser windows, a Kali Linux terminal window shows the command `curl -X POST http://10.201.121.177/login -d "username=admin&password=password123"` being run, which successfully logs in as the administrator.

At the bottom, a screenshot of the Burp Suite interface shows a captured session. It includes a 'Packets' tab with network traffic, a 'Proxy' tab showing the captured request, and an 'Attack' tab where an 'Intruder' attack is being configured against the target host `10.201.121.177` on port `80`. The attack is set to use HTTP and is configured to send multiple requests simultaneously to test for session reuse.

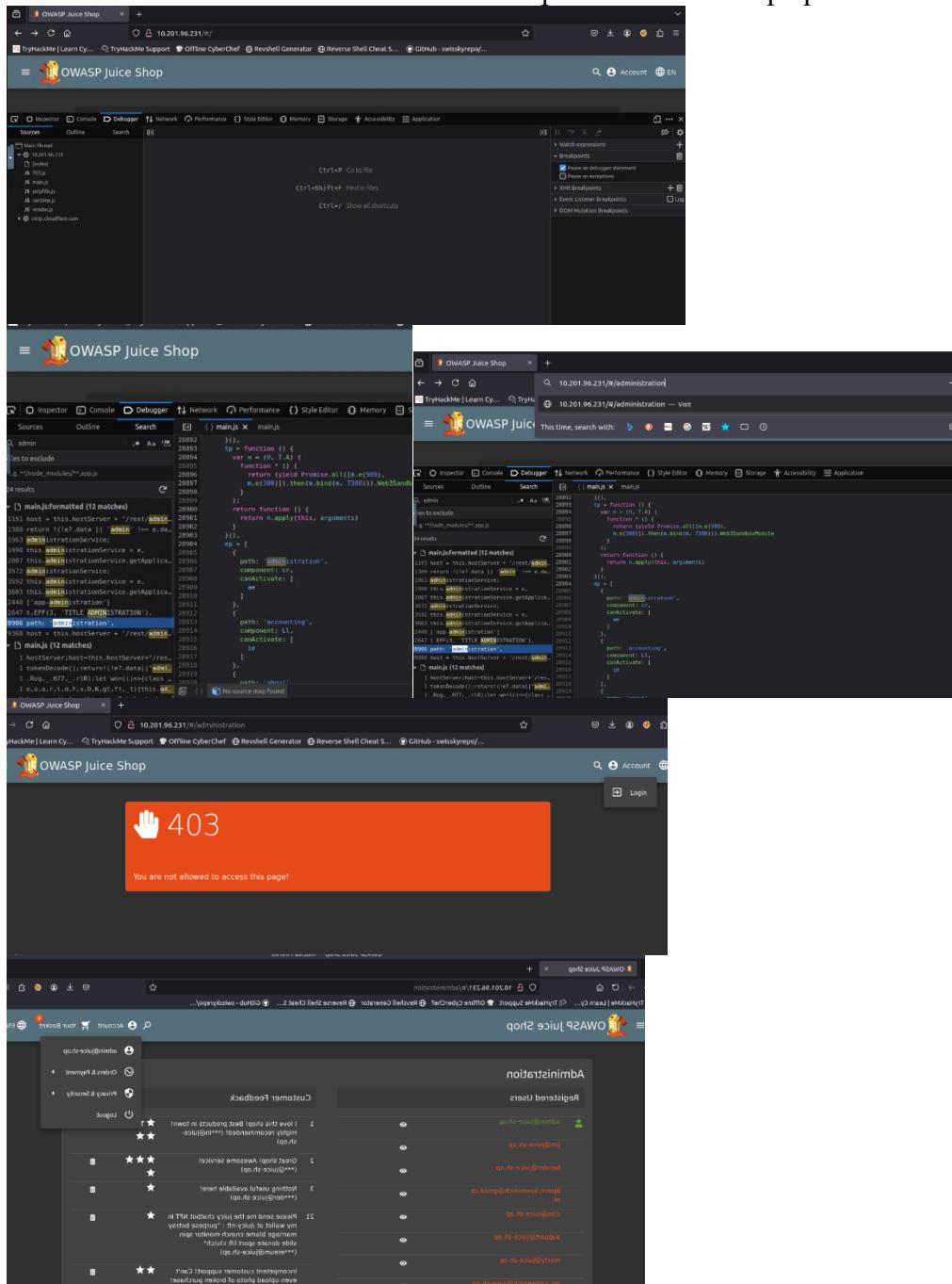
DOM XSS — Medium

- **Issue:** Client-side JavaScript reads untrusted data from the URL fragment or localStorage and injects it into the DOM unsafely.
- **PoC (summary):** Visiting a crafted URL fragment triggered script execution without server interaction.
- **Impact:** Script runs in user's browser context; similar impacts to other XSS variants.
- **Remediation:** Avoid `innerHTML` with untrusted data, use `textContent`, sanitize inputs, apply CSP.



Broken Access Control — High

- **Issue:** API endpoints return resources based on client-supplied IDs without verifying ownership.
- **PoC (summary):** Modified GET /rest/orders/{id} from one ID to another and retrieved another user's data.
- **Impact:** Unauthorized access to sensitive data.
- **Remediation:** Enforce server-side ownership checks and use opaque identifiers.



Recommendations

- Regularly update and patch vulnerable components.
 - Implement input validation and output encoding to prevent XSS and SQLi.
 - Enforce strong authentication and role-based access controls.
 - Avoid exposing sensitive information in responses or error messages.
 - Use HTTPS and secure session handling to protect user data.
-

Learning Outcomes

- Understood how common web vulnerabilities work in real environments.
 - Gained hands-on experience using **Burp Suite**, **browser proxy setup**, and **Juice Shop**.
 - Learned to identify, exploit, and document findings like a SOC or pentest analyst.
 - Improved practical understanding of **OWASP Top 10** risks.
-

Conclusion

Through the OWASP Juice Shop simulation, I explored and exploited multiple real-world vulnerabilities such as XSS, SQL Injection, Broken Auth, and Sensitive Data Exposure. This task strengthened my practical cybersecurity skills and improved my awareness of secure coding and web application defense practices.
