Cyber Security Project: Bounty Hunt 101

HAMDAN BIN HASHIM

January 17, 2026

1 Executive Summary

This report details the security assessment conducted on the locally hosted **OWASP Juice Shop** application. The objective was to identify, analyze, and document security vulnerabilities within the application's scope. The assessment followed a standard penetration testing methodology, encompassing reconnaissance, vulnerability scanning, and manual exploitation.

Three high-severity vulnerabilities were identified during the engagement:

1. **Reflected Cross-Site Scripting (XSS)**: Allowing arbitrary JavaScript execution in the user's browser.
2. **SQL Injection (SQLi)**: Enabling authentication bypass and unauthorized administrative access.
3. **Insecure Direct Object Reference (IDOR)**: Permitting unauthorized access to other users' sensitive shopping basket data.

2 Methodology & Reconnaissance

2.1 Target Information

- **Application Name**: OWASP Juice Shop
- **Environment**: Localhost (Lab Environment)
- **Tools Used**: Nmap, Burp Suite Community, Browser DevTools

2.2 Reconnaissance Findings

A network scan was performed using `nmap` to identify open ports and service versions.

Command Executed:

```
nmap -sC -sV localhost
```

Scan Results (Excerpt): The scan confirmed the application is hosted on TCP port 3000.

```
PORT      STATE    SERVICE       VERSION
3000/tcp  open     http          Node.js Express framework
| http-title: OWASP Juice Shop
|_http-server-header: Access-Control-Allow-Origin: *
```

3 Vulnerability Analysis

3.1 1. Reflected Cross-Site Scripting (XSS)

Vulnerability Type: Client-Side Injection
Severity: High
Affected Component: Search Functionality (`/rest/products/search`)

3.1.1 Description

The application's product search feature fails to properly sanitize user-supplied input before reflecting it in the HTTP response. This allows an attacker to inject malicious JavaScript code, which is executed by the victim's browser within the context of the vulnerable site.

3.1.2 Proof of Concept (Reproduction Steps)

1. Navigate to the OWASP Juice Shop homepage.
2. Locate the "Search" (magnifying glass) icon in the navigation bar.
3. Inject the following Polyglot XSS payload into the search field:

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

4. Submit the search query by pressing Enter.
5. **Observation:** An alert box is triggered immediately, confirming that the JavaScript payload was executed by the browser.

3.1.3 Evidence

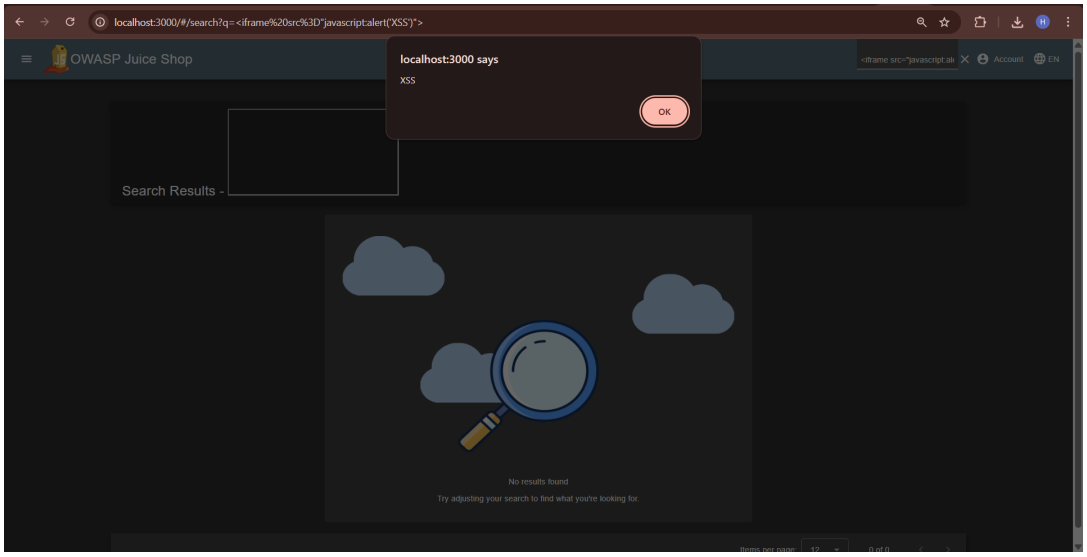

Figure 1: Execution of Reflected XSS payload.

3.2 2. SQL Injection (Authentication Bypass)

Vulnerability Type: Injection (SQLi)
Severity: Critical
Affected Component: User Login (`/rest/user/login`)

3.2.1 Description

The user login mechanism is vulnerable to SQL injection. The application constructs database queries using unsanitized user input in the ”Email” field. By injecting SQL logic operators, an attacker can alter the query structure to evaluate as true, bypassing the password check and authenticating as the first user in the database (Administrator).

3.2.2 Proof of Concept (Reproduction Steps)

1. Navigate to the Login page via the ”Account” menu.
2. In the **Email** field, enter the following injection payload:

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

   Note: The payload closes the string literal, forces a true condition, and comments out the remainder of the query.
3. Leave the **Password** field empty or enter random characters.
4. Click the ”Log in” button.
5. **Observation:** The application logs the user in as `admin@juice-sh.op` without a valid password, granting full administrative privileges.

3.2.3 Evidence

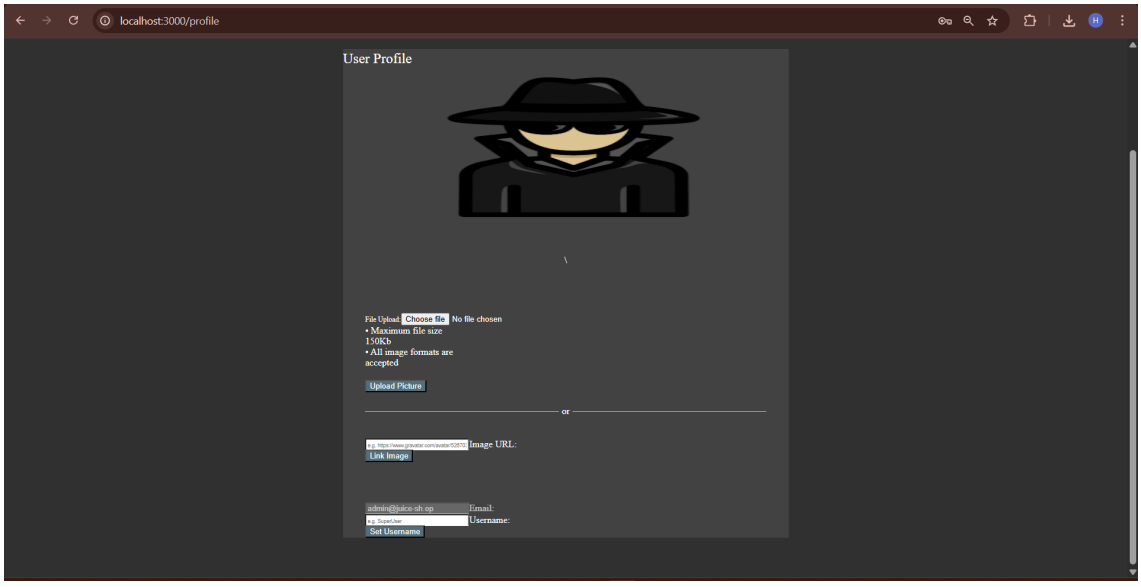

Figure 2: Successful authentication bypass via SQL Injection.

3.3 3. Insecure Direct Object Reference (IDOR)

Vulnerability Type: Broken Access Control (IDOR)
Severity: High
Affected Component: Basket API (`/rest/basket/{id}`)

3.3.1 Description

The application uses predictable numeric identifiers to reference user shopping baskets in API calls. The server fails to validate that the authenticated user is authorized to access the requested basket ID. This allows an attacker to enumerate and view the shopping baskets of other users by simply modifying the basket ID in the HTTP request.

3.3.2 Proof of Concept (Reproduction Steps)

1. Log in to the application with a standard user account.
2. Add a unique item (e.g., "Best Juice Shop Salesman Artwork") to your basket to distinguish it.
3. Intercept the HTTP traffic using **Burp Suite Proxy** or view the **Network Tab** in browser DevTools.
4. Locate the API request fetching the basket details: `GET /rest/basket/6 HTTP/1.1` (assuming ID 6 is the current user).
5. Send this request to **Burp Repeater**.
6. **Test 1 (Baseline):** Send the request as-is. Verify it returns your basket contents ("Best Juice Shop Salesman Artwork").
7. **Test 2 (Exploit):** Modify the URL path to increment the ID: `GET /rest/basket/2 HTTP/1.1`
8. Send the modified request.
9. **Observation:** The server responds with `200 OK` and returns the JSON data for Basket ID 2 (containing different items like "Raspberry Juice"), confirming unauthorized access.

3.3.3 Evidence

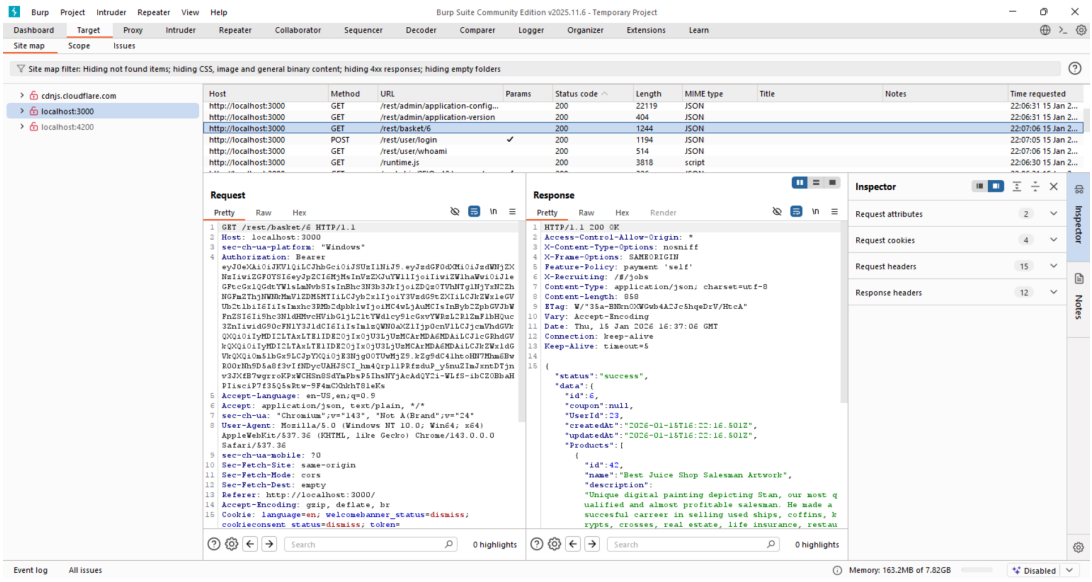

Figure 3: Baseline: Authorized access to own basket.

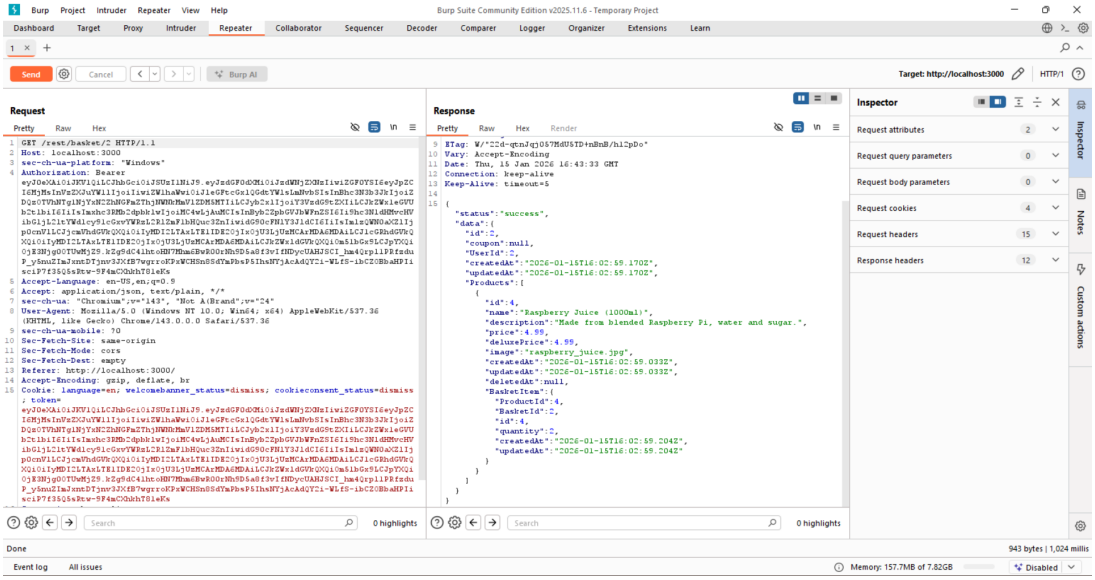

Figure 4: Exploit: Unauthorized access to victim's basket via ID manipulation.

4 Conclusion

The assessment revealed critical security flaws in the OWASP Juice Shop application. The lack of input validation and broken access controls presents a significant risk to data confidentiality and integrity. Immediate remediation is recommended, specifically implementing parameterized queries for SQLi, output encoding for XSS, and server-side authorization checks for IDOR.