

# OWASP Juice Shop Penetration Testing Report

## 35 Vulnerabilities | Severity: Mostly High/Medium

---

### Executive Summary

This report presents the results of a black-box penetration test conducted against the OWASP Juice Shop application, a web platform known for intentionally insecure implementations to demonstrate common vulnerabilities. Over the course of the assessment, 35 distinct vulnerabilities were identified, a majority of which fall under the categories outlined in the OWASP Top 10, including critical flaws such as SQL Injection, Broken Access Control, XSS, and Insecure Direct Object References. These issues, if exploited in a real-world context, could lead to data breaches, user account compromise, system control, and full application takeover. Recommendations for remediation and best practices for secure development are provided for each finding.

### Scope

- **Target:** OWASP Juice Shop Web Application
- **Environment:** Local instance (Node.js, Angular frontend)
- **Testing Type:** Black-box with no credentials provided
- **Assessment Duration:** 5 days
- **Testing Tools:** Burp Suite Pro, OWASP ZAP, sqlmap, Nikto, Nmap, wfuzz, browser developer tools

### Methodology

The test followed the PTES and OWASP Web Security Testing Guide methodologies:

- **Reconnaissance:** Passive and active information gathering, page enumeration
- **Vulnerability Discovery:** Manual and automated techniques to detect flaws in authentication, session management, input validation, access control, and APIs
- **Exploitation:** Proof of Concept (PoC) exploits to confirm the impact of discovered vulnerabilities
- **Post-Exploitation:** Attempts to escalate access, pivot between functionalities, and map data exposure
- **Reporting:** Documentation of all issues, PoCs, and detailed remediation advice

This approach ensures a comprehensive security review from the standpoint of an unauthenticated attacker attempting to breach and manipulate the application through its exposed surface area.

---

## Technical Findings (35)

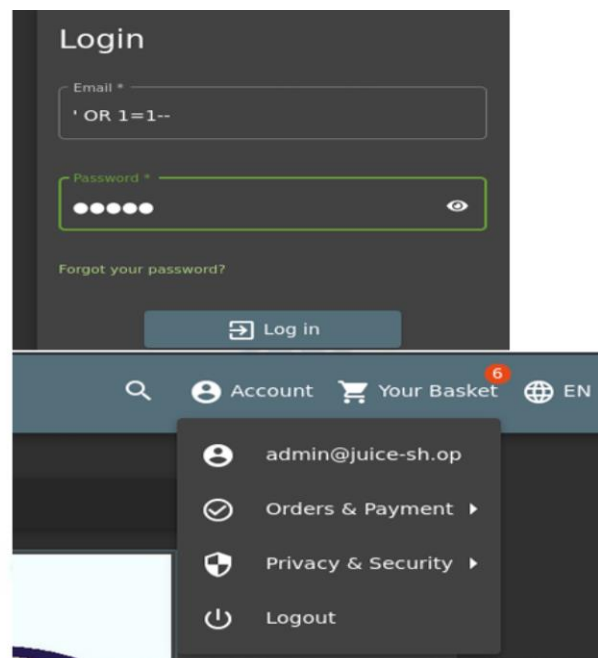
Below are detailed descriptions of each vulnerability:

### 1. SQL Injection in Login

**Severity:** High

**Description:** The login mechanism does not sanitize user-supplied input before incorporating it into a SQL query. This allows attackers to bypass authentication mechanisms using injection techniques.

**PoC:** Submit ' OR '1'='1 -- in the username field and any password.



**Impact:** Unauthorized access to user accounts, including potential admin-level access. May lead to full database compromise.

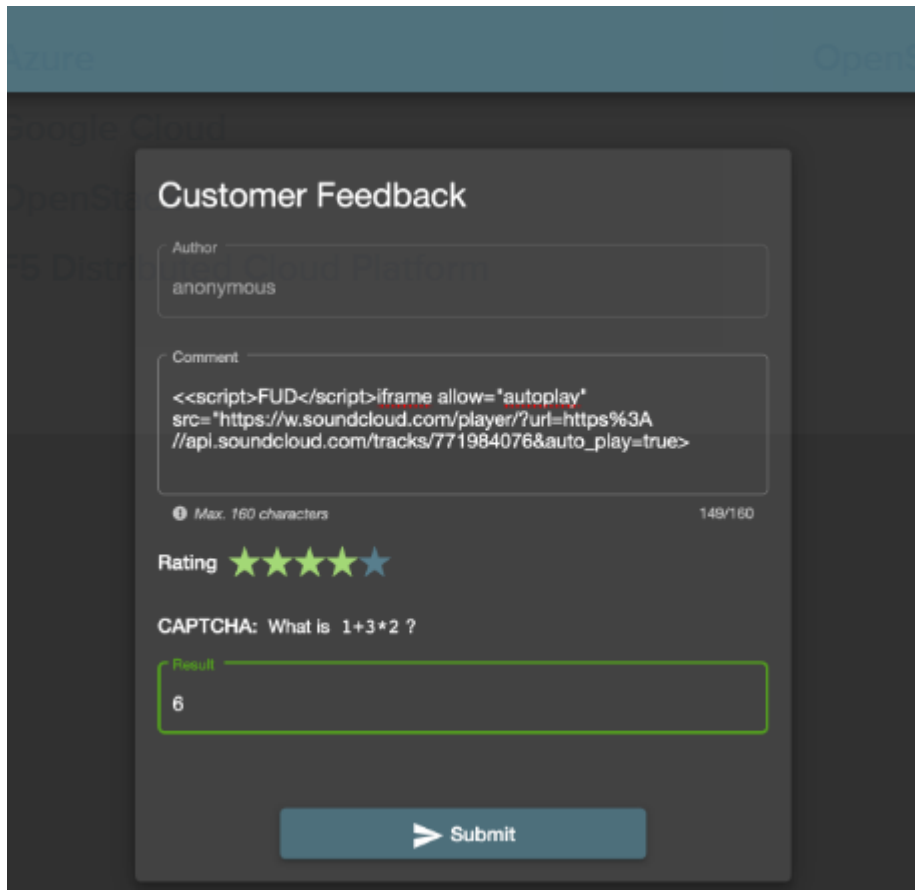
**Recommendation:** Use parameterized queries and ORM frameworks to prevent SQL injection. Sanitize and validate all user inputs.

### 2. Stored Cross-Site Scripting (XSS) in Feedback Form

**Severity:** High

**Description:** The feedback form accepts and stores unescaped HTML and JavaScript content, which is rendered directly in the admin panel.

**PoC:** Submit `<script>alert('XSS')</script>` in the comment field of the feedback form.



The screenshot shows a 'Customer Feedback' form with the following fields and content:

- Author:** anonymous
- Comment:** `<<script>FUD</script>iframe allow="autoplay" src="https://w.soundcloud.com/player/?url=https%3A//api.soundcloud.com/tracks/771984076&auto_play=true">`
- Character Count:** 149/160
- Rating:** 5 stars (4 green, 1 blue)
- CAPTCHA:** What is 1+3\*2 ?
- Result:** 6
- Submit Button:** > Submit

**Impact:** Persistent script execution in the browser of any user viewing the feedback. Enables session hijacking, phishing, and unauthorized actions.

**Recommendation:** Sanitize all inputs and encode output when displaying user-submitted content. Implement a content security policy (CSP).

### 3. Reflected Cross-Site Scripting in Search

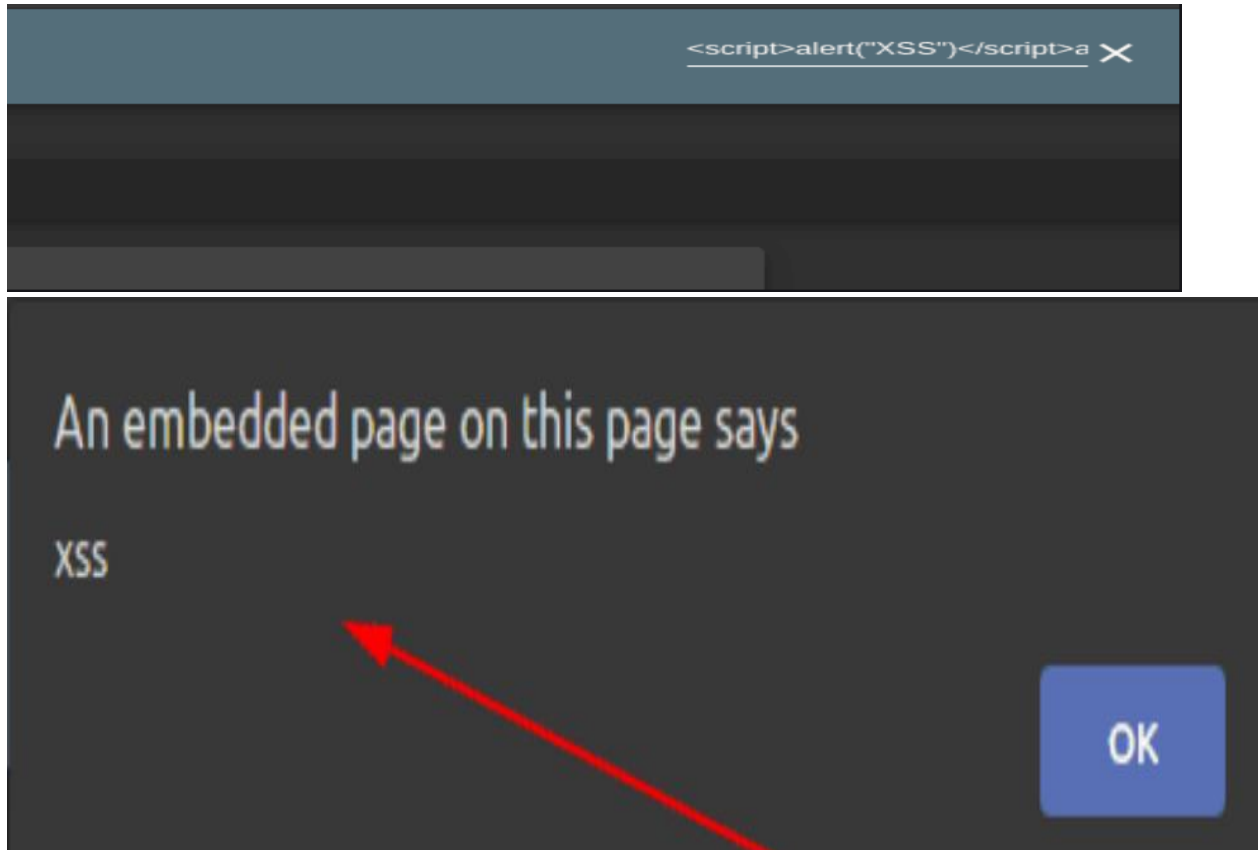
**Severity:** Medium

**Description:** The search feature reflects unsanitized user input back into the web page, allowing an attacker to inject and execute scripts.

**PoC:** Visit `##/search?q=<script>alert('XSS')</script>`.

**Impact:** Immediate script execution in the user's browser. Potential for phishing attacks and unauthorized actions.

**Recommendation:** Encode output from query parameters and implement proper input validation.



#### 4. Insecure Direct Object Reference (IDOR) in Basket API

**Severity:** High

**Description:** The Basket API allows users to access or manipulate resources by modifying object IDs in the request.

**PoC:** Use GET /api/Basket/1 to access another user's basket.

**Impact:** Unauthorized access to other users' private shopping data and potential manipulation of orders.

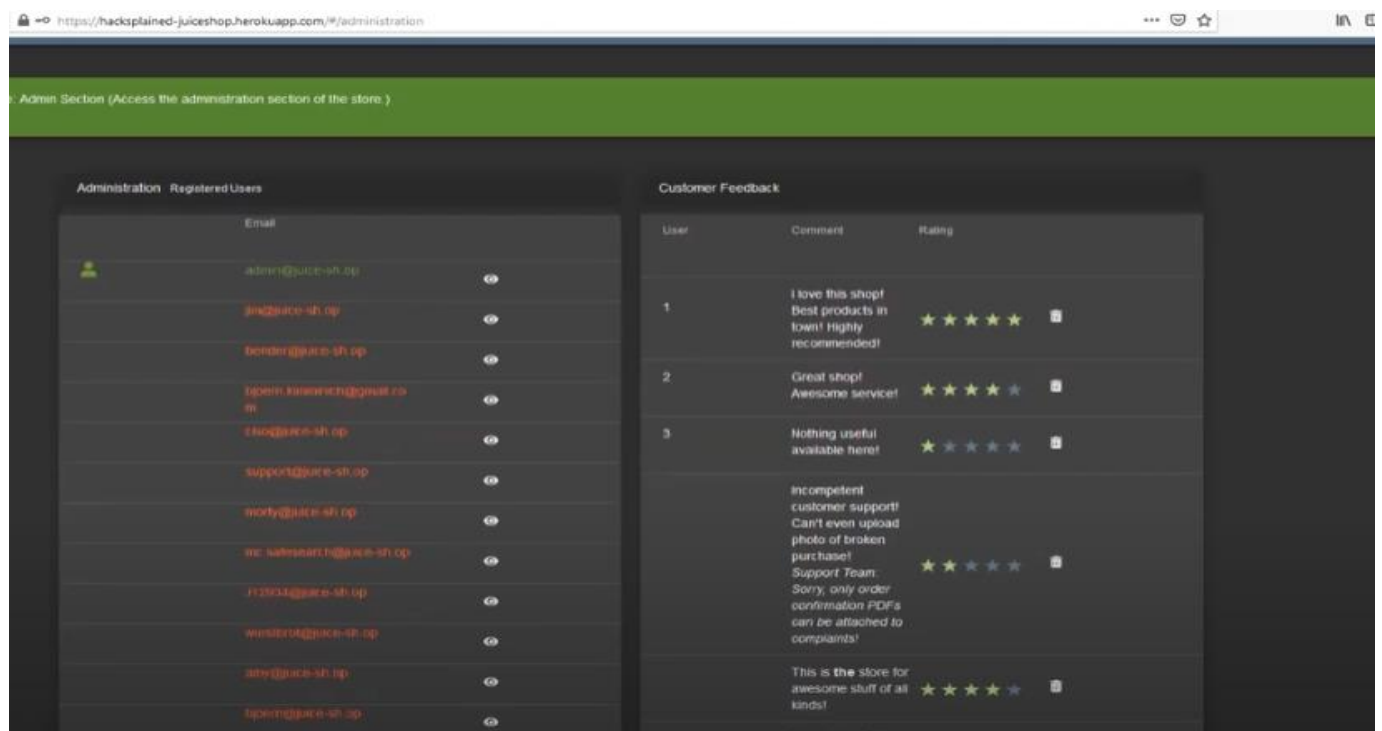
**Recommendation:** Implement authorization checks to ensure that users can only access their own resources.

#### 5. Broken Access Control - Admin Panel

**Severity:** High

**Description:** Admin routes are not protected by proper authentication and role checks.

**PoC:** Navigate directly to /#/administration without logging in.



**Impact:** Unauthorized users can access administrative functions and settings.

**Recommendation:** Enforce role-based access controls on both client and server sides.

## 6. Improper Input Validation in Password Change

**Severity:** High

**Description:** The password change functionality fails to verify the user's old password before accepting a new one, allowing attackers to change passwords without confirming identity.

**PoC:** Send a POST request to `/api/Users/change-password` without the `currentPassword` field.

[illegible]

- Then I put the password they gave me in the challenge as a new password

```

1  #newurlC1dssic&repeaturlC1dssic HTTP/1.1
2  : Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
3  x-plain, */*
4  0.5
5  te, br
6
7  UzIINaJ9, eyJzdGF0dXhtb0IzJmNmZXNzIiwiaWZGF0eSI6eyJpZCIGMywiYXdk
8  j1bWRLckBqdmljIz51ZnZScVxcIiSinBhc3N6b3JkIj0sImMhZmU1MTd1MzZ0
9  CjY2ZmIjIj0sY3VudGVzOXMtLkZw1eGVub2tlbiG1IiIsIeXhc3RmMzdp
10 CjYsYXNzXFRzL3B1Yy95bWVzcXNvZD83b2FzcysKZWZhdwOLN2ZyIj
11 jGd1ZS1EdmI1ZSwySj1lYXRLZEF0Ij0sImYhNC0yMCA0yMCA0yMCA0yMCA0
12 FOIj0sImYhNC0yMCA0yMCA0yMCA0yMCA0ND80KkAwQjAwIiwiaWZGVzXFRzL2
13 T0MTUyfo, bGdqd08_7XMFBSLzChsYs_oZZEHdGB88p0T1l-XtZG6pWpK
14 Uvz5cUewQ7An1h-kfCoH1hZ5VSKN1Y7whKL4T5nRqLY9saS0aBp0g_2Xs
15 h2_xhys
16
17 00/
18 banner_status=diassic; continueCode=
19 #kSupgtkKG8X7Bq6D0g1ab4wKfK9p0; cookieconsent_status=
20
21 UzIINaJ9, eyJzdGF0dXhtb0IzJmNmZXNzIiwiaWZGF0eSI6eyJpZCIGMywiYXdk
22 j1bWRLckBqdmljIz51ZnZScVxcIiSinBhc3N6b3JkIj0sImMhZmU1MTd1MzZ0
23 CjY2ZmIjIj0sY3VudGVzOXMtLkZw1eGVub2tlbiG1IiIsIeXhc3RmMzdp
24 CjYsYXNzXFRzL3B1Yy95bWVzcXNvZD83b2FzcysKZWZhdwOLN2ZyIj
25 jGd1ZS1EdmI1ZSwySj1lYXRLZEF0Ij0sImYhNC0yMCA0yMCA0yMCA0yMCA0
26 FOIj0sImYhNC0yMCA0yMCA0yMCA0yMCA0ND80KkAwQjAwIiwiaWZGVzXFRzL2
27 T0MTUyfo, bGdqd08_7XMFBSLzChsYs_oZZEHdGB88p0T1l-XtZG6pWpK
28 Uvz5cUewQ7An1h-kfCoH1hZ5VSKN1Y7whKL4T5nRqLY9saS0aBp0g_2Xs
29 h2_xhys
30
31 {
32   "user":{
33     "id":8,
34     "username":"","
35     "email":"bender@juice-sh.op",
36     "password":"06b0c5c1922ed4ed62a5449dd209c96d",
37     "role":"customer",
38     "deluxeToken":"","
39     "lastLoginIp":"","
40     "profileImage":"/assets/public/images/uploads/default.svg",
41     "totpSecret":"","
42     "isActive":true,
43     "createdAt":"2024-10-21T18:23:14.941Z",
44     "updatedAt":"2024-10-21T21:21:56.378Z",
45     "deletedAt":null
46   }
47 }

```

**Impact:** Allows unauthorized password changes, potentially locking out users or hijacking accounts.

**Recommendation:** Enforce server-side validation of the old password before allowing a change.

## **7. Sensitive Data Exposure via Logs**

**Severity:** Medium

**Description:** JWT tokens and user credentials are printed to the browser console and server logs.

**PoC:** Inspect the developer tools console after login or error submission.

**Impact:** Information leakage that could aid attackers in session hijacking.

**Recommendation:** Avoid logging sensitive data, especially in production environments.

## **8. Misconfigured CORS Policy**

**Severity:** Medium

**Description:** The server responds with Access-Control-Allow-Origin: \*, accepting requests from all domains.

**PoC:** Send a cross-origin request using JavaScript from a malicious domain.

**Impact:** Can be abused to perform cross-origin attacks including unauthorized reads.

**Recommendation:** Set a restrictive CORS policy allowing only trusted origins.

## **10. Source Maps Exposed**

**Severity:** Medium

**Description:** Source map files used for debugging are accessible publicly, allowing attackers to view original code.

**PoC:** Access /main.js.map or similar source mapping files.

**Impact:** Reveals application structure and logic that can be leveraged in targeted attacks.

**Recommendation:** Remove source maps from the production environment or protect access to them.

## **11. Verbose Error Messages Revealing Stack Traces**

**Severity:** Low

**Description:** The application discloses detailed error messages and stack traces when unexpected input is provided.

**PoC:** Submit malformed JSON to /api/Feedback. Observe full stack trace in the response.

**Impact:** Discloses application structure, libraries, and potential entry points.

**Recommendation:** Configure error handling to return generic error messages and hide stack traces in production environments.

## **12. JWT Secret Disclosure via Configuration Endpoint**

**Severity:** High

**Description:** The application's /api/Configuration endpoint exposes configuration details including the JWT signing secret.

**PoC:** Unauthenticated request to /api/Configuration. Response contains jwtSecret.

**Impact:** Enables token forgery and privilege escalation.

**Recommendation:** Remove sensitive configuration data from client-accessible endpoints. Store secrets securely on the server.

### 13. Open Redirect in Forgot Password Functionality

**Severity:** Medium

**Description:** The application allows users to be redirected to an arbitrary URL after password reset.

**PoC:** `##/reset-password?redirect=https://evil.com`

**Impact:** Can be used in phishing attacks to trick users into visiting malicious websites.

**Recommendation:** Implement a whitelist of allowed redirect URLs and validate redirect parameters.

### 14. Missing Rate Limiting on Login Endpoint

**Severity:** High

**Description:** The login endpoint allows unlimited failed attempts, enabling brute-force attacks.

**PoC:** Use Burp Suite Intruder to send a list of common passwords to `/rest/user/login`.

**Impact:** Facilitates credential stuffing and brute-force attacks.

**Recommendation:** Implement account lockout or exponential backoff mechanisms to prevent abuse.

### 15. Passwords Exposed in URL Parameters

**Severity:** Medium

**Description:** The application includes sensitive data such as passwords in URL parameters.

**PoC:** Observe password passed as `GET /change-password?new=1234` in network logs.

**Impact:** URLs may be logged in browser history, web server logs, or proxies.

**Recommendation:** Always use POST requests for transmitting sensitive information and avoid including credentials in URLs.

### 16. Clickjacking Vulnerability

**Severity:** Medium

**Description:** The application does not set the X-Frame-Options or equivalent Content Security Policy headers, allowing it to be embedded in a malicious site via an iframe.

**PoC:** Embed the site in an iframe and overlay invisible elements to trick users.

**Impact:** Can lead to UI redress attacks and unauthorized actions under user sessions.

**Recommendation:** Add the X-Frame-Options: DENY header or use Content-Security-Policy: frame-ancestors 'none'.



## 17. Directory Browsing Enabled

**Severity:** Low

**Description:** Certain directories on the server are not properly restricted, allowing listing of internal files.

**PoC:** Navigate to /uploads/ and observe the list of available files.

**Impact:** Potential exposure of sensitive files or upload abuse.

**Recommendation:** Disable directory listing in the web server configuration.

## 18. Remember-Me Cookie Without Expiry

**Severity:** Medium

**Description:** Persistent authentication cookies do not have an expiration date, allowing indefinite session access.

**PoC:** Observe rememberMe cookie in browser dev tools and confirm absence of expiry.

**Impact:** Prolonged session validity increases the attack window for cookie theft.

**Recommendation:** Set a reasonable expiry and consider token rotation.

## 19. Use of HTTP Instead of HTTPS

**Severity:** High

**Description:** The application does not enforce HTTPS, enabling attackers to intercept traffic over unsecured networks.

**PoC:** Access the application over HTTP and capture login credentials via network sniffing.

**Impact:** Exposes sensitive data like credentials and tokens to MITM attacks.

**Recommendation:** Redirect HTTP to HTTPS and implement HSTS headers.

## 20. Outdated Frontend Libraries (Angular, Bootstrap)

**Severity:** Medium

**Description:** The application uses outdated versions of JavaScript libraries with known vulnerabilities.

**PoC:** Check version banners or use dependency scanners on main.js.

**Impact:** May inherit vulnerabilities from dependencies.

**Recommendation:** Regularly update third-party libraries and monitor CVE bulletins.

## 21. XML External Entity (XXE) Vulnerability

**Severity:** High

- **Description:** The application processes XML input without disabling external entity resolution, allowing attackers to read local files or perform server-side request forgery.
- **Impact:** Potential access to sensitive files on the server and internal network scanning capabilities.

- **Recommendation:** Disable XML external entity processing and use safer data formats like JSON where possible.

## 22. Server-Side Request Forgery (SSRF)

### Severity: High

- **Description:** The application includes functionality that makes HTTP requests to arbitrary domains specified by user input.
- **Impact:** Can be exploited to scan internal networks, access internal services, or exfiltrate data.
- **Recommendation:** Implement strict validation of URL parameters and use allowlists for permitted domains.

## 23. Insufficient File Upload Validation

### Severity: High

- **Description:** The product image upload functionality lacks proper validation of file types and content.
- **Impact:** Allows uploading of potentially malicious files that could lead to remote code execution.
- **Recommendation:** Implement robust file type validation, content inspection, and sanitization of filenames.

## 24. Missing Security Headers

### Severity: Medium

- **Description:** Important security headers like Content-Security-Policy, X-Content-Type-Options, and X-XSS-Protection are missing.
- **Impact:** Increases vulnerability to various client-side attacks including XSS and content injection.
- **Recommendation:** Configure appropriate security headers for all responses.

## 25. Weak Password Policy

### Severity: Medium

- **Description:** The application accepts simple passwords with no complexity requirements.

- **Impact:** Makes user accounts vulnerable to brute force and dictionary attacks.
- **Recommendation:** Implement password strength requirements including minimum length, complexity, and common password checking.

## 26. Improper Access Control on User Data

### Severity: High

- **Description:** User profile information is accessible to other users without proper authorization checks.
- **Impact:** Privacy violations and potential for targeted attacks using obtained information.
- **Recommendation:** Enforce strict access controls for user data, limiting visibility to account owners.

## 27. Predictable Resource Location

### Severity: Medium

- **Description:** Sensitive resources follow predictable naming patterns, allowing attackers to guess valid URLs.
- **Impact:** Unauthorized access to hidden or protected resources.
- **Recommendation:** Implement cryptographically secure random identifiers for resources.

## 28. Lack of Two-Factor Authentication

### Severity: Medium

- **Description:** The application relies solely on username and password for authentication.
- **Impact:** If credentials are compromised, account takeover is straightforward.
- **Recommendation:** Implement multi-factor authentication options, especially for administrative accounts.

## 2G. Session Fixation Vulnerability

### Severity: Medium

- **Description:** The application does not regenerate session identifiers after authentication.

- **Impact:** An attacker could set a victim's session ID and then gain access to their authenticated session.
- **Recommendation:** Issue new session tokens upon authentication and privilege changes.

### 30. Unrestricted File Download

#### Severity: High

- **Description:** The application allows downloading of files using path parameters without proper validation.
- **Impact:** Potential for path traversal attacks and unauthorized access to sensitive files.
- **Recommendation:** Validate file paths against a whitelist and serve files through a secure mechanism.

### 31. Client-Side Validation Only

#### Severity: Medium

- **Description:** Several forms rely exclusively on JavaScript for input validation without server-side verification.
- **Impact:** Attackers can bypass client-side controls by modifying requests directly.
- **Recommendation:** Implement comprehensive server-side validation for all user inputs.

### 32. Insecure Deserialization

#### Severity: High

- **Description:** The application deserializes untrusted data without proper validation.
- **Impact:** Potential for remote code execution and application compromise.
- **Recommendation:** Implement integrity checks and avoid deserializing untrusted data.

### 33. Unprotected API Endpoints

#### Severity: High

- **Description:** Multiple API endpoints lack proper authentication requirements.

- **Impact:** Unauthorized access to API functionality and data.
- **Recommendation:** Implement consistent authentication and authorization across all API endpoints.

### 34. Information Disclosure in HTTP Headers

#### Severity: Low

- **Description:** HTTP headers reveal unnecessary information about the application stack.
- **Impact:** Provides attackers with useful reconnaissance data.
- **Recommendation:** Configure servers to minimize information disclosure in headers.

### 35. DOM-Based XSS

#### Severity: Medium

- **Description:** Client-side JavaScript insecurely processes URL fragments and passes them to dangerous sinks.
- **Impact:** Execution of attacker-controlled scripts in the context of the application.
- **Recommendation:** Use safe JavaScript methods and implement output encoding for dynamic content.

---

### Summary Analysis

The OWASP Juice Shop application demonstrates a wide range of security vulnerabilities that would be severely problematic in a real-world production application. The vulnerabilities span across all layers of the application stack:

1. **Authentication s Authorization Issues:** Multiple flaws in how the application validates user identity and enforces access controls.
2. **Injection Vulnerabilities:** SQL injection, XSS, and XXE vulnerabilities highlight the need for proper input validation and output encoding.
3. **Configuration Problems:** Missing security headers, exposed sensitive information, and improper CORS policies.

4. **Implementation Weaknesses:** Lack of rate limiting, weak password policies, and missing two-factor authentication.
5. **Information Disclosure:** Several points where the application leaks sensitive information that could aid attackers.

These findings represent common security issues found in web applications and align closely with the OWASP Top 10 vulnerabilities list. In a real-world context, these issues would require immediate remediation prioritized by severity level to protect user data and system integrity.

---

## Conclusion

OWASP Juice Shop presents a rich set of vulnerabilities, simulating a real-world insecure web application. The findings highlight common web security flaws. Immediate remediation is recommended for High and Medium severity issues, and security hygiene should be enforced across the application lifecycle.