OWASP Juice Shop: Web Application Security Assessment

Target System: OWASP Juice Shop (Docker Container)
Assessment Date: January 2026
Security Analyst: Klyne Zyro C. Reyes
Confidentiality Level: Internal / Educational Use Only

Executive Summary

A comprehensive security assessment was conducted on the **OWASP Juice Shop** web application to identify and exploit critical application-level vulnerabilities. The assessment utilized the **OWASP Top 10** framework to evaluate the target's resilience against common web attacks. The engagement revealed systemic flaws in input validation, access control, and data protection mechanisms.

Key Findings:

- **Administrative Account Compromise:** Critical **Injection (SQLi)** vulnerabilities in the login portal allowed for authentication bypass, granting immediate administrative access without valid credentials.
- **Weak Authentication & Credential Leaks:** Hardcoded administrative credentials were discovered within client-side JavaScript files. Additionally, weak user passwords were compromised via **Open Source Intelligence (OSINT)** and social engineering vectors (e.g., Security Question bypass).
- **Sensitive Data Exposure:** The application fails to restrict access to internal assets. Confidential documents, developer backup files (.bak), and "soft-deleted" product data were successfully exfiltrated using directory enumeration and SQL injection techniques.
- **Broken Access Control:** Improperly secured administrative interfaces (**Scoreboard**) and Insecure Direct Object References (IDOR) allowed for the manipulation of user baskets and unauthorized data access.
- **Client-Side Attacks:** Multiple **Cross-Site Scripting (XSS)** flaws were identified, allowing an attacker to execute arbitrary JavaScript in victim browsers to hijack sessions.

Critical Risk: The combination of hardcoded secrets, injection flaws, and weak authentication presents a **Critical Risk** to the application. An unauthenticated attacker can trivially escalate privileges to Administrator, recover "deleted" sensitive data, and take over user accounts. Immediate remediation of credential management and input sanitization is required.

1.0 Assessment Methodology

1.1 Scope & Objectives The objective of this assessment was to evaluate the security posture of the OWASP Juice Shop web application (hosted via Docker). The assessment focused on identifying vulnerabilities listed in the **OWASP Top 10 (2021)** framework, specifically targeting:

- Broken Access Control
- Injection Flaws (SQLi, NoSQL)
- Cryptographic Failures
- Security Misconfiguration

1.2 Tools Utilized The assessment was conducted using a "Black Box" approach (no prior knowledge of source code) utilizing the following toolset:

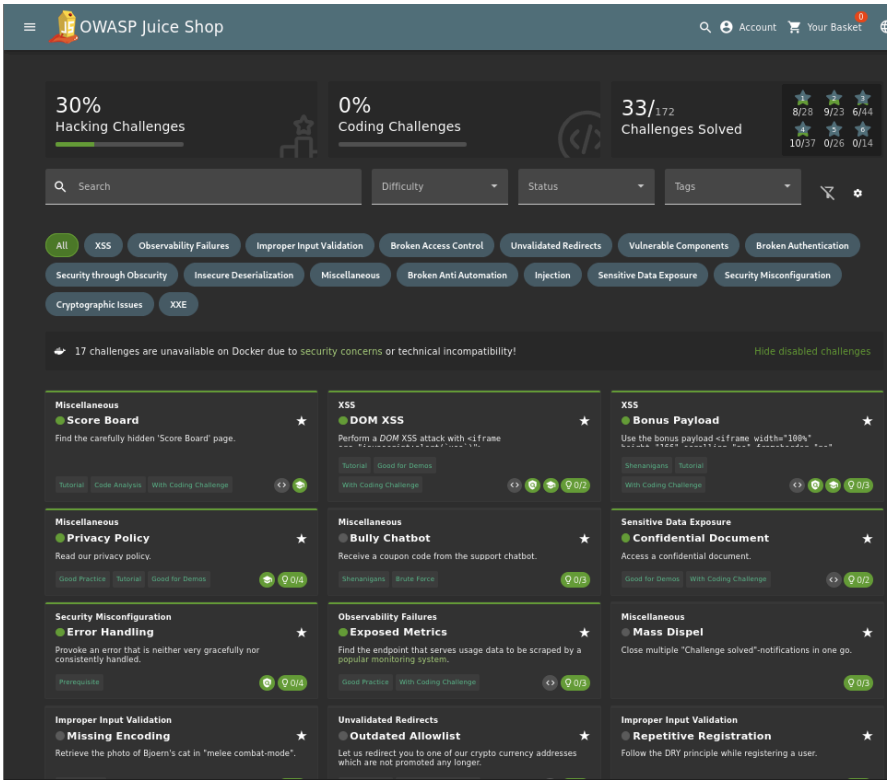
- **Burp Suite Community Edition:** For HTTP request interception and manipulation.
- **Firefox Developer Tools:** For client-side debugging, JavaScript analysis, and network inspection.
- **Manual Enumeration:** Payload injection and logic analysis.

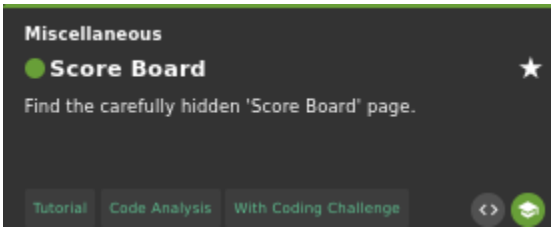
2.0 Security Findings

2.1 Information Disclosure & Reconnaissance

2.1.1 Exposure of Administrative Interfaces (Scoreboard)

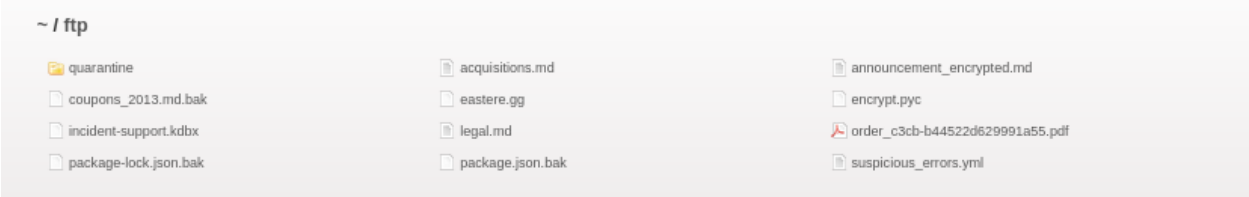
- **Severity: Medium**
- **OWASP Category:** A05:2021-Security Misconfiguration
- **Description:** The application fails to hide sensitive administrative interfaces from unauthenticated users. During the reconnaissance phase, a "Scoreboard" page was discovered which tracks the completion status of various hacking challenges. While intended for gamification, in a real-world scenario, this exposes the application's internal logic and potential weak points.
- **Evidence:**
    1. Inspected the application's JavaScript source code.
    2. Identified a route pointing to **/score-board**.
    3. Navigated to **http://localhost:3000/#/score-board**.
    4. **Result:** Successfully accessed the tracking interface without administrative privileges.

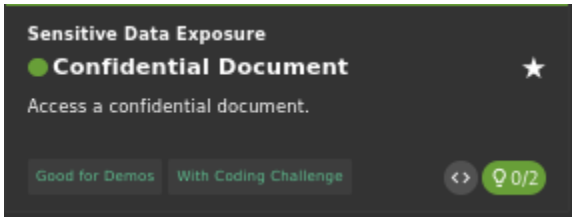




2.1.2 Sensitive Directory Enumeration (/ftp)

- **Severity: Medium**
- **OWASP Category:** A01:2021-Broken Access Control
- **Description:** The application permits directory listing on the /ftp endpoint. This allows unauthenticated users to browse the file system structure and identify potential assets for exfiltration.
- **Exploitation Steps:**
    1. Analyzed the "**About Us**" page and identified a link to **legal.md**.
    2. Truncated the URL to access the parent directory: **http://localhost:3000/ftp**.
    3. **Result:** The server returned a full index of the directory, revealing file names such as **acquisitions.md**, **coupons_2013.md.bak**, and **package.json.bak**.





2.1.3 Obfuscated Path Disclosure (Security via Obscurity)

- **Severity: Low**
- **OWASP Category:** A05:2021-Security Misconfiguration
- **Description:** The application attempts to hide sensitive "Easter Egg" paths using weak encoding (Base64 + ROT13) instead of proper access controls.
- **Exploitation Steps:**
    1. Located a suspicious Base64 string in the application source: **L2d1ci9xcmlmL25lci9mYi9zaGFhbC9ndXJsL3V2cS9uYS9ybmZncmUvcnR0L2p2Z3V2YS9ndXlvcm5mZ3JlL3J0dA==**
    2. **Decoded:** Base64 To

       /gur/qrif/ner/fb/shaal/gurl/uvq/na/rnfgre/rtt/jvguva/gur/rnfgre/rtt

       Then to ROT13 Cipher

       /the/devs/are/so/funny/they/hid/an/easter/egg/within/the/easter/egg

    3. **Result:** Revealed the hidden path /the/devs/are/so/funny/they/hid/an/easter/egg....

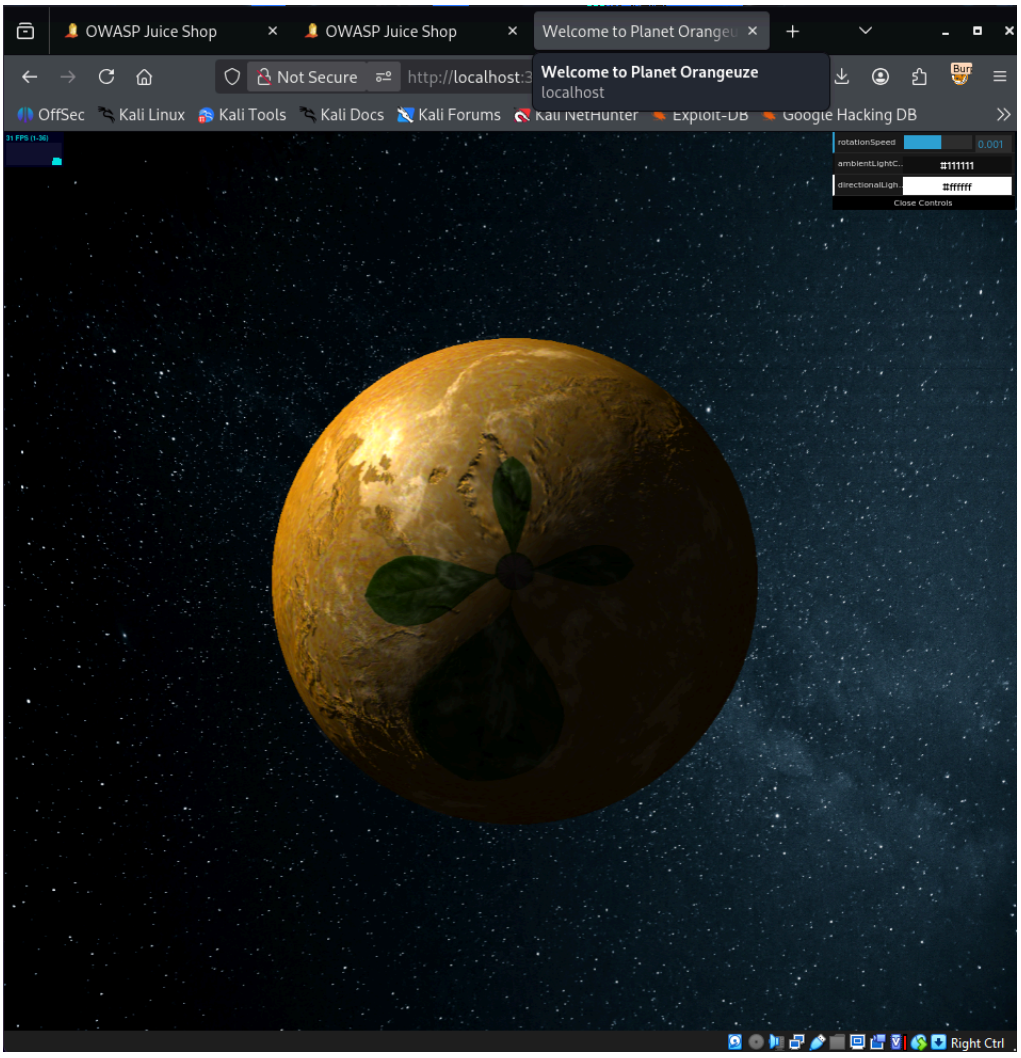
- **Impact:** While currently harmless, relying on obscurity to hide paths is a bad practice that could mask genuine administrative backdoors.

Open

eastere.gg_00.md
~/Downloads

Save

```
"Congratulations, you found the easter egg!"
- The incredibly funny developers

...

...

...

Oh' wait, this isn't an easter egg at all! It's just a boring text file! The real easter egg can be found
here:

L2d1ci9xcmlmL25lci9mYi9zaGFhbC9ndXJsL3V2cS9uYS9ybmZncmUvcnR0L2p2Z3V2YS9ndXIvcm5mZ3JlL3J0dA==

Good luck, egg hunter!
```

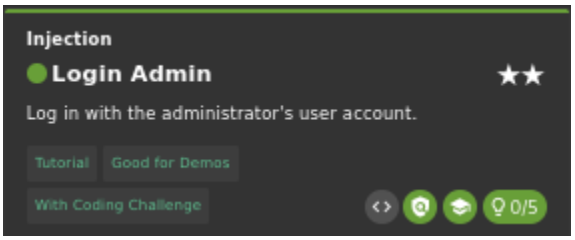


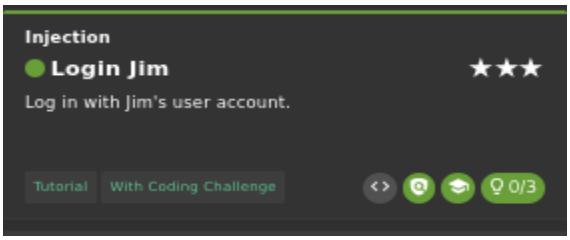
http://localhost:3000/the/devs/are/so/funny/they/hid/an/easter/egg/within/the/easter/egg

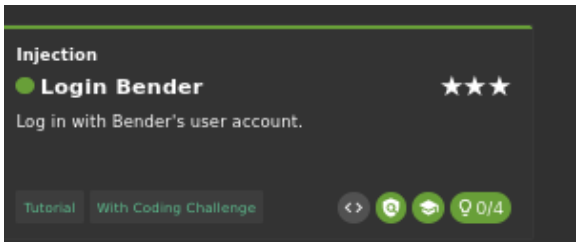
2.2 Injection Vulnerabilities

2.2.1 Authentication Bypass via SQL Injection

- **Severity: High**
- **OWASP Category:** A03:2021-Injection
- **Description:** The application's login mechanism fails to properly sanitize user input. By injecting standard SQL tautologies or commenting out the remainder of the query, an attacker can bypass authentication checks.
- **Impact:** Full account takeover of Administrative and Privileged users (e.g., jim@juice-sh.op, bender@juice-sh.op) without knowing the password.
- **Exploitation Steps:**
    1. Navigated to the /login page.
    2. **Scenario A (Generic Admin):** Injected ' OR 1=1 -- into the email field.
    3. **Scenario B (Targeted Attack):** Targeted specific high-value users by injecting jim@juice-sh.op' --.
    4. Observed successful login and session establishment.







2.2.2 Database Schema Extraction & Data Exfiltration (UNION-Based SQLi)

- **Severity: Critical**
- **Description:** The product search endpoint (**/rest/products/search**) is vulnerable to UNION-based SQL injection. By manipulating the search parameter, it is possible to append results from other database tables to the legitimate product list.
- **Technical Analysis:**

    - Column Enumeration: Through manual fuzzing with ORDER BY, it was determined the current table uses **9 columns**.

- Data Type Discovery: It was identified that columns 2 and 3 accept string data, allowing for data extraction.

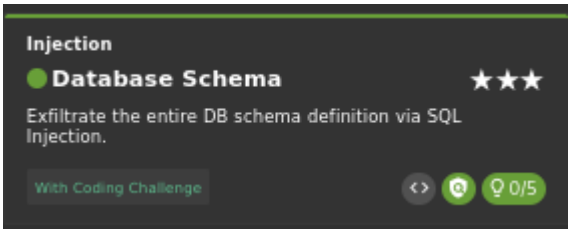
- **Exploitation Payload:** To dump the database structure (Table Names):

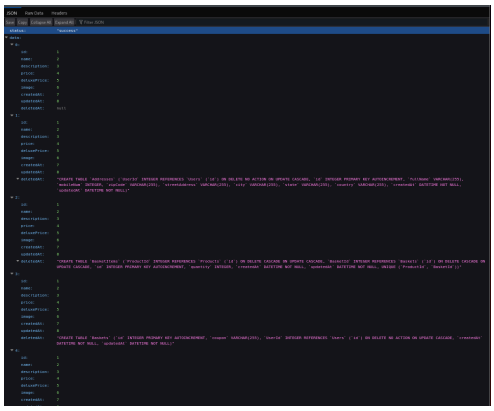
  test')) UNION SELECT 1, 2, 3, 4, 5, 6, 7, 8, sql FROM sqlite_master–

  To dump User Credentials (Email & Password):

  qwert')) UNION SELECT 1, email, password, '4', '5', '6', '7', '8', '9' FROM Users--

- **Impact:** Complete exposure of the Users table, including password hashes, and full visibility of the database schema (sqlite_master), allowing for mapped attacks against the backend.



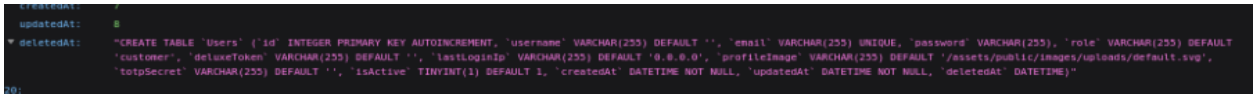


2.2.3 Ephemeral Account Creation (Privilege Escalation)

- **Severity: High**
- **Description:** Using the previously identified UNION vulnerability, an attacker can force the database to return a "fabricated" user record. The application accepts this injected row as a valid user session.
- **Exploitation Steps:**
    1. Intercepted a login request via Burp Suite.
    2. Modified the email parameter to inject a full user row:

' UNION SELECT 1, 'hack', 'acc0unt4nt@juice-sh.op', '12345', 'accounting', '', '1.2.3.4', 'lmg', '', 1, '2024-01-01', '2024-01-01', null FROM Users–

3. The application processed the injected data and logged the attacker in as the non-existent "Accountant" user with administrative privileges.



2.2.4 Integrity Violation via NoSQL Injection

- **Severity: Medium**
- **Description:** The application utilizes a MongoDB (NoSQL) backend for product reviews. The patch endpoint fails to validate JSON input, allowing the use of MongoDB operators to bypass ID checks.
- **Exploitation Steps:**
    1. Intercepted a POST request when editing a user review.
    2. Modified the id parameter to use the "Not Equal" ($ne) operator:

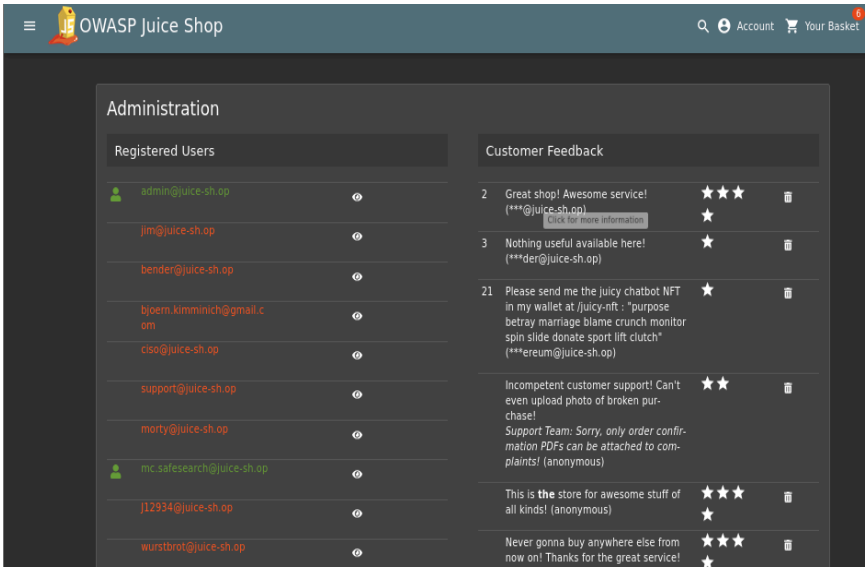
"id": { "$ne": -1 }

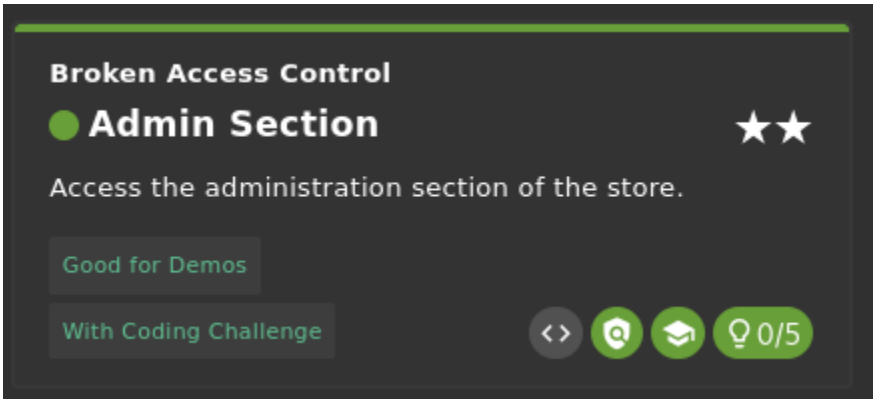
    3. This condition evaluates to true for all documents, allowing the attacker to modify reviews belonging to other users or bypass ownership checks.

2.3 Broken Access Control & Privilege Escalation

2.3.1 Improper Access Control on Administrative Routes

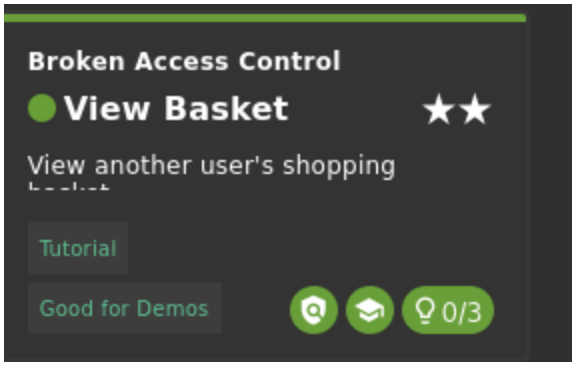
- **Severity: High**
- **OWASP Category:** A01:2021-Broken Access Control
- **Description:** The application utilizes client-side routing for its administrative interface. While the backend API attempts to verify roles, the frontend route **/#/administration** is discoverable.
- **Vulnerability:**
    1. **Scenario A (Unauthenticated):** Accessing the route reveals the existence of the page (Information Disclosure), though API calls may fail (403 Error).
    2. **Scenario B (Authenticated via SQLi):** After exploiting the SQL Injection (Finding 2.2.1), the attacker acts as the admin user. The application then fully renders the /administration page, granting the ability to view all registered users (jim, bender, mc.safesearch) and delete arbitrary reviews.
- **Exploitation Steps:**
    1. Performed the SQL Injection login (**' OR 1=1 --**) to establish an administrative session.
    2. Manually navigated to http://localhost:3000/#/administration.
    3. **Result:** The application rendered the full User Management interface, confirming that the injected session holds elevated privileges.





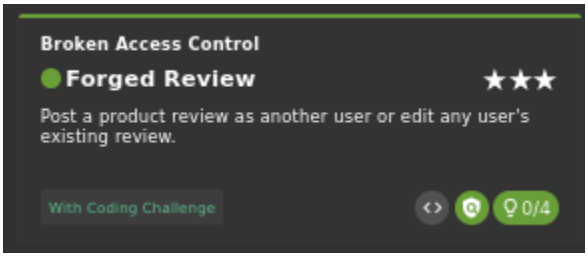
2.3.2 Insecure Direct Object Reference (IDOR) - Shopping Basket

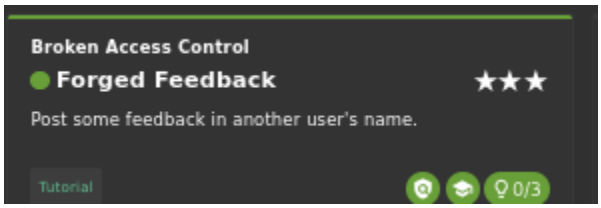
- **Severity: High**
- **Description:** The application relies on client-side input (**bid or Basket ID**) to retrieve shopping cart contents. By modifying this ID in the HTTP GET/POST requests, an attacker can view or manipulate other users' baskets.
- **Exploitation Steps:**
    1. Intercepted a request to **/rest/basket/1** using Burp Suite.
    2. **Horizontal Escalation:** Modified the bid parameter to 2.
    3. **Result:** Successfully viewed items in another user's cart.
    4. **Manipulation:** Modified the "Add to Basket" request to place items into a victim's cart, disrupting business logic.



2.3.3 Integrity Violation via Review Forgery

- **Severity: Medium**
- **Description:** The product review functionality trusts user input for the "Author" and "Rating" fields without validation on the server side.
- **Exploitation Steps:**
    1. Submitted a standard review.
    2. Intercepted the request and modified the user email to admin@juice-sh.op.
    3. **Input Validation Bypass:** Changed the star rating to 0 (which is not possible via the UI).
    4. The server accepted the forged review, violating data integrity.





2.4 Client-Side Injection (XSS)

2.4.1 Reflected Cross-Site Scripting (XSS)

- **Severity: Medium**
- **OWASP Category:** A03:2021-Injection
- **Description:** The application's "**Search**" functionality reflects user input directly into the **Document Object Model** (**DOM**) without proper escaping. This allows an attacker to execute arbitrary JavaScript in the victim's browser.

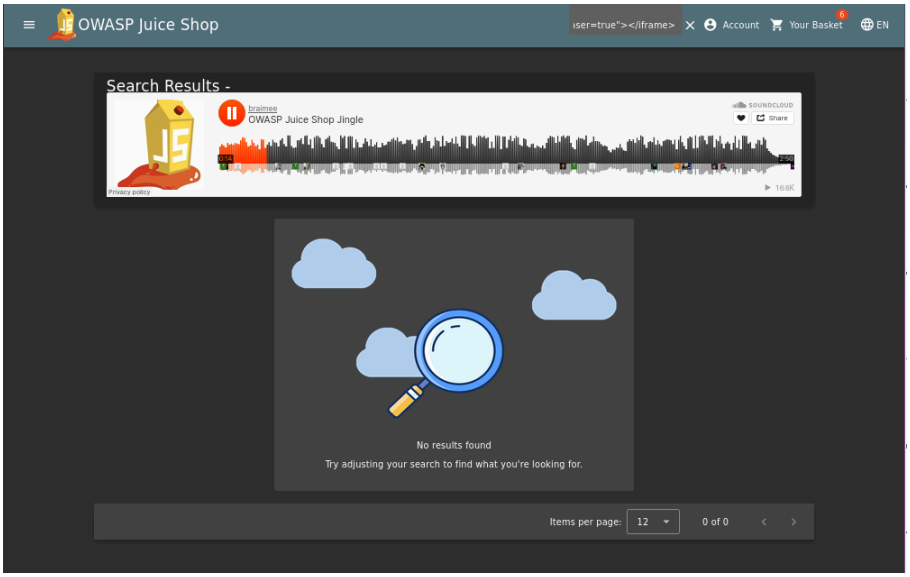
- **Exploitation Steps:**
    1. Navigated to the search bar.
    2. **Proof of Concept:** Entered the following payload to trigger an alert:

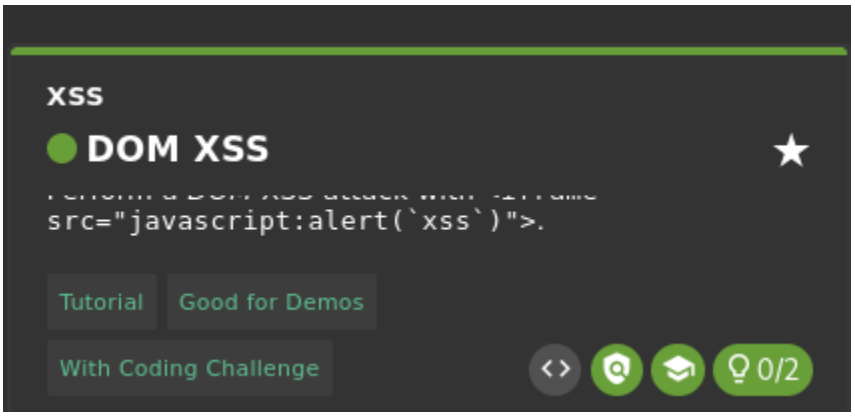
<iframe src="javascript:alert(`xss`)">

    3. **Defacement POC:** Successfully injected an external SoundCloud widget to demonstrate the ability to alter page content and potentially Phish users:

<iframe width="100%" height="166" scrolling="no" frameborder="no" allow="autoplay" src="https://w.soundcloud.com/player/?url=https%3A//api.soundcloud.com/tracks/771984076&color=%23ff5500&auto_play=true&hide_related=false&show_comments=true&show_user=true&show_reposts=false&show_teaser=true"></iframe>

- **Impact:** An attacker could steal session cookies (document.cookie), redirect users to malicious sites, or perform actions on behalf of the victim.





2.5 Sensitive Data Exposure & Improper Asset Management

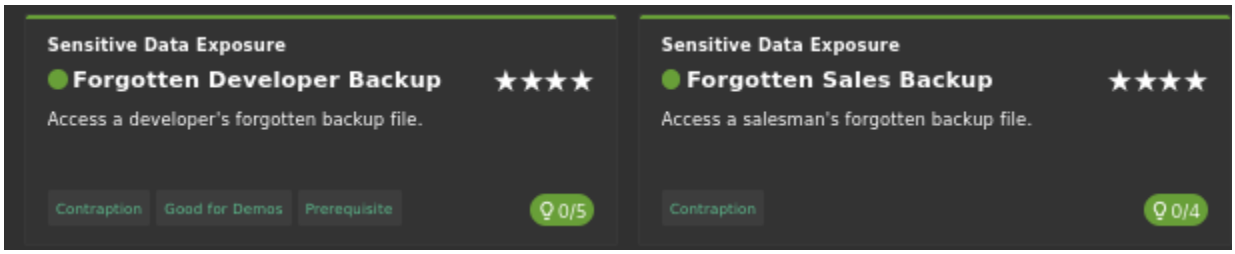
2.5.1 Retrieval of Backup Files (Poison Null Byte Bypass)

- **Severity: High**
- **OWASP Category:** A05:2021-Security Misconfiguration
- **Description:** Developers left backup files (**.bak**) in the public web directory. While the server attempts to block access to non-standard file types, this restriction was bypassed using a "**Poison Null Byte**" (**%2500**) injection, tricking the server into treating the request as a valid .md file.
- **Exploitation Steps:**
    1. Enumerated the /ftp directory.
    2. Identified two critical backup files:
        - package.json.bak (Reveals dependencies).
        - coupons_2013.md.bak (Reveals unreleased discount codes).
    3. **Bypass Payload:** Appended **%2500.md** to the URL:

/ftp/package.json.bak%2500.md

/ftp/coupons_2013.md.bak%2500.md

- **Impact:** Exfiltration of package.json reveals the exact version numbers of backend libraries (e.g., sanitize-html v1.4.2, express-jwt v0.1.3). This allows an attacker to map these specific versions against the **CVE database** to identify unpatched vulnerabilities in the server's supply chain, potentially leading to Remote Code Execution (RCE) without further interaction.

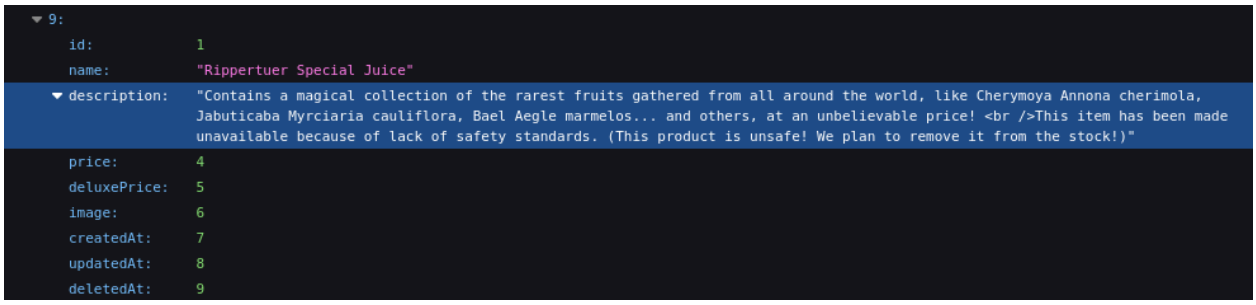


2.5.2 Recovery of "Deleted" Data via SQL Injection

- **Severity: High**
- **Description:** The application "**soft deletes**" products (marking them hidden rather than removing them). Using the previously identified SQL Injection, it is possible to recover products removed for safety violations.
- **Exploitation Steps:**
    1. Used UNION SELECT to identify the columns.
    2. Injected a query to find rows where the **deletedAt date** is **NOT null**:

test')) UNION SELECT 1, name, description, 4, 5, 6, 7, 8, 9 FROM Products WHERE deletedAt IS NOT NULL--

3. **Result:** Recovered the "**Rippertuer Special Juice**" , a product removed from the store, proving that sensitive "**deleted**" data remains accessible.

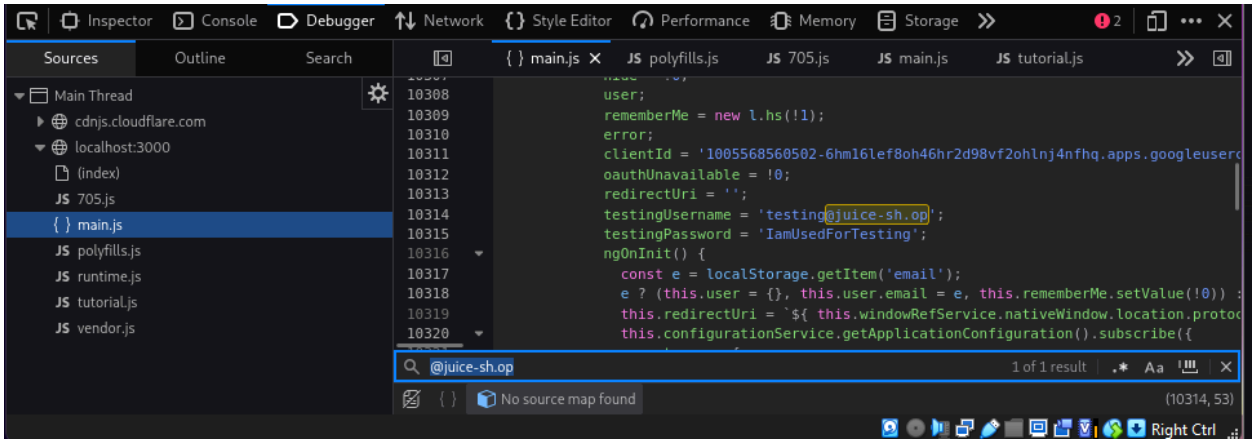


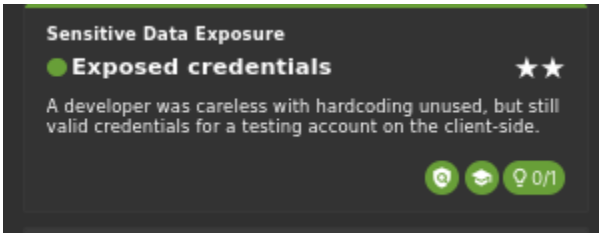
2.6 Weak Authentication & OSINT (Open Source Intelligence)

2.6.1 Hardcoded Credentials in Client-Side Code

- **Severity: Critical**
- **OWASP Category:** A07:2021-Identification and Authentication Failures
- **Description:** Administrative credentials were found hardcoded in the application's JavaScript bundles, visible to any user via the browser debugger.
- **Exploitation Steps:**
    1. Inspected **main.js** using **Firefox Developer Tools**.
    2. Located a **hardcoded** email/password string.

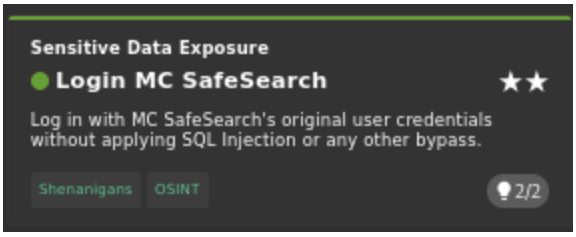
3. **Result:** Successfully authenticated as the user, bypassing all security controls.

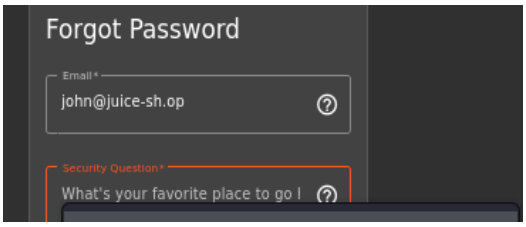


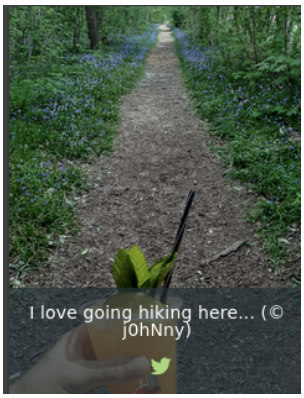


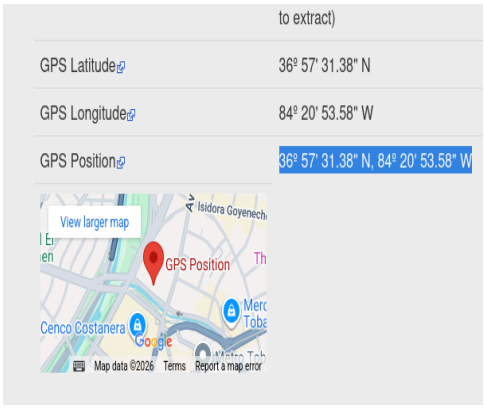
2.6.2 Compromise via Open Source Intelligence (OSINT)

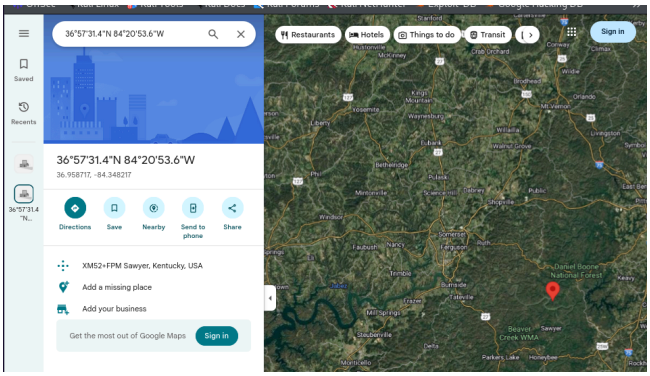
- **Severity: High**
- **Description:** Multiple high-privileged accounts were compromised by leveraging publicly available information (social media, metadata, and pop culture references) to guess passwords and answer security questions.
- **Findings:**
    1. **MC SafeSearch:** Password ("Mr. N00dles") was derived by analyzing the lyrics of the user's music video to answer the "Pet Name" security question.
    2. **John's Account:** Location metadata from an uploaded photo revealed "Daniel Boone National Forest." This answer was used to reset the account password via the "Security Question" bypass.
    3. **Emma's Account:** Visual analysis of a photo on the "Photo Wall" revealed "ITsec" written on a whiteboard, which answered the security question regarding her "First Job."

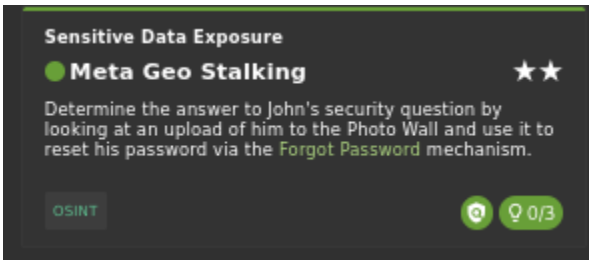

Sensitive Data Exposure
Login MC SafeSearch
★★
Log in with MC SafeSearch's original user credentials without applying SQL Injection or any other bypass.
Shenanigans
OSINT
2/2


Forgot Password
Email *
john@juice-sh.op
Security Question *
What's your favorite place to go I


I love going hiking here... (© jOhNny)


to extract)
GPS Latitude
36° 57' 31.38" N
GPS Longitude
84° 20' 53.58" W
GPS Position
36° 57' 31.38" N, 84° 20' 53.58" W
View larger map
GPS Position
Cenco Costanera
Google
Map data ©2026
Terms
Report a map error


36°57'31.4"N 84°20'53.6"W
36.958717, -84.348217
Directions
Save
Nearby
Send to phone
Share
XM52+FPM Sawyer, Kentucky, USA
Add a missing place
Add your business
Get the most out of Google Maps
Sign in
Restaurants
Hotels
Things to do
Transit
Sign in

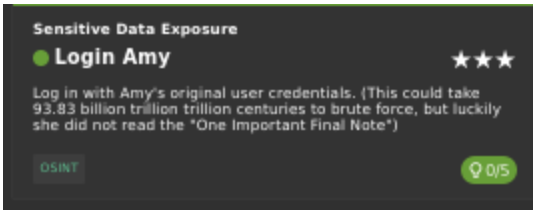

Sensitive Data Exposure
Meta Geo Stalking
★★
Determine the answer to John's security question by looking at an upload of him to the Photo Wall and use it to reset his password via the Forgot Password mechanism.
OSINT
0/3

2.6.3 Crypto-Asset Takeover via Seed Phrase Leakage

- **Severity: Critical**
- **Description:** A cryptocurrency wallet seed phrase was leaked in fragmented parts across public feedback forms.
- **Exploitation Steps:**
    1. Scraped the /administration and "About Us" pages for **Review text**.
    2. Reassembled the mnemonic phrase: *"**purpose betray marriage blame crunch monitor spin slide donate sport lift clutch**"*.
    3. Converted the mnemonic to a Private Key (**0x5bcc3e9d38baa06e7bfaab80ae5957bbe8ef059e640311d7d6d465e6bc948e3e**) and took over the user's NFT assets.

2.6.4 Pattern-Based Password Deduction (Amy's Account)

- **Severity: High**
- **Description:** The user **amy@juice-sh.op** utilized a password pattern based on a specific compliance article (GRC "Perfect Passwords"). While technically complex, the pattern was predictable based on the user's background context and public references.
- **Technical Analysis:**
    1. Identified the user's relationship context (Character "**Kif**" from **Futurama**).
    2. Located the "**One Important Final Note**" in the referenced GRC article, which suggests padding passwords with dots.
    3. **Password Construction:** Replaced the example **D0g...** with the user's partner **K1f...** while maintaining the required length.
    4. **Recovered Credential: K1f.....**
- **Impact:** Demonstrates that complexity rules (length/characters) are **ineffective** if the generation *logic* is predictable.



3.0 Remediation Strategy

To secure the OWASP Juice Shop application against the identified vulnerabilities, the following remediation steps are recommended:

1. **Input Validation (Defense against SQLi/XSS):**
- Implement strict Allow-List validation on all user inputs.
- Use **Parameterized Queries (Prepared Statements)** for all database interactions to prevent SQL Injection.
- Implement Context-Aware Encoding for output to neutralize Cross-Site Scripting (XSS) payloads.
2. **Access Control Enforcement:**
- Implement robust server-side role checks (**RBAC**).
- Ensure that administrative routes (e.g., /administration) verify the user's session token and role **before** rendering any data or interface elements.
- Replace sequential identifiers (like Basket IDs) with **UUIDs** to prevent Insecure Direct Object Reference (IDOR) attacks.
3. **Secrets Management:**
- **Remove** all **hardcoded** credentials and secrets from client-side code (main.js).
- Use environment variables and secure vaults for sensitive keys.
- **Disable** directory listing on the web server to prevent file enumeration in /ftp.
4. **Security Misconfiguration:**
- **Remove** backup files (.bak) from the production environment.
- Validate file extensions strictly on the server side, ensuring that "**Null Byte**" bypasses are rejected.

4.0 Technical Lessons Learned

Beyond the specific vulnerabilities identified, this assessment highlighted several recurring themes regarding application security architecture:

- **Never Trust the Client:** The majority of critical flaws (Basket manipulation, Review forgery, Hardcoded credentials) stemmed from the server trusting data sent from the browser. Security controls must always be enforced on the backend API, never solely in the frontend JavaScript.
- **Obscurity is Not Security:** Hiding administrative paths (like /score-board or /ftp) or using weak encoding (Base64/ROT13) provided no real protection. Tools like Burp Suite and directory brute-forcers easily bypass these "hiding" spots.
- **"Soft Deletion" Risks:** Marking database rows as "**deleted**" (e.g., deletedAt: 2014...) without actually removing them allows attackers to recover sensitive data via SQL Injection. Sensitive data should be permanently purged or moved to a disconnected archive.
- **Input Sanitization is Non-Negotiable:** A single missing validation check on the login form compromised the entire administrative tier. This reinforces that Input Validation must be applied globally to every parameter, header, and cookie.

5.0 Conclusion & Strategic Perspective

This assessment highlights that while perimeter security is essential, application-layer flaws present a massive risk. A single SQL injection provided administrative access, bypassing all network-level controls.

Professional Reflection: While this engagement focused on web-specific vectors, my analysis confirms that my primary strength and interest lie in **Infrastructure and Network Security**. The "Web3 Sandbox" and "Null Byte" findings demonstrated my ability to understand underlying system architecture and server configurations. However, I found the client-side manipulation (XSS) less engaging than the direct server-side exploitation found in my previous vsftpd work.

Future Focus: My future research will focus on **Post-Exploitation**—specifically how to pivot from a compromised web server (like this one) into an internal Active Directory environment to achieve total network dominance.